Stack- and Queue-like Dynamics in Recurrent Neural Networks

ANDRÉ GRÜNING*

Max-Planck-Institute for Mathematics in the Science

Inselstr. 22–26

04103 Leipzig, Germany

Department of Psychology,

University of Warwick,

Coventry, CV4 7AL, United Kingdom

Telephone +39-040-3787-504

Fax: +39-040-3787-528

E-mail: gruening@sissa.it

August 15, 2005

*Current address: Cognitive Neuroscience Sector, S.I.S.S.A, 34104 Trieste, Italy.

Abstract

What dynamics do simple recurrent networks (SRNs) develop to represent stack-like and queue-like memories? SRNs have been widely used as models in cognitive science. However, they are interesting in their own right as non-symbolic computing devices from the view points of analogue computing and dynamical systems theory. We train SRNs on two prototypical formal languages with recursive structures that need stack-like or queue-like memories for processing, respectively. We analyse the evolved dynamics, interpret them in terms of simple dynamical systems and relate the different ease with which SRNs aquire them to the properties of these simple dynamical systems. Within the dynamical systems framework, we conclude that the stack-like language is simpler than the queue-like language without making use of arguments from symbolic computation theory.

Keywords: simple current networks, network dynamics, formal languages, dynamical complexity

1 Introduction

Artificial neural networks have been used for modelling cognitive processing as an alternative to classical symbol-based methods (Ellis and Humphreys, 1999, Elman *et al.*, 1996). This is especially true for simple recurrent networks (SRNs) in natural language and sequence processing (Christiansen and Chater, 1999a), and evidence exists that recurrent network networks (RNNs) are capable of learning the essentials of syntactic processing from scratch and that their performance matches human performance as regards for example the decay of performance with longer constructions or the preference of certain structures over others (Christiansen and Chater, 1994, Christiansen and Devlin, 1997, Christiansen and Chater, 1999b, Ellefson and Christiansen, 2000, Lupyan and Christiansen, 2002, Grüning, 2003, Monaghan *et al.*, 2003).

Yet SRNs (Elman, 1990) deserve attention in their own right as *non-symbolic* computing devices, and are of interest from the perspective of theoretical computer science since they are instantiations of analogue computers, whose computational capabilities have been theoretically assessed and compared to those of well-known symbolic computers or automata (Hopcroft and Ullmann, 1979): it has been shown that RNNs can emulate a Turing machine, and might even perform Super-Turing computation (e. g. Siegelmann and Sontag, 1995, Hammer, 1997, Moore, 1998). While these theoretical assessments often restrict themselves to hand-coding weights in order to implement a certain dynamic desired for the emulation of a symbolic automaton in an SRN, and thus did not lay weight on the process of training, most of the simulations on language processing have focused on the learning process and the final performance and have been interested in the inner SRN dynamics to a lesser extent from a computational point of view.

However, it is not a-priori obvious that the dynamics suggested in the theoretical approaches can serve as an idealisation of the dynamics that evolve when SRNs are actually trained on a natural or formal language. On the contrary, the hand-coded network dynamics implement symbolic automata in such a naive one-to-one way that the advantages of an analogue architecture are lost, e. g. a quantitative metric similarity measure between related processing states (van Gelder, 1998, 1999, Visser *et al.*, 2004). Furthermore SRNs have emergent

architectural biases that might be different from the biases of a human tailoring an RNN fitted for a certain dynamical system (Casey, 1996, Hammer and Tiňo, 2003, Tiňo *et al.*, 2004, Tiňo and Hammer, 2003, Visser *et al.*, 2004), and an emerging representation of a formal language might look quite different from the one in symbolic automata. So far, little is known about the *systematic* relationship of the type and complexity of a formal language to the complexity of its (emerging) network dynamics. For example, Tabor (2000) shows that when changing parameters minimally, essentially the same network dynamics moves from processing a context-free language to a non context-free one. Hence the classical strict boundaries of representational power between different language classes and automata do not play a primary role in analogue computing.

With this work we would like to contribute to filling on the one hand the gap between our knowledge of the theoretical power of SRNs and what has been practically achieved by learning in network simulations. On the other hand we want to analyse what kind of structured dynamics RNNs or more specifically SRNs with second-order weights suggest as representations of prototypical formal languages when they are trained, and how these dynamical representations relate to language properties.

The underlying general idea of training SRNs on languages is that the network learns to decide which properties of the input symbols are relevant to produce the desired output and that it to this end must reconstruct the relevant or causal states (Crutchfield, 1993, 1991) for the output as a suitable summary over the inputs seen so far. It has been shown that the causal states emerge as clusters of state vectors (Cleeremans *et al.*, 1989).

In this article we concentrate on two simple formal languages, a deterministic palindrome language and a deterministic duplication language. These two languages have been essential for defining the standard view on symbolic computing since they exhibit two basic kinds of recursion, namely a stack-like and a queue-like structure. Furthermore these structures are also found in the syntax of natural languages (Savitch *et al.*, 1987) and are thus also of interest from a cognitive point of view. We are especially interested in the dynamical differences of the representations of our two languages and want to see how these differences are related to the properties of the underlying formal languages.

2 Background

Let us start stating important differences between symbolic and analogue automata. The main properties of symbolic automata (Hopcroft and Ullmann, 1979) are (i) a discrete state space, (ii) a single powerful finite control, (iii) the distinction between finite control and (possibly infinite) memory resources, and (iv) hard limits as the typical behaviour when resource limits are reached. Usually the limited resources take the form of restricting the amount of memory or time consumed as a function of the data length, and it is well known how a given language is represented in such an automaton using rewrite and transition rules of a particular type. Opposed to this, the main features of recurrent neural networks acting as analogue computing devices (in the guise of dynamical recognisers, Moore 1998) are (i) a continuous but bounded state space, (ii) many elementary processors (model neurons) but connected in a complex way, (iii) no distinction between control and memory, both are encoded in the connecting weights (model synapses), (iv) the primary limited resource is numeric precision of dynamic variables, and results in soft limits and graceful degradation. While the questions of representation and complexity in a symbolic setting have found answers, these questions are largely unanswered in an artificial neural network setting.

It was recognised early that neural networks with discrete neurons, i. e. neurons whose activation can only take on a finite number of values, are closely related to finite state automata (FSAs). In fact, McCullough and Pitts (1943) gave birth to the concept of both artifical networks and FSAs. And accordingly, simulations (e. g. Cleeremans *et al.* 1989, Giles *et al.* 1992, Omlin 2001, Lawrence *et al.* 1998, for a comprehensive overview see Jacobsson 2005) with continuous neurons soon revealed that approximations to the original FSA could be extracted from an SRN trained on the corresponding language. Casey (1996) proved that SRNs must organise their state space such that regions in state space correspond to states of a finite state automaton (FSA) when successfully trained on a regular language under the assumption of noisy trajectories ((with a bounded noise process, e. g. a consequence of limited numerical precision). Thus, for regular languages there is a good match between theory and simulations.

In practical cases however, the SRN hardly represents a FSA perfectly (e. g. due to finite training). A perfect representation would for example entail that closed loops in the FSA correspond to—in the simplest case—attractive fixed points of certain combinations of SRN dynamics. In simulations, SRNs might instead of such a fixed point use a slowly transient region that suffices to keep the network activation in a certain region for a certain number of time steps (Omlin, 2001, Zeng *et al.*, 1993): for practical purposes it is not important to have a real attractive fixed point as long as there is a slowly transient region that is *functionally equivalent* at least for the string lengths encountered during training and test.

A general remark on idealisation in symbolic and analogue systems might be appropriate here. Every concrete symbolic automaton (for example a desktop computer) has only a finite amount of memory. Theoretically speaking, it is therefore only equivalent to a (very complex) FSA and thus can only process regular languages. For any practical purposes, however, it is fine to treat it as a Turing machine (or rather as a Turing-equivalent random access memory machine) assuming there are no memory limitations: the structure of this complex FSA is better captured by the idea of it being infinite (Savitch, 1992).

The resource in analogue computation that compares to memory is precision. And likewise there are results (Casey, 1996, Moore, 1998, Maass and Orponen, 1998) that with limited precision (for example from ubiquitous noise or numeric imprecisions) at best regular languages can be processed reliably (when all the fixed points are really attractive fixed points and not slowly transient regions). But similar to the example of the symbolic desktop computer, the essentials of a dynamics are often better captured in the spirit of analogue computation when we abstract away from limited numeric precision and imperfect fixed points.

We have noted above that the state space of an SRN is not discrete and can thus accommodate an infinite number of different states. Already in the case of regular languages the SRNs show a tendency to represent some languages in a 'fractal' way, a tendency to make finer subdistinctions in the state clusters (Cleeremans *et al.*, 1989) than would functionally be necessary for producing the correct output or correct state space transition upon the next input: states tend to cluster closer to each other not only when they have a similar future

but also when they have a longer similar past. Thus the dynamics are sensitive for previous context.

While this sometimes impedes processing of regular languages it enables SRNs to process non-regular languages using an infinite number of states: for non-regular languages the finer structures within clusters often do play a functional role due to expanding network dynamics that eventually map nearby states onto very different regions with different output and transition behaviour. From the present experimental evidence (see below), one can conclude that SRNs use distance between states to encode similarities of the past and future trajectories of these states. Dynamics are tuned such as to diminish or enhance appropriately the distances between states just according to how relevant the differences between these states are for the next few outputs or transitions, thereby making efficient use of the compact state space. In other words, functional relationships between states are encoded in the geometrical relationships between clusters and their finer sub-structures (see also Tiňo and Hammer 2003 for the encoding of statistical properties of network input into its state space geometry). Of course, in view of Casey's theorem representations of non-regular languages cannot be robust to noise since due to the compactness of the state space there are state clusters that get arbitrarily close as their number increases however small they might be.

To wrap up the arguments, due to 'imperfect' attractive sets, noise and compactness of the state space, we cannot expect any learnt dynamics to process the whole infinity of a language. However, the network will still try to adapt its dynamics such that the distinctions between states most relevant for the time being are most easily accessible geometrically. Therefore the geometric distribution of clusters can still serve as a starting point for the interpretation of the dynamics. This has found evidence in the following simulational work:

For context-free structures, simulations focusing on both cognitive and dynamical system aspects have been carried through. Pollack (1991) suggested how to embed a counter required for the simplest non-regular language $a^n b^n$ (i. e. *ab*, *aabb*, *aaabbb*, ...) in a single dimension, anticipating later results of Wiles and Elman (1995), Bodén *et al.* (1999), Wiles *et al.* (2001), Bodén and Blair (2003), Bodén and Wiles (2000). Rodriguez (2001) analysed

some more sophisticated context-free languages, finding dynamics for the palindrome language comparable to ours.

For context-sensitive structures only few simulations have been carried out: Christiansen and Chater (1999b) focused on the performance of SRNs as compared to humans, and Tabor (2002) centred the dynamical analysis of trained networks on decay properties of the evolving network dynamics. Rodriguez (2001) also analysed some simple context-sensitive languages and Bodén and Wiles (2000) concentrated on $a^n b^n c^n$. All of these languages fall into the queue automaton (QA) subclass QA_1 of languages (Cherubini *et al.*, 1991) properly contained in the class of context-sensitive languages, but they still make only marginal use of the cross-serial type of recursion typical for this class.

In our simulations we will—to our knowledge for the first time—use a duplication language that will make more thoroughly use of cross-serial recursion. In fact, it is almost as prototypical a language for this type of recursion as is the palindrome language for context-free centre-embedding recursion (Cherubini *et al.*, 1991). And we will contrast the evolved dynamics for these two languages in order to relate their dynamical and formal representation.

3 Simulations

In the following, we will firstly define what we mean when we talk about the palindrome and duplication languages. Then we will describe what the network's task will be when learning these languages. We will, thirdly, define our second-order variant of SRNs and, fourthly, introduce a training scheme that includes a hill-climbing aspect, and, finally, we will describe the simulations. However a detailed discussion of the successful networks' dynamics will be deferred until the next section.

3.1 Languages

In order to keep the simulations as simple as possible, we concentrate on simple formal languages exhibiting centre-embeddings and cross-serial recursiveness over a binary alphabet $\{a, b\}$. Let w by an arbitrary string over a, b.

The palindrome language (figure 1) then is the set of all strings of the form ww^r, i. e. a string w followed by a mirror-reversed copy w^r of w. It is a context-free language and therefore needs a stack-like memory to keep track of its centre-embedding recursion. The duplication language (figure 2) is the set of strings ww, a string w followed by an identical copy w of the same string. It belongs to the context-sensitive class of formal languages and needs for its cross-serial dependencies a queue-like memory.[1]

By a 'stack-like memory' we mean a memory that can—like the stack in a push down automaton (PDA)—only be accessed in a 'first-in last-out' manner, i. e. before reading out a particular item, all other items that were stored after it have to be read out and removed from the memory. A 'queue-like memory' is a memory that can only be accessed in first-in first-out manner: if you want to read out a particular item, all other items stored before it have to be read out and removed from the queue before access is granted.

[Figure 1 about here.]

[Figure 2 about here.]

3.2 Task

The network learns these languages in the following *deterministic* transducer task (Meduna, 2000): First a string w is fed to the network symbol by symbol, then upon input of a special symbol ¶ it is to produce the second part w^r or w symbol by symbol as its output. When there are no more symbols to reproduce for input ¶, the network should reply with the special symbol E (see figure 3 for example input/target pairs).

Note that a recognition task on ww^r or ww would require a non-deterministic automaton since the transition from the first w to the second w^r or w cannot be determined deterministically. However w can be transduced deterministically into w^r or w since ¶ explicitely indicates that the input of the first w is finished.

[Figure 3 about here.]

[1] In fact, cross-serial structures do by far not exhaust syntactic structures allowed for by context-sensitive grammars. The appropriate grammar class and automata type are context-free breadth-first grammars and real-time QA (Cherubini *et al.*, 1991).

This transducer task is a variant of prediction learning. We chose it, since the second part w^r or w of a string ww or ww^r is perfectly predictable once the first part w has been encountered.[2] On the other hand since w is constructed randomly of as and bs, no symbol in the first part w contains any hint about any other symbol in this first part, and the same argument applies to the second part. Thus the network would not gain from being supplied with previous symbols (as in a prediction task). It must simply organise its dynamics as to allow for an appropriate stack-like (for ww^r) or queue-like (for ww) memory to evolve.

3.3 Network

We slightly broaden the notion of SRNs and use for the simulation networks with second-order weights similar to the networks used for example in Pollack (1991) with the following update function. Let $i^{(t)}$ denote the input vector at time t, $x^{(t)}$ the vector of activations of the neurons in the hidden layer, and $o^{(t)}$ the output vector at time t. Then

$$x_i^{(t)} = f(w_{ijk} x_j^{(t-1)} i_k^{(t)}), \quad o_i^{(t)} = f(v_{ijk} x_j^{(t)} i_k^{(t)}) \tag{1}$$

where w_{ijk} and v_{ijk} are the weight matrices, f the standard sigmoid function $x \mapsto 1/(1+e^{-x})$, and summation over the indices j and k is implicitly understood. The bias term is realised as a neuron with constant activation equal to one. According to the taxonomy introduced in Kremer (2001) we use a spatiotemporal connectionist network (STCN) with second-order context memory.[3]

We use this second-order variant of an SRN since the search space for possible solutions is bigger than for standard first-order SRNs (Bodén and Blair, 2003), at least if applied in conjunction with gradient based learning algorithms (Bodén *et al.*, 2000). The standard back-propagation through time (BPTT)(n) algorithm (e. g. Williams and Peng, 1990) is straight-forwardly adapted to include second-order weights. In addition, we added some minor modifications

[2] In a prediction task the information that the second part of a string has been reached is usually also supplied explicitly by drawing the symbols for the second part from a different alphabet, i. e. one uses wW, where W is a copy of w in which all letters are replaced by the uppercase equivalents.

[3] Using Kremer's notation and variable names, our update function of the hidden layer reads: $f_{\vec{s}}(\vec{s}(t-1), \vec{x}(t), (W)) \equiv \vec{\sigma}(\mathbf{W}\vec{x}(t)(1 \oplus \vec{s}(t-1)))$ and the output function reads: $f_{\vec{y}}(\vec{s}(t), \vec{x}(t), (W)) \equiv \vec{\sigma}(\mathbf{W}^{(\mathbf{out})})\vec{x}(t)(1 \oplus \vec{s}(t))$.

to improve speed (see Appendix: Training algorithm). We stick to a variant of BPTT and SRNs despite their well known technical short-comings (e. g. Bengio *et al.*, 1994, Bodén *et al.*, 1999) since it is a standard tool in cognitive science and furthermore the hidden state space is more amenable to analysis than in technically more versed types of networks and learning algorithms (Hochreiter and Schmidhuber, 1997, Schmidhuber *et al.*, 2002).

If one—as we do—employs input symbol patterns in which all but one input neuron are 0, we get for each input symbol $a, b, ¶$ a function $F_S : x_i \rightarrow f(w_{ijk} x_j i_k^S)$ where i^S is the input vector corresponding to the input symbol S that maps the hidden state space X of the network onto itself. The F_a, F_b and $F_¶$ make thus up a system of iterated functions on X which can be treated, together with the partition of the state space given by the output mapping, as a type of dynamical recogniser (Moore, 1998).

3.4 Systematic retraining

In previous work on non-regular languages and SRNs, researchers had a hard time training their networks. Often only a fraction converged successfully on the target languages, and therefore they frequently concentrated on the best networks which were analysed as exemplary in-principle solutions.

Frequently, they subjected these best networks to retraining with a smaller learning rate η to improve and fine-tune their performance (Rodriguez and Wiles, 1998, Rodriguez *et al.*, 1999, Bodén and Wiles, 2000, Rodriguez, 2001). In this paper we want to follow this line, but do the retraining in a more systematic manner. We therefore introduce a hill-climbing aspect as a systematised version of retraining with a lower learning rate η, see figure 4: a network is trained for a fixed number of epochs on the training set with a fixed learning rate. After each training epoch, the network is run on a test corpus on which its performance in terms of the number of correctly processed strings is determined, and finally its weights are saved. Afterwards those weights that have given the best performance on the test set are restored and subjected to retraining with a smaller learning rate. For testing, a string counts as an error when its second half is not correctly reproduced.[4]

[4]For testing, we account for deviations from the target output, interpreting the output

[Figure 4 about here.]

3.5 Preliminary simulations

Let PAL(n) and DUP(n) denote the finite subsets of all ww^r and all ww respectively where w is allowed to have a maximal length n.

In preliminary experiments (with set-up and parameters similar to the case described in more detail in the next section, see footnote 6) we trained PAL(n) and DUP(n) with n ranging from four to seven on at least 100 second-order SRNs with three to ten neurons in the hidden layer. The training sets of size 1000 were drawn randomly from the language under consideration with a bias for shorter strings. The test sets consisted of all strings of the actual language ordered according to, firstly, increasing length, and secondly lexicographic order (Jacobsson and Ziemke, 2003). Thus the test sets overlapped with the training sets.

In these preliminary experiments, we wanted to include strings from at least PAL(4) and DUP(4) because we judged this the minimal length for which the structure of their infinite generalisations would be sufficiently clear. It turned out that the hidden layer needed at least five neurons to produce some networks that processed DUP(4) without error. Even though a higher number of hidden units (HUs) allowed to process longer strings correctly we judged that the network dynamics would be easier to identify in networks with only five HUs. The palindrome language PAL(4) was learnt without error by a majority of networks with five HUs, so we decided to focus on the more difficult PAL(5), for it is important to train the networks at the edge of learnability for structured dynamics to emerge (Cleeremans *et al.*, 1989).

Results are that five out of 125 networks learnt PAL(5) without error and eleven out of 100 learnt DUP(4) without error.[5]

activation as follows: E if the activation of the corresponding output neuron is ≥ 0.5, otherwise a or b whichever has higher activation.

[5]See Rodriguez (2001) for a similar approach on the palindrome language. Rodriguez used first-order SRNs and trained them in a classical prediction task on PAL(6). In a prediction task the appearance of the first symbol of the second part cannot be foreseen. Therefore the string lengths are comparable to ours (PAL(5)). The trained first-order networks showed dynamics similar to ours. While we found some networks that could learn PAL(5) without error, Rodriguez's best network got 78% of the strings in PAL(6) right. However Rodriguez enriched PAL(6) from the outset with longer homogeneous strings, and was able to report on

In sum, we decided to concentrate on networks with five HUs since they seemed to be an acceptable comprise between computational power and amenability to analysis. We will describe these simulations with SRNs with five hidden units trained on PAL(5) and DUP(4) in more detail in the next section.[6]

The fact that we were not able to train networks with five HUs to perform without error on DUP(5) (the best networks still got 18 out of 62 strings wrong) reveals, as a first result, that the palindrome language is easier to learn than the duplication language.

4 Dynamics

However we are not primarily interested in the networks' performance *per se* but rather in the evolved dynamics and an explanation of the performance differences of ww^r and ww in terms of the dynamics. To this end, we concentrate on the simulations with SRNs with five neurons in the hidden layer trained on PAL(5) and DUP(4), respectively, from the previous section.

We take those networks that did not make any error on their respective language and have a closer look on their evolved dynamics before suggesting a dynamics that abstracts away from peculiarities and imprecisions in individual network dynamics.

4.1 Evolved dynamics

In order to analyse the dynamics, we took those networks that had learnt their respective language without error and recorded the state of the hidden units while they were processing 1000 strings of their languages. Not surprisingly, state vectors show a tendency to cluster close to each other (Cleeremans *et al.*,

some generalisation beyond this training set (the best network got about 34% of PAL(7) right). Our own attempts towards generalisation were equally inconclusive. Therefore we refrain from reporting them.

[6]For these language we used the following parameters: Strings of lengths 1, 2, 3, 4, 5 were selected with probabilities $0.05, 0.15, 0.2, 0.4, 0.2$ for PAL(5) and strings of lengths 1, 2, 3, 4 with probabilities $0.05, 0.15, 0.4, 0.4$ for DUP(4). All strings of equal length were equally probable. The parameters of the training (see the Appendix and figure 4) were for PAL(5): $e_{max} = 1200$, depth of unfolding $n = 11$ (the length of the longest string in PAL(5) plus an extra ¶ to test for empty storage). For DUP(4) we had $e_{max} = 600$ and $n = 9$. For both languages, we started with a learning rate $\eta = 0.1$, divided by 2 for each hill-climbing step until it fell below 0.001.

1989) according to the contents a symbolic stack or queue (see figures 1 and 2) would have after having seen the same input string of as, bs, and ¶. Furthermore we note that clusters of state vectors are not distributed randomly in the network's state space: the stack- or queue-like memory that evolved in training is encoded in the spatial relation of the clusters and the dynamics of transition between them effected by the different inputs.

We visualised the five-dimensional (5D) hidden state space of the SRNs as a sequence of two-dimensional (2D) projections in which principal component analysis (PCA) proved helpful or not, depending on the individual network.[7] By inspection, we found that 2D structures of clusters such as in figure 5(a) for ww^r and in figure 6(a) for ww were recurrent in many of the well-trained networks during the input of the first half w of a string, and thus we decided to focus our analysis on these structures. In the sequel we concentrate on the two particular networks from which the graphs in figures 5 and 6 were derived since they are prototypical examples of the evolved dynamics in the other networks.

[Figure 5 about here.]

[Figure 6 about here.]

We note that along one direction, the number of preceding input symbols is counted: for ww^r roughly from right to left in figure 5(a), and for ww from top to bottom in figure 6(a). The distances between clusters projected along this direction show a tendency to decrease with an increasing *number* of inputs. Thus this direction is reminiscent of a monotonous multiplicative counter (Bodén and Blair, 2003) and can be said to encode the memory (stack or queue) *depth*. The type of the inputs seen so far, i. e. the *contents* of the (stack- or queue-like) memory is, in contrast, encoded in a direction roughly perpendicular to the first: travelling from low to high memory depth, the trajectories have a tendency to branch always to the same side according to the last input a or b.

But the dynamics for ww^r and ww are qualitatively different. For ww^r, inputs cause large jumps (length ≈ 0.8 in figure 5(a)) in state space, even for

[7] We considered as well three-dimensional projections which however did not reveal any structure above and beyond the 2D ones. In fact, it would be desirable to also take into account non-linear sub-manifolds of the state space, and not only the linear ones defined by projections and PCA. However here we content ourselves with linear sub-manifolds as in most previous work in this field.

greater stack depth, and thus trajectories cross each other frequently.[8] For ww we get a different picture. Here inputs tend to cause shorter jumps in state space with increasing queue depth, and trajectories tend not to cross each other.[9] The structures are less pronounced for higher memory depth. This might be caused by nonlinear distortions due to the neurons' activation functions reaching saturation. Still it is easy to see how these structures could be extended to higher depth in the absence of such distortion.

4.2 Essential dynamics

Of course, for each individual network the dynamics look a bit different from the prototypical ones presented, but we want to understand the essentials of the dynamics underlying the representation of the ww^r and ww languages in SRNs. This is, we aim to find an idealised 2D dynamical system that solves the task, resembles the actually encountered dynamics closely enough, and with infinite precision would be able to deal with strings from ww^r or ww of unbounded length. Previous work (e. g. Bodén and Blair, 2003, Rodriguez, 2001) has demonstrated that the dynamics of networks in formal language tasks can often be interpreted in terms of simple dynamical systems that abstract away from the peculiarities and distortions of its concrete realisations in the networks. Is there such an interpretation even in the case of ww^r and ww?

How, then, can we implement a stack-like memory in 2D in a simple way? Remember the activation of the hidden neurons are confined to $[0, 1]$.[10] And assume the symbols of our alphabet a, b correspond to the numeric values 0 and 1.

The key idea here is to find a real number $x \in [0, 1)$ with binary expansion $x = 0.x_n x_{n-1} \cdots x_0$ given an input sequence $x_0 x_1 \cdots x_n$ where all x_i are binary digits. Let us start with $x = 0.0$ and insert 0 or 1 according to input a or b in the most significant digit of its binary expansion. Dividing x by 2 at each new

[8] This is clear for the projections shown here and need not necessarily be so for the full 5D dynamics. These results have also been confirmed calculating the full 5D Euclidian distance between cluster centres.

[9] They do so only close to the border of the state space where nonlinear distortion is high.

[10] In fact, 0 and 1 can only be reached as asymptotic activation values. Essential for the following argument is only that the state space is bounded. Then by rescaling we can make it fit to whatever range of activation we like.

input symbol corresponds to shifting right the binary expansion by one digit. In the new vacant first position we can then insert a 0 or 1 whichever the input. Reading out the sequence now stored in x we simply have to multiply x by 2 and check whether x is less (output a) or greater than 1 (output b) and dispose off the digit before the colon. Essentially this implements a dynamical systems version of the full one-sided 2-shift on the digits of the binary expansion of x, i. e. by multiplying x by 2 each digit in this representation is moved one position to the left (Kitchens, 1998).

But how do we know when x corresponds to the empty storage? Both no input symbol and any input sequence of only as would be assigned $x = 0$. We simply use a second dynamic variable y to act as a multiplicative counter (Bodén and Wiles, 2000) whose counting constant we assume for the sake of simplicity to be 2: we initially start with $y = 1$ and divide it by 2 each time a symbol a or b is input. We multiply by 2 if symbol ¶ is presented. When $y = 1$ again, the memory is empty and thus we are done (output E).[11]

The suggested essential dynamics is summarised in figure 7(a) and its phase space portrait is given in figure 5(b), contrasting with the actual dynamics. Most characteristically the most recent symbol input has the greatest influence on the real number state since it corresponds to the most significant digit of x and thus leads to rather long trajectories which cross, even for high stack depth. This solution is very natural since the most significant digit that stands for the symbol which is top on the stack memory is most easily accessible by comparison operations in an analogue computing setting.

[Figure 7 about here.]

For ww we need a queue-like memory. Thus the key idea here is to find a dynamical system that delivers a real number $x \in [0, 1)$ with binary expansion $x = 0.x_0x_1 \cdots x_n$ given an input sequence $x_0x_1 \cdots x_n$. We encounter the difficulty that we have to directly manipulate a less significant digit of x in order

[11] For the essential dynamics, in fact, one should divide by a number slightly greater than 2 in order to get a gap between the real number representations of $0.0111\cdots$ and $0.111\cdots$ (which does not make a big difference unless one really wants to process infinitely long strings). One could also use a ternary or quaternary encoding for x in order to get rid of the separate memory depth counter y, see Siegelmann and Sontag (1995). However dynamics with both x and y are closer to the trained dynamics.

to insert the new input symbol into its binary expansion. And for this we need to keep track of the next vacant digit, i. e. the up to now least significant digit. This is done by assigning a double role to the multiplicative counter y otherwise identical to the stack case: y is such that its value corresponds to the last used digit of x, and thus $y/2$ can act as a pointer to the corresponding next vacant digit.

Thus we start again from $x = 0, y = 1$ and for every b input we add $y/2$ to x which we leave unchanged for input a, while y is divided by 2. Thus the value added to x does not solely depend on the input symbol, but also on the counter y that masks the relevant digit of x. Reading out proceeds exactly as for ww^r: x is left-shifted and y multiplied by 2 for each ¶ until $y = 1$ is reached.

Note that the essential dynamics for ww^r and ww share the same state vectors but connect them in a different way. And while the dynamics for input ¶ are the same as for the stack, the a and b dynamics are different: the symbol first in queue memory has the biggest influence on the real number value of x and thus the trajectory lengths decay with increasing memory depth (and thus decreasing value of y) such that the trajectories do not cross each other. This solution requires that small changes are effected on comparatively large numbers.

In figures 5(b) (centre-embedding) and 6(b) (cross-serial) the essential dynamics are drawn. They yield a good match with the actual dynamics given in figures 5(a) and 6(a), i. e. the essential dynamics capture the recurrent properties of the actual dynamics well.

4.3 Further corroborations

The idealised dynamics we have suggested so far were essentially inspired by looking at the actual dynamics during the input phase (of as and bs) in figure 5(a) and 6(a). However, they also make a statement about the reproduction phase during ¶ input. In fact, the idealised dynamics in the reproduction phase look the same for ww^r and ww, compare the lines for input ¶ in both figure 7(a) and (b), and for both dynamics the trajectories look like those in figure 5(b) but are of course traversed in the opposite direction.

However, even though the dynamics during the a,b input phase is formed

mainly by the errors injected during the reproduction phase (since the outputs are not relevant in the input phase and the network learns fast to keep all of them close to zero for that phase), its dynamical visualisation gives much clearer an interpretable impression than those during the reproduction phase for which direct error feed-back is available.

[Figure 8 about here.]

First of all, the reproduction dynamics for input ¶ often do not take place in the same 2D plane as the input dynamics which is not surprising since we did not train the networks on alternating a,b versus ¶ inputs.

Secondly, the reproduction dynamics find a different set of dimensions to accommodate the clusters depending on the symbol with is topmost in memory: in figure 8 e. g. clusters corresponding to A top on stack are coded along HU5 while those with top B are along HU2. The rough counting direction for both groups of clusters is given by HU4.

Thirdly clusters are split according to the number of ways they can be reached. For example state vectors that correspond to three stored symbols split up into three subclusters since the network keeps track of the states' history (Cleeremans *et al.*, 1989): states for ABB can be reached from the clusters for $BABB$ and $AABB$ by input ¶ but also directly from BB by input a. State vectors corresponding to two symbols on the stack can spilt into up to seven clusters: six clusters by input ¶ and one directly by input a or b. Even more clusters exist for the states with only one symbol, but here the subclusters already start to overlap. Finally this is an explanation of why the cluster corresponding to empty memory is fairly dispersed: it can be reached by input ¶ from the even bigger number of subclusters corresponding to a single symbol in memory. Therefore it has been left out in figure 5(a) and 6(b) and now also in figure 8.

Nevertheless for ww^r the picture given by figure 8 still resembles the idealised reproduction dynamics in so far as HU4 is clearly a kind of counting dimension as the suggested y, and clusters are ordered along HU2 (or HU5) according to the corresponding memory contents in a similar fashion as in figure 5(a) and (b).

The emerging dynamics for the duplication language ww during ¶ input also show a spilt-up of the clusters. We can still identify a rough direction

states move in with decreasing queue depth. However, a clear order of the clusters perpendicular to this direction cannot be identified. At least clusters corresponding to the topmost queue symbol A (representing queue contents as a string of As and Bs as in figure 2) lie in the one half perpendicular to this direction, but partly so do subclusters with topmost symbol B that derive from a state with topmost A by input ¶. Still the dynamics are not just random but structured and we find it more revealing to present a flow diagram of the ww dynamics for all three input conditions a, b and ¶, see figure 9, derived as follows.

The recorded state vectors for ww were PCA transformed and the two dimensions with the largest variance determined. Then we laid a regular grid over the plane spanned by these two dimensions and iterated the network dynamics for each input symbol on the points of the grid. The down-scaled difference vectors were then projected back to the selected principal component (PC) plane and set off from the points in the grid.

[Figure 9 about here.]

The vertical direction (PC5) is the direction in which counting the number of stored symbols takes places (corresponding to y in the idealised dynamics) with the state for empty memory roughly at the top edge of the triangular outline, whereas the horizontal direction is used to differentiate between a and b inputs, taking on the role suggested by variable x. While the a and b flow are pointing away from the empty memory state, the flow for input ¶ points back to it and spans about equally big but opposite angles with the a and b flow, respectively. Furthermore the flow gets weaker for all input conditions the greater the distance to the empty memory state. Apart from the distortion in the lower right corner, this agrees with the flow the idealised dynamics in figure 4(b) suggests.

4.4 Discussion

In sum, we find that SRNs that are trained on initial parts (i. e. limited string length) of the centre-embedding language ww^r and on the cross-serial language ww develop dynamics that allow for an interpretation in terms of simple dynamical systems: the networks do not spatially distribute state clusters in an

unstructured way, but tend to geometrically organise the clusters as modelled by the essential, idealised dynamics in figure 7. We can thus conclude that the networks do not learn these finite initial parts of non-regular language by rote, but grasp the recursive aspects of the full infinite versions of these languages, even though they are not able to process the full infinite languages due to dynamic imprecisions such as 'imperfect fixed points', i. e. slowly transient regions replacing real attractive fixed points that however still suffice to keep activity in a certain region for some finite time, or noise and numeric imprecisions.

Thus having found an appropriate interpretation of the dynamics, we note that in the case of the centre-embedding language ww^r the dynamics for the dynamic variables x and y are independent whereas for the cross-serial task the value of y directly influences the value of x. This means imprecisions in the dynamics of y carry over to x and accumulate there, while in the centre-embedding case imprecisions of x and y are independent. Hence we need a more precise tuning of the weights for the queue-like memory dynamics during training to achieve the same performance which makes the cross-serial language more difficult to learn. We can say that ww has a higher complexity than ww^r purely on dynamical reasons in this SRN setting, independent of argumentation with the symbolic complexity classes in the Chomsky hierarchy.

For the counting language $a^n b^n$, only one type of network solutions has been found up to now: the multiplicative counter. However it can appear in three guises: monotonic, oscillating or spiral (Bodén and Blair, 2003). We could not identify the dynamics in some well trained networks for $ww^r$and ww: instead of counting in a monotonic way in y they might employ oscillatory or spiral counting or even combinations thereof. This would of course obscure the clearer dynamical picture of figures 5(a) and 6(a) rendering the dynamics less amenable to inspection. This will be explored in more detail in forth-coming work.

5 The opposite task

In a certain respect, our simulations on the palindrome and duplication languages correspond to those of Tonkes *et al.* (1998). They studied how two jointly trained artificial neural networks communicated a semantic concept modelled

as a real number over a binary time-discrete channel within a fixed time. Semantic concepts often can only be approximately communicated between two individuals, similarly to how bounded real numbers are approximated by a finite binary expansion. Thus the semantic concepts were fixed while a symbolic code developed. However, the needs of the networks acting as the encoder and the decoder are different. While the encoder tended to implement the leftward 2-shift $x \mapsto 2x \mod 1$ and thus to transmit the most significant digit first, the decoder preferred to implement the rightward 2-shift $x \mapsto \frac{1}{2}x$ and thus to receive the least significant digit first. A code similar to the binary representation of x developed if the symbol string was reversed before it was fed into the decoder (stack-like task). If there was no such string reversal (queue-like task) the task was much harder to learn and the resulting symbolic code became a compromise between the different needs of encoder and decoder.

Thus also in Tonkes *et al.* (1998)'s simulations the queue-like task was harder to aquire than the stack-like task. But there is as well a symmetry on the level of cognitive interpretation. In Tonkes *et al.* (1998) the semantic concept (the real number) was given and the symbolic code had to evolve. In our simulations the symbolic code is given and to be encoded as a real vector during input of a and b and decoded into a symbolic sequence again in the reproduction phase with input ¶. Thus here the mental representation (the real state vector) has to evolve from a symbolic code.

6 Conclusion

We trained second-order SRNs on short strings from the centre-embedding ww^r and the cross-serial language ww. Most networks did not learn these languages well. But as we were mainly interested in how a network can represent such languages in principle we concentrated on those networks that did well on these languages. For the interpretation of the dynamics we visualised different 2D projections of the state spaces of these networks and focused on those trajectory structures that reappeared in many of them.

Even though the networks did not learn the whole infinite languages (i. e. for unbounded stack and queue depths) they still grasped important aspects of

their recursiveness. They did not learn the strings of the language by rote but in a systematic fashion: the geometrical arrangement of clusters in state space is used to encode functional relationships between them.

It turned out that the duplication language is more difficult to learn than the palindrome language since it is dynamically more demanding: the sub-dynamics for the dynamic variables x and y are directly coupled for ww while they are not for ww^r. Thus, in this analogue setting, the two languages have different complexity due to their dynamics, and not because they would require different symbolic automata. These results have bearing on the following:

Cognitive Science and Cognitive Linguistics. Adopting a dynamical systems view of cognition, i.e. that artificial neural networks and analogue computing are suitable for modelling cognition (the so-called 'dynamical metaphor' of cognitive processing as opposed to the classical 'symbolic metaphor', Bates and Elman 1993, Jaeger 1996, van Gelder 1999), our findings can be a building block in how complex cognitive processes are accommodated in analogue neural hardware and finally in human brains and why generic cross-serial structures are more difficult than centre-embedding ones even in an analogue setting. We have to leave open the question of a potential bearing on linguistic issues such as why centre-embedded syntactic structures are much more common than cross-serial ones (Savitch *et al.*, 1987), even though cross-serial dependencies seem to be easier to generalise to further embeddings than centre-embeddings (Bach *et al.*, 1986, Christiansen and Chater, 1999b).

Analogue computation theory and complexity. First of all, this work helps clarify the view on what an evolved dynamical system representation of a language is as opposed to the hand-coded representations that for example can show that SRNs are Turing-equivalent (Siegelmann and Sontag, 1995). Those hand-coded solutions are fairly inspired by the well-known symbolic ones and thus do not make proper use of the advantages and biases of analogue hardware (see the discussion of this in Visser *et al.*, 2004).

Clarifying the concept of network or dynamical system based analogue computing will contribute to developing methods that assess computational complexity from an analogue point of view. There do exist various concepts on

both how to assess the complexity of a symbolic automaton, its associated language class and even individual symbolic strings, for example the Chomsky hierarchy (Chomsky, 1956) that classifies formal languages according to what type of rewrite rules and what type of recognition automaton they require; computational complexity theory that takes into account resource limits (Hopcroft and Ullmann, 1979); and finally there is Kolmogoroff or algorithmic complexity that formalises the concept of a shortest description of a set of strings (Li and Vitanyi, 1997). In their current formulations all these approaches are based on the symbolic conception of computing.

However analogue hardware has quite different biases from symbolic hardware and we have seen that the greater complexity of ww compared to ww^r can be explained with arguments rooted in the dynamics, and it is not obvious how this dynamic explanation relates to the symbolic concepts. What we envision therefore is to supplement these with a dynamic computation perspective. To our knowledge the following approaches in this direction have been undertaken:

Moore (1998) introduced formal language classes depending on what function types are used to piece together a dynamical system and examined their relation to the standard classes from the Chomsky hierarchy and computational complexity.

Tabor (2000) addressed the question of similarity of formal languages. In a symbolic setting the step from a regular to a context-free language, or from a context-free to a context-sensitive language is a discrete and discontinuous one and requires a radical change of the hardware,[12] whereas in SRNs often only a small change of a weight is needed to alter the processing type. Thus Tabor (2000) suggests to take the Euclidian distance between weight matrices that implement different languages as a measure for their similarity. This measure cuts across the Chomsky hierarchy as there are non-context-free (and even non-computable) languages arbitrarily close in weight space to context-free languages.

However this very interesting approach has some problems: There exist

[12] At least in the standard conception of symbolic automata theory. However, a PDA can also be seen as an infinite state automaton where there is a certain structure imposed on its states. As such it can be approximated by a sequence of FSAs of increasing size under certain conditions, similar to how a countable state Markov shift can be approximated by finite shifts (Crutchfield and Young, 1990, Crutchfield, 1993, Kitchens, 1998).

dynamical systems with very different weight matrices or even with different dimensionalities that process the same language. Is there a canonic one for a given language? Or—similar to Kolmogoroff complexity—a minimal one? When are two dynamical systems equivalent from the perspective of language processing?

Perhaps we can find answers to some of these open questions when instead of taking properties of the functions the dynamical systems are built of, or the similarity of weight matrices of a certain dynamical representation, we take properties that are more intrinsic to the dynamics, such as the types and numbers of fixed and periodic points and other invariant sets needed to process a language. For example the language $a^n b^n$ needs an attractive fixed point for F_a and a matching repelling one for F_b so that each combination $F_b^n \circ F_a^n \equiv id$ (Bodén and Blair, 2003). In the case of ww^r and ww discussed in this paper the line $y = 0$ is an attractor for the joined dynamics of F_a and F_b and each point $(x, 0)$ on this line specifies an infinite input sequence of as and bs, namely the one corresponding to the binary expansion of x.

Proceeding thus, a classification of formal languages similar to Chomsky hierarchy might be feasible using concepts from dynamical system theory. Similarity of two languages could for example be related to their dynamics' invariant sets. It would be interesting to find out how such a classification relates to the standard Chomsky hierarchy and its more versed progeny.

Acknowledgements

The author was supported by a research grant of the Max-Planck-Society and by the EC Research Training Network Project 'Basic Mechanisms of learning and forgetting'. I enjoyed discussions on the subject matter of this work with Jürgen Jost, Thomas Wennekers and Henrik Jacobsson. I am also thankful to three anonymous referees whose comments helped to improve the article substantially.

Appendix: Training algorithm

Preliminary simulation runs with full BPTT where the error is injected at every time step but the weights are only updated at end-of-string and subsequently the activation of the HUs reset did not work as well as BPTT(n) where the error is injected only for the current output in every time step and where subsequently the weights updated are with the network unfolded for the past n time steps. This procedure showed better performance but took a considerably longer time.

We sped up BPTT(n) with the following modifications: 1. the error injection and weight update procedure is only started randomly on average at about every nth time step; then 2. the output error is calculated and injected for the last n time steps with the network unfolded for the past n time steps. Thus ignoring the moving target problem, on average the error equals the one in standard BPTT. These modifications sped up the training considerably while the training results were equal to the standard BPTT(n).

We use the minimal Euclidean distance between output and target as the error function. Output values that differ less than 0.2 from their target values are considered as correct and no error is back-propagated for them. This avoids overlearning of instances where the output tends to agree with the target but can only reach the precise target values by oversaturating, and driving some weights far off from zero.

References

Bach, E., Brown, C. and Marslen-Wilson, W., 1986, Crossed and nested dependencies in German and Dutch: a psycholinguistic study. *Language and Cognitive Processes*, *1*, 249–262.

Bates, E. A. and Elman, J. L., 1993, Connectionism and the study of change, in *Brain Development and Cognition: A reader*, edited by M. Johnson (Oxford: Blackwell), pp. 623–642.

Bengio, Y., Simard, P. and Frasconi, P., 1994, Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157–166.

Bodén, M. and Blair, A., 2003, Learning the dynamics of embedded clauses. *Applied Intelligence*, *19*, 51–63.

Bodén, M., Jacobsson, H. and Ziemke, T., 2000, Evolving context-free language predictors, in *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO), pp. 1033–1040.

Bodén, M. and Wiles, J., 2000, Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, *12*, 197–210.

Bodén, M., Wiles, J., Tonkes, B. and Blair, A., 1999, Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units, in *Proceedings of the Nineth International Conference on Artificial Neural Networks (ICANN 99)*, edited by D. Willshaw and A. Murray, pp. 359–364.

Casey, M., 1996, The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, *8*, 1135–1178.

Cherubini, A., Citrini, C., Reghizzi, S. C. and Mandrioli, D., 1991, QRT FIFO automata, breadth-first grammars and their relations. *Theoretical Computer Science*, *85*, 171–203.

Chomsky, N., 1956, Three models for the description of language. *IRE Trans. on Information Theory*, *2*, 113–124.

Christiansen, M. H. and Chater, N., 1994, Generalization and connectionist language learning. *Mind & Language*, *9*, 273–287.

Christiansen, M. H. and Chater, N., 1999a, Connectionist natural language processing: The state of the art. *Cognitive Science*, *28*, 417–437.

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

Christiansen, M. H. and Devlin, J. T., 1997, Recursive inconsistencies are hard to learn: A connectionist perspective on universal word order correlations, in *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, edited by N. J. Mahwah (Lawrence Erlbaum Associates), pp. 113–118.

Cleeremans, A., Servan-Schreiber, D. and McClelland, J. L., 1989, Finite state automata and simple recurrent networks. *Neural Computation*, *1*, 372–381.

Crutchfield, J. and Young, K., 1990, Computation at the onset of chaos, in *Complexity, Entropy, and the physics of Information, SFI Studies in the Sciences of Complexity*, edited by W. Zurek (Addison-Wesley), pp. 223–269.

Crutchfield, J. P., 1991, Reconstructing language hierarchies, in *Information Dynamics*, edited by H. A. Atmanspracher and H. Scheingraber (New York: Plenum Press), pp. 45–60.

Crutchfield, J. P., 1993, The calculi of emergence: Computation, dynamics, and induction, in *Proceedings of the Oji International Seminar: Complex Systems – from Complex Dynamics to Artificial Reality* (Numazu, Japan). Special Issue of Physika D (1994), 11–54.

Ellefson, M. R. and Christiansen, M. H., 2000, Subjacency constraints without universal grammar: Evidence from artificial language learning and connectionist modeling, in *The Proceedings of the 22^{nd} Annual Conference of the Cognitive Science Society*, pp. 645–650.

Ellis, R. and Humphreys, G., 1999, *Connectionist Psychology* (Hove, East Sussex: Psychology Press).

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

Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D. and Plunkett, K., 1996, *Rethinking Innateness: A Connectionist Perspective on Development* (Cambridge, Mass.: MIT Press).

van Gelder, T., 1998, The dynamical hypothesis in cognitive science. *Behavioral and Brain Sciences*, *21*, 615–665.

van Gelder, T. J., 1999, Defending the dynamical hypothesis, in *Dynamics, Synergetics, Autonomous Agents: Nonlinear Systems Approaches to Cognitive Psychology and Cognitive Science*, edited by W. Tschacher and J.-P. Dauwalder (Singapore: World Scientific), pp. 13–28.

Giles, C., Miller, C., Chen, D., Chen, H., Sun, G. and Lee, Y., 1992, Learning and extracting finite state automata with second-order-recurrent neural networks. *Neural Computation*, *4*, 393–405.

Grüning, A., 2003, Why verb-initial languages are not frequent, MPI-MIS Preprint Series 10/2003, Max-Planck-Institute for Mathematics in the Sciences (MPI-MIS), Leipzig.

Hammer, B., 1997, On the generalization of Elman networks, in *International Conference on Artificial Neural Networks '97*, edited by W. Gerstner, A. Germond and M. Hasler (Berlin: Springer), pp. 404–414.

Hammer, B. and Tiňo, P., 2003, Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, *15*, 1897–1929.

Hochreiter, S. and Schmidhuber, J., 1997, Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Hopcroft, J. E. and Ullmann, J. D., 1979, *Introduction to Automata Theory, Languages, and Computation* (Mass.: Addison-Wesley).

Jacobsson, H., 2005, Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, *17*, 1223–1263.

Jacobsson, H. and Ziemke, T., 2003, Improving procedures for evaluation of connectionist context-free language predictors. *IEEE Transactions on Neural Networks*, *14*, 963–966.

Jaeger, H., 1996, Dynamische Systeme in der Kognitionswissenschaft. *Kognitionswissenschaft*, *5*, 151–174.

Kitchens, B. P., 1998, *Symbolic Dynamics* (Berlin: Springer).

Kolen, J. F. and Kremer, S. C. (eds.), 2001, *A Field Guide to Dynamical Recurrent Networks* (New York: IEEE Press).

Kremer, S. C., 2001, Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, *13*, 249–306.

Lawrence, S., Giles, C. L. and Fong, S., 1998, Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, *12*, 126–140.

Li, M. and Vitanyi, P., 1997, *An Introduction to Kolmogorov Complexity and Its Applications* (Berlin: Springer).

Lupyan, G. and Christiansen, M. H., 2002, Case, word order and language learnability: Insights from connectionist modeling, in *Proceedings of the 24^{th} Annual Conference of the Cognitive Science Society*.

Maass, W. and Orponen, P., 1998, On the effect of analog noise in discrete-time analog computations. *Neural Computation*, *10*, 1071–1095.

McCullough, W. S. and Pitts, E., 1943, A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*, 115–133.

Meduna, A., 2000, *Automata and Languages. Theory and Applications* (Berlin: Springer).

Monaghan, P., Gonitzke, M. and Chater, N., 2003, Two wrongs make a right: Learnibility and word order consistency, in *Proceedings of the 25^{th} Annual Conference of the Cognitive Science Society*.

Moore, C., 1998, Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, *201*, 99–136.

Omlin, C., 2001, Understanding and explaining DRN behaviour, in Kolen and Kremer (2001), chapter 8, pp. 207–228.

Pollack, J. B., 1991, The induction of dynamical recognizers. *Machine Learning*, *7*, 227–252.

Rodriguez, P., 2001, Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*, 2093–2118.

Rodriguez, P. and Wiles, J., 1998, Recurrent neural networks can learn to implement symbol-sensitive counting, in *Advances in Neural Information Processing Systems*, edited by M. Jordan, M. Kearns and S. Solla, volume 10 (Cambridge, Mass.: MIT Press), pp. 87–93.

Rodriguez, P., Wiles, J. and Elman, J. L., 1999, A recurrent neural network that learns to count. *Connection Science*, *11*, 5–40.

Savitch, W. J., 1992, Why it might pay to assume that languages are infinite, Newsletter 6(4), Center for Research in Language, UCSD.

Savitch, W. J., Bach, E. and Marsh, W. (eds.), 1987, *The Formal Complexity of Natural Language* (Dordrecht: D. Reidel Publishing Company).

Schmidhuber, J., Gers, F. and Eck, D., 2002, Learning nonregular languages: A comparison of simple recurrent networks and lstm. *Neural Computation*, *14*, 2039–2041.

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

Tabor, W., 2000, Fractal encoding of context free grammars in connectionist networks. *Expert Systems*, *17*, 41–56.

Tabor, W., 2002, The value of symbolic computation. *Ecological Psychology*, *14*, 21–51.

Tiňo, P. and Hammer, B., 2003, Architectural bias in recurrent neural networks – fractal analysis. *Neural Computation*, *15*, 1931–1957.

Tiňo, P., Čerňanský, M. and Ľubica Beňušková, 2004, Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, *15*, 6–15.

Tonkes, B., Blair, A. and Wiles, J., 1998, A paradox of neural encoders and decoders or why don't we talk backwards, in *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL98)*, edited by B. McKay, X. Yao, C. S. Newton, J.-H. Kim and T. Furuhashi (Berlin. New York: Springer), pp. 357–364.

Visser, I., 2002, *Rules and Associations*, Ph.D. thesis, Afdeling Psychologie, Faculteit Maatschappij- en Gedragswetenschappen, Universiteit van Amsterdam.

Visser, I., Raijmakers, M. E. J. and Molenaar, P. C. M., 2004, On the computational capabilities of symbolic and subsymbolic models. *Submitted*. See also chapter 2 in Visser (2002).

Wiles, J., Blair, A. D. and Bodén, M., 2001, Representation beyond finite states: Alternatives to push-down automata, in Kolen and Kremer (2001), pp. 129–142.

Wiles, J. and Elman, J., 1995, Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks, in *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (Cambrigde, Mass.: MIT Press), pp. 482–487.

Williams, R. J. and Peng, J., 1990, An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, *2*, 490–501.

Zeng, Z., Goodman, R. M. and Smyth, P., 1993, Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, *5*, 976–990.

List of Figures

abbbba, *bbaabbaabb*

$$\epsilon \xrightarrow[\epsilon]{a} A \xrightarrow[\epsilon]{b} BA \xrightarrow[\epsilon]{b} BBA \xrightarrow[b]{\P} BA \xrightarrow[b]{\P} A \xrightarrow[a]{\P} \epsilon \xrightarrow[E]{\P} \epsilon$$

(a) Two example strings.

(b) Example symbolic stack contents during processing of *abbbba* in a transducer task.

Figure 1: The language ww^r. A symbolic stack would keep track of each input symbol a and b writing e. g. A and B into its memory. The memory contents can therefore be represented as a string of As and Bs where new symbols are added or deleted at its left end. ϵ denotes the empty string. Symbols over the $\rightarrow$ are input symbols, symbols under it are output symbols.

$$abbabb,\ bbaabbbaab \quad \epsilon \underset{\epsilon}{\overset{a}{\rightarrow}} A \underset{\epsilon}{\overset{b}{\rightarrow}} AB \underset{\epsilon}{\overset{b}{\rightarrow}} ABB \underset{a}{\overset{\P}{\rightarrow}} BB \underset{b}{\overset{\P}{\rightarrow}} B \underset{b}{\overset{\P}{\rightarrow}} \epsilon \underset{E}{\overset{\P}{\rightarrow}} \epsilon$$

(a) Two example strings of ww.

(b) Example symbolic queue contents during processing of *abbabb*.

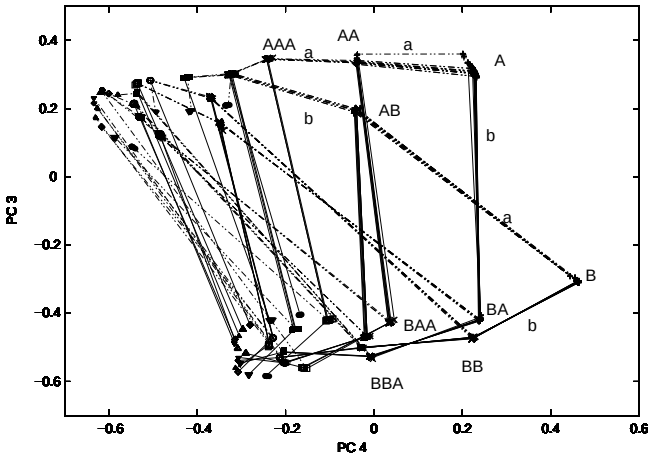
Figure 2: The cross-serial duplication language ww. A legend as in figure 1 applies. In a transducer task the symbolic queue keeps track of a and b inputs by writing A and B into its memory. The left end of a string is the reading end of the queue, the right one the writing end.

input:	$b\ a\ b\ b\ \P\P\P\P\P$
target for ww^r:	$----bbabE$
output for ww:	$----babbE$

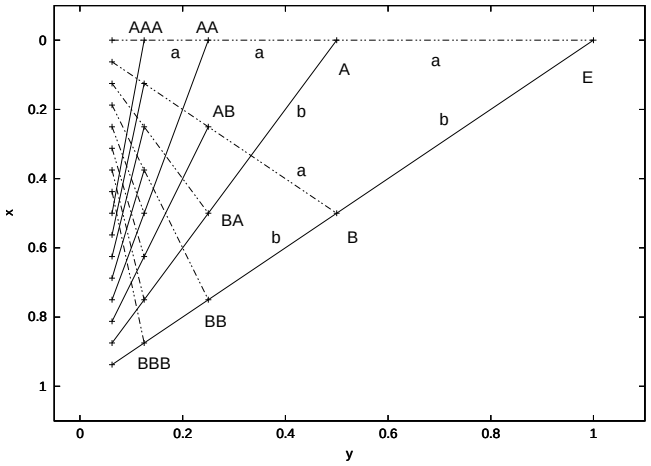
Figure 3: Example input and target pairs for ww^r and ww. '$-$' indicates that the output is irrelevant.

```
initialise weights randomly from [−1, 1).
initialise η = 0.1.
while η ≥ 0.001 do
  for e_max times do
    train net on training corpus
    test net on test corpus
    if net is better than previous best then
      save weights
    end if
  end for
  η ↦ η/2
  restore best weights
end while
```

Figure 4: Training scheme. Test performance is evaluated counting the correctly processed strings.

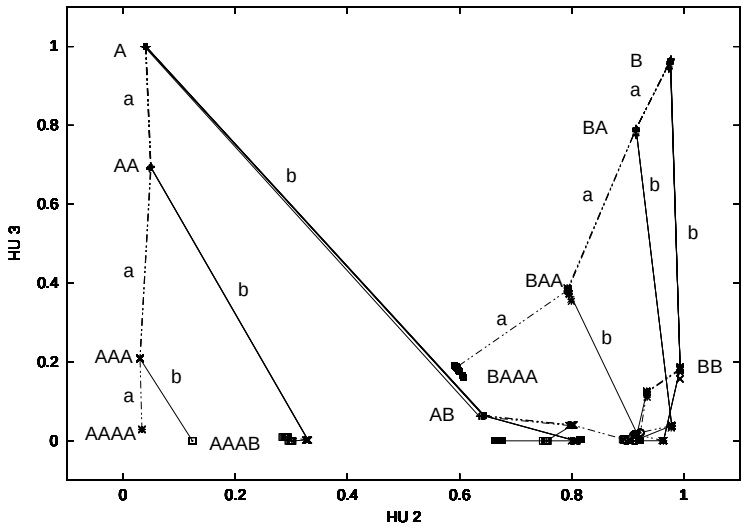


(a) Actual Dynamics for DUP(5), i.e. $ww^r, |w| \leq 5$.

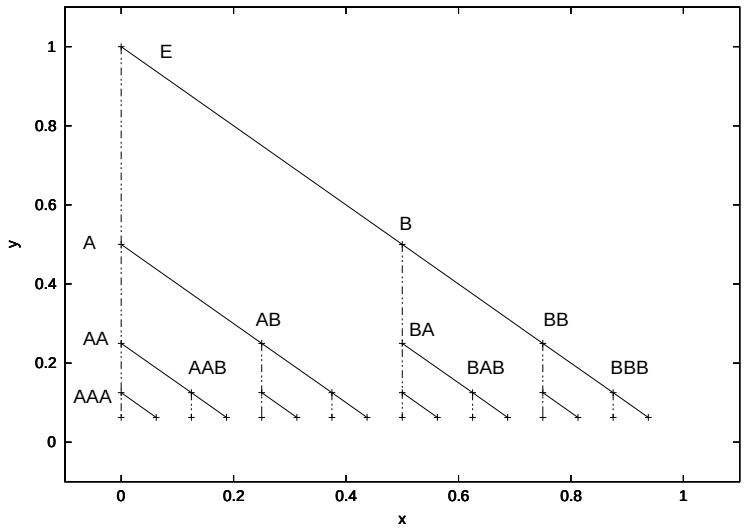


(b) Essential Dynamics for $ww^r, |w| \leq 4$.

Figure 5: Actual and essential dynamics for the centre-embedding language $ww^r, |w| \leq 5$. For the actual dynamics, principal component (PC) analysis helped find the most interesting 2D projection. The labels given for some of the clusters denote the equivalent context of a symbolic stack as in figure 1. The rather dispersed cluster E for empty memory is left out for the sake of clarity. Trajectories start in cluster A and B and run left from there. Dash-dotted lines mark trajectories between the clusters caused by an input a, and solid ones correspond to input b. Roughly, the horizontal axis encodes the number of inputs, and the vertical the contents of the memory: clusters with most recent input symbol a lie in the upper half. For the essential dynamics, imagine the empty stack cluster E smeared out and its solid b trajectory bend to the bottom to get a close fit with the actual dynamics.



(a) Actual Dynamics for DUP(4), i. e. $ww, |w| \leq 4$



(b) Essential Dynamics for $ww, |w| \leq 4$.

Figure 6: Actual and essential dynamics of the cross-serial language $ww, |w| \leq 4$. The same legend as in figure 5 applies and, for the actual dynamics, the cluster for empty memory has again been left out. Cluster labels correspond to the string representing the contents of a symbolic queue as in figure 2. Trajectories start in cluster A or B and essentially run downwards from there. The vertical axis is interpreted as queue depth, the horizontal one as encoding queue contents: clusters with first input symbol a lie in the left half. The branch for first input symbol b suffers from non-linear distortion due to saturation. Otherwise it looks like a twisted version of the essential dynamics.

init:	$x = 0$	$y = 1$
F_a:	$x \mapsto (x + \mathbf{0})/2$	$y \mapsto y/2$
F_b:	$x \mapsto (x + \mathbf{1})/2$	$y \mapsto y/2$
$F_¶$:	$x \mapsto 2x \mod 1$	$y \mapsto 2y$

(a) Dynamics for ww^r, requiring a stack-like memory

init:	$x := 0$	$y = 1$
F_a:	$x \mapsto x + \mathbf{0}$	$y \mapsto y/2$
F_b:	$x \mapsto x + \mathbf{y}/2$	$y \mapsto y/2$
$F_¶$:	$x \mapsto 2x \mod 1$	$y \mapsto 2y$

(b) Dynamics for ww, requiring a queue-like memory

Figure 7: The essential dynamical systems in 2D. After applying the corresponding mappings F_a and F_b for a sequence of inputs a and b, the binary expansion of x responds to the reversed input sequence of as and bs in (a) and to the sequence itself in (b).

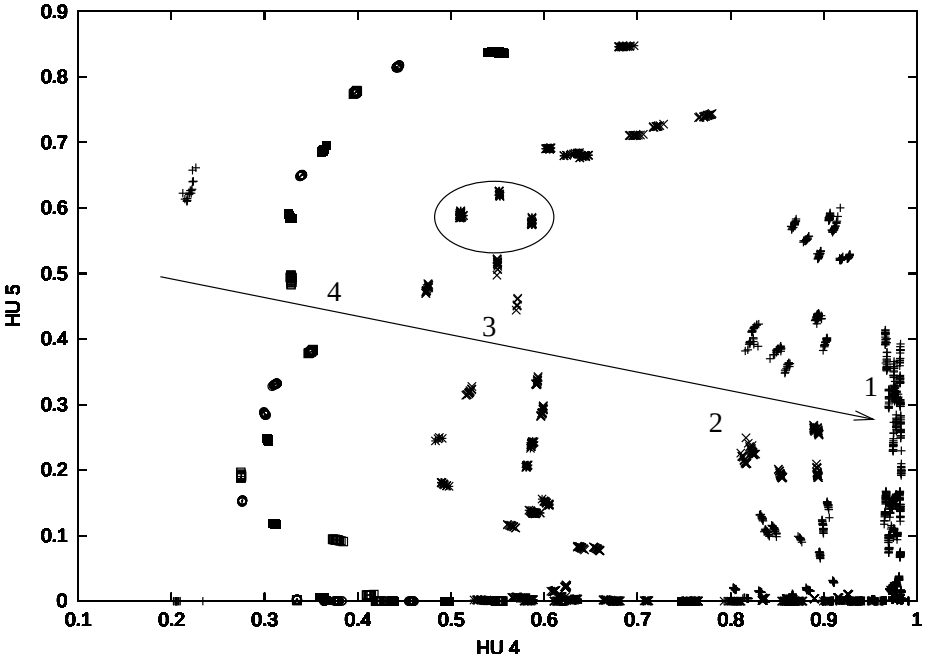


Figure 8: The same network as in 5(a), but in reproduction phase during ¶ input for strings with $|w| \leq 4$. Clusters that would output a upon next input ¶ (i. e. that correspond to a stack memory with top symbol A in the sense of figure 1) are accommodated along HU5 and HU4. The clusters squeezed at the bottom for HU5 $= 0$ have next output symbol b (i. e. B top on stack) and can be seen to be spread out along HU2 (not shown here) in a very similar fashion as the a clusters are along HU5. We ringed one of the threefold split clusters in to highlight the splitting of clustered discussed in the main text. Trajectories are not displayed since they would frequently originate and end at the bottom and thus rather obscure the structure than clarify it. The arrow indicates the direction in which the number of symbols in memory is counted down, numbers next to it indicate the number of symbols that are encoded with this geometrical position along the arrow. The arrow furthermore divides the clusters in two classes; above it one finds all clusters with next-to-topmost symbol B and below with next-to-topmost symbol A (all of them having topmost symbol A).

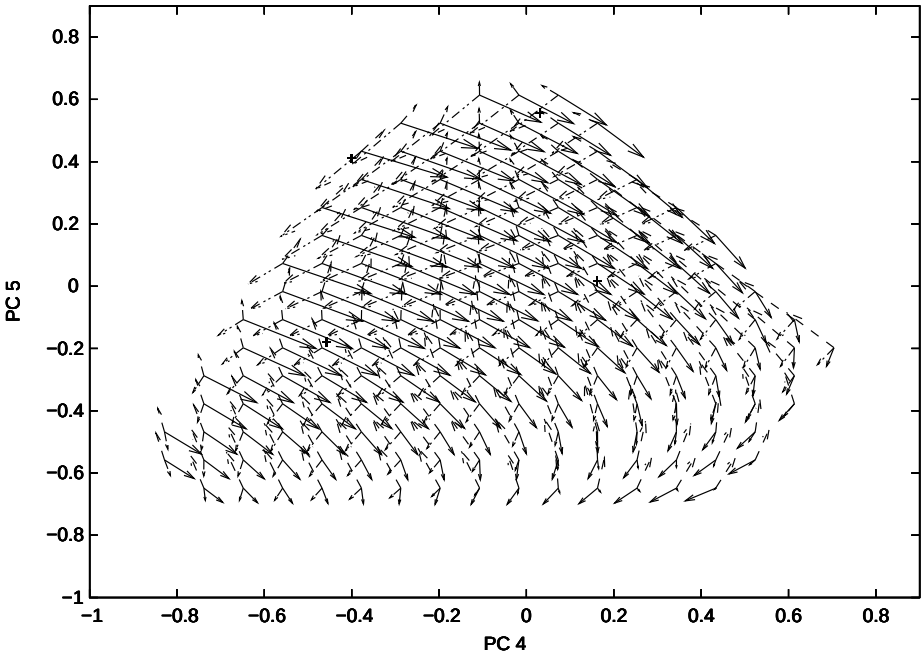


Figure 9: *ww*: The flow diagram for the same network as in figure 6(a) in the 2D plane spanned by the two most significant PCs. Dotted arrows correspond to the mapping F_a corresponding to input a, solid ones to F_b and, finally, dashed ones to $F_\P$. The network dynamics was applied to grid vectors that in the original set of coordinates of HUs lay within the admissible range $(0, 1)$ of HU activations and only these grid vectors were kept leading to the irregular outline of the 'grid'. In addition, difference vectors at the outline of the flow plot can point outward for the combined effect of projection and confinement to the original domain of definition of the dynamics.