Short Term Memory in Echo State Networks

Article · January 2002

CITATIONS
310

READS
1,146

1 author:

Herbert Jaeger
Jacobs University
85 PUBLICATIONS **5,896** CITATIONS

Short term memory in echo state networks [1]

Herbert Jaeger
Fraunhofer Institute for Autonomous Intelligent Systems

May 28, 2002

[1]Appeared as technical report, GMD Report 152, GMD - Forschungszentrum Informationstechnik GmbH. Publication date March 28, 2002.

Abstract. The report investigates the short-term memory capacity of echo state recurrent neural networks. A quantitative measure MC of short-term memory capacity is introduced. The main result is that MC $\leq N$ for networks with linear output units and i.i.d. input, where N is network size. Conditions under which these maximal memory capacities are realized are described. Several theoretical and practical examples demonstrate how the short-term memory capacities of echo state networks can be exploited for dynamic pattern recognition and stochastic sequence modeling tasks.

key words: recurrent neural networks, echo state networks, short-term memory, supervised learning

Zusammenfassung. Die Kurzzeitgedächtnis-Kapazität von Echo-State-Netzwerken wird untersucht. Ein quantitatives Mass MC für diese Kapazität wird eingeführt. Das Hauptresultat ist MC $\leq N$ für Netze mit linearen Ausgabeneuronen und unabhängig identisch verteiltem Input, wobei N die Netzwerkgrösse ist. Es werden Bedingungen beschrieben, unter denen die maximale Gedächtnis-Kapazität erreicht wird. Eine Reihe von theoretischen und praktischen Beispielen demonstrieren, wie das Kurzzeitgedächtnis von Echo-State Netzen für die Klassifikation dynamischer Muster und die Modellierung stochastischer Sequenzen ausgenutzt werden kann.

Stichwörter: rekurrente neuronale Netze, Echo-State Netze, überwachtes Lernen, Kurzzeitgedächtnis

1 Introduction

Echo state networks are a novel approach to analysing and training recurrent neural networks (RNNs). It leads to a fast, simple and constructive algorithm for supervised training of RNNs. A detailed introduction to the approach is given in [9]. The present article, while self-contained, is essentially a continuation of that work.

This article is concerned with short-term memory (STM) effects in RNNs. We use the term "short term memory" here to denote memory effects connected with the *transient* activation dynamics of networks, as opposed to (i) the long-term memory effects afforded by synaptic weight changes in learning, and (ii) memory effects brought about by switching phenomena in attractor dynamics.

Many tasks in signal analysis and control require system models with significant STM spans, i.e. the system output $\mathbf{y}(n)$ should depend significantly on the input history $\mathbf{u}(n), \mathbf{u}(n-1), \ldots$. The standard approach in systems engineering to achieve such STM capabilities is to make a finite input history $\mathbf{u}(n), \ldots, \mathbf{u}(n-k)$ available to the model system by sliding window techniques: the current input is "tapped" from a delay line.

Recurrent neural networks offer an alternative solution to STM demands. From a mathematical point of view, RNNs are dynamical systems with a high-dimensional internal state $\mathbf{x}(n)$. When driven by external input $\mathbf{u}(n)$, the state $\mathbf{x}(n)$ preserves some information about the input history. It is therefore not necessary to feed delayed input versions into the network. Even better, it is in principle possible to achieve arbitrarily long memory spans even with small RNNs.

This is witnessed most strikingly by the "long short-term memory" networks investigated by Gers and Schmidhuber (e.g., [6]). Those memory spans are achieved with a highly specialized network architecture which seems to work well in specific tasks (most prominently regular language learning), but poorly on other standard tasks requiring STM, like chaotic systems prediction [5].

More generic approaches to RNN design and training use gradient descent on error surfaces to obtain RNNs from teacher data (overviews: [11], [2]). The STM spans achievable are severely limited by the fact that gradient information has to be propagated iteratively through the network. Since this information degrades quickly through propagation, only a small propagation depth (order of 10) can be efficiently mastered. This effective propagation depth coincides with the effective memory spans achievable.

A recent, novel approach to RNN analysis and training, "echo state networks" [9] holds promise to overcome this situation.

The basic idea of echo state networks is to use a large "reservoir" RNN as a supplier of interesting dynamics from which the desired output is combined. This idea has been independently discovered and investigated under the name of "liquid state machines" by Wolfgang Maass and collaborators [10]. The two approaches are complementary in many respects: (i) Maass et al. analyze the question which input-output dynamics are at all realizable in the class of echo state/liquid state networks, whereas the work on echo state networks focusses on concrete conditions that enable a particular network to function as an echo state network. (ii) The work on liquid state machines is rooted in a biological setting of continuous-time, spiking networks, while the ideas on echo state networks were first conceived in a framework of discrete-time, non-spiking networks in engineering applications. (iii) Maass et al. consider a very general class of "readout" functions for transforming the network state into the desired output signal, which can basically be realized by postprocessing the network state by a feedforward network. By contrast, research on echo state networks has so far concentrated on linear or linear-plus-sigmoid readout functions which can be realized simply by attaching an output unit to a network.

This article investigates the STM capacity of echo state networks. It is organized as follows.

Section 2 gives a brief re-introduction to echo state networks. As a didactic example, it is shown how an echo state network can be trained to function as a delay line.

Section 3 presents the main theoretical findings, including a formal definition MC of STM capacity, and a theorem that gives an upper bound on $\mathrm{MC} \leq N$ for networks with linear output units and i.i.d. input, where N is network size.

Section 4 shows how an echo state network trained as a delay line can serve as a rehearsing mechanism, repeating an input signal over and over, without involving attractor dynamics.

Section 5 treats a number of basic tasks of dynamic pattern recognition, including instances of one-shot learning and discrimination learning. These techniques are then taken to a robotics task: an echo state network is trained to recognize facts like "robot has passed through a door from a corridor into a room" from sensor and motor variables available to the robot.

Section 6 shows how their STM capabilities enable echo state networks to model stochastic symbol processes generated by hidden-Markov models or even more expressive sources. The technique described is applied to a real-world modeling task in Section 7, where an echo state network is trained as a model of a fairytale text.

Section 8 wraps things up with a brief discussion.

2 Echo state networks: a brief introduction

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 column vectors $\mathbf{u}(n) = (u_1(n) \dots u_K(n))^\top$, of internal units $\mathbf{x}(n) = (x_1(n) \dots x_N(n))^\top$, and of output units $\mathbf{y}(n) = (y_1(n) \dots y_L(n))^\top$. Input $\mathbf{u}(n)$ comes from a compact set $U \subset \mathbb{R}^K$ of *admissible* inputs. 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, and in an $L \times (K + N + L)$ matrix $\mathbf{W}^{\text{out}} = (w^{\text{out}}_{ij})$ for the connections to the output units. We do not admit backprojections from the output units to the internal units or connections between output units (unlike in [9] where such connections are included). Note that connections directly from the input to the output units are allowed. We will 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. We remark at this point that the training algorithms for echo state networks adjusts only the output connection weights, which are therefore highlighted in Figure 1.

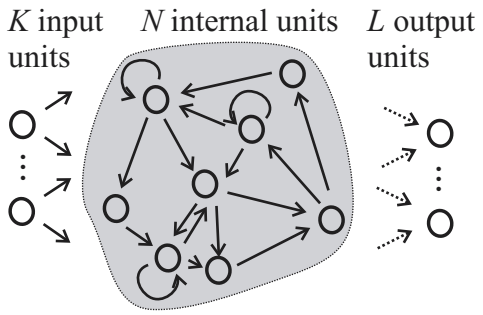


Figure 1: The basic network architecture assumed in this article. Dashed arrows indicate trainable connections.

The activation of internal units is updated according to

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

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

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

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

We introduce a network state update operator T and write $\mathbf{x}(n+h) = T(\mathbf{x}(n), \bar{\mathbf{u}}^h)$ to denote the network state that results from an iterated application of Eq. (1) when the input sequence $\bar{\mathbf{u}}^h = \mathbf{u}(n+1), \ldots, \mathbf{u}(n+h)$ is fed into the network which at time n is in state $\mathbf{x}(n)$.

2.1 Echo states

Under certain conditions (detailed out in [9]), the activation state $\mathbf{x}(n)$ of a recurrent neural network (RNN) is a function of the (infinite) input history $\mathbf{u}(n), \mathbf{u}(n-1), \ldots$ presented to the network. More precisely, under certain conditions there exists an *echo function* $\mathbf{E} = (e_1, \ldots, e_N)$, 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 repeat here a proposition from [9] which gives (a) a sufficient condition for echo states, and (b) a sufficient condition for the non-existence of echo states.

Proposition 1 *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 the network has echo states for all admissible inputs* $\mathbf{u}$. *(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 no echo states if the zero input sequence* $\mathbf{u}(n) = \mathbf{0}$ *is an admissible input sequence.*

These 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 (if $\mathbf{0}$ is an admissible input). My 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.

2.2 Training echo state networks

As mentioned above in passing, echo state networks are trained in a way which only changes the output connection weights. Obtaining an echo state network with a desired (trained) I/O-performance is therefore a two-stage task. First, one procures a RNN, including input units and input connections, which has the echo state property. According to experience, it is sufficient to ensure that $|\lambda_{\max}| < 1$. Second, one attaches output units and trains suitable output connection weights. We now describe the basic idea underlying this strategy.

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 history:

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

Note that (4) is rather restrictive in that we do not admit that the output $\mathbf{y}(n+1)$ depends on previous outputs $\mathbf{y}(n), \mathbf{y}(n-1), \ldots$ in an auto-regressive way. The echo state approach can also deal with auto-regressive systems ([9] actually deals exclusively with that case), but in this article we restrict ourselves to the purely non-auto-regressive case.

We proceed by stating the intuitions of echo state network training in informal terms, using a simple example for illustration. Assume that we have some echo state network. Assume that a long input sequence is presented to the network. Due to the echo state property, after some initial transient the internal unit's activation can be be written as follows (with some liberty of notation)

$$x_i(n) \;\approx\; e_i(\ldots, \mathbf{u}(n), \mathbf{u}(n+1)), \tag{5}$$

where e_i is the echo function of the i-th unit. If the network is suitably inhomogeneous, the various echo functions will significantly differ from each other. For an illustration of this fact, consider a single-channel random input $u(n) = \nu(n)$, where the $\nu(n)$ is an i.i.d. random signal from a uniform

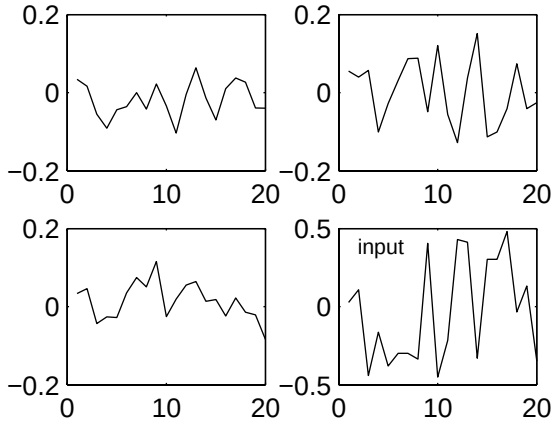


Figure 2: Traces of three arbitrarily selected units of a 20-unit echo state network driven by random input. The input signal is shown in the last trace.

distribution over $[-0.5, 0.5]$. Figure 2 shows traces of some units of a 20-unit echo state network that was driven by this input signal.

Now assume we want to train this 20-unit network to reproduce delayed versions of the input signal at its output units. Concretely, we attach 4 output units to the network and wish to approximate an output vector $\mathbf{y}(n) \approx (\nu(n-1), \nu(n-5), \nu(n-10), \nu(n-15))^\top$ as closely as possible. In terms of Eq. 4, this means we want to approximate the system $\mathbf{y}_{\text{teach}}(n+1) = \mathbf{G}_{\text{teach}}(\ldots, u(n), u(n+1)) = (\nu(n-1), \nu(n-5), \nu(n-10), \nu(n-15))^\top$ through the network output . The idea is to approximate $\mathbf{G}_{\text{teach}}$ through combining the echo functions $\mathbf{E} = (e_1, \ldots, e_{20})$ in a mean square error minimizing way. To this end, recall that the network output is given by Eq. 2. We use $f_l^{\text{out}} = \tanh$, $(l = 1, \ldots, 4)$, which are invertible, therefore (2) is equivalent to

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

Inserting the echo function $\mathbf{E}$ yields

$$(\mathbf{f}^{\text{out}})^{-1}\mathbf{y}(n+1) = \mathbf{W}^{\text{out}}(u(n+1), \mathbf{E}(\ldots, u(n-1), u(n), u(n+1)). \tag{7}$$

Now we determine the weights $\mathbf{W}^{\text{out}}$ such that the 4-element error vector

$$\begin{aligned} \epsilon_{\text{train}}(n) &= (\mathbf{f}^{\text{out}})^{-1}\mathbf{y}_{\text{teach}}(n) - (\mathbf{f}^{\text{out}})^{-1}\mathbf{y}(n) \\ &= (\mathbf{f}^{\text{out}})^{-1}\mathbf{y}_{\text{teach}}(n) - \mathbf{W}^{\text{out}}(u(n), \mathbf{E}(\ldots, u(n-1), u(n)) \end{aligned} \tag{8}$$

is component-wise minimized in the mean square error (MSE) sense, i.e. such that the 4-element mean square error

$$\mathrm{mse}_{\mathrm{train}} = 1/(n_{\max} - n_{\min}) \sum_{i=n_{\min},\ldots,n_{\max}} \epsilon^2_{\mathrm{train}}(n) \tag{9}$$

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 $\mathrm{mse}_{\mathrm{train}}$ as *training error*. Inspection of (8) reveals that minimizing (9) is a simple task of computing a linear regression, to be carried out separately and independently for the 4 output channels.

Concretely, in our little delay learning demo task (i) we let the network run for $n = 0$ to $n_{\max} = 200$, 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, (iii) collect the network states $\mathbf{x}(n)$ from $n_{\min} = 101$ to $n_{\max} = 200$, and (iv) compute the weights $\mathbf{W}^{\mathrm{out}}$ offline from these collected states, such that the error (9) becomes minimal.

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

$$\mathbf{G}' \approx \mathbf{W}^{\mathrm{out}}\mathbf{E} \tag{10}$$

becomes a MSE approximation of $\mathbf{G}'_{\mathrm{teach}}$ by a weighted combination of the echo functions $\mathbf{E}$.

Figure 3 shows the output of the trained network. It appears that up to a delay of 10, the task is basically mastered. We will later investigate findings of this kind in more detail. For now, suffice it to say that we found training errors of $\mathrm{mse}_{\mathrm{train}} = (0.000049, 0.00030, 0.014, 0.062)$. When the trained network was tested, a test error $\mathrm{mse}_{\mathrm{test}} = (0.000047, 0.00035, 0.016, 0.060)$ was obtained (estimated from averaging over a 100-step run).

Two important points should be highlighted:

1. The 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 $\mathbf{G}'_{\mathrm{teach}} \approx \mathbf{W}^{\mathrm{out}}\mathbf{E}$, 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

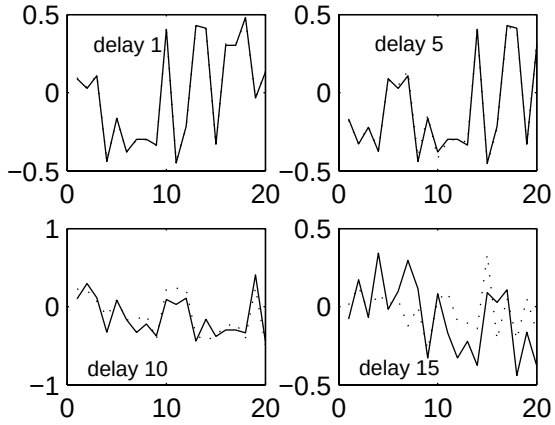


Figure 3: Testing the trained delay network. Correct delayed input signal (solid line) is superimposed on output of trained network (dashed line) for delays 1, 5, 10, 15.

> 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 20-unit network used in the example was randomly connected; weights were set to values of 0, $+0.47$ and -0.47 with probabilities 0.8, 0.1, 0.1 respectively. This means a sparse connectivity of 20 %. (The value of 0.47 for non-null weights resulted from a global scaling such that $\mid \lambda_{\max} \mid \approx 0.9 < 1$ was obtained – see again [9] for details about this scaling).

The input weights were set in an ad hoc decision (without any optimization) to values of +0.1, -0.1 with equal probability.

Calculations were done with the Matlab software package[1]. The linear regression was done by first calculating the pseudoinverse A^- of the matrix A whose rows consist of the network states collected during the training run, and then multiplying A^- with the vector consisting of the teacher outputs of that run to obtain the desired output weight vector.

We conclude the section with a more rigorous and general formulation of the training procedure.

[1] The author gratefully acknowledges the re-implementation of the original Mathematica routines in Matlab by Christina Jager.

Task. Given: a teacher I/O time series $(\mathbf{u}_{\text{teach}}(n), \mathbf{y}_{\text{teach}}(n))_{n=0,\ldots,n_{\max}}$.
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. As a general rule, this RNN should be the larger the more complex the task, or the higher the desired precision, or the longer the required short term memory.

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 $n = 0, \ldots, n_{\max}$:

$$\mathbf{x}(n+1) = \mathbf{f}(\mathbf{W}^{\text{in}}(\mathbf{u}_{\text{teach}}(n+1)) + \mathbf{W}\mathbf{x}(n)). \tag{11}$$

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 $\mathbf{y}_{\text{teach}}(n) = (y_{\text{teach},1}(n), \ldots, y_{\text{teach},L}(n))$, put $\mathbf{G}'_{\text{teach}}(n) = (\mathbf{f}^{\text{out}})^{-1}\mathbf{y}_{\text{teach}}(n)$. Compute output weights $\mathbf{W}^{\text{out}}$ such that the MSE vector $\text{mse}_{\text{train}}$

$$\frac{\sum_{n=n_{\min}}^{n_{\max}} \left(\mathbf{G}'_{\text{teach}}(n) - \mathbf{W}^{\text{out}}(\mathbf{u}_{\text{teach}}(n), \mathbf{E}(\ldots, \mathbf{u}_{\text{teach}}(n-1), \mathbf{u}_{\text{teach}}(n)))\right)^2}{(n_{\max} - n_{\min})} \tag{12}$$

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

3 Analysis of STM capacity

In this section we investigate the short-term memory capacity of echo state networks. The basic question is, if we train an echo state network to generate at its output units delayed versions $\nu(n-k)$ of a single-channel input $\nu(n)$, what memory spans can we expect, and which precision in the outputs?

We first define a quantitative measure MC of STM capacity.

Definition 1 *Let $\nu(n) \in U$ (where $-\infty < n < +\infty$ and $U \subset \mathbb{R}$ is a compact interval) be a single-channel stationary input signal. Assume that we have a RNN, specified by its internal weight matrix $\mathbf{W}$, its input weight (column)*

vector $\mathbf{w}^{in}$ *and the the unit output functions* $\mathbf{f}, \mathbf{f}^{out}$*. The network receives* $\nu(n)$ *at its input unit. For a given delay* k *and an output unit* y_k *with connection weight (row) vector* $\mathbf{w}_k^{out}$ *we consider the determination coefficient*

$$\begin{aligned} d[\mathbf{w}_k^{out}](\nu(n-k), y_k(n)) = & \qquad (13) \\ = \quad & d(\nu(n-k), \mathbf{w}_k^{out} \begin{pmatrix} \nu(n) \\ \mathbf{x}(n) \end{pmatrix} \\ = \quad & \frac{cov^2(\nu(n-k), y_k(n))}{\sigma^2(\nu(n))\sigma^2(y_k(n))}, \end{aligned}$$

where cov denotes covariance and σ^2 *variance.*

1. *The* k*-delay STM capacity of the network is defined by*

$$\mathrm{MC}_k = \max_{\mathbf{w}_k^{out}} d[\mathbf{w}_k^{out}](\nu(n-k), y_k(n)). \qquad (14)$$

2. *The STM capacity of the network is*

$$\mathrm{MC} = \sum_{k=1}^{\infty} \mathrm{MC}_k. \qquad (15)$$

The determination coefficient of two signals is the squared correlation coefficient. It ranges between 0 and 1 and represents the fraction of variance explainable in one signal by the other. Thus, the STM capacity measures how much variance of the delayed input signal can be recovered from optimally trained output units, summed over all delays. Note that the output units do not interfere with one another; arbitrarily many output units y_k can be attached to the same network.

We start by investigating the special case where we have linear output units, namely, where $\mathbf{f}^{\mathrm{out}} = \mathbf{id}$. Our fist goal is to show that if the input signal is i.i.d., then $\mathrm{MC} \leq N$.

With linear output units, the output signal y_k is a linear combination of the signal $\nu(n)$ and the internal unit activations $x_i(n)$ $(i = 1, \ldots, N)$. Denote by $\mathbf{v}(n) = (v_1(n) \ldots v_{N+1}(n))^\top$ the vector of signals $(\nu(n+1)\ x_1(n)\ \ldots\ x_N(n))^\top$. For convenience we assume that the $N+1 \times N+1$ correlation matrix $\mathbf{R} = E[\mathbf{v}(n)\mathbf{v}^\top]$ has full rank (we will later see that we can drop this assumption).

We treat random signals $s(n)$ as vectors (in the vector space of numerical random variables) and introduce the inner product $\langle s(n), s'(n) \rangle =$

$\mathrm{corr}(s(n), s'(n)) = E[s(n)s'(n)]$, which allows us to define $s(n), s'(n)$ to be orthogonal if $\langle s(n), s'(n)\rangle = 0$. Because $\mathbf{R}$ has full rank, the dimension of the signal space S spanned by the v_i is $N+1$. We choose new signals $\tilde{\mathbf{v}} = (\tilde{v}_1(n), \ldots, \tilde{v}_{N+1}(n))^\top$ which constitute an orthonormal basis of this space. (We will generally indicate by $\tilde{\cdot}$ orthonormal vectors and related quantities.) The choice is arbitrary, with the provision that if the constant signal $\mathbf{1}(n) \equiv 1$ is contained in S, we require $\tilde{v}_1(n) = \mathbf{1}(n)$. Note that $\mathbf{1}(n) \in S$ if at least one of the signals v_i has nonzero expectation.

It is easy to see that the change to the new signal basis $\tilde{v}_i(n)$ does not affect the k-delay STM capacity of the network, in the sense that the k-delay STM capacity defined in (14) is equivalently given by

$$\mathrm{MC}_k = \max_{\mathbf{w}_k^{\mathrm{out}}} d(\nu(n-k), \mathbf{w}_k^{\mathrm{out}} \tilde{\mathbf{v}}(n)). \tag{16}$$

By the well-known Wiener-Hopf equation (see e.g. [4] Sec. 3.2 for an introduction), the least mean square error weight vector (it minimizes $E[(\nu(n-k) - \mathbf{w}_k^{\mathrm{out}} \tilde{\mathbf{v}}(n))^2]$) is given by

$$\hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}} = (\tilde{\mathbf{R}}^{-1} \tilde{\mathbf{p}}_k)^\top, \tag{17}$$

where $\tilde{\mathbf{R}}$ is the correlation matrix of the signals $\tilde{v}_i(n)$ and $\mathbf{p}_k = E[\tilde{\mathbf{v}}(n)\, \nu(n-k)]$. $\hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}}$ yields the maximum in (16), i.e. $\mathrm{MC}_k = d(\nu(n-k), \hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}} \tilde{\mathbf{v}}(n))$.

Denote by $\hat{y}_k(n) = \hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}} \tilde{\mathbf{v}}$ the signal obtained when we tap the signals $\tilde{v}_i(n)$ with $\hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}}$. ($\hat{y}_k(n)$ is also the signal that we would get as the trained network's output if we could train the network on an infinite training data set.) We find the following alternative characterization of $\hat{y}_k(n)$:

$$\begin{aligned}
\hat{y}_k(n) &= \hat{\tilde{\mathbf{w}}}_k^{\mathrm{out}}\ \tilde{\mathbf{v}}(n) \ =\ (\tilde{\mathbf{R}}^{-1} \tilde{\mathbf{p}}_k)^\top \tilde{\mathbf{v}}(n) \\
&= (\mathbf{I}\, \tilde{\mathbf{p}}_k)^\top \tilde{\mathbf{v}}(n) \ =\ \tilde{\mathbf{p}}_k^\top \tilde{\mathbf{v}}(n) \\
&= (E[\tilde{v}_1(n)\, \nu(n-k)], \ldots, E[\tilde{v}_{N+1}(n)\, \nu(n-k)])\ \tilde{\mathbf{v}}(n) \\
&= (\langle \tilde{v}_1(n), \nu(n-k)\rangle, \ldots, (\langle \tilde{v}_{N+1}(n), \nu(n-k)\rangle)\ \tilde{\mathbf{v}}(n) \\
&= \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \nu(n-k)\rangle\ \tilde{v}_i(n).
\end{aligned} \tag{18}$$

We continue by making the assumption that the $\nu(n-k)$ (where $k = 1, 2, \ldots$) are i.i.d. We distinguish two cases, (i) $E[\nu(n)] = 0$, (ii) $E[\nu(n)] = a \neq 0$. We treat only the second (more complicated) case. The signal $\nu(n-k)$ can be split into two signals $\nu(n-k) = a\mathbf{1}(n) + \mu(n-k)$, where $E[\mu(n-k)] = 0$. The representation (18) of $\hat{y}_k(n)$ then becomes

$$\hat{y}_k(n) = a\mathbf{1}(n) + \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \tilde{v}_i(n). \tag{19}$$

We compute the variance of $\hat{y}_k(n)$

$$\begin{aligned} \sigma^2(\hat{y}_k(n)) &= \|\hat{y}_k(n) - E[\hat{y}_k(n)]\|^2 \\ &= \|\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \tilde{v}_i(n)\|^2 \\ &= \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle^2 \end{aligned} \tag{20}$$

and the correlation coefficient of $\hat{y}_k(n)$ with $\nu(n-k)$:

$$\begin{aligned} & r(\hat{y}_k(n), \nu(n-k)) = \\ &= \frac{\mathrm{cov}(\hat{y}_k(n), \nu(n-k))}{\sigma(\hat{y}_k(n)) \; \sigma(\nu(n-k))} \\ &= \frac{\langle \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \tilde{v}_i(n), \; \mu(n-k) \rangle}{\sigma(\hat{y}_k(n)) \; \sigma(\nu(n-k))} \\ &= \frac{\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \langle \tilde{v}_i(n), \mu(n-k) \rangle}{\sigma(\hat{y}_k(n)) \; \sigma(\nu(n-k))} \\ &= \frac{\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle^2}{\sigma(\hat{y}_k(n)) \; \sigma(\nu(n-k))} \; = \; \frac{\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \mu(n-k) \rangle^2}{\sigma(\hat{y}_k(n)) \; \sigma(\nu(n-k))}. \end{aligned} \tag{21}$$

Observing that $d = r^2$, from (20) and (21) it follows that

$$d(\hat{y}_k(n), \nu(n-k)) = \frac{\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \mu(n-k) \rangle^2}{\sigma^2(\nu(n-k))}. \tag{22}$$

Combining the findings from (16) to (21), we can compute a boundary on the STM memory capacity, using the notation $\tilde{\mu}(n-k) = \mu(n-k)/\|\mu(n-k)\| = \mu(n-k)/\sigma^2(\nu(n))$ for the normalized signals $\mu(n-k)$:

$$\begin{aligned} \mathrm{MC} &= \sum_{k=1}^{\infty} d(\hat{y}_k(n), \mu(n-k)) \\ &= \sum_{k=1}^{\infty} \frac{1}{\sigma^2(\nu(n))} \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \mu(n-k) \rangle^2 \qquad \text{[use that the } \mu(n-k) \text{ are i.i.d.]} \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{\sigma^2(\mu(n))} \sum_{i=2}^{N+1} \sum_{k=1}^{\infty} \langle \tilde{v}_i(n), \mu(n-k) \rangle^2 \\
&= \frac{1}{\sigma^2(\mu(n))} \sum_{i=2}^{N+1} \sum_{k=1}^{\infty} \sigma^2(\mu(n)) \langle \tilde{v}_i(n), \tilde{\mu}(n-k) \rangle^2 \\
&= \sum_{i=2}^{N+1} \sum_{k=1}^{\infty} \langle \tilde{v}_i(n), \tilde{\mu}(n-k) \rangle^2 \\
&\leq N, \qquad (23)
\end{aligned}
$$

where the last inequality follows from $\sum_{k=1}^{\infty} \langle \tilde{v}_i(n), \tilde{\mu}(n-k) \rangle^2 \leq 1$ (note that the $\tilde{\mu}(n-k)$ are orthonormal).

The case (i) $E[\nu(n)] = 0$ can be treated in a similar fashion. Furthermore, it is clear that if the correlation matrix $\mathbf{R}$ is rank deficient, the STM memory can only be smaller than with a full-rank $\mathbf{R}$. We sum up our insight:

Proposition 2 *The memory capacity for recalling an i.i.d. input by a N-unit RNN with linear output units is bounded by N.*

Both conditions (i.i.d. input and linear output units) are necessary for this bound.

This is clear for the first condition. Consider, for example, the case of a constant input signal $\nu n = \mathbf{1}(n)$. Then, any linear-output network with output weights $\mathbf{w}_k^{\text{out}} = (1, 0, \ldots, 0)$ would "recover" all (identical) delayed versions of the input perfectly, formally yielding an infinite memory capacity. This suggests the following refinement of the definition of STM memory capacity, which we briefly indicate for zero mean input signals. Compute from the original input signals $\nu(n-k)$ a new set of signals $\nu'(n-k) = \nu(n-k) - \pi(\nu(n-k), S(\nu(n-k-1), \nu(n-k-2), ...))$, where $S(\nu(n-k-1), \nu(n-k-2), ...)$ is the signal space spanned by the preceding inputs, and $\pi(\nu, S)$ denotes the projection of a vector ν into a space S. $\nu'(n-k)$ thus is reduced to the information contribution of the signal $\nu(n-k)$ which is novel, i.e. is not already contained in previous inputs. Use $\nu'(n-k)$ instead of $\nu(n-k)$ in the above definitions. The resulting alternative version of Prop. 2 would then not need the i.i.d. condition.

Echo state networks are indeed capable to exploit dependencies in the input data, to achieve a memory capacity exceeding N. For a demonstration, the 20-unit network used above was re-used, but modified a little (now: linear output units, scaling $|\lambda_{\max}| = 0.95$, input connection weights randomly sampled from $\{-0.001, +0.001\}$). The input signal jumped to a new random value every 10 steps, thus introducing temporal dependencies which can be

exploited by the network in the STM task. The network was equipped with 40 output units, which were trained on the delays $1 - 40$. This setup resulted in a memory capacity of $\mathrm{MC} = 25.5 > 20 = N$. Figure 4 shows the network output in the testing phase for output units trained on delays 10, 15, 20, 25.

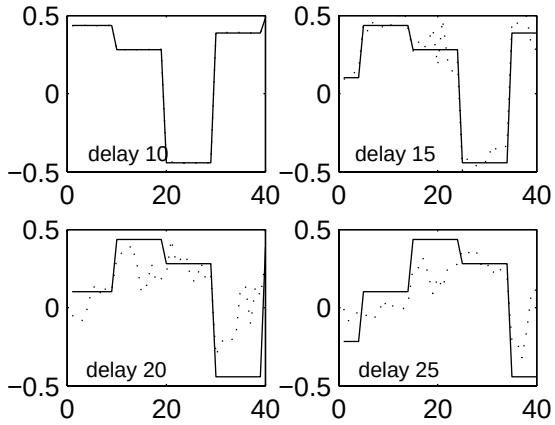


Figure 4: Testing the trained delay network for delays 10, 15, 20, 25. Correct delayed input signal (solid line) is superimposed on output of trained network (dashed line).

Figure 5 shows the "forgetting curve", a plot of the k-delay memory capacities. It exhibits a close-to-100% recall for delays up to 13, followed by an rugged slope that still is significantly above zero at a delay of 40.

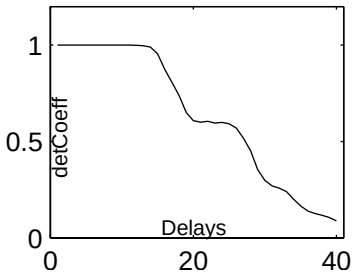


Figure 5: The forgetting curve of the trained network. The plot shows MC_k vs. delay k, estimated numerically over a 3000 step run.

If we drop the other condition from Proposition 2, namely, linear output units, we can likewise obtain memory capacities exceeding N. Specifically, if we go as far as to admit different output functions f_k for different delays,

infinite memory capacity can be obtained even with a single-unit network. Consider a "network" whose single, linear unit x is updated according to $x(n+1) = 1/2\, x(n) + 1/2\, \nu(n+1)$, where $\nu(n)$ is a random binary 0-1-signal. Then, the network state at time n is $x(n) = \sum_{k=0}^{\infty} 2^{-(k+1)} \nu(n-k)$, which, written out in digits to base 2, is just the input history. If we use for f_k the (highly nonlinear) function that picks from the digit-base-2 representation of $x(n)$ the k-th digit, we recover the full input history from the network state in the output units. This is of course an extreme and theoretical example, but it underlines the necessity of the conditions in Proposition 2.

We can shed more light on the STM mechanism in echo state networks when we assume that the network is linear throughout, i.e. when both $\mathbf{f}$ and $\mathbf{f}^{\text{out}}$ are identity functions $\mathbf{id}$. We will show that in this case the forgetting curve is monotonically decreasing, and that generically $\text{MC} = N$. Again, we work under the assumption that the input $\nu(n)$ is i.i.d.

To reduce notational complexity, we assume that $E[\nu(n)] = 0$ (the case $E[\nu(n)] \neq 0$ can be dealt with in a similar fashion as above).

We assume again that the signals $\nu(n), x_j(n)$ (where $j = 1, \ldots, N$) are linearly independent and introduce a set $\tilde{v}_i(n)$ ($i = 1, \ldots, N+1$) of orthonormal signals spanning the same space as the $\nu(n), x_j(n)$ ($j = 1, \ldots, N$), with the convention that $\tilde{v}_1(n) = \nu(n)/\|\nu(n)\|$. The main step toward showing that the forgetting curve decreases monotonically is expressed in the following "masking lemma". It states that in (18) we can replace the input $\nu(n-k)$ by $\hat{y}_{k-l}(n-l)$, where $l \leq k$. Intuitively, this means that any information about $\nu(n-k)$ contained in the current state $\tilde{v}_i(n)$ is the same as the information about $\hat{y}_{k-l}(n-l)$ in the current state – i.e., optimal estimates $\hat{y}_{k-l}(n-l)$ of $\nu(n-k)$ from intermediate states $\tilde{v}_i(n-l)$ "mask" the propagation of $\nu(n-k)$ through the temporal sequence of states $\tilde{v}_i(n-k), \tilde{v}_i(n-k+1), \ldots$ to the present state $\tilde{v}_i(n)$.

Lemma 1 *("Masking lemma") For all $k \geq 1$, for all $0 \leq l \leq k$, it holds that*

$$\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \tilde{v}_i(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l}(n-l) \rangle \; \tilde{v}_i(n). \qquad (24)$$

The first equality in (24) is a variant of (18) (note that $\langle \tilde{v}_1(n), \nu(n-k) \rangle = 0$); the proof of the second equality is given in the Appendix.

The masking lemma is actually stronger that what we need for our purposes. From (24) we only make use of the case $l = 1$, which implies that $\hat{y}_k(n)$ is the projection of $\hat{y}_{k-1}(n-1)$ into the space spanned by the $\tilde{v}_i(n)$. Therefore, $\|\hat{y}_k(n)\| \leq \|\hat{y}_{k-1}(n-1)\|$. By an argument of shift invariance,

$\|\hat{y}_{k-1}(n-1)\| = \|\hat{y}_{k-1}(n)\|$. Thus, we obtain $\|\hat{y}_k(n)\| \leq \|\hat{y}_{k-1}(n)\|$. We observe that the following variant of (22) holds for zero-mean inputs:

$$d(\hat{y}_k(n), \nu(n-k)) = \frac{\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \nu(n-k)\rangle^2}{\sigma^2(\nu(n-k))}. \tag{25}$$

From (25) and (18) we can infer that $d(\hat{y}_k(n), \nu(n-k))$ is proportional to $\|\hat{y}_k(n)\|^2$ by a factor of $1/\sigma^2(\nu(n))$; likewise $d(\hat{y}_{k-1}(n), \nu(n-k+1))$ is proportional to $\|\hat{y}_{k-1}(n)\|^2$ by the same factor. Therefore, $d(\hat{y}_k(n), \nu(n-k)) \leq d(\hat{y}_{k-1}(n), \nu(n-k+1))$. A similar argument yields $d(\hat{y}_{k-1}(n), \nu(n-k+1)) \leq d(\hat{y}_{k-2}(n), \nu(n-k+2))$, etc. The case where the signals $\nu(n), x_j(n)$ (where $j = 1, \ldots, N$) are not linearly independent can be dealt with in an analog fashion, by limiting the argument on the space spanned by those signals. We collect our findings:

Proposition 3 *The k-delay STM capacity for an i.i.d. input of a N-unit linear RNN decreases monotonically with k.*

We stay with linear networks and ask under which conditions the full STM capacity N is achieved. An obvious necessary requirement is that the internal states $x_1(n), \ldots, x_N(n)$ are linearly independent (which amounts to saying the the correlation matrix $\mathbf{R}$ has full rank). We show that this is also a sufficient condition provided the linear network has the echo state property.

Observe that the memory capacity of a linear echo state network is independent of input scaling, so we can assume normalized input $\nu(n) = \nu(n)/\|\nu(n)\| = \tilde{\nu}(n)$.

An iterated expansion of the state update equation (1) represents the internal states in terms of the input history:

$$\begin{aligned} \mathbf{x}(n) &= \mathbf{W}\mathbf{x}(n-1) + \mathbf{w}^{\text{in}}\,\tilde{\nu}(n) \\ &= \sum_{k=0}^{\infty} \mathbf{W}^k \mathbf{w}^{\text{in}}\,\tilde{\nu}(n-k). \end{aligned} \tag{26}$$

(Note that (26) presupposes the echo state property). Let $[\mathbf{w}]_i$ denote the i-th component of a vector $\mathbf{w}$. Putting $\alpha_{ik} = [\mathbf{W}^k \mathbf{w}^{\text{in}}]_i$, (26) can be rewritten component-wise ($i = 1, \ldots, N$):

$$x_i(n) = \sum_{k=0}^{\infty} \alpha_{ik}\,\tilde{\nu}(n-k). \tag{27}$$

Likewise, the input can formally be written as $\tilde{\nu}(n) = 1\tilde{\nu}(n) =: \alpha_{00}\,\tilde{\nu}(n) = \sum_{k=0}^{\infty} \alpha_{0k}\,\tilde{\nu}(n)$, where $\alpha_{0k} = 0$ for $k > 0$.

We rename the signals $\tilde{\nu}(n), x_1(n), \ldots, x_N(n)$ to $v_1(n), \ldots, v_{N+1}(n)$ and obtain, for $i = 1, \ldots, N+1$

$$v_i(n) = \sum_{k=0}^{\infty} \alpha_{(i-1)k}\, \tilde{\nu}(n-k). \tag{28}$$

Now we again assume that the signals $\mathbf{v}(n) = (v_1(n) \ldots v_{N+1}(n))^\top$ are linearly independent, and carry out a coordinate transformation to novel, orthonormal signals $\tilde{\mathbf{v}}(n) = (\tilde{v}_1(n) \ldots \tilde{v}_{N+1}(n))^\top$ with the understanding that $\tilde{v}_1(n) = \tilde{\nu}(n)$. The transformation is effected by a matrix $\mathbf{T}$ defined by $\forall n : \tilde{\mathbf{v}}(n) = \mathbf{T}\mathbf{v}(n)$. Let $\alpha_k = (\alpha_{0k} \ldots \alpha_{Nk})^\top$, and let $\mathbf{T}_i$ denote the i-th row of $\mathbf{T}$. For $i = 2, \ldots, N+1$ derive a representation of $\tilde{v}_i(n)$ in terms of the input history, as follows:

$$\begin{aligned}
\tilde{v}_i(n) &= \sum_{k=0}^{\infty} \mathbf{T}_i \alpha_k \tilde{\nu}(n-k) \\
&= \sum_{k=1}^{\infty} \mathbf{T}_i \alpha_k \tilde{\nu}(n-k) \quad [\text{use } \langle \tilde{v}_i(n), \nu(n) \rangle = 0] \\
&=: \sum_{k=1}^{\infty} \beta_{ik} \tilde{\nu}(n-k).
\end{aligned} \tag{29}$$

Because $\|\tilde{v}_i(n)\| = 1$, it holds that

$$\sum_{k=1}^{\infty} \beta_{ik}^2 = 1. \tag{30}$$

Observing orthogonality of the $\tilde{\nu}(n-k)$ $(k \geq 0)$, from (29) we can conclude

$$\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \tilde{\nu}(n-k) \rangle\, \tilde{v}_i(n) = \sum_{i=2}^{N+1} \beta_{ik}\, \tilde{v}_i(n). \tag{31}$$

Combining (30) with (31) we arrive at our goal:

$$\begin{aligned}
\text{MC} &= \sum_{k=1}^{\infty} d(\hat{y}_k(n), \tilde{\nu}(n-k)) \\
&= \sum_{k=1}^{\infty} \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \tilde{\nu}(n-k) \rangle^2 \quad [\text{use (25)}]
\end{aligned}$$

$$= \sum_{k=1}^{\infty} \|\hat{y}_k(n)\|^2$$
$$= \sum_{k=1}^{\infty} \sum_{i=2}^{N+1} \beta_{ik}^2 = \sum_{i=2}^{N+1} \sum_{k=1}^{\infty} \beta_{ik}^2 = N. \quad (32)$$

To round up, we give a criterium to decide when the signals $\mathbf{v}(n) = (v_1(n) \dots v_{N+1}(n))^\top$ are linearly independent. We consider the first $N+1$ terms in the expansion (28):

$$\begin{pmatrix} \nu(n) \\ \mathbf{x}(n) \end{pmatrix} = \quad (33)$$
$$= \begin{pmatrix} 1 \\ \mathbf{W}^0 \mathbf{w}^{\text{in}} \end{pmatrix} \nu(n) + \begin{pmatrix} 0 \\ \mathbf{W}^1 \mathbf{w}^{\text{in}} \end{pmatrix} \nu(n-1) + \cdots + \begin{pmatrix} 0 \\ \mathbf{W}^N \mathbf{w}^{\text{in}} \end{pmatrix} \nu(n-N) + \cdots$$

Let $\mathbf{M}_0$ denote the $(N+1) \times (N+1)$ matrix made from the column vectors appearing on the rhs. of (33). Obviously, a sufficient criterium for $\nu(n), x_1(n), \dots, x_N(n)$ to be linearly independent is that $\mathbf{M}_0$ has full rank. By virtue of the zeros in its first row, $\mathbf{M}_0$ has full rank iff the $N \times N$ right lower submatrix $\mathbf{M}$ made from columns $\mathbf{W}^1\mathbf{w}^{\text{in}}, \dots, \mathbf{W}^N\mathbf{w}^{\text{in}}$ has full rank. This latter condition is also necessary for $\nu(n), x_1(n), \dots, x_N(n)$ to be linearly independent. This follows from a basic fact of linear algebra, viz., that for a vector $\mathbf{v}_1 \in \mathbb{R}^N$, an $N \times N$ matrix $\mathbf{M}$, vectors $\mathbf{v}_i$ defined by $\mathbf{v}_i = \mathbf{M}\mathbf{v}_{i-1}$ $(i \geq 2)$ it holds that if $\mathbf{v}_i \in \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_{i-1})$, then $\forall j \geq 1 : \mathbf{v}_{i+j} \in \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_{i-1})$.

We assemble our findings:

Proposition 4 *The STM memory capacity of a linear network is N iff the matrix $\mathbf{M}_N = (\mathbf{W}^1\mathbf{w}^{in} \cdots \mathbf{W}^N\mathbf{w}^{in})$ has full rank.*

For a demonstration, the same 20-unit network as used previously was re-run with linear unit output functions, input weights randomly chosen to be $+0.5, -0.5$, i.i.d. input, and a spectral radius $|\lambda_{\max}| = 0.98$. The network was trained on a 2000 step teacher sequence, of which the first 1000 steps were discarded (the initial transient is long when $|\lambda_{\max}|$ is close to unity, which requires discarding a long initial portion of the training run). This yielded a network with a memory capacity of 19.2 (estimated numerically from a 3000 step test run). Fig. 6A shows the forgetting curve. Note that a substantial portion of the memory capacity resides in delays greater than the network size.

The matrix $\mathbf{M}_N$ in this example had full rank – so why falls the obtained MC = 19.2 short of the theoretical memory capacity of 20.0? I cannot provide a conclusive analysis, but the singular values of the weight matrix $\mathbf{W}$ suggest that numerical error accumulation is the answer (Fig. 6B). The ratio of the largest vs. the smallest singular value is about 50, which means that $\mathbf{W}$ is moderately ill-conditioned. At each iteration, when $\mathbf{W}$ is applied to the current state vector, it is "compressed" in the direction of the axis in $\mathbb{R}^N$ corresponding to the smallest singular value. The loss of state information due to roundoff errors is particularly salient in this direction. Considering that the MC computed here results from up to 40 iterated mappings of $\mathbf{W}$, it is not surprising that 0.8 of the theoretical MC of 20 is lost.

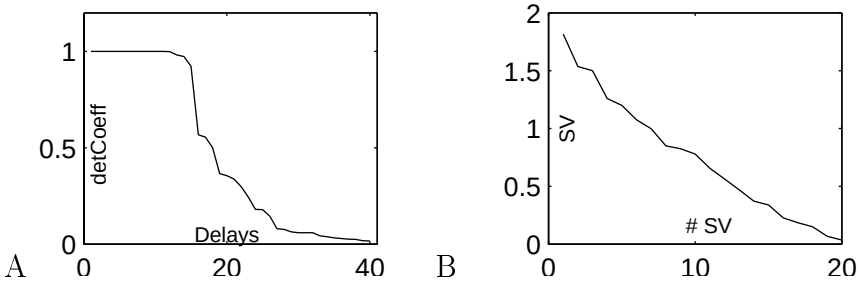


Figure 6: A. The forgetting curve of a trained, full-memory capacity linear network with 20 units. The plot shows MC_k vs. delay k, estimated numerically over a 3000 step test run to be 19.2. B. Singular values (logarithmic scale) of the matrix $\mathbf{M}_N$. For explanation see text.

To explore STM capacity a bit further, a linear 400-unit network was trained (input weights chosen from $+0.5, -0.5$, i.i.d. input, spectral radius of $\mathbf{W}$ was $\mid \lambda_{\max} \mid = 0.95$, 1500 training steps of which 500 were discarded, testing on 1000 steps). A value of MC ≈ 145 was found. On the one hand, this falls dramatically short of the value of MC = 400.0 that we would expect from Prop. 4. This demonstrates clearly that Prop. 4 assumes infinite-precision state representations and calculations. But on the other hand, if one *does* consider the effects of numerical error accumulation, the forgetting curve in Fig. 7 is rather surprising: the input information is preserved in the network state almost perfectly for about 130 update steps. Again, I cannot present a rigorous analysis of this observation; a hand-waving explanation might attribute the long memory span to network redundancy in the sense that as the input signal $\nu(n)$ is passed through the successive network states $\mathbf{x}(n+1), \mathbf{x}(n+2), \ldots$, there are always some directions in $\mathbb{R}^{400}$ in which the current update does not "squeeze" the $\nu(n)$ component of $\mathbf{x}(n+i)$ too much.

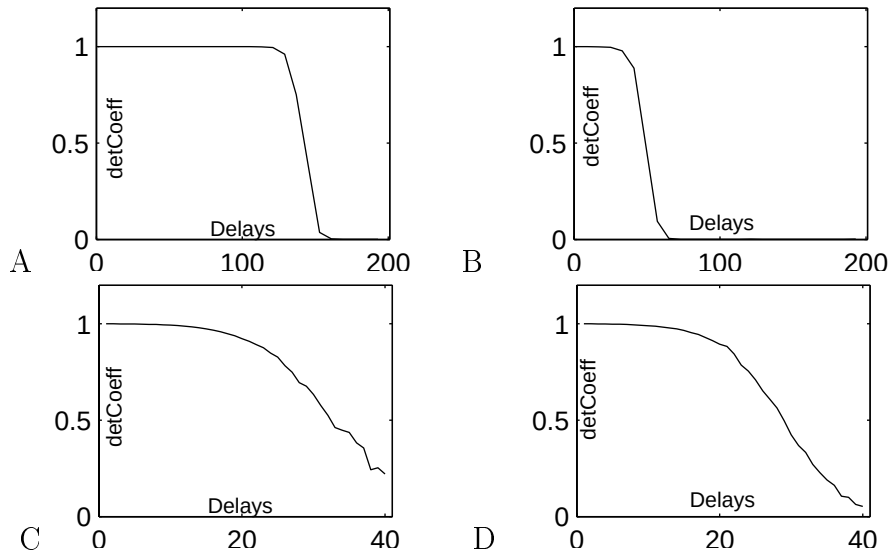


Figure 7: A. The forgetting curve of a trained, full-memory capacity linear network with 400 units, plotted for delays up to 200. B. Same as A, but with a sigmoid network. C. Same as A, but with noisy network update, plotted for delays up to 40. D. Same as B, but with noisy network update.

If the same experiment is repeated with the same network, but with sigmoid units instead of linear ones, the memory capacity shrinks to a value of about 51. Fig. 7B shows the forgetting curve.

One might assume that the long memory span obtained here crucially exploits of the double-precision arithmetics used in the Matlab simulation of the network. What happens if noise is added to the network update equation, i.e. when instead of (1) we use

$$\mathbf{x}(n+1) = \mathbf{W}\mathbf{x}(n) + \mathbf{w}^{\text{in}}\nu(n+1) + \mu(n+1), \tag{34}$$

where $\mu(n)$ is a noise vector? The previous experiment was repeated using (34) during training and testing. The noise was sampled uniformly from $[-0.01, 0.01]$. As expected, the memory capacity shrinks. Trained from 2500 input points (initial 500 discarded), a memory capacity $MC \approx 31$ was calculated (only 40 output units contributed to this number, which therefore underestimates the true MC a bit). Figure 7C shows the resulting forgetting curve.

Finally, when we redo the experiment a fourth time, now with sigmoid units and noisy state update, the memory capacity shrinks further, but only a last little bit, to MC ≈ 28 (Figure 7D).

At this point a comment on the choice of input connection weights in the example from the previous section should be made. They were set to very small values of 0.001. The reason for doing so was that such small-sized input made the network work with an activation level close to zero, in a range where the sigmoids $\mathbf{f}$ are almost linear. Thus, the network ran almost in the conditions of Proposition 3. Similarly, if the 400-unit network experiment without noise, but with sigmoid units, is repeated with very small input connection weights, memory spans approaching the linear network can be reached.

The reduction in STM span by noisy state update observed above – from a memory capacity of 145 to 31 – is rather dramatic. Noisy state update is an important phenomenon. In engineering, noise enters digital simulations of networks in the form of rounding errors, and it pervades analog VLSI neural network chips in the form of imprecise and shifting physical gate properties. Biological spiking neural networks, one could say, are made of noise. Therefore, a theoretical analysis of the effects of noise on the STM memory phenomena treated here is desirable. In the remainder of this section, we describe preliminary explorations in this direction, and suggest a remedy.

There are at least two mechanisms involved in the rapid decay of memory span through noise:

1. As the input signal is propagated through the network's state space through repeated state updates, its norm shrinks (asymptotically exponentially). By contrast, the noise contribution is a (sphere of) constant-norm signal(s). When the target signal norm becomes comparable in size with the diameter of the noise sphere, it cannot be recovered from the network state with a high precision any more.

   More precisely, let $E[\mathbf{W}^k \mathbf{w}^{\text{in}} \nu(n-k)] = \mathbf{W}^k \mathbf{w}^{\text{in}} E[\nu(n-k)] =: \mathbf{x}_\nu(k)$ denote the averaged contribution of the $k-$delayed input signal on the network state. Then if the spectral radius of $\mathbf{W}$ is less than unity (which we may assume here), the norms $\|\mathbf{x}_\nu(k)\|$ will (under some genericity assumptions which are granted with random matrices $\mathbf{W}$) asymptotically shrink exponentially with increasing k. Figure 8 is a logarithmic plot of these norms for the first 200 delays. It reveals that decreasing signal strength (in the sense of decreasing $\|\mathbf{x}_\nu(k)\|$) cannot be the sole reason for the breakdown of memory span through noise. With noisy state update (cf. Fig. 7C), the STM span collapses between delays 20 and 40; by contrast, Fig. 8 shows that the signal strength falls down to the magnitude of the noise only after about delay 100.

2. A second mechanism which explains more of the STM breakdown un-

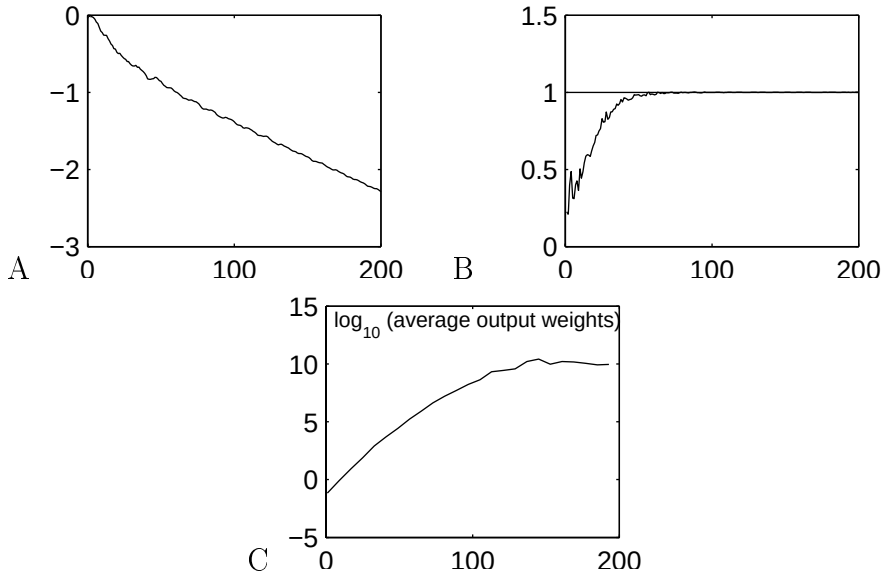


Figure 8: A. The norms $\|\mathbf{x}_\nu(k)\|$ of the iterated input traces in the network state (logarithmic plot) for the first 200 iterations. B. The portion of iterated input traces falling into the subspace spanned by the input traces with delays $k = 191, \ldots, 200$. For explanation see text. C. Averaged absolute output weights of trained outputs for delays $1, 8, 16, \ldots, 200$ (logarithmic plot).

der noise resides in the fact that repeated iterations $\mathbf{W}^k$ of a matrix $\mathbf{W}$ for large k generically yield mappings which are projections into a low-dimensional subspace of $\mathbb{R}^N$, after a suitable normalization. More precisely, consider for any $\mathbf{v} \in \mathbb{R}^N$ the sequence $\mathbf{v}_1 = \mathbf{W}^1\mathbf{v}/\|\mathbf{W}^1\mathbf{v}\|, \mathbf{v}_2 = \mathbf{W}^2\mathbf{v}/\|\mathbf{W}^2\mathbf{v}\|, \ldots$. Then, as k increases, most of the normalized vectors $\mathbf{v}_k$ will become confined in a low-dimensional linear subspace S of $\mathbb{R}^N$, in the sense that $\|\pi_S\mathbf{v}_k\| \to 1$ as $k \to \infty$, where π_S denotes projection into S. (Actually, generically S is one-dimensional; however, convergence of $\mathbf{v}_k$ into the one-dimensional asymptotic subspace can be very slow; we use here that there are small-but-greater-than-one-dimensional subspaces into which the $\mathbf{v}_k$ converge quickly).

To witness this fact, we compute $\|\pi_S\tilde{\mathbf{x}}_\nu(k)\|$ for the normalized signals $\tilde{\mathbf{x}}_\nu(k) = \mathbf{x}_\nu(k)/\|\mathbf{x}_\nu(k)\|$ and the 10-dimensional linear subspace spanned by the averaged signals $\mathbf{x}_\nu(191), \ldots, \mathbf{x}_\nu(200)$. Figure 8B is a plot of $\|\pi_S\tilde{\mathbf{x}}_\nu(k)\|$ for $k = 1, \ldots, 200$. After about 100 updates, the network state traces of the input signals become almost completely immersed in this 10-dimensional subspace.

Consider the case of noise-free training (forgetting curve in Fig. 7A). As k increases, increasingly greater portions of the signal $\mathbf{W}^k\mathbf{w}^{\text{in}}\nu(n-k)$ fall into the subspace S, which is also heavily "populated" by projections of competing signals $\mathbf{W}^{k'}\mathbf{w}^{\text{in}}\nu(n-k')$. In order to sort $\mathbf{W}^k\mathbf{w}^{\text{in}}\nu(n-k)$ out from among its competitors, the least mean square algorithm has to come up with increasingly large weights. Fig. 8C is a logarithmic plot of the average absolute output weights of the trained networks in the non-noise training scheme, for delays up to 200. Very fast growing, and eventually very large weights indeed are required for the extraction of delayed signals.

This works only well as long as there is no noise in the state update. With noise, large output weights would transport and magnify the noise into the output signals, leading to high mean square errors. The least mean square weight estimation prevents this by computing small weights. The necessity to have small weights (in order not to boost noise into the output) quickly overwhelms the necessity to have large weights (in order to sort out signals from each other which populate the same small subspace).

The two mechanisms discerned here suggest a remedy for the noise susceptibility of STM learning. When the matrix $\mathbf{W}$ is unitary, both of these memory-destructive mechanisms are circumvented: The norms $\|\mathbf{x}_\nu(k)\|$ will stay constant, and the linear subspace S into which signal traces are forced remains the full $\mathbf{R}^N$. Now we cannot use a precisely unitary $\mathbf{W}$ because its spectral radius $|\lambda_{\max}|$ would be unity, whereby we would lose the echo state property. However, when $\mathbf{W}$ is unitary, and C is a constant slightly smaller than 1, $C\mathbf{W}$ is "almost" unitary and yields a network with the echo state property.

To test whether our reasoning is sound, we procured a random unitary matrix $\mathbf{W}_0$ [technical note: this was done by replacing the singular value diagonal matrix in the SVD of our original weight matrix by the identity matrix] and scaled it with the scaling factor $C = 0.98$ standardly used in the above experiments. Using a linear network with this almost unitary weight matrix, a memory capacity of 395 was obtained for the no-noise setup. When noise sampled again from $[-0.01, 0.01]$ was inserted to the states during training and testing, the memory capacity decreased again, but not so dramatically as before: MC = 138 was obtained. Figure 9A,B shows the forgetting curves for these two trials.

Finally, we tested what happens when the scaling factor C grows toward unity. For delays smaller than the network size 400, the signal trace $\mathbf{x}_\nu(k)$ is propagated once through a set of orthogonal directions of $\mathbb{R}^N$. For $k > 400$,

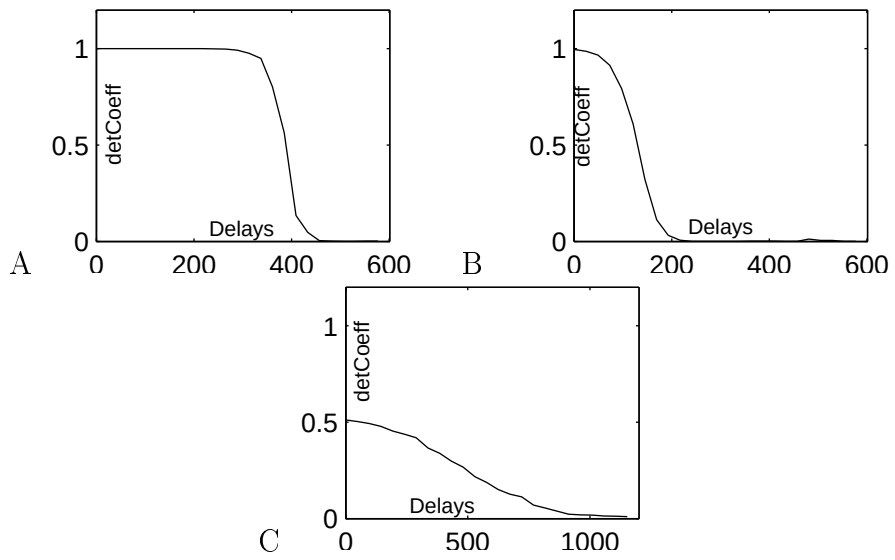


Figure 9: A. Memory curve of almost unitary 400 unit linear network with scaling factor $C = 0.98$, no noise inserted. B. Same as A, with noise. C. Same as A, but with a scaling factor $C = 0.999$.

the signal trace $\mathbf{x}_\nu(k)$ folds into a second cyclic tour through $\mathbb{R}^N$, and so forth for $k > 800, k > 1200, \ldots$. The magnitude shrink of these higher iterations is determined by C. As $C \to 1$, we would expect that the repeated, overlayed "folds" of the signal trace are unraveled by the mean square error minimization algorithm, at the cost of a reduction of MC_k for each k due to competition with other "folds". In fact, this is what is obtained. Figure 9C shows the forgetting curve for $C = 0.999$, (MC = 269 was found), featuring a long tail beyond $k = 400$.

Short term memory is a fundamental property of recurrent neural networks. The treatment given here is only a first step toward understanding STM phenomena. We conclude this section by pointing out two kinds of open questions.

- Asymptotic properties: e.g., given noisy update, how does the optimally achievable MC grow with network size?
- Nonlinear networks: e.g., is the forgetting curve monotonically decreasing in nonlinear networks? Are linear networks always optimal for large MC?

4 Something like phone number rehearsal

In the remainder of this paper we present theoretical and practical examples of RNN training tasks which make direct use of the STM capability of echo state networks.

We begin with a network which mimicks a familiar everyday phenomenon. If I want to memorize a short sequence, e.g. a phone number, I recite it repetitively in my mind (my short-term memory? working memory?) – until I use it a few moments later, or until it is transferred to my long-term memory. Echo state networks, trained as short-term memories, can be used for this cyclic rehearsal task, by feeding their output back into the input channel. The interesting point about this is that there is *no attractor involved.*

We demonstrate this effect with a little example. A 100-unit single-input, single-output linear network was trained on an i.i.d. training signal as a 50-step-delay system (input weights sampled randomly from $[-0.1, 0.1]$, spectral radius of $\mathbf{W}$ was 0.98, test error for $\mathrm{mse}_{\mathrm{test}} = 0.0000053$). In the exploitation phase, the network was started by running it for 50 steps during which a custom-made signal was fed into the input unit. After the 50th step, the network ran on its own, with its output fed back into its input unit. Figure 10 provides plots of the network output in the first 50 steps of the free run (corresponding to the first re-cycling of the fancy signal), then of the 10th, 50th, 200th repetition period.

As should be expected from a non-attractor dynamics, the 50-step target sequence is reproduced with increasing loss in precision as the number of repetitions grows. By contrast, I can repeat a phone number in my mind without it getting fuzzier over time. Apparently there are stabilizing mechanisms available in humans which reach beyond the simple repetition effect demonstrated here.

5 Dynamical pattern recognition

A fundamental task in signal processing is dynamical pattern detection. For our present purpose we define a *pattern* as a set $P = \{\mathbf{p}_j(1) \cdots \mathbf{p}_j(l_j) \mid j \in J\}$, where $\mathbf{p}$ is a numerical vector, J is an index set and the sequences $\mathbf{p}_j(1) \cdots \mathbf{p}_j(l_j)$ are the *realizations* of the pattern. One way to formalize the pattern detection task is to demand a binary output signal $d(n)$ that classifies the input signal $\mathbf{u}(n)$ as follows:

$$d(n) = \begin{cases} 0.5, \text{ if } \mathbf{u}(n-l+1) \cdots \mathbf{u}(n) \in P \text{ for some } l \geq 0 \\ 0, \text{ else.} \end{cases} \tag{35}$$

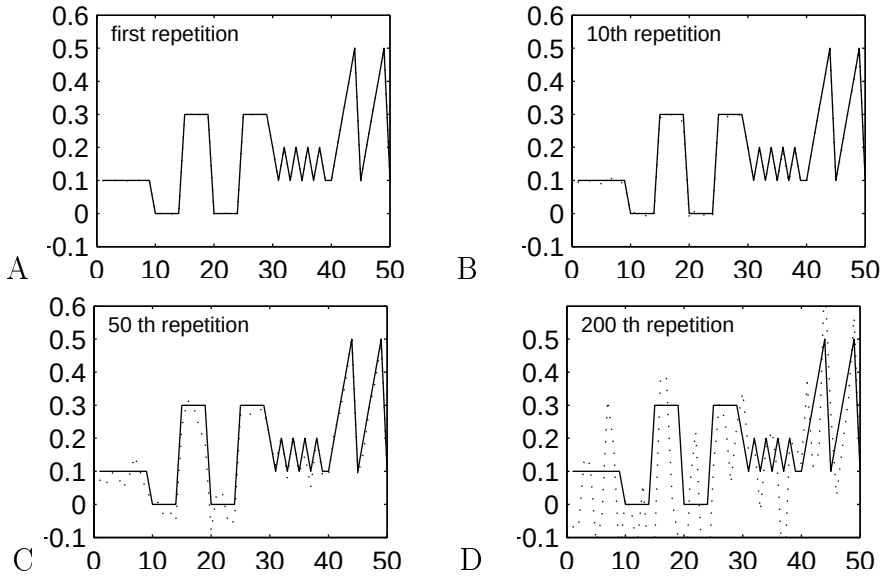


Figure 10: Overlays of network output (dashed) and correct signal (solid) in the testing phase of the cyclic rehearsal task. Plots A – D show repetitions 1, 10, 50 and 200.

(We use the value $d(n) = 0.5$ instead of the more standard $d(n) = 1$, because we will use standard sigmoid output units to produce $d(n)$; but such units can only generate values properly smaller than 1.)

We will investigate how echo state networks can be trained as dynamical pattern detectors. First, we will explore some elementary types of patterns with synthetic data (Subsection 5.1), then demonstrate discrimination learning (Subsection 5.2), and finally report on a robot application (Subsection 5.3). All calculations described in this section and the next one were done with the Mathematica software package.

5.1 Basic types of patterns, and their detection

In this section we consider various sorts of one-dimensional random patterns which occur as rather rare events, embedded a random "background" signal. All patterns which we will consider have a uniform length of 10, i.e. $\forall j \in J : l_j = \{10\}$. The training and test sequences are all prepared in the following way:

1. The pattern is defined by specifying a set $P = \{p_j(1) \cdots p_j(10) \,|\, j \in J\}$. Typically, the pattern instances $p_j(1) \cdots p_j(10)$ will be drawn from from

a uniform distribution over $[-0.5, 0.5]^{10}$.

2. An i.i.d. "background" signal $u_0(1), \ldots, u_0(n_{\max})$ is prepared by drawing each $u_0(n)$ from a uniform distribution over $[-0.5, 0.5]$.

3. A collection of "pattern insertion points" is chosen. This is a subset $R = \{m_1 < \cdots < m_r\} \subset \{1, \ldots, n_{\max}\}$, selected such that (i) $m_1 \geq 10$ and (ii) m_i and m_{i+1} are at least 10 points apart.

4. For each $m_i \in R$, one instance $p_j(1) \cdots p_j(10) \in P$ is chosen and inserted into the background signal by replacing $u_0(m_i - 9), \ldots, u_0(m_i)$ with $p_j(1), \ldots p_j(10)$. This yields an input signal $u(1), \ldots, u(n_{\max})$ which is mostly the background noise, speckled with mutually separated instances of the pattern, each ending at one of the pattern insertion points.

5. A correct (teacher or testing) output signal $y_{\text{teach}}(1), \ldots, y_{\text{teach}}(n_{\max})$ is prepared by putting $y_{\text{teach}}(n) = 0.5$ if $n \in R$, else $y_{\text{teach}}(n) = 0$. I.e., the teacher signal goes up to 0.5 each time a pattern instance is completed in the input signal.

Since the background signal is a random sequence, chance patterns very close to the original pattern may pop up in the background. These "spurious" pattern instances are not reflected in the teacher output.

We begin by investigating how a singleton pattern $P = \{p(1) \cdots p(10)\}$ is learnt.

The pattern was obtained by a random draw from a uniform distribution over $[-0.5, 0.5]^{10}$. Figure 11A shows the single instance of the pattern. A 500-step background signal $u_0(n)$ was generated, and the pattern inserted at points $200, 350, 400, 450, 500$ to yield the final input signal $u(n)$. Accordingly, the teaching signal was a 500-step sequence of zeros except at the insertion points, where it was 0.5.

The 500-step sequence was split into subsequences of lengthes $100 + 200 + 200 = 500$, which were used as initial transient, training data, and testing data, respectively. This means that the portion used for actual training contained only a single positive instance of the pattern, and the last 200 steps used for testing contained four such instances.

Fig. 11B shows the last 200 steps of the input signal. The inserted patterns (ending at steps 50, 100, 150, 200 in this plot) are not easily discerned by visual inspection, because the "figure" here has locally the same statistical properties as the "ground". This impression of difficult detectability is underlined if we try to detect the pattern with a linear filter.

The optimal linear filter in this case (modolo normalization) is $d_{\text{linear}}(n) = \sum_{k=1,\ldots,10} u(n-10+k)p(k)$. Applying this filter to the input signal gives a rather poor pattern detection performance. Fig. 11C shows $d_{\text{linear}}(n)$. We characterize the performance of a pattern detection filter $d(n)$ by the discrimination ratio

$$\text{DR} = \sqrt{E[d^2(n^+)]/E[d^2(n^-)]}, \tag{36}$$

where n^+ ranges over the pattern insertion points and n^- over the others. The DR of the linear detector $d_{\text{linear}}(n)$ is about DR ≈ 3.47 (numerical estimate from data shown in Fig. 11C).

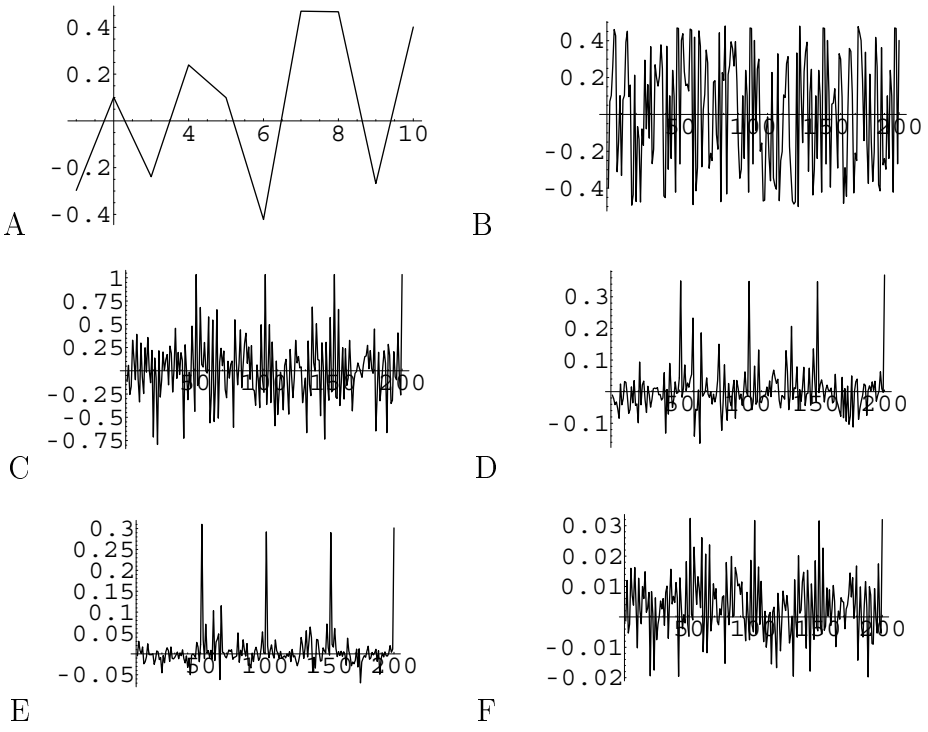


Figure 11: The singleton pattern detection task. A. The single-instance pattern used. B. Last 200 steps of the input signal (used for testing), with instances of the pattern at insertion points 50, 100, 150, 200. C. Output of a linear detection filter applied to the test sequence. D. Output of a 100-unit echo state network trained on only a single occurence of the pattern. E. Output of the same network, trained with a teacher data set containing 5 instances of the pattern. F. Same as E., but with 0.01-sized noise inserted into network during training.

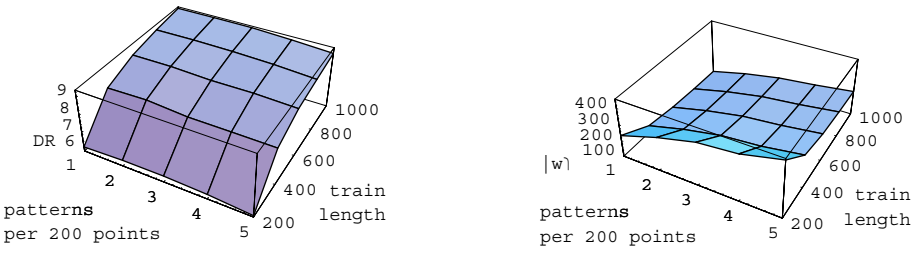
A 100-unit network was trained with the training data thus prepared. The network was essentially the same as the one used in [9], scaled to $|\lambda_{\max}| = 0.6$. Two input units were attached to the network, with input connection weights chosen randomly to be 0.5 or -0.5 (first input unit) or 0.25 / -0.25 (second input unit). The first input unit was fed with a constant bias signal of size 0.2, the second was used to feed the 500-step input signal $u(n)$ into the network. A single output unit was trained with the teacher sequence described previously. The output function was the sigmoid $\mathbf{f}^{\text{out}} = \tanh$. The first 100 steps of the run were discarded, and the steps $n = 101, \ldots, 300$ were used for training the network. Note that within this period, only one instance of the pattern occurred, so we have here a single-shot learning task. Before step 301, the output weights were computed and the then trained network was run on the remaining 200 test data points. Fig. 11D shows the performance of the signal detector thus obtained. Its discrimination ratio was DR ≈ 6.35, about twice as good as with the linear detector. Note that one might obtain a perfect binary signal detection by thresholding the output signal shown in this figure (and others).

The performance improves when more instances of the pattern are available in a longer training sequence. Repeating the experiment with a longer training sequence (1000 steps of actual training data, containing 5 occurences of the pattern) yielded a performance of DR ≈ 12.4 (see Fig. 11E for a plot of the detection signal).

The achievable detection ratio obviously depends significantly on the size of training data and the number of positive patterns contained in it. This dependency was elucidated by systematic variation of (i) length of input data actually used for training, (ii) density of positive examples of patterns in the training data. The training data length was varied from 200 to 1000 in increments of 200, the density of positive examples was incremented from 1 to 5 instances per 200 training steps. This yielded a 5×5 scheme of conditions. 50 to-be-detected patterns were drawn randomly from $[-0.5, 0.5]^{10}$, and a network was trained in each condition on each of the patterns. The 50 discrimination ratios achieved in each condition were averaged.

Figure 12A is a plot of the findings. Two observations are conspicuous: (i), an increase of the density of positive patterns has almost no effect, if any, it is detremental; (ii) increasing the length of training data leads to a significant improvement of discrimination performance. This suggests that an important factor to obtain good "singleton" pattern detectors is to present the network with a large number of negative examples. This seems to be more important than to increase the number of positive examples. This requires a more thorough investigation in the future.

The variation of DR across the 50 patterns in each condition is not shown; it was quite large. For instance, in the single-shot learning condition (leftmost cell in the plot of Fig. 12A) the best DR found among the 50 trials was 11.7 and the worst was 2.5. One may assume that much of this large variation is due to the random background signal, which will lead to good DR's when the background is mostly dissimilar to the pattern and vice versa. The example considered above (plots in Fig. 11) is representative of the average in this 50-trial sample.



A B

Figure 12: A. Discrimination ratio achieved with different setups of training data. B. Average absolute output weights across same setups. For explanation see text.

Another observation worth pointing out is the absolute size of the output weights computed by the learning algorithm. Fig. 12B shows the average absolute weights obtained across the 5×5 training conditions. The order of magnitude is 100, which is very large. We have met large output weights before: in the discussion of why learning long delays is so susceptible to noise. We can make a similar experiment here and retry the pattern detection learning task with noise injected into the network during the training phase. The observation is that the network performance strongly deteriorates. When, for instance, the experiment shown in Fig. 11E was repeated with noise of size 0.01 added to network states during training, the DR went down from 12.4 to 3.4. Fig. 11F shows the output of the resulting pattern detector. The average absolute output weight size was 0.14 in this condition. The following table lists the relationships between size of inserted noise, DR and average output weights for these and a few other noiselevels, all obtained in otherwise identical replications of the experiment.

noiselevel	DR	weights
0.0	12.4	104.0
0.0001	10.5	13.8
0.001	6.2	1.7
0.01	3.4	0.14

I include all these findings concerning noise, output weights, and performance, because I believe here is a key to a better understanding of how echo state networks might eventually be optimized by suitable network design. However, currently I can offer no theoretical account of these issues.

In the remaineder of this subsection we give a brief account of other experiments where different types of patterns were tested. The same network and training setup as described above was used throughout these experiments.

Noisy patterns. A more natural type of pattern than the pure singleton pattern is a pattern that consists of a "prototype" pattern and variations of it. In this sense, a length-10 pattern was defined as a set of random variations of a basis pattern $p_0(1) \cdots p_0(10)$

$$P = \{p_0(1) + \nu(1) \ \cdots \ p_0(10) + \nu(10) \mid \nu(k) \in [-0.2, 0.2]\}, \qquad (37)$$

where $\nu(k)$ is a uniformly distributed random variable with values from $[-0.2, 0.2]$. The same network as previously was used. The prototype pattern $p_0(1) \cdots p_0(10)$ was generated by a random draw from $[-0.5, 0.5]^{10}$. The training sequence had a total length of 100 initial plus 1000 actual training plus 200 test points. In the 1000 steps used for training, 19 random instances of P were inserted at 50-step intervals; in the 200 test steps, 4 test instances were placed at positions 50, 100, 150, 200 within this test sequence.

Figure 13A shows how a linear filter optimized to detect the prototype pattern $p_0(1) \cdots p_0(10)$ responds to the complete 1300-step training sequence. The regularly spaced peak spikes mark the positions of the inserted pattern instances; their variablity is an indicator of the variations between the pattern instances.

The discrimiation ratio obtained in this single trial was $DR \approx 11.8$; the trained output connections featured average absolute weights of 194. Both values are similar to what was obtained in the pure singleton pattern task.

Shifted patterns. Another elementary kind of variations of a prototype are shift transformations. Thus, a pattern was defined by shifting a basis pattern $p_0(1) \cdots p_0(10)$ by a random amount between -0.5 and 0.5:

$$P = \{p_0(1) + \nu \ \cdots \ p_0(10) + \nu \mid \nu \in [-0.5, 0.5]\}. \qquad (38)$$

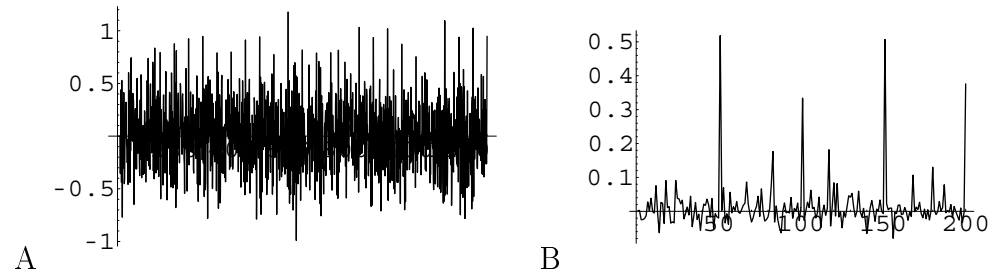


Figure 13: A. Output of a linear filter for detecting the prototype pattern in the training sequence. B. Output of the trained network on the test sequence. For explanation see text.

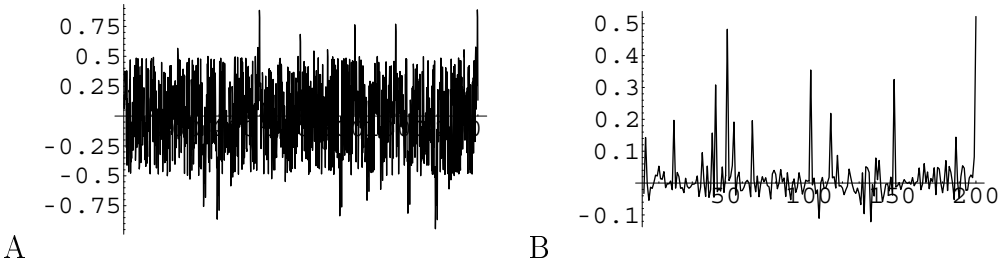


Figure 14: A. Training and testing sequence for the shifted pattern task (overall length 1300). B. Output of the trained network on the test sequence.

The training sequence was prepared like in the previous example. Fig. 14A shows the training input signal; the shifted versions of the pattern stick out at places. Fig. 14B is a plot of the trained network's output on the test sequence. Again, no substantial difference from the pure singleton pattern is apparent. The discrimination ratio found numerically in the short test sequence was $DR \approx 8.3$; the average absolute output weight was 188.

Two-element patterns. Yet another basic kind of patterns are disjunctive patterns. To test a simple exemplar, a pattern was defined as a set $P = \{p_1(1) \cdots p_1(10), p_2(1) \cdots p_2(10)\}$ of two independently generated patterns. Fig. 15A shows the two patterns used, and Fig. 15B the trained network's output on the test sequence (training setup as in previous two examples). The first two response spikes are triggered by two occurrences of the first pattern in the test data, the second two spikes by the other pattern. The discrimination ratio in this example was $DR \approx 12.1$; the average absolute output weight was 200.

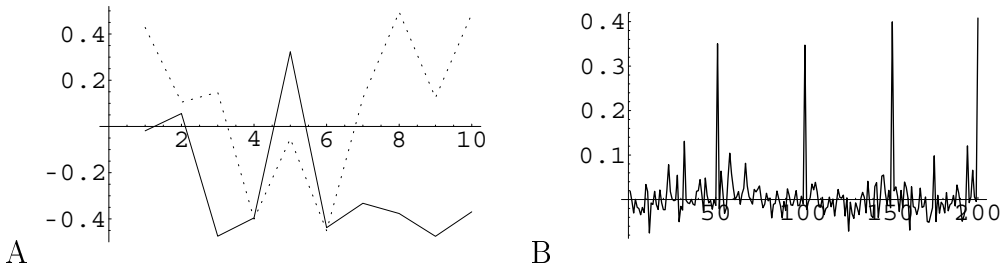


Figure 15: A. The two instances making for the pattern in the disjunctive pattern experiment. B. Output of the trained network on the test sequence. For explanation see text.

Pattern defined by a short description. Another standard type of pattern is a collection of instances which superficially may look quite different from each other, but which share a simple description. We procure a synthetic pattern of this kind by putting

$$P = \{p(1) \cdots p(10) \mid D(p(1), \ldots, p(10)) > C\}, \tag{39}$$

where D is a nonlinear decision function and C some threshold. Concretely, and quite arbitrarily, we used

$$D(u_1, \ldots, u_{10}) = \sin(u_5) + \tan^2(u_6)/10 + \sum_{i=1}^{10} u_i, \tag{40}$$

and $C = 2$. A 1300-step input signal $u(n)$ signal was obtained by drawing a random sequence from the interval $[-0.5, 0.5]$. The teacher signal was constructed by putting $y_{\text{teach}}(n) = 0.5$ when $D(u(n-9), \ldots, u(n)) > 2$, else $y_{\text{teach}}(n) = 0$.

It was found in preliminary trials that for this kind of task a different setting of the global network parameters was preferable: the spectral radius of our by now familiar 100-unit network was scaled to $|\lambda_{\max}| = 0.5$; the input connection weights for the constant bias input (of size 0.2) were randomly chosen to be 2 or -2; the input connection weights for the $u(n)$ signal input unit were chosen from -0.1 and 0.1.

Figure 16A gives a plot of the teacher signal in the last 200 steps used for testing, overlayed with the trained network output. The discrimination ratio was DR ≈ 5.3. When the network output is thresholded at a value of 0.25, a binary detector signal is obtained, which is opposed to the correct

(teacher) signal in Fig. 16B. The thresholded network output gives a correct indication of the pattern in 97.5% of the 200 test points.

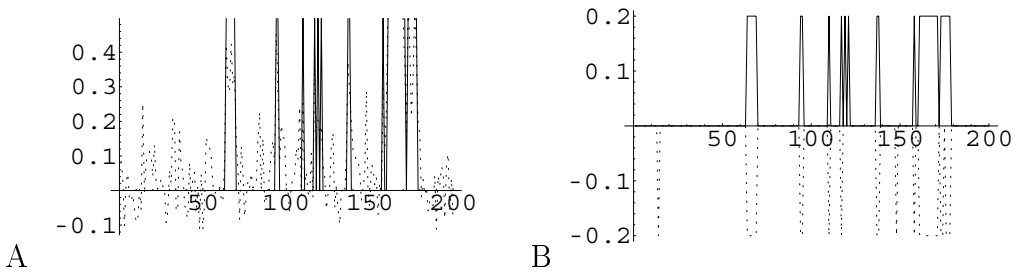


Figure 16: The "short description" pattern. A. Teacher and network output on the testing sequence. B. Teacher (solid line) and thresholded network output (dashed line) in a mirrored representation. For explanation see text.

Pattern defined by a system with a state. The types of patterns considered so far were all static – sets of 10-dimensional vectors with no inherent temporal aspect. Our final example is a type of pattern which *is* inherently dynamic; and furthermore, it depends on historical context. Real-life examples of such patterns would be e.g. gestures; plant failures; thunderstorms; economical crises – actually, almost every real-life dynamical phenomenon which we can detect and classify. Our original definition of a pattern as a set of finite-length instances, $P = \{\mathbf{p}_j(1) \cdots \mathbf{p}_j(l_j) \mid j \in J\}$, should actually be accomodated in deference to the circumstance that the beginning of a dynamical pattern is often not well-defined. The variant $P = \{\ldots \mathbf{p}_j(-2)\mathbf{p}_j(-1)\mathbf{p}_j(0) \mid j \in J\}$ would be more suitable, but we will not dwell on this subject.

A simple mathematical model of a truly dynamical pattern is to model it by a decision function which is actually the output of a dynamical system driven by the input signal $u(n)$:

$$P = \{\cdots \mathbf{p}(-2)\mathbf{p}(-1)\mathbf{p}(0) \mid D(\ldots, \mathbf{p}(-2), \mathbf{p}(-1), \mathbf{p}(0)) > C\}, \qquad (41)$$

where $D(\ldots, u(-2), u(-1), u(0))$ is the output of a dynamical system at time 0 after an input history $\ldots, u(-2), u(-1), u(0)$.

Again, we choose D quite ad hoc, by defining a dynamical system with two state variables x_1, x_2 and an output variable D by

$$x_1(n+1) \;=\; \frac{\|\sqrt{x_1(n)}\|}{1 + 5\,u^2(n)} - \frac{x_2^3(n)}{1 + x_1^2(n) + x_2^2(n)}$$

$$
\begin{aligned}
x_2(n+1) &= u^3(n) - \frac{x_1(n)\,x_2(n)}{1+x_1^2(n)} \qquad (42)\\
D(n) &= 5\,x_1(n-1)\,u(n) - x_2(n-1)\,\sin(x_1(n-1)).
\end{aligned}
$$

A 1300 step random sequence was prepared to serve as input $u(n)$. Figure 17A shows the last 200 steps of the dynamical system's (42) output $D(n)$ when driven by $u(n)$. A cutoff $C = 1$ was used to generate a binary pattern indicator teacher signal from $D(n)$. The same network and training setup as in the previous task was used. Fig. 17B shows a plot of the teacher signal (rescaled from 0.5 to 0.2) vs. the thresholded (cutoff 0.25) trained network output (rescaled and mirrored) in the final 200 steps used for testing. The discrimination coefficient (for the unthresholded network output) was 4.2. The thresholded network output is equal to the teacher in 95.5% of the 200 test points.

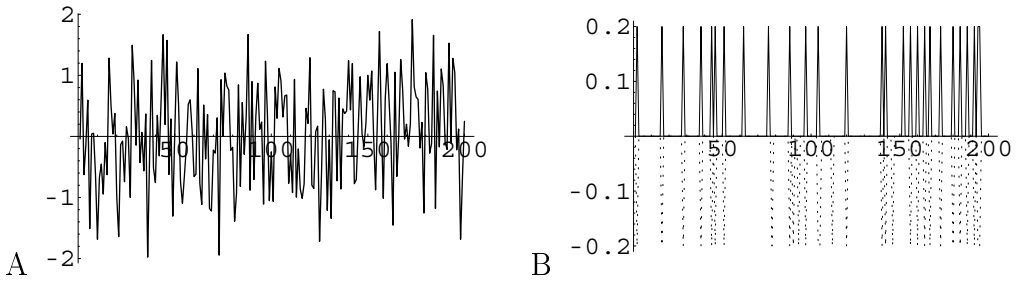


Figure 17: The "truly dynamic" pattern. A. Output $D(n)$ of the dynamical system (40) driven by the random input signal $u(n)$. B. Teacher (solid line) and thresholded network output (dashed line).

5.2 Discrimination learning

Discrimination learning refers to a situation where a learnt pattern detector first falsely includes some signals into the pattern, and then has to be changed in order to exclude those signals. In this section we will present an illustrative demonstration of this important task within the echo state network approach.

First, we trained a network on a singleton pattern again. The pattern is a simple ramp (Fig. 18A, solid line). This time we chose a 400-unit network (the same as in [9]), which was scaled to $|\,\lambda_{\max}\,| = 0.6$, with two input units for a constant bias and the input signal (input connection weights 0.5 / -0.5 and 0.25 / -0.25, respectively; sigmoid output function of output units; all like the 100-unit network used futher above). A background sequence of

length 1300 was prepared (100 initial steps, 1000 actual training steps, 200 testing steps). 14 instances of the ramp signal were inserted into the 1000-step training subsequence, and 2 further instances at points 150 and 200 in the 200-step testing subsequence. The teacher signal consisted of 0.5's at all of these insertion points and was zero elsewhere. Additionally, a modified version of the ramp, – let's call it a "wedged ramp" – (Fig. 18A, dashed line) was inserted into the test sequence at positions 50 and 100.

Fig. 18B shows the output of the trained network on the testing sequence. The ramp signal is correctly detected (last two spikes), but the modified ramp signal also leads to a clear detection spike (first two), although during training the wedged ramp was not presented to the network. The network obviously has generalized from the teaching examples. To gain a clearer impression of the network's response to the two kinds of ramps, the network was re-run on another 10 test sequences with different background signals, but with the ramps at the same places (the kind of signal augmentation known from averaging EEG signals!). Fig. 18C shows the averaged response, which reveals no significant difference between the responses to the ramp vs. the wedged ramp.

Of course, generalization from training examples is often desirable. However, the converse of generalization, discrimination, is as important. We now train the network to discriminate between the two ramps. Two trials were carried out, with training data made up differently for illustration. They contained R instances of the original ramp *and* W instances of the wedged ramp, where $R = W = 1$ in the first trial and $R = W = 14$ in the second. The teacher signal was 0.5 only at the positions of the original ramp, and zero elsewhere. Thus, the network encountered negative examples during training in these two trials. Figures 18D,E show averaged responses of networks trained on the two teaching sequences: D: $R = W = 1$, E: $R = W = 14$. It is apparent that a exposure to a single negative instance in training already significantly weakens the network response to the wedged ramp, and that a repeated presentation of both the positive and the negative instances yields a convincing discriminiation between the two signals.

The 100-unit network, put to the same set of discrimination learning tasks, performed much like the 400-unit network except for the $R = W = 14$ condition, where its performance was not markedly better than in the $R = W = 1$ case. It seems that the 100-unit network was not expressive enough to achieve the desired discrimination as well as the 400 unit network.

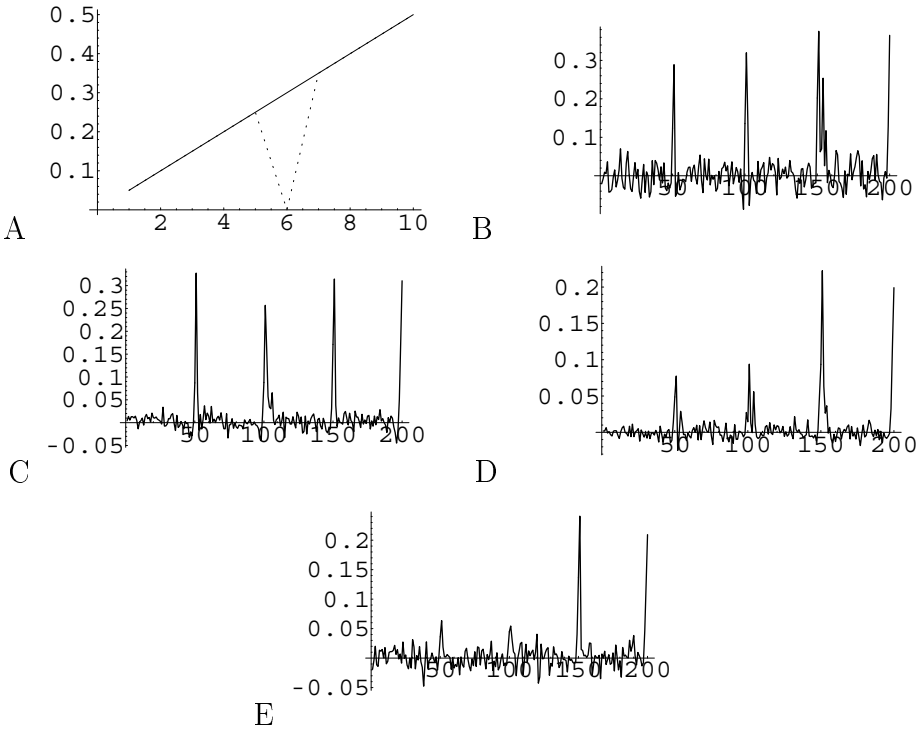


Figure 18: Discrimination learning. A. The ramp signal (solid line) and the "wedged ramp" (dashed line). B. Network response of the network trained on 14 positive examples of the ramp. The input signal contained two instances of the wedged ramp (positions 50 and 100) and two of the original ramp (before points 150 and 200). C. Averaged response of network trained on 13 positive examples (ten trials superimposed). D. Averaged response, trained on $R = 1$ ramps and $W = 1$ wedged ramps. E. Same as D, with $R = W = 14$.

5.3 A robot application: event detection

In this subsection we sketch a first application of echo state networks as dynamical pattern recognizers in a robotics task. The work reported here was carried out in collaboration with Joachim Hertzberg and Frank Schönherr of AiS, who provided the task and training data. A more detailed account can be found in [7].

A simulated robot was made to wander through an office environment for a simulated duration of approximately 21 minutes. The environment consisted of two rooms and a corridor with connecting doors between corridor and rooms and between the two rooms. The environment contained a few obstacles and featured mostly 90 degree corners but also a few "rounded"

ones. Various variables available to the robot were sampled during this run: raw sensor data (e.g., infrared sensor readings, together 10 channels), processed sensor data (e.g., robot heading, 3 channels), motor measurement data (robot speed and angular velocity readings, 3 channels), and activations of the various behavior control modules of the robot (i.e., modules like "aviod collision", "follow left/right wall", "go through door", altoghether 9 channels). The sampling rate was 4 per simulated second. All in all, 25 channels with sensor and control related quantities were collected.

We consider only facts that correspond to events of short duration (as opposed to "state" facts that hold true for an extended period of time). From among the numerous event categories trained, we select here the following five:

1. φ_1 = "robot passes by a 90 degree corner, excluding corners at corridor ends" [at the end of corridors, there are two 90 degree corner close to each other]

2. φ_2 = "robot passes by a 90 degree corner, including corners at corridor ends"

3. φ_3 = "robot passes by a corridor end"

4. φ_4 = "robot passes trough any door"

5. φ_5 = "robot passes through a door, thereby entering a room from the corridor" [this excludes room-to-corridor and room-to-room door passages]

The first three and the last two event types each were rather similar to each other; distinguishing between events $\varphi_1, \varphi_2, \varphi_3$ and between φ_4, φ_5 is not apparently trivial.

The variables $\varphi_1, \ldots, \varphi_5$ were put from 0 to 1 for one simulated second by the human experimenter immediately after the corresponding event occured.

Fig. 19A shows a portion of the training data.

These data were used to train an echo state network with 5 output units for $\varphi_1, \ldots, \varphi_5$ – note that these output units are trained and operate independently and simultaneously. The network was our familiar 100 unit RNN, scaled to a spectral radius of $\mid \lambda_{\max} \mid = 0.95$, with sigmoid output units attached. Weights from input units to the network were put at random to values of 0, -0.5, 0.5 with probabilities 0.7, 0.15, 0.15 respectively.

From the data collected in the simulation run, roughly the first simulated 15 minutes (4000 update cycles) were used for training and the remaining

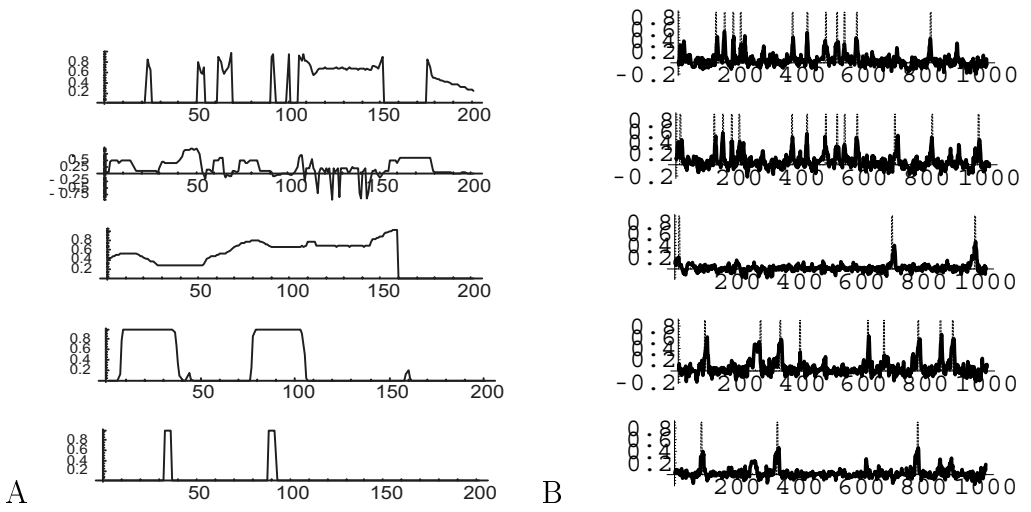


Figure 19: A. Some of the input and teacher channels in the event detection training for a robot. From top, four input channels: an infrared distance sensor reading, a motor velocity, an estimated robot heading, a behavior control module activation, and (last plot) a teacher output signal. B. The trained network's output (fat solid line) vs. the human event detection (dashed line). From top: φ_1 = "90 degree corner W/O corridor ends", φ_2 = "90 degree corner with corridor ends", φ_3 = "corridor ends", φ_4 = "through any door", φ_5 = "through door from corridor into room".

approx. 5 minutes (1000 cycles) for testing. Fig. 19B shows the network output for the five event detector channels during the testing phase. We point out some main observations:

- The network outputs provide reasonable event hypotheses. With a suitable cutoff threshold, event detection with very few false positives or false negatives could be achieved.
- The event class distinctions between the first three and last two event types was mastered. They illustrate not quite trivial instances of discrimination learning.
- Hand-coding event detection routines with classical AI techniques would require substantial effort and time for each class of event. In comparison, the supervised training of an echo-state network was done in one afternoon, for all event types in one sweep, including data collection and teacher output signal definition.

We also mention some observations from similar experiments not presented here:

- A network size of 100 units is roughly appropriate for this task: a 400 unit network did not lead to significantly better results, whereas a 25 unit network performed poorly.
- Relevant global network parameters, especially $|\lambda_{\max}|$, can be scaled in a wide range without significant changes in performance. From the perspective of echo state network theory, the present filtering task appears to be rather simple; "any reasonably sized and scaled network does the job".
- Longer training sequences result in a better discrimination ratio of the trained network output, but useful networks can be obtained already from very scarce training date (for conspicuous, apparently context-insensitive events like "go through door", training from a single positive example was satisfactory).

6 Stochastic symbol sequences

In this section we consider how echo state networks can be trained to model stochastic symbol dynamics of the kind usually modeled by (higher-order) Markov chains, hidden Markov models, or more recently, observable operator models [8].

The general setup is the following. Let $E = \{a_1, \ldots, a_M\}$ be a finite alphabet and $(Z_n)_{n\in\mathbb{Z}}$ be a familiy of random variables with values in the observation space E. We assume that the random variables Z_n are a stationary stochastic process. Stationarity implies that the distribution of the process is determined by the finite-length sequence probabilities $P_k(Z_1 = a_{i_1}, \ldots, Z_k = a_{i_k})$, where $k \geq 1$ and $a_{i_j} \in E$. We use the shorthand notation $P_k(a_{i_1} \ldots a_{i_k})$ for $P_k(Z_1 = a_{i_1}, \ldots, Z_k = a_{i_k})$.

A model of the process Z_n is another stationary stochastic process $(\hat{Z}_n)_{n\in\mathbb{Z}}$, usually embodied in a generative algorithm, which has a "similar" distribution. It has marginal distributions $\hat{P}_k(\hat{Z}_1 = a_{i_1}, \ldots, \hat{Z}_k = a_{i_k})$. Similarity can be quantified in a number of ways. We will employ the Kullback-Leibler distance of the finite marginal distributions,

$$D_k(\hat{P}_k \| P_k) = \sum_{a_{i_1} \ldots a_{i_k} \in E^k} \hat{P}_k(a_{i_1} \ldots a_{i_k}) \log \frac{\hat{P}_k(a_{i_1} \ldots a_{i_k})}{P_k(a_{i_1} \ldots a_{i_k})}, \qquad (43)$$

to quantify similarity between original and model.

Our goal is to use a training sequence $a_{i_1} \ldots a_{i_L}$ generated from the original process Z_n to teach an echo state network, which can then be used as a generator of stochastic sequences whose finite-length distributions $\hat{P}_k$ should have small KL-distances to the original distributions P_k.

The echo state network is equipped with M input units and M linear output units with no feedback connections (as in all examples in this article). During training, the symbols from the training sequence are read into the input units, one symbol per network update cycle, by coding a_{i_n} through a unit signal in the i_n-th input unit:

$$\mathbf{u}(n) = (0, \ldots, 0, 1, 0, \ldots, 0), \tag{44}$$

where the "1" occurs in the i_n-th input unit. The teacher signal is a similar encoding of the *next* symbol, i.e.

$$\mathbf{y}_{\text{teach}}(n) = (0, \ldots, 0, 1, 0, \ldots, 0), \tag{45}$$

where the "1" occurs in the i_{n+1}-th output unit. The network should thus learn to predict the next symbol.

After training, the network is used to generate a random symbol sequence $a_{j_1}, \ldots, a_{j_{L'}}$ in the following way.

1. Start the network from an arbitrary initial network state. Compute the output activation vector $(y_1(1) \ldots y_M(1))$.

2. Turn the output vector into a probability vector $(p_1(1) \ldots p_n(1))$, by setting all negative entries to zero and renormalizing the remaining entries such that they sum to unity.

3. From the obtained probability vector, select one index position j_1 by a weighted random draw, where the weights are given by $(p_1, \ldots, p_n)$. For example, with $(p_1(1)\, p_2(1)) = (0.2\, 0.8)$ we would choose $j_1 = 1$ with probability 0.2. Write a_{j_1} out as the first generated symbol.

4. Feed a_{j_1} back into the input units of the network by the coding scheme described above.

5. Update the network with this input and compute the output activation vector $(y_1(2) \ldots y_M(2))$. Iterate the previous steps until a sequence of desired length is generated.

We expect that network-generated symbol sequences $a_{j_1}, \ldots, a_{j_{L'}}$ have similar finite distributions as the original process after initial transients due to the arbitrary starting state have died out. We give first two reasons that justify this expectation, and then report on experiments to show that indeed complex processes can be modeled in this way.

The first argument why our setup should work is of a general, somewhat intuitive nature. Many stationary stochastic processes (among them, Markov chains of any order, hidden-Markov processes and observable operator processes) are asymptotically forgetting. Concretely, this means the following. Let $\ldots, b_{-2}, b_{-1}, b_0$ be a left-infinite path of Z_n, and $b_1, \ldots, b_k$ a finite observation sequence. Consider the conditional probabilities $P_l = P(Z_1 = b_1, \ldots, Z_k = b_k \mid Z_{-l} = b_{-l}, \ldots, Z_0 = b_0)$, where $l = 0, 1, \ldots$. Then, the sequence $P_0, P_1, \ldots$, converges for almost every history $\ldots, b_{-2}, b_{-1}, b_0$. Stated more simply: the recent past is more important for predicting the future than the distant past, and the effects of ever more distant inputs vanish asymptotically. Now consider an echo state network that is fed with an input sequence $\ldots, b_{-2}, b_{-1}, b_0$. As we have seen in the forgetting curves in Section 3, the state of the network contains information about the past input history in a way which also reflects the recent history well and decays with the delay time. Thus, the information about the past which is most relevant for predicting the future of $\ldots, b_{-2}, b_{-1}, b_0$ is readily available in the network state.

The second argument why things should work out concerns the question why we should be allowed to treat the output vector as a probability vector. For a moment, let us forget about sequences and networks, and consider a simple numerical experiment. Let $y_1, \ldots, y_L$ be a path of an i.i.d. process Y_n with values 0 and 1, where $P(Y_n = 1) = p$ and $P(Y_n = 0) = 1 - p$. Then, the least mean square error estimation $\hat{p}$ of p from the path is $\hat{p} = (y_1 + \ldots + y_L)/L$, which converges to p almost surely with growing L. Now, equipped with this little reminder, let us return to our sequences and networks. Assume that we have a very long training sequence in which all possible subsequences $b_{-l+1}, \ldots, b_0$ of some length l appear very often, and where l is suitably large to allow us good predictions of the future distribution from a history of length l. Consider a symbol $b \in E$ and the probability $p = P(Z_1 = b \mid Z_{-l+1} = b_{-l+1}, \ldots, Z_0 = b_0)$. Then, the training procedure outlined above will lead to a trained network whose activation in the output unit corresponding to b is approximately p at times when the recent input history was $b_{-l+1}, \ldots, b_0$. This holds for all such input histories of length l. Therefore, the network output activation at the node corresponding to any symbol b should be a sensible estimate of the probability of this symbol to occur next in the sequence after all preceding histories.

We demonstrate the viability of this approach with two synthetic examples. (In the next section, a real-world process will be treated.)

The first example is a process generated by a hidden Markov model (HMM). We assume that the reader is familiar with the basic concepts of HMMs. The alphabet of emitted (observable) symbols is $E = \{1, 2, 3\}$, and there are three hidden states $\{s_1, s_2, s_3\}$. The probabilities of the hidden state transitions are given in the following stochastic matrix (we use an obvious shorthand):

$$(P(s_i \to s_j))_{i,j=1,2,3} = \begin{pmatrix} 0.9 & 0.1 & 0.0 \\ 0.0 & 0.9 & 0.1 \\ 0.1 & 0.0 & 0.9 \end{pmatrix}, \tag{46}$$

i.e., the hidden state process basically cycles through the three states, remaining in each state with probability 0.9. The emission probabilities for the three observable symbols 1, 2, 3 are $(0.1, 0.2, 0.7)$ in the first state, $(0.5, 0.05, 0.45)$ in the second state, and $(0.5, 0.4, 0.1)$ in the third state.

Three training sequences of length 500, 2000, and 5000 were generated from this HMM, which was started from the asymptotic hidden state distribution vector $(1/3, 1/3, 1/3)$, so the training sequences were samples from a stationary process.

A 100-unit echo state network with four input and three output units was employed (the same network as used previously in this article). The weight matrix was scaled to a spectral radius of 0.95. The first input unit served to feed a constant bias of size 0.2 into the network. This unit was connected to the network by weights randomly chosen from the interval $[-1, 1]$. The remaining three input units served for reading in the symbols; each of them was connected to the network by weights randomly put to 1 or -1.

Figure 20 shows the output of the network trained from the 5000 step sequence when it was fed with a HMM-generated sequence. It becomes apparent that the output units produce varying "hypotheses" about the next symbol. Actually, at each time the three output signals sum to unity with a 10-digit precision, and no negative outputs occurred, so the renormalization step 2. in the generation procedure was unneccessary.

For a quantitative judgement of model quality, various Kullback-Leibler distances were estimated from data, in the following way. Consider first the network estimated from the 500 step sequence. It was used to generate a symbol sequence $S_{\mathrm{net}_{500}}$ of length 2000. Furthermore, a test sequence S_{test} of same length was generated from the original HMM. Probabilities $\hat{P}_l(a_{i_1} \ldots a_{i_l})$ of finite-length subsequences of length $l = 1, \ldots, 7$ were estimated from $S_{\mathrm{net}_{500}}$ by frequency counts (which gives the maximum-likelihood estimates of these probabilities). Analogously, probabilities $P_l(a_{i_1} \ldots a_{i_l})$ were estimated from

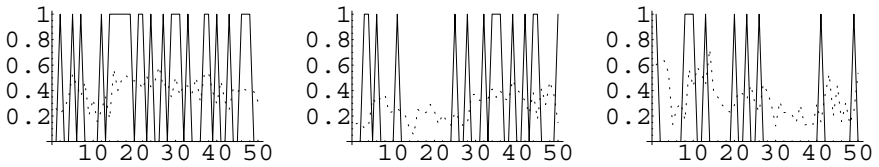


Figure 20: HMM process learning. Output of the trained network (dashed line) and teacher signal (solid line) during a 50 step episode. Each plot corresponds to one output unit. For details compare text.

the test sequence. Using these empirical estimates, for each l the Kullback-Leibler distance (43) was computed. (A technical detail: for larger l, some of the subsequences occurring in $S_{\text{net}_{500}}$ did not occur in S_{test}. This would lead to a zero denominator in the rhs. of Eq. (43). To circumvent this impasse, in these cases an apriori probability of 3^{-l} was substituted for the non-defined $P_l(a_{i_1} \ldots a_{i_l})$ in the computation of Eq. (43). Since with l larger than 7, the portion of subsequences not occurring in S_{test} becomes noticable, only subsequences up to this length were considered).

Furthermore, the Kullback-Leibler distances of finite-length subsequence distributions between the test sequence and the original 2000-step training sequence was also computed in the same manner. These distances yield a baseline, being the empirical KL distances that would be obtained with a perfect model.

Figure 21A shows the findings. The larger the training sample, the better the model quality. The network trained from the 5000 step sequence gives a very good approximation to the original distributions.

For a further comparison, the length 2000 training sequence was also used to estimate an observable operator model (OOM) of the HMM process. We cannot give an introduction to OOMs and their estimation here and refer the reader to [8] for an introduction. Suffice it to say that OOMs are superior to standard HMM models estimated by means of the EM algorithm and can be considered the best available modeling tool for symbol processes of the kind considered here. Figure 21B reveals that the model obtained with this dedicated technique is only marginally better than the echo state network model. When this comparison was repeated on the 500 and 5000 step sequences, the OOM model was slightly worse than the corresponding echo state network models. (It should be noted however that OOM models – like HMM models – allow one to compute explicit probabilities for subsequences; this is not possible with echo state network models, which are purely generative). A detailed investigation remains to be carried out.

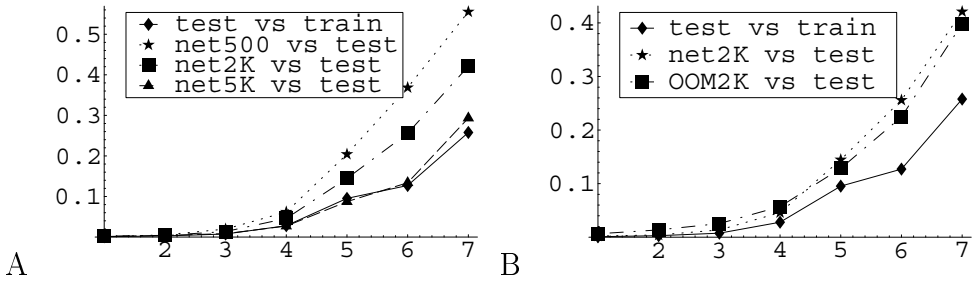


Figure 21: A. Empirical KL distances for subsequences of length 1 to 7. Solid line: baseline KL distance corresponding to perfect model. Star, box, triangle lines: KL distances between processes generated from the networks trained on 500, 2000, 5000 steps, respectively. B. Comparison with dedicated model estimation technique (OOM modeling). Empirical KL distances for subsequences of length 1 to 7. Solid line: baseline. Starred line: KL distances from network trained from 2000 steps. Boxes: KL-distances from OOM model trained with state-of-the-art technique.

The second example uses the "probability clock" process Z_n described in [8]. It is a stationary process with two symbols $E = \{1, 2\}$ whose central property is that the conditional probabilities $P(Z_1 = 2 \mid Z_{-l} = 1, Z_{-l+1} = 2, \ldots, Z_0 = 2)$ yield an undamped sinusoidal oscillation in l of period length 2π, so roughly (but not exactly) 6 update steps make one oscillation. The probability clock cannot be modeled by finite-dimensional HMMs. It can however be generated by a 3-dimensional OOM, and such OOM models can be learnt from data. The previous experiment was identically repeated with this process. Figure 22 summarizes the findings, which turn out to be qualitatively the same as in the first experiment.

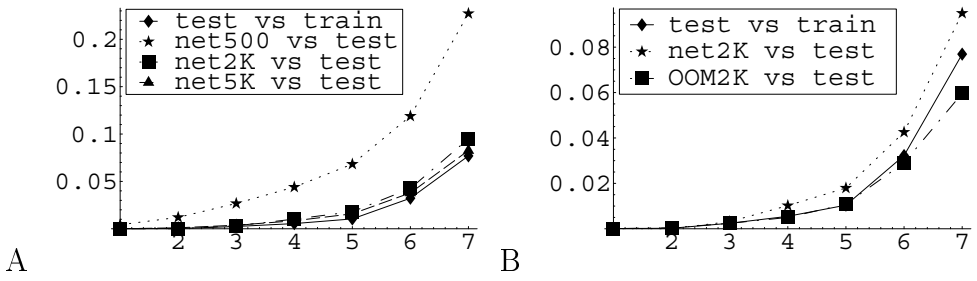


Figure 22: The analog of Fig. 21 for the probability clock process.

The network setup used in these two experiments was chosen quite ad

hoc; no optimization of spectral radius of the weight matrix and of input weights was undertaken.

All in all, these preliminary observations made in this section suggest that for learning short subsequence distributions of stochastic symbol processes, echo state networks are an approach worthy of further investigation.

7 Little red riding hood

The modeling approach used in the previous section will now be taken to a real-world stochastic sequence: a fairytale.

As a text source, the tale of Little Red Riding Hood (LRRH) was chosen.[2] The text was simplified by putting all letters to smallcaps and reducing the set of interpunctuation symbols to ",", "." and "␣", which left a total of 29 different symbols. This gave a teacher sequence $\mathrm{LRRH}_{\mathrm{train}} = s_1 \ldots s_{3412}$ of length 3412. It is given in Appendix B.

A number of researchers have trained recurrent neural networks on text symbol prediction tasks. The classical work is [3], a recent example in the same vein is [1]. The goal of those investigations is to analyse whether the trained network represents in its states linguistic categories. The ratio training sample size / network size is large in these approaches (e.g., 10,000,000/120 words/units in [1]). This makes sense for the particular investigation goal, because grammatical structure can be inferred by statistical learning only when a particular grammatical role filler (e.g., a word) occurs in many combinations with other role fillers in a particular grammatical structure. When there are many different role fillers – a condition obtained in real-world texts –, a large text corpus is mandatory.

The goal that we pursue in this section is more moderate. Like in the previous section, we only want to obtain a black-box model of the symbol subsequence distribution. Only a single, comparatively short text is used for training. The ratio training sample size / network size is small (3412/400 symbols/units). This ratio becomes effectively even smaller because the training text contains numerous repeated subsequences. We do not want to discover structure "under" the text surface but want to capture in the model the surface itself.

A 400-unit network (the same as used in other tasks described previously

[2]Source: Little Red Riding Hood & The History of Tom Thumb, illus. H. Isabel Adams (London: J. M. Dent & Co., 1893). Electronic text file obtained from http://www-dept.usm.edu/ ~engdept/lrrh/lrrhhome.htm, the website of Michael N. Salda's "The Little Red Riding Hood Project" at The de Grummond Children's Literature Research Collection, University of Southern Mississippi.

in this article) was scaled to $|\lambda_{\max}| = 0.95$, and equipped with 30 input units and 29 linear output units. The first input unit served to provide a constant bias of size 0.2; its connection weights were set at random values between -1 and 1. The remaining 29 input units were used to feed text into the network; their weights were set to either +1 or -1 randomly. As in all examples covered in this article, no feedback projections from the output units were present. Like in the previous section, the network choice was ad hoc and no optimization was attempted.

The training was done in the same way as described in the previous section. In order not to waste training text through discarding an initial transient, the first 100 steps of $\mathrm{LRRH}_{\mathrm{train}}$ were duplicated and prepended to the original training sequence; the network was then trained on the 100+3412-step total sequence and the first 100 steps of that run were discarded.

When the trained network is fed with text, it generates in its output units a "hypothesis vector" of the next symbol. Fig. 23 shows the output of the trained network during a 50 step episode, where the input sequence was the following passage taken from the training text:

`she_live_a_great_way_off_said_the_wolf._oh_yes_sai`

The output activation traces in Fig. 23 show significant peaks in the correct places. It also becomes apparent that negative output values *do* occur, so the renormalization step 2. in the generation procedure from the previous section will be necessary. A curious fact is that the output values at each time sum to unity with 10-digit precision, as in the previous section. This observation is somewhat surprising and waits for an explanation.

The trained network was tested in various ways. First, it was checked how the network performed in the next-symbol-prediction task on the training data set. It was fed with $\mathrm{LRRH}_{\mathrm{train}}$. The output unit with the largest activation was taken to code the network's next-symbol-prediction. 70.5 % of the symbols were correctly predicted. This is possible because in English texts very often the next symbol can be predicted with certainty; the network has learnt to exploit this determinism.

Next, the network was used to generate a symbol sequence $\mathrm{LRRH}_{\mathrm{net}}$ of length 3412 on its own, using the generation procedure from the previous section. This is a 50-step subsequence of the network production thus obtained:

`hey_go_aut_hograndmothe_pot_at,trromdt,__telht_e_w...`

This looks like a perfect mess with an English flavor. The latter derives from fragments with a correct English text morphology, like `hograndmothe`, and

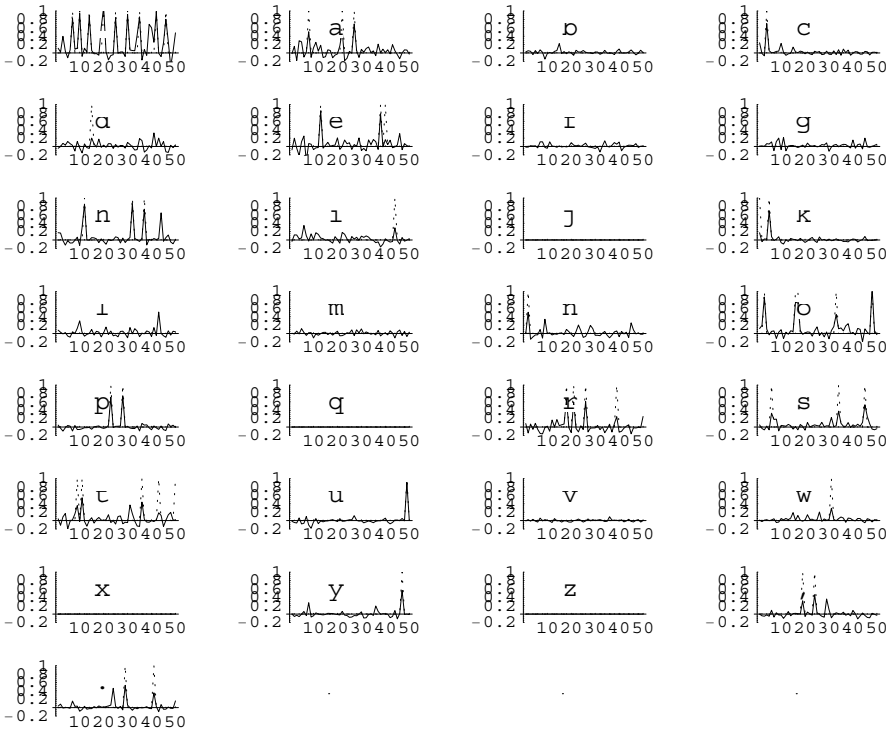


Figure 23: 50-step output sample of trained LRRH network. Letters q, x, z did not occur in training text, therefore network output is zero here.

from altogether correct words like _go_. English text is different from random sequences like the ones considered in the previous section, in that there are correct and incorrect subsequences. Therefore, it becomes an interesting question to quantify the proportion of correct subsequences.

To this end, the network production was compared to the training sequence by checking which subsequences in $\mathrm{LRRH}_{\mathrm{net}}$ occurred also in $\mathrm{LRRH}_{\mathrm{train}}$ and vice versa. Figure 24A shows the fraction of subsequences of lengthes $l = 2, \ldots, 10$ in $\mathrm{LRRH}_{\mathrm{net}}$ that also occur in $\mathrm{LRRH}_{\mathrm{train}}$ (solid line with diamonds). Multiple occurrences of a subsequence in $\mathrm{LRRH}_{\mathrm{net}}$ are counted multiple times. (Example: with $\mathrm{LRRH}_{\mathrm{net}}$ = "ababcd" and $\mathrm{LRRH}_{\mathrm{train}}$ = "xabyz" the diamond plot would give 2/5 for subsequence length 2). The dashed line with stars gives the converse statistics, i.e. the fraction of subsequences from $\mathrm{LRRH}_{\mathrm{train}}$ also appearing in $\mathrm{LRRH}_{\mathrm{net}}$.

In English text, most of the time most of the existing 29 symbols are impossible as next symbols. In terms of our learning task, "impossible" next

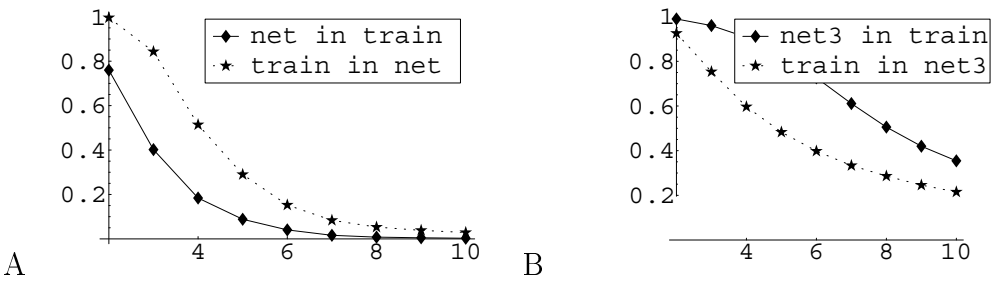


Figure 24: Comparing LRRH network babble with original. A. Network production generated by random draw. Solid line: proportion of "correct" subsequences in network production (for lengthes 2 – 10). Dashed line: proportion of subsequences in test sequence which also appear in network production. B. Same as A., network production generated by concentrated random draw.

symbols in a given context are symbols that never occurred in the context in the training sequence. A closer inspection of Figure 23 shows that the network's next-step-probability hypotheses for impossible successors are not zero, but jitter around zero. The generation procedure used so far will therefore not infrequently choose impossible next symbols. When such a "wrong" symbol is fed back into the network input, the resulting network state will be dissimilar from any network states obtained during training; consequentially, the next network output vector will even be more "jittery" and the likelihood of another impossible selection will increase. Figure 25 shows traces of some output units during the production of the `hey_go_aut...` string. It becomes apparent that the network runs more jittery than when fed with correct input (as in Fig. 23), and in the second half of the shown trace truly "stumbles over its own feet".

Figure 25: 50-step output sample of trained LRRH network in free-running generation. Outputs of nodes corresponding to letters d,e,f,g are shown.

This self-induced messing up of the network state can be diminished by a modification of the generation procedure. Obviously, the proportion of correct productions can be increased when higher output values are favored over-proportionally. This can be effected for instance by a potentiation of

the outputs with a "favor-factor" F. Step 2. from the generation procedure is then generalized to

2'. Turn the output vector into a probability vector $(p_1(1), \ldots, p_n(1))$, by setting all negative entries to zero, raise all remaining entries to the F-th power, and renormalize such that they sum to unity.

For $F = \infty$, we obtain a winner-takes-all selection. For illustration and pleasure, we present five instances of network babble generated with exponents $F = 1, 2, 3, 4, \infty$ (note that $F = 1$ corresponds to the original procedure).

$F = 1$:

```
li_ghill_ngwogp._tbie_nctogrhn,_md,_woein_and,otne_tdw_king,_there_ofhithd_.
fdylcgoreo_hee,ycmichff_umfforoug,_hmtter_cha,wanrmot_,_oelnriwftvwvg_
yoidgurlfy_grangttou_uet_auhrecheegat_thoeylsfrloet_ubfonungirpbuormahy_e.,_
hyonugetdiong_yufr_w_yawithmttbnield_grasfnevere_retas,_up_tle_rh_r_vitls_to_
```

$F = 2$:

```
rned_riding_hood_red_the_better_to_atter_top_fckngirgoghtf_butter_butter_to_
her_nigr_a__the_wolf_sone_chgeyouvery_gobbled_he_tging_hooding_hood_who_whher_
days_the_door_chhu_w_do_hnolf_little_red_a_little_red_riding_hood_ma_she_was_
grandmamma,__gat_let_sayou_hhe_bed_riding_hood_pulleget_a_t__upst_wha
```

$F = 3$:

```
who_wolf,_who_were_the_wolf_cold_grandmamma,_grand_and_see_wor_sd_a__ahe_bed_
downlt_ove_and_the_lited_her_grandmamma,_grandmamma,_who_have__the_wolf_so_en,
_her_see_her_see_her_a_the_wolf_so_nt,_grandmother_thir_of_so_ding_hood_a_
cake_and_a_little_red_her_gobbin_and_ther_see_her_the_l_the_bet__he__he
```

$F = 4$:

```
what_grandmother_see_her_grandmamma,_what_great_the_little_red_riding_hood_
wood_see_her_see_her_see_door_her_the_wolf_some_wolf_said_the_wolf_so_not_a__
the_wolf_so_ding_hood_who_was_in_the_wolf_so_did_the_wolf_so_her_see_her_the_
bobbin_and_the_bed,_she_little_red_riding_hood_who_was_in_the_better_to
```

$F = \infty$:

```
e_got_a_little_red_riding_hood_wolf_so_her_grandmother_the_better_the_wolf_
so_her_grandmother_the_better_the_wolf_so_her_grandmother_the_better_the_wolf_
so_her_grandmother_the_better_the_wolf_so_her_grandmother_the_better_the_wolf_
so_her_grandmother_the_better_the_wolf_so_her_grandmother_the_better_
```

We find that with increasing F, the fraction of correct subsequences increases – but the variability, or "interestingness", decreases at the same time. With winner-take-all selection, the generation process becomes deterministic and runs into a cycle which is made from substrings that occur in the training sequence very often: the essence of LRRH!

Figure 24B shows the fractions of correct subsequences for $F = 3$, the favor factor which yields the intuitively most pleasing network talk.

We conclude our excursion into fairyland with an inspection of KL distances computed like in the previous section. In order to obtain some sort of baseline plot, the original training sequence was segmented into length 30 subsequences, which were alternatingly assigned to two test sequences $\mathrm{LRRH}_{\mathrm{test1}}$ and $\mathrm{LRRH}_{\mathrm{test2}}$, each of approximate length 1700. For $F = 1, 2, 3, 4$, network productions of length 1700 were generated. The KL distances between the two test sequences, and between the four network productions and $\mathrm{LRRH}_{\mathrm{test1}}$ were computed. Figure 26 plots the results.

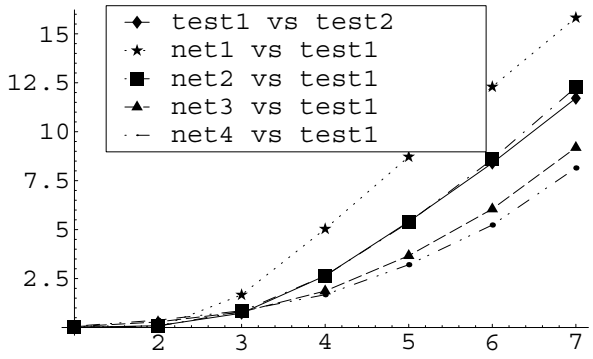


Figure 26: KL distances for $\mathrm{LRRH}_{\mathrm{test1}}$ vs. $\mathrm{LRRH}_{\mathrm{test2}}$ (solid line), and for network productions with $F = 1, 2, 3, 4$ vs. $\mathrm{LRRH}_{\mathrm{test1}}$ (star, box, triangle, dot).

KL distances do not lend themselves easily to an intuitive interpretation. What can be gleaned from Fig. 26 is that with larger F one obtains network productions that are increasingly similar to $\mathrm{LRRH}_{\mathrm{test1}}$ – for $F = 3, 4$ even more similar than $\mathrm{LRRH}_{\mathrm{test2}}$. Intuitively, this means that larger F yield productions containing the most frequent subsequences in the right fractions.

The material presented in this section is only a first, admittedly unsystematic step toward an exploration of echo state networks as text or language "surface models". Many interesting ramifications offer themselves for further investigation, of which I would like to point out the following. Because there

are no feedback connections from the output units into the network, it is possible to attach additional sets of 29 output units to the same network and train them on other texts. Each set of output units would then incorporate a "local" language model, or in other words, a language model in a particular context. It should be possible to train another network to recognize contexts (these context-recognition networks should be trained with word input rather than with symbol input). The context-recognition network could then be run in parallel with the symbol-prediction network, exploiting the context hypotheses of the first to switch between appropriate sets of output units in the latter. In this way one might obtain an altogether quite compact and at the same time, quite rich model of a language. A language, in this sense, would of course be understood merely as a nonstationary stochastic process with alternating "expert" generators (= contexts, particular training texts). The charm of this all lies in the fact that current mixture of experts models for nonstationary sequence modeling require multiple *networks* to model contexts; in our approach, only multiple *sets of output units* would be required. This promises interesting savings in required space allocation.

8 Discussion

All results presented in this article are variations on a common theme:

- A suitably configured recurrent neural network preserves in its current state information about the input history: the state is an "echo state".
- By training output units in a supervised way, systems can be obtained whose output depends on the input history.
- Because echo state networks are high-dimensional, nonlinear systems, the echo state represents in its components a rich reservoir of nonlinear transformations of the input history. This wealth of input-history-related dynamics can be eploited to realize complex nonlinear filters in a simple and computationally efficient way.
- Because there are no feedback connections from the output units into the network, an arbitrary number of output units can be attached to a single echo state network, and can be trained on different tasks simultaneously. This allows one to obtain compact multi-purpose filters, as witnessed especially in the robotics example.

Having memory is a fundamental precondition for non-trivial information processing. There are many phenomena that are usually counted as memory effects. A rough classification would be the following:

Static long-term memory realized by parameter change. During learning, parameters (aka "synaptic weights") are adjusted and remain fixed thereafter, equipping a learning system with acquired properties.

Dynamic long-term and short-term memory realized through attractor dynamics. In a previous article on echo state networks [9], various attractor dynamics were trained which realized sequence generators and switchable stable state dynamics (multi-state flipflops). Such systems can be seen as incorporating diverse forms of memory that express themselves in ongoing attractor dynamics. For instance, a periodic attractor which cycles through the output sequence $0, 0, 0, 0, 0, 1$ must "remember" at each zero output how many zeros it has already generated, in order to produce the "1" at the appropriate place. This is a case of an attractor-based memory of finite duration. A switchable multe-state attractor preserves in its current dynamic state the information about the last triggering input event, which can lie back in an arbitrarily deep past. This is an instance of an unbounded-duration dynamic memory. In the echo state framework, these kinds of memory dynamics require feedback connections from the output units to the network.

Dynamic short-term memory realized through transient dynamics. This kind of STM was the running theme in the present article, which contained a collection of variations ranging from pure delay lines to transient rehearsal to recognition and prediction tasks that implicitly are also short-term memory tasks.

The upshot of this brief and incomplete list is that there are many kinds of "memory" which complex dynamical systems can acquire, display and exploit. A principled classification would be welcome, but I doubt it can be achieved: high-dimensional, adjustable nonlinear dynamical systems are beautiful beasts which very likely can never be completely tamed. Put into this perspective, echo state networks procure *one* further strategy to make friends with these wild things.

Acknowledgments. I am greatly indebted to Thomas Christaller for his unfaltering support. Wolfgang Maass contributed inspiring discussions and valuable references. Christina Jager re-implemented the algorithms in Matlab (from the author's original implementation in Mathematica). 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 lemma 1

We show $\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l}(n-l) \rangle \; \tilde{v}_i(n)$ by induction over l.

$l = 0$: The fact that $\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_k(n) \rangle \; \tilde{v}_i(n)$ for all $k \geq 1$ is clear from the first equality in (24).

$l = 1$: We have to show that $\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-1}(n-1) \rangle \; \tilde{v}_i(n)$ for all k. Observe that a state $\tilde{v}_i(n)$ is a linear combination $\tilde{v}_i(n) = \sum_{\kappa=1}^{N+1} \alpha_\kappa \tilde{v}_\kappa(n-1)$ of the previous states (we assume a linear network!). Let δ_{ij} denote the Kronecker delta. Then conclude

$$
\begin{aligned}
&\sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-1}(n-1) \rangle \; \tilde{v}_i(n) \;\; = \\
&= \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-1}(n-1) \rangle \; \tilde{v}_i(n) \quad [\text{use } \langle \tilde{v}_1(n), \hat{y}_{k-1}(n-1) \rangle = 0 \;] \\
&= \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \sum_{j=1}^{N+1} \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \; \tilde{v}_j(n-1) \rangle \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \langle \tilde{v}_i(n), \tilde{v}_j(n-1) \rangle \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \langle \sum_{\kappa=1}^{N+1} \alpha_\kappa \tilde{v}_\kappa(n-1), \tilde{v}_j(n-1) \rangle \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \sum_{\kappa=1}^{N+1} \alpha_\kappa \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \langle \tilde{v}_\kappa(n-1), \tilde{v}_j(n-1) \rangle \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \sum_{\kappa=1}^{N+1} \alpha_\kappa \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \delta_{\kappa j} \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \alpha_j \langle \tilde{v}_j(n-1), \nu(n-k) \rangle \; \tilde{v}_i(n) \\
&= \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \nu(n-k) \rangle \; \tilde{v}_i(n) \\
&= \hat{y}_k(n). \qquad (47)
\end{aligned}
$$

$l \to l+1$: We can assume that for all $k \geq 1$ it holds that $\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l}(n-l) \rangle \; \tilde{v}_i(n)$. We show that $\hat{y}_k(n) = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l-1}(n-l-1) \rangle \; \tilde{v}_i(n)$:

$$
\begin{aligned}
&\hat{y}_k(n) = \\
&\quad = \sum_{i=2}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l}(n-l) \rangle \; \tilde{v}_i(n) \\
&\quad = \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \hat{y}_{k-l}(n-l) \rangle \; \tilde{v}_i(n) \\
&\quad = \sum_{i=1}^{N+1} \langle \tilde{v}_i(n), \sum_{j=1}^{N+1} \langle \tilde{v}_j(n-l), \hat{y}_{k-l-1}(n-l-1) \rangle, \tilde{v}_j(n-l) \rangle \; \tilde{v}_i(n) \\
&\qquad\qquad [\text{use (47) and shift invariance}] \\
&\quad = \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} \langle \hat{y}_{k-l-1}(n-l-1), \tilde{v}_j(n-l) \rangle \langle \tilde{v}_j(n-l), \tilde{v}_i(n) \rangle \; \tilde{v}_i(n) \\
&\quad = \sum_{i=1}^{N+1} \langle \hat{y}_{k-l-1}(n-l-1), \tilde{v}_i(n) \rangle \; \tilde{v}_i(n) \\
&\qquad\qquad [\text{use that} \\
&\qquad\qquad \tilde{v}_i(n) \in \text{ span}(\tilde{v}_1(n-l), \ldots, \tilde{v}_{N+1}(n-l), \nu(n-l+1), \ldots, \nu(n)) \\
&\qquad\qquad \text{and that } \langle \hat{y}_{k-l-1}(n-l-1), \nu(n-l+l') \rangle = 0 \text{ for } l' = 1, \ldots, l] \\
&\quad = \sum_{i=2}^{N+1} \langle \hat{y}_{k-l-1}(n-l-1), \tilde{v}_i(n) \rangle \; \tilde{v}_i(n). \qquad (48)
\end{aligned}
$$

B Text source for the LRRH task

$\text{LRRH}_{\text{train}} =$

once_upon_a_time_there_was_a_little_village_girl,_the_prettiest_ever_seen_her_mother_doted_upon_her,_and_so_did_her_grandmother._she,_good_woman,_made_for_her_a_little_red_hood_which_suited_her_so_well,_that_everyone_called_her_little_red_riding_hood._one_day_her_mother,_who_had_just_made_some_cakes,_said_to_her_my_dear,_you_shall_go_and_see_how_your_grandmother_is,_for_i_have_heard_she_is_ailing,_take_her_this_cake_and_this_little_pot_of_butter._little_red_riding_hood_started_off_at_once_for_her_grandmothers_cottage,_which_was_in_another_village._while_passing_through_a_wood_she_met_a_wolf,_who_would_have_liked_well_to_have_eaten_her,_but_he_dared_not,_because_of_some_woodcutters_who_were_hard_by_in_the_forest._so_he_asked_her_where_she_was_going._the_poor_child,_who_did_not_know_it_was_dangerous_to_listen_to_a_wolf,_answered,_i_am_going_to_see_my_grandmother,_to_take_her_a_cake_and_a_little_

pot_of_butter_that_my_mother_sends_her._does_she_live_a_great_way_off_said_the_wolf._oh_yes_said_little_red_riding_hood,_she_lives_beyond_the_mill_you_see_right_down_there,_in_the_first_house_in_the_village._well,_said_the_wolf,_i_shall_go_and_see_her_too._i_shall_take_this_road,_and_do_you_take_that_one,_and_let_us_see_who_will_get_there_first._the_wolf_set_off_at_a_gallop_along_the_shortest_road,_but_the_little_girl_took_the_longest_way_and_amused_herself_by_gathering_nuts,_running_after_butterflies,_and_plucking_daisies_and_buttercups._the_wolf_soon_reached_her_grandmothers_cottage,_he_knocks_at_the_door,_rap,_rap._whos_there_tis_your_granddaughter_little_red_riding_hood,_said_the_wolf_in_a_shrill_voice,_and_i_have_brought_you_a_cake_and_a_little_pot_of_butter_that_my_mother_sends_you._the_good_old_grandmother,_who_was_ill_in_bed,_called_out,_pull_the_bobbin_and_the_latch_will_go_up_the_wolf_pulled_the_bobbin,_and_the_door_opened._he_leaped_on_the_old_woman_and_gobbled_her_up_in_a_minute,_for_he_had_had_no_dinnerfor_three_days_past._then_he_shut_the_door_and_rolled_himself_up_in_the_grandmothers_bed,_to_wait_for_little_red_riding_hood._in_a_while_she_came_knocking_at_the_door,_rap,_rap._whos_there_little_red_riding_hood,_who_heard_the_gruff_voice_of_the_wolf,_was_frightened_at_first,_but_thinking_that_her_grandmother_had_a_cold,_she_answered,_tis_your_granddaughter,_little_red_riding_hood,_and_i_have_brought_you_a_cake_and_a_little_pot_of_butter_that_my_mother_sends_you._then_the_wolf_called_to_her_in_as_soft_a_voice_as_he_could,_pull_the_bobbin_and_the_latch_will_go_up._little_red_riding_hood_pulled_the_bobbin_and_the_door_opened._when_the_wolf_saw_her_come_in,_he_covered_himself_up_with_the_clothes,_and_said,_put_the_cake_and_the_little_pot_of_butter_on_the_chest,_and_come_and_lie_down_beside_me._little_red_ridinghood_took_off_her_cloak_and_went_over_to_the_bed,_she_was_full_of_surprise_to_see_how_strange_her_grandmother_looked_in_her_nightcap._she_said_to_her_then,_oh,_grandmamma,_grandmamma,_what_great_arms_you_have_got_all_the_better_to_hug_you_with,_my_dear_oh,_grandmamma,_grandmamma,_what_great_legs_you_have_got_all_the_better_to_run_with,_my_dear_oh,_grandmamma,_grandmamma,_what_great_eyes_you_have_got_all_the_better_to_see_with,_my_dear_oh,_grandmamma,_grandmamma,_what_great_teeth_you_have_got_all_the_better_to_gobble_you_up_so_saying,_the_wicked_wolf_leaped_on_little_red_riding_hood_and_gobbled_her_up._here_endeth_the_tale_of_little_red_riding_hood.

References

[1] M.W. Andrews. Processing and recognition of symbol sequences. In *Proc. 23rd Ann. Conf. Cognitive Science Society*, pages 61–65, 2001.

[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] J.L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

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

[5] 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.

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

[7] J. Hertzberg, H. Jaeger, and F. Schönherr. Learning to ground fact symbols in behavior-based robots. Submitted, 2002.

[8] H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000. Draft version: http://www.gmd.de/People/Herbert.Jaeger/.

[9] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report 148, GMD - German National Research Institute for Computer Science, 2001. Ftp'able from http://www.gmd.de/People/Herbert.Jaeger/Publications.html.

[10] W. Maass, T. Natschlaeger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. Submitted, 2001.

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

Received and put to print March 28, 2002.