

# Context-free and context-sensitive dynamics in recurrent neural networks

Mikael Bodén and Janet Wiles  
Department of Computer Science and Electrical Engineering  
University of Queensland, QLD 4072, Australia

January 25, 2001

**Running head:** Context-free and context-sensitive dynamics

**Keywords:** Context-free grammar, Context-sensitive grammar, Dynamical system, Language, Learning, Recurrent neural network, Recursive structure

**Statement:** This paper has not been submitted elsewhere in identical or similar form neither before nor after its submission to Connection Science.

**Author for correspondence and proofing:** Mikael Bodén, Tel. +61 (0)7 3365 8302, Fax. +61 (0)7 3365 4999, Email mikael@csee.uq.edu.au

## Abstract

Continuous-valued recurrent neural networks can learn mechanisms for processing context-free languages. The dynamics of such networks is usually based on damped oscillation around fixed points in state space and requires that the dynamical components are arranged in certain ways. It is shown that qualitatively similar dynamics with similar constraints hold for  $a^n b^n c^n$ , a context-sensitive language. The additional difficulty with  $a^n b^n c^n$ , compared with the context-free language  $a^n b^n$ , consists of “counting up” and “counting down” letters simultaneously. The network solution is to oscillate in two principal dimensions, one for counting up and one for counting down. This study focuses on the dynamics employed by the Sequential Cascaded Network, in contrast with the Simple Recurrent Network, and the use of Backpropagation Through Time. Found solutions generalize well beyond training data, however, learning is not reliable. The contribution of this study lies in demonstrating how the dynamics in recurrent neural networks that process context-free languages can also be employed in processing some context-sensitive languages (traditionally thought of as requiring additional computation resources). This continuity of mechanism between language classes contributes to our understanding of neural networks in modeling language learning and processing.

# 1 Introduction

It is well-known that recurrent neural networks (RNNs) can emulate finite state automata and thus perform operations such as recognition on the class of regular languages (Cleeremans et al., 1989; Pollack, 1991; Giles et al., 1992; Casey, 1996). It has been argued that for an RNN to correspond to a finite state automaton, it must organize its state-space to create attractors corresponding to the distinct states of the finite state automaton (Casey, 1996).

It is also well known that a large class of RNNs are based on analog components and are not limited to finite state computation (Siegelmann, 1999; Moore, 1998; Blair and Pollack, 1997; Kolen, 1994).

The power of a recurrent neural network to process languages from a given class needs to be distinguished from its ability to learn those languages. In learning, the issue is not restricted to the states that the recurrent network can represent, but rather, also requires that the network generalizes to additional strings from the language. For example, a Simple Recurrent Network (SRN; Elman, 1990) can, in the limit, exhibit infinitely many states. The SRN has been shown to use qualitatively different dynamics from the finite state automata emulations, which allow it to generalize when learning context-free languages (Wiles and Elman, 1995; Rodriguez and Wiles, 1998; Rodriguez et al., 1999; Bodén et al., 1999).

There is a direct correspondence between the types of grammars in formal language theory and models of computing machines (Hopcroft and Ullman, 1979). The relation between formal grammars and learnable dynamics in recurrent neural networks, however, is not obvious. This paper sets out to establish that the qualitative dynamics utilized for context-free languages is also applicable to at least some context-sensitive languages – a step that is traditionally thought of as requiring additional computation resources.

RNNs have been handcrafted, sometimes using external stacks, to handle context-free (Das et al., 1993; Zeng et al., 1994; Hölldobler et al., 1997; Tabor, 2000) and context-sensitive languages (Steijvers and Grünwald, 1996). We are, however, concerned with learned behaviors. Manually designed systems based on their classical counterparts make an artificial distinction between the processing requirements of context-free and context-sensitive languages. Recurrent neural networks that learn can be used to study how the network handles the transition between context-free and context-sensitive languages.

We will show that a second order network, the Sequential Cascaded Network (SCN; described in Pollack, 1991; Giles et al., 1992), trained by Backpropagation Through Time (BPTT), can learn to predict a (finite) subset of the context-sensitive language  $a^n b^n c^n$  with good generalization capabilities. The results are comparable to those of Chalup and Blair (1999) who evolved (using a hill-climbing algorithm) an SRN to predict a subset of  $a^n b^n c^n$  but did not establish if the solution generalized beyond the maximum string length in the training data. The difficulty by which a solution was found led us to initially explore the SCN. Moreover, we provide a detailed analysis based on dynamical systems theory describing how the network solves the problem. The relation between the

employed dynamics and the dynamics for predicting the context-free language  $a^n b^n$  is discussed in detail.

At the outset it is important to note that when assessing the network as having learned a context-free or context-sensitive language (both of which are infinite), we are in practice imposing a limit on the level of embedding used for training and testing – rendering both training and test sets as regular. The analysis suggests, however, that the network essentially processes the presented strings as if they were drawn from an infinite language.

## 2 The network

The SCN (Pollack, 1991; Giles et al., 1992) operates by propagating the outer product between the current input pattern and the previous state, together with the input and state patterns themselves, through a set of weights ( $\mathbf{W}$ ) to the output layer. The same product and patterns are propagated through a different set of weights ( $\mathbf{V}$ ) to the state layer – producing the state to be used in the next time step.

The output of unit  $i$ , at time  $t$ , of the SCN is defined as

$$y_i^t = f\left(\sum_{j,k} W_{ijk} z_j^{t-1} x_k^t + \sum_j W_{ij\theta} z_j^{t-1} + \sum_k W_{i\theta k} x_k^t + W_{i\theta\theta}\right) \quad (1)$$

$$z_i^t = f\left(\sum_{j,k} V_{ijk} z_j^{t-1} x_k^t + \sum_j V_{ij\theta} z_j^{t-1} + \sum_k V_{i\theta k} x_k^t + V_{i\theta\theta}\right) \quad (2)$$

where  $\mathbf{x}$  is the externally set input vector and  $\mathbf{z}$  is a vector of internal *state* units (see Figure 1). The index  $i$  is used for identifying individual output and state units,  $j$  is used for the state of the previous time step, and  $k$  is used for the current input. Biases are introduced as additional elements in the weight matrices (indexed with  $\theta$ ).<sup>1</sup> Thus, as an example of notation,  $W_{ijk}$  is a second order weight (used for connecting the product, between the  $j$ th state unit and the  $k$ th input, with the  $i$ th output unit),  $W_{ij\theta}$  and  $W_{i\theta k}$  are first order weights (feeding the  $i$ th output with either the  $j$ th state unit or the  $k$ th input), and  $W_{i\theta\theta}$  is a first order bias (associated with the  $i$ th output unit). The logistic output function,  $f(net) = 1/(1 + e^{-net})$ , is used.

[Figure 1 about here.]

The network is trained using BPTT. Errors for  $\mathbf{y}^t$  are propagated through  $\mathbf{W}$  to  $\mathbf{z}^{t-1}$  and, subsequently, to  $\mathbf{z}^{t-2}$  through  $\mathbf{V}$ . If propagation is limited to one step, the learning regime corresponds to what Pollack (1991) calls the “backspace trick.” Further unfolding of layers, however, enables errors to be

<sup>1</sup>The addition of biases on the outer product has its origin in Pollack’s (1991) formulation of the SCN which distinguishes between a function and a context network, each with separate sets of output biases.

propagated iteratively through  $\mathbf{V}$  for  $\mathbf{z}^{t-3}$ , ...,  $\mathbf{z}^{t-h}$  for any  $h$ . The unfolding process may introduce learning instability since penalties can conflict (as noted by Pollack, 1991) but unfolding through time was shown to be critical for learning context-free dynamics in SRNs (Bodén et al., 1999).

### 3 Experiments

To study the dynamics of networks that process a context-sensitive language, networks were trained on a finite subset of  $a^n b^n c^n$ .

The task for the network is to predict the next letter in strings from  $a^n b^n c^n$  where  $1 \leq n \leq 10$ . That is, in strings with the same number of  $as$ ,  $bs$  and  $cs$ , the number of  $bs$  and  $cs$  should be reliably predicted. Strings are presented consecutively in random order with a distribution skewed towards shorter strings.<sup>2</sup> Thus, when  $a$  is presented to the network, it cannot deterministically predict the first  $b$ . The network does not initially know which  $n$  it is dealing with. It is not until the first  $b$  has appeared at the input that the network can be expected to successfully predict the remaining letters in the current string and the first letter of the following string (see Figure 2). We use a local encoding where  $a$  is represented by  $[1 \ 0 \ 0]$ ,  $b$  by  $[0 \ 1 \ 0]$  and  $c$  by  $[0 \ 0 \ 1]$ . The criterion for a successfully predicted string is that whenever  $b$  or  $c$  is presented at the input (as part of a string from the language) the output pattern must uniquely identify the correct next letter (by having exactly one unit activation over 0.5 in the position as noted above) as defined by the language.

A *solution* is a network which correctly predicts all strings  $1 \leq n \leq 10$ .<sup>3</sup> <sup>4</sup> A *generalizing* network predicts at least all strings  $1 \leq n \leq 11$ .

[Figure 2 about here.]

We varied the learning rate,  $\eta$ , between 0.1 and 0.5 and the momentum,  $\alpha$ , between 0.0 and 0.9. The network was unfolded during BPTT training for  $h = 10, 15$  or 20 time steps. Two hundred networks with different initial weights (Gaussian distribution with a 0.1 variance around zero) and two state units were trained for each configuration.

Reliability was measured as the number of networks that converged on a solution within the presentation of 10,000 strings. In preliminary trials, reliability was low in all cases. The best reliability, 15/200 solutions, was achieved

<sup>2</sup>The strings were drawn from the distribution  $P(n = 1) = 0.2, P(n = 2) = 0.3, P(n = 3) = 0.17, P(n = 4) = 0.1, P(n = 5) = 0.07, P(n = 6) = \dots, P(n=10) = 0.03$ . The network weights were updated after each fully presented string.

<sup>3</sup>Our definition of a solution is not tailored to exclude networks which are able to predict related supersets of the language (e.g. balanced parenthesis language, see Section 4.2). This is not to be confused with a *recognizer* that overgeneralizes to less constrained languages (e.g.  $a^*b^*c^*$ ). The purpose of the definition is rather to qualify network behaviors which at least predict strings from  $a^n b^n c^n$ .

<sup>4</sup>During testing all strings were presented in descending order starting with a large  $n$ , ending with  $n = 1$ . The state units were never reset.

when  $\eta = 0.5$ ,  $\alpha = 0.0$  and  $h = 20$ . In a related study, it was established that bifurcations of dynamics in an SRN (trained on the context-free language  $a^n b^n$ ) cause gradient based learning to be unstable close to where solutions are found (Bodén et al., 1999). Similar patterns of instability were noted for the SCN. The scope of the current study is limited to the successful networks.

To determine the potential of generalization, each solution (saved at the point it first predicted all strings in the training set correctly) was subject to further training (up to another 10,000 strings) but with a lower learning rate ( $\eta = 0.01$ ,  $\alpha = 0.0$ ) to traverse the error landscape more cautiously. One network was observed to generalize to all strings up to  $n = 18$ . The average generalization ability for 20 successful networks (including those that only managed up to  $n = 10$ ) was  $n = 12.1$ .

## 4 Results and Analysis

Discrete time recurrent neural networks can usefully be characterized as discrete time dynamical systems (Casey, 1996; Tino et al., 1998; Moore, 1998; Rodriguez et al., 1999) with inputs, outputs, and state variables.

For a linear autonomous dynamical system  $\mathbf{s}^t = \mathbf{F}\mathbf{s}^{t-1}$  where  $\mathbf{s} \in \mathbb{R}^d$ , an *eigenvalue* of  $\mathbf{F}$  is a scalar  $\lambda$  and an *eigenvector* of  $\mathbf{F}$  is a vector  $\mathbf{v} \neq \mathbf{0}$  such that  $\mathbf{F}\mathbf{v} = \lambda\mathbf{v}$ . The eigenvalue gives the rate of contraction or expansion of  $\mathbf{F}$  in the direction given by the corresponding eigenvector.

For  $a^n b^n c^n$  there are three possible inputs, allowing us to distinguish between three *autonomous* dynamical systems:  $\mathbf{F}_a$ ,  $\mathbf{F}_b$  and  $\mathbf{F}_c$  corresponding to the inputs  $a$ ,  $b$  and  $c$ , respectively.

A *fixed point* is a point in state space,  $\bar{\mathbf{s}}$ , for which  $\bar{\mathbf{s}} = \mathbf{F}\bar{\mathbf{s}}$  and can be determined by solving the roots of the system of equations (for which  $\bar{\mathbf{s}} = \mathbf{F}\bar{\mathbf{s}}$  is satisfied) for each system. When studied in the vicinity of a fixed point of the system, the Jacobian (the partial derivative matrix) of the state units has eigenvalues and eigenvectors that express how the non-linear system changes over time around that point (Devaney, 1989; Tino et al., 1998; Rodriguez et al., 1999). The non-linearity is introduced by the output function.

When the eigenvalue is positive, 1-periodic (monotonic) behavior occurs in the direction of the associated eigenvector, and when the eigenvalue is negative 2-periodic (oscillating) behavior occurs. When the absolute eigenvalue is below 1 the fixed point is an attractor in the direction of the eigenvector, and when the absolute eigenvalue is above 1 it is a repeller (Tino et al., 1998; Rodriguez et al., 1999). If a fixed point is attracting in one direction and repelling in the other, it is also called a saddle point. In some cases the eigenvalue is a complex value, which indicates a rotation around the fixed point (Hirsch and Smale, 1974; Devaney, 1989). In the following, state spaces are 2-dimensional ( $z_1$  and  $z_2$ ), which implies that each fixed point is associated with two eigenvalues and their corresponding eigenvectors.

## 4.1 Network Dynamics

All the successfully generalizing networks used similar dynamics. For the sake of clarity, we first focus on one representative solution. The state trajectory (the trace of points visited in state space over time) when  $a^8b^8c^8$  is presented to the network is shown together with decision boundaries implemented by the output units, fixed points and eigenvectors in Figure 3. The decision boundaries are determined by regarding the output function as imposing a virtual threshold on states,  $f_H(net) = 0$  if  $net < 0$ , 1 otherwise.

[Figure 3 about here.]

When the network enters  $\mathbf{F}_a$ , states quickly align with the horizontal eigenvector. The fixed point of  $\mathbf{F}_a$  is attracting in both directions but with a double period along the horizontal eigenvector as indicated by the eigenvalue -0.73. When the state is horizontally aligned this behavior takes control and the system starts oscillating *to* the fixed point.

A similar effect is noted for  $\mathbf{F}_c$ . States not aligned with the vertical eigenvector are strongly attracted to the fixed point (eigenvalue 0.05), then, as the eigenvalue associated with the vertical eigenvector indicates (-1.34), the system oscillates *away* from the saddle point. The oscillation continues until  $\mathbf{F}_a$  is entered again. The shift is predicted by the network when the output decision boundaries for the first and third output units are crossed one time step ahead of the actual transition (see Equation 1).<sup>5</sup>

So far the solution is very similar to those found for the context-free language  $a^n b^n$  in previous work. The final state of the first system,  $\mathbf{F}_a$ , uniquely identifies the current count and affects the starting state for processing the remaining letters, as exemplified by  $\mathbf{F}_c$ . The critical behavior for  $a^n b^n c^n$ , however, is found in  $\mathbf{F}_b$ . Here, the final state of  $\mathbf{F}_a$  (which tells us how many  $a$  have been presented) is transferred (over multiple time steps) to  $\mathbf{F}_c$ . It is helpful to look at each eigenvalue (and associated eigenvector) in turn. Along the horizontal eigenvector, states oscillate away from the fixed point as the eigenvalue -1.42 indicates.  $\mathbf{F}_b$  continues doing so until  $\mathbf{F}_c$  takes over. While the  $\mathbf{F}_b$  state oscillates along the horizontal eigenvector, it crosses the decision boundaries implemented by the output units to correctly predict the first  $c$ . The vertical eigenvector (with eigenvalue -0.76) entertains simultaneous oscillation towards the fixed point. The vertical oscillation has the effect that the state gradually aligns with the horizontal eigenvector. Along the vertical eigenvector, it looks like the state simply oscillates towards the fixed point, but, together with the horizontal behavior we observe that the network employs a two-pronged counting strategy forming a star-shaped trajectory around the fixed point of  $\mathbf{F}_b$ .

---

<sup>5</sup>The current output is determined by the previous state. For  $a^8b^8c^8$  the boundaries separate the 6th from the 7th  $b$  and the 6th from the 7th  $c$  to predict the first  $c$  and the first  $a$ , respectively.

## 4.2 Context-free vs. context-sensitive dynamics

A computational model for a context-free language requires mechanisms similar to those of a stack or a counter. A push-down automaton (PDA) is basically a finite state automaton equipped with a stack. A PDA is able to recognize all context-free languages (Hopcroft and Ullman, 1979). Importantly, a context-sensitive language cannot be modelled by a PDA alone and represents, for the computational linguist and theoretical computer scientist, a qualitative step in terms of computational power. A linear bounded automaton (LBA) is basically a Turing Machine with limited tape access and is able to recognize all context-sensitive languages (Hopcroft and Ullman, 1979).

Experimentation with RNNs on the simple context-free language  $a^n b^n$  reveals that the behaviors employed by the SCN are intimately related to those we found with  $a^n b^n c^n$ . A typical state trajectory and the eigenvectors for an SCN predicting  $a^n b^n$  are shown in Figure 4.

[Figure 4 about here.]

Rodriguez et al. (1999) analysed four different SRN weight sets generated by BPTT on training data from  $a^n b^n$  and identified conditions for the SRN to represent and predict  $a^n b^n$  correctly to a level beyond its training data. First, the largest absolute eigenvalue of each system should be inversely proportional to one another (meaning in practice one being around -0.7 and the other around -1.4). The inverse proportionality ensures that the rate of contraction around the fixed point of  $\mathbf{F}_a$  matches the rate of expansion around the saddle point of  $\mathbf{F}_b$  (Rodriguez et al., 1999, p. 23).

The second criterion is that the fixed point of  $\mathbf{F}_a$  should lie on the line determined by the eigenvector corresponding to the smallest absolute eigenvalue of  $\mathbf{F}_b$  (the direction in which the saddle point attracts) when the eigenvector is projected through the  $\mathbf{F}_b$  fixed point. This basically means that the first thing that happens in  $\mathbf{F}_b$  is a long-distance state shift in the direction of its eigenvector to a part in state space close to the fixed point of  $\mathbf{F}_b$ . The direction of the eigenvector ensures that the final state of  $\mathbf{F}_a$  (which identifies the  $a$ -count) correctly affects the starting point for the expansion in  $\mathbf{F}_b$ . The small eigenvalue ensures the direct transition.

There is also a third informal criterion concerned with the state prior to  $\mathbf{F}_a$ . The state trajectory needs to reset the position after processing a string (of any  $n$ ) so that  $\mathbf{F}_a$  can correctly start processing a new string. Without giving a clear condition for the additional criterion, Rodriguez et al. (1999) supplied a stronger example when it is met. It was noted that, in the context of the balanced parenthesis language,<sup>6</sup> by having the fixed point of  $\mathbf{F}_b$  lie on the eigenvector for the smaller (absolute) eigenvalue of  $\mathbf{F}_a$ , both systems can return to the correct state at any time.

---

<sup>6</sup>The balanced parenthesis language is a superset of  $a^n b^n$  allowing  $a$  to re-appear after the first  $b$  but still requiring the total number of each letter to match.

The rates of expansion and contraction match in a similar way in the technically distinct, second order SCN predicting  $a^n b^n c^n$ . The eigenvalues for each system-shift are approximately inversely proportional:  $-0.73 \cdot -1.42 = 1.04$  and  $-0.76 \cdot -1.34 = 1.02$ . The similarities with the oscillating solution for  $a^n b^n$  are substantial. The various  $\mathbf{F}_a$  for  $a^n b^n$  and  $a^n b^n c^n$  make use of the same dynamics. For  $\mathbf{F}_b$  involved in predicting  $a^n b^n c^n$ , both eigenvectors cross the fixed points of  $\mathbf{F}_a$  and  $\mathbf{F}_c$ , of which the first enables a correct transfer of the counting offset and the second opens up the possibility of returning to  $\mathbf{F}_b$  from  $\mathbf{F}_c$  at any time for counting up more  $b$ s (similar to the balanced parenthesis language). Moreover, the eigenvector of  $\mathbf{F}_c$  crosses the  $\mathbf{F}_b$  fixed point and ensures that the starting point of  $\mathbf{F}_c$  is correctly set with respect to the final state of  $\mathbf{F}_b$ .

To verify the formal criteria identified by Rodriguez et al. (1999) on a larger set of trained networks and with a different recurrent architecture, we performed a large number of additional simulations with the SCN and the SRN on  $a^n b^n$ . Learning rate, momentum and level of unfolding were varied. Each network was equipped with two state units. Inputs  $a$  and  $b$  were presented as  $[1 \ 0]$  and  $[0 \ 1]$ , respectively.<sup>7</sup> We observed that, in general, there are two basic ways SRNs and SCNs implement the counting necessary for  $a^n b^n$  and one exclusive to SCNs: By damped oscillation (as described above), by monotonic state-changes and by entangled spiralling.<sup>8</sup> The former type is found in a majority of the networks so our comparison focuses on SCNs that implement damped oscillation.

From a variety of trials, 60 SCNs that successfully predicted  $a^n b^n$  ( $1 \leq n \leq 10$ ) using an oscillating solution were selected and compared with 20 SCNs that successfully predicted  $a^n b^n c^n$  ( $1 \leq n \leq 10$ ).<sup>9</sup> We restrict our investigation to the eigenvalues found along the oscillating direction for all systems: two values for the  $a^n b^n$  networks (referred to as  $\kappa_a$  and  $\kappa_b$ , corresponding to the eigenvectors along the oscillating direction in  $\mathbf{F}_a$  and  $\mathbf{F}_b$ , respectively) and four values for the  $a^n b^n c^n$  networks (referred to as  $\lambda_a$ ,  $\lambda_{ba}$ ,  $\lambda_{bc}$  and  $\lambda_c$ , corresponding to the eigenvectors along the oscillating direction of  $\mathbf{F}_a$ , the expanding oscillating direction of  $\mathbf{F}_b$ , the contracting oscillating direction of  $\mathbf{F}_b$ , and the expanding oscillating direction of  $\mathbf{F}_c$ , respectively).

Of the eigenvalues found in solutions for  $a^n b^n$ ,  $\kappa_a$  is approximately inversely proportional to  $\kappa_b$  (see Table I). For  $a^n b^n c^n$ ,  $\lambda_a$  is also approximately inversely proportional to  $\lambda_{ba}$ , and  $\lambda_{bc}$  is approximately inversely proportional to  $\lambda_c$ . Notably,  $\lambda_{bc}$  is sufficiently small (still attracting) to play a similar role to the smaller (absolute) eigenvalue of  $\mathbf{F}_b$  in the  $a^n b^n$  networks (usually close to zero, ensuring the long-distance state transition). All standard deviations are small and verify that all networks conform to the same type.

<sup>7</sup>All other simulation details (including the distribution of  $n$ ) were identical to those for  $a^n b^n c^n$ .

<sup>8</sup>In networks employing entangled spiralling, when  $a$  is presented, states are slowly attracted towards a fixed point by rotating around it. During the presentation of  $b$ , states are rotating away (in the opposite direction) from a nearby, repelling, fixed point. The eigenvalues of such fixed points are complex.

<sup>9</sup>Networks were trained until they first discovered a solution. Fine-tuned networks were not included. Networks which solved  $a^n b^n$  by entangled spiralling and monotonic state-changes were excluded.

[Table I about here.]

The distances between the lines determined by the eigenvectors and the fixed points according to the second criterion are listed in Table II. The distance of interest for  $a^n b^n$  networks is  $\delta_{ba}$ , the distance between the line for the eigenvector for the smaller (absolute) eigenvalue of  $\mathbf{F}_b$  and the fixed point of  $\mathbf{F}_a$ . For  $a^n b^n c^n$  networks, let  $\epsilon_{ba}$  be the distance between the line for the eigenvector for the contracting eigenvalue of  $\mathbf{F}_b$  and the fixed point of  $\mathbf{F}_a$ , and let  $\epsilon_{cb}$  be the distance between the line for the eigenvector for the smaller (absolute) eigenvalue of  $\mathbf{F}_c$  and the fixed point of  $\mathbf{F}_b$ . The averages of  $\delta_{ba}$ ,  $\epsilon_{ba}$ , and  $\epsilon_{ca}$  are close to zero, with small standard deviations, indicating that all the fixed points lie on the corresponding lines for the eigenvectors. Hence, correct transitions between the systems are ensured.

[Table II about here.]

The results demonstrate that the criteria identified by Rodriguez et al. (1999) hold not only for a considerably larger set of networks and a different network type but also for a language from a different language class altogether.

The latter result is significant for at least two reasons. First, the classical view within formal language theory maintains that context-sensitive languages cannot be processed using the computation resources minimally sufficient for context-free languages. Traditionally, this discrepancy is commonly used as the defining reason for distinguishing between the language classes (Hopcroft and Ullman, 1979).

Second, the computational power of RNNs is, in theory, equivalent to that of Turing Machines (Siegelmann, 1999) implying that RNNs are capable of processing all recursively enumerable languages. Our experiment is a demonstration of how context-sensitive computation can be implemented and exhibit generalization abilities within a *learning* framework using the same intrinsic learning biases as for learning a context-free language.

### 4.3 SRN vs. SCN

Is the result on  $a^n b^n c^n$  only supported by a particular network? It turns out that it is difficult to find solutions for  $a^n b^n c^n$  with the SRN. We performed four hundred additional simulations (with two configurations:  $\eta = 0.5$ ,  $\alpha = 0.0$  and  $h = 20$ , and  $\eta = 0.3$ ,  $\alpha = 0.5$  and  $h = 20$ ) with an SRN, equipped with three fully recurrent hidden units, and BPTT. No solutions were found for either configuration within the presentation of 10,000 strings.

However, Chalup and Blair (1999) showed that, by using a specialized hill-climbing algorithm utilizing weight mutation and incremental learning, it is possible to find SRNs correctly predicting the full training set (in this case  $1 \leq n \leq 12$ ) with three recurrent hidden units. Generalization performance beyond  $n = 12$  was not addressed – hence, the demonstration was inconclusive

regarding the potential of capturing the infinite nature of the language. The dynamics employed in their SRN is similar to that observed here (Alan Blair, personal communication) but due to the higher dimensionality of the state space the similarities are difficult to assess visually. To that end, we performed some additional simulations to investigate the characteristics of SCNs when three state units were used. When  $\eta = 0.5$ ,  $\alpha = 0.0$ ,  $h = 20$  and  $1 \leq n \leq 10$ , 5/200 solutions were found. The solutions were subject to further training with a smaller learning rate ( $\eta = 0.01$ ) and the average generalization was  $n = 11.8$ .

The increased dimensionality of the state space allows for additional fixed points and associated eigenvalues and eigenvectors. Nevertheless, all except one of the solutions complied with the studied criteria: On average  $\lambda_a \lambda_{ba} = 0.97$  ( $SD = 0.13$ ), and  $\lambda_{bc} \lambda_c = 1.09$  ( $SD = 0.07$ ). The other solution attempted to use a hybrid solution:  $\mathbf{F}_a$  used a 2-periodic attractor as in previous solutions ( $\lambda_a = 0.63$ ), but both  $\mathbf{F}_b$  and  $\mathbf{F}_c$  were partly utilizing 1-periodic attractors (as indicated by multiple small, mainly positive, eigenvalues). The hybrid network never generalized beyond the training data ( $n = 10$ ).

One obvious advantage with the SCN compared with the SRN is that the SCN, in effect, employs one set of output decision boundaries in state space for each input pattern. The output activation is determined by the previous state through the multiplicative connection. Thus, any influence the previous state has on the output is dependent on the current input (see Jacobsson, 1999, for a mathematical account). The input dependence makes the search for possible dynamics less constrained and enables the SCN to find solutions not allowed by the SRN. The output decision boundaries for the SRN are independent of the input activation and consequently effective at all times. Another possible disadvantage for the SRN compared with the SCN is that, equipped with the same number of units, the SRN has a smaller number of weights.

#### 4.4 Finite vs. infinite languages

A serious point can be raised regarding the assessment of the network as correctly processing the infinite languages  $a^n b^n$  and  $a^n b^n c^n$  based solely on tests with the finite (and regular) languages  $a^n b^n$  and  $a^n b^n c^n$  where  $1 \leq n \leq m$  and  $m$  is any finite positive integer.

It is indeed possible to construct a (large) finite state automaton which correctly processes the regular subsets of the two languages. Such automata would not naturally lend themselves to generalization beyond the training set. Increasing  $m$  would require new states and associated transformations to be added. If the recursive nature of the languages can be discovered by the learner, a stack or a counter (equipped with an infinite memory) remedy this problem.

A distinction can be made between language *competence* and language *performance*. Importantly, the network has been observed as being able to go beyond the training set. The number of strings correctly processed is, however, not infinite. As with human linguistic performance, the network performance is

not perfect and degrades with higher  $m$  (Christiansen and Chater, 1999).<sup>10</sup>

As for competence, the means by which the processing is implemented are in essence capable of infinite levels of embedding. Theoretically it has been shown that RNNs can process all context-free, context-sensitive and even recursively enumerable languages (Siegelmann, 1999). The proofs are based on stacks implemented using simple neural components as fractal encoders and decoders in continuous space. Specifically, Cantor sets have been used to encode level of embedding by dividing the state space into smaller and smaller regions. Since infinite numerical precision is assumed, fractal encoding and decoding can be performed error-free (Siegelmann, 1999; see also Tabor, 2000; Moore, 1998; Hölldobler et al., 1997).<sup>11</sup> The dynamics obtained by networks trained to predict  $a^n b^n$  and  $a^n b^n c^n$  is similar to fractal encoding/decoding. When the network is counting up, one principal dimension of the state space is divided into smaller and smaller intervals of possible values, and when the network is counting down, state spaces expand accordingly. Hence, the technique that is utilized by the trained network has the competence of processing infinitely deep strings but performance is inhibited by noise and lack of precision of dynamical components.

## 5 Conclusions

This paper has shown that the simple context-sensitive language  $a^n b^n c^n$  can be learned (in the sense that generalization is beyond the training data) with BPTT using an SCN with two state units. On average, networks generalized from training data of strings for  $1 \leq n \leq 10$  to  $n = 12.1$ . Training is, however, not reliable. The best networks needed to be unfolded through multiple time steps. The best configuration resulted in a solution in about 8% of the runs.

The SCN exploits the analog nature of internal representations and divides the state space in smaller and smaller regions counting up letters. When counting down, each step is proportionally larger. The additional difficulty with  $a^n b^n c^n$  compared with the context-free language  $a^n b^n$  consists of counting up and counting down simultaneously. The network solves this by oscillating in two principal dimensions, one for counting up and one for counting down.

The solutions for predicting  $a^n b^n c^n$  can be characterized in terms of eigenvalues and eigenvectors of the three dynamical systems within the network. According to the eigenvalue analysis all successful networks found with BPTT made use of similar dynamics.

We have shown that the type of dynamics found for  $a^n b^n$  and the balanced parenthesis language extends naturally to a language of a different complexity class and also to a different network type, that of the second order SCN. We also verified that similar dynamics occur in 3-dimensional state spaces, as originally

---

<sup>10</sup>Christiansen and Chater also showed differences in learnability of languages from the same language class, differing in types of recursion (e.g. counting recursion and center-embedding).

<sup>11</sup>The assumption that state spaces can be accessed with infinite precision can be compared to the assumption in classical computation theory of access to infinite memory.

suggested by Chalup and Blair (1999).

Gold (1967) proved that no superfinite language (context-free languages and beyond) is learnable without negative examples. The most popular solution, proposed by Gold, is to endow the learner with restrictive biases. The SRN, trained by prediction, has intrinsic learning biases which allow it to pick up some aspects of context-free languages and generalize beyond training data (Elman, 1993; Rohde and Plaut, 1999; Rodriguez et al., 1999). The results herein demonstrate that the SCN exhibits similar learning biases for the prediction task, enabling learning of a simple context-free language. Importantly, the same minimal learning biases are sufficient for learning (and generalizing to) a context-sensitive language.

It is likely that none of the classes of classical models of computation corresponds to RNNs trained using gradient descent. That is, RNNs may prove incapable of learning the whole class of context-free languages, yet be capable of capturing some context-sensitive languages. Similarly, it is likely that the class of human languages cuts across the classical hierarchy constrained by learning issues.

## Acknowledgements

This work was supported by an ARC grant to JW. The authors thank Stephan Chalup, Henrik Jacobsson, Paul Rodriguez, Tom Ziemke and four anonymous reviewers for useful comments on a draft of this paper.

## References

- Blair, A. D. and Pollack, J. B. (1997). Analysis of dynamical recognizers. *Neural Computation*, 9(5):1127–1142.
- 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 International Conference on Artificial Neural Networks*, pages 359–364, Edinburgh. IEE.
- Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135–1178.
- Chalup, S. and Blair, A. D. (1999). Hill climbing in recurrent neural networks for learning the  $a^n b^n c^n$  language. In *Proceedings of the 6th International Conference on Neural Information Processing*, pages 508–513, Perth.
- Christiansen, M. and Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157–205.

- Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381.
- Das, S., Giles, C. L., and Sun, G.-Z. (1993). Using prior knowledge in a NNPD to learn context-free languages. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems*, volume 5, pages 65–72. Morgan Kaufmann, San Mateo, CA.
- Devaney, R. L. (1989). *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.
- Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99.
- Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., and Lee, Y. C. (1992). Learning and extracted finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.
- Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 16:447–474.
- Hirsch, M. W. and Smale, S. (1974). *Differential Equations, Dynamical Systems, and Linear Algebra*. Academic Press, New York.
- Hölldobler, S., Kalinke, Y., , and Lehmann, H. (1997). Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of KI-97: Advances in Artificial Intelligence*, pages 313–324. Springer Verlag.
- Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison–Wesley, Reading, MA.
- Jacobsson, H. (1999). A comparison of simple recurrent and sequential cascaded networks for formal language recognition. Master’s thesis, University of Skövde, Sweden.
- Kolen, J. F. (1994). Fool’s gold: Extracting finite state machines from recurrent network dynamics. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 501–508. Morgan Kaufmann Publishers, Inc.
- Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, 201:99–136.
- Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227.

- Rodriguez, P. and Wiles, J. (1998). Recurrent neural networks can learn to implement symbol-sensitive counting. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.
- Rodriguez, P., Wiles, J., and Elman, J. L. (1999). A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40.
- Rohde, D. L. T. and Plaut, D. C. (1999). Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72:67–109.
- Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser.
- Steijvers, M. and Grünwald, P. (1996). A recurrent network that performs a context-sensitive prediction task. Technical Report NC-TR-96-035, NeuroCOLT, Royal Holloway, University of London.
- Tabor, W. (2000). Fractal encoding of context free grammars in connectionist networks. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, 17(1):41–56.
- Tino, P., Horne, B. G., Giles, C. L., and Collingwood, P. C. (1998). Finite state machines and recurrent neural networks – automata and dynamical systems approaches. In Dayhoff, J. and Omidvar, O., editors, *Neural Networks and Pattern Recognition*, pages 171–220. Academic Press.
- Wiles, J. and Elman, J. L. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the Seventeenth Annual Meeting of the Cognitive Science Society*, pages 482–487. Lawrence Erlbaum.
- Zeng, Z., Goodman, R. M., and Smyth, P. (1994). Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330.

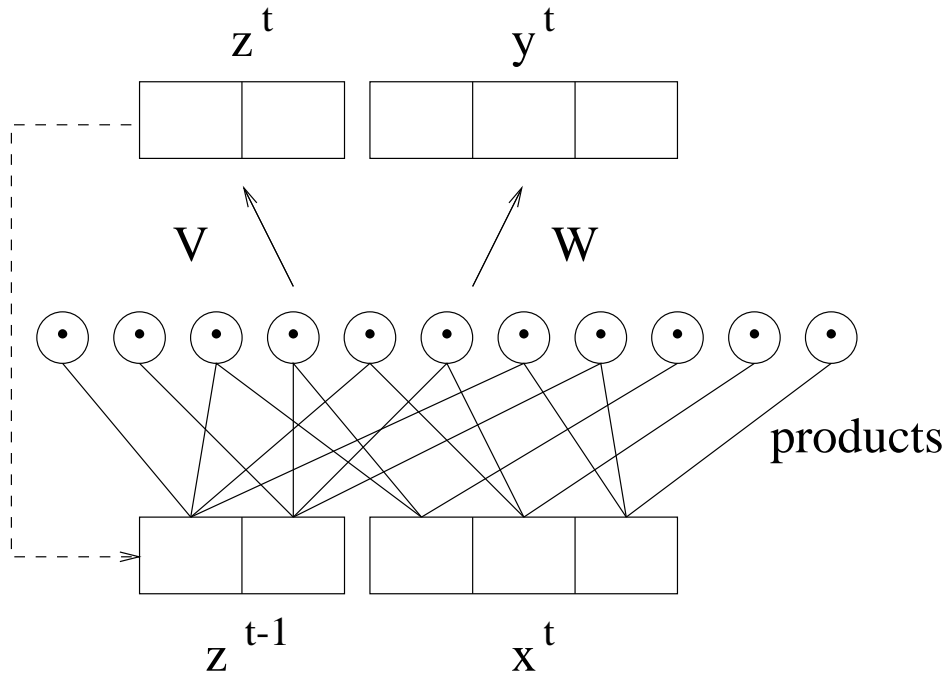


Figure 1: The Sequential Cascaded Network with 3 input units, 3 output units and 2 state units. The previous state vector,  $\mathbf{z}^{t-1}$ , is used as an additional input together with the current input vector,  $\mathbf{x}^t$ , as indicated by the dashed arrow. Each activation of the previous state is multiplied with each activation of the input as illustrated by the connected circles. The products are fed through the fully connected weight layer ( $\mathbf{W}$  and  $\mathbf{V}$ ) of the network. Biases are included in the weight matrices.

```

INPUT:  aaaabbbbcccccaabbccaaaaabbbbcccc
OUTPUT: aaabbbbcccccaabbccaaaaabbbbccccca
          *****  ****      *****

```

Figure 2: An example of the synchronization of inputs and the target outputs along a time axis. The sequence shown is  $a^4b^4c^4a^2b^2c^2a^5b^5c^5$ . The points in time when error checking is performed are marked with  $*$ .

	$\kappa_a$	$\kappa_b$	$\kappa_a\kappa_b$	$\lambda_a$	$\lambda_{ba}$	$\lambda_a\lambda_{ba}$	$\lambda_{bc}$	$\lambda_c$	$\lambda_{bc}\lambda_c$
Mean	-0.68	-1.42	0.96	-0.77	-1.38	1.06	-0.76	-1.33	1.02
SD	0.07	0.10	0.07	0.04	0.05	0.03	0.03	0.05	0.02

Table I: Eigenvalues and their products in the oscillating direction for the  $a^n b^n$  networks ( $\kappa_a$  to  $\kappa_a\kappa_b$ ) and  $a^n b^n c^n$  networks ( $\lambda_a$  to  $\lambda_{bc}\lambda_c$ ). If products are close to 1.0, convergence and divergence rates (expressed by the terms) are matched.

	$\delta_{ba}$	$\epsilon_{ba}$	$\epsilon_{cb}$
Mean	0.00	0.01	0.02
SD	0.00	0.01	0.01

Table II: Distances between fixed points and the lines for the eigenvectors.  $\delta_{ba}$  is for  $a^n b^n$  networks, and  $\epsilon_{ba}$  and  $\epsilon_{cb}$  are for  $a^n b^n c^n$  networks. Small distances indicate that the associated systems are matched and transitions (carrying counting information) are precise.

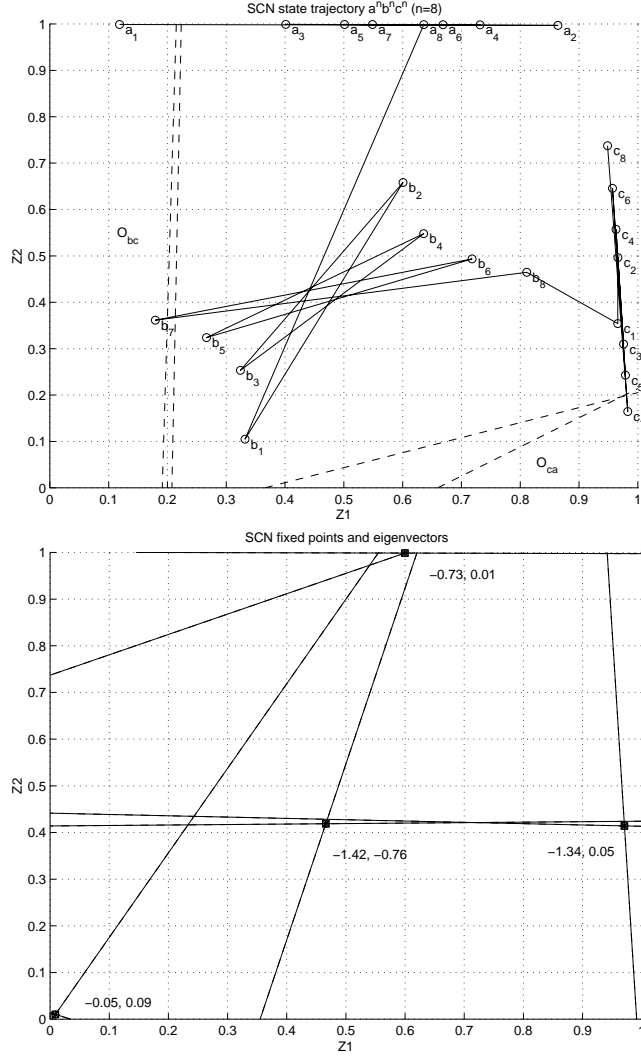


Figure 3: Top: The state trajectory for predicting  $a^8 b^8 c^8$  with the output decision boundaries projected into state space (dashed lines). The drawn  $O_{bc}$  decision, is based on the weights utilized as a result of the outer product involving  $[0 \ 1 \ 0]$  (the  $b$  input), and predicts the first  $c$ . The  $O_{ca}$  decision boundary, is based on the weights which are similarly made effective by the outer product involving  $[0 \ 0 \ 1]$  (the  $c$  input), and predicts the first  $a$  of the next string. Bottom: The fixed points, associated eigenvectors (plotted through the corresponding fixed point) and eigenvalues (printed near the fixed point) for each system in state space.

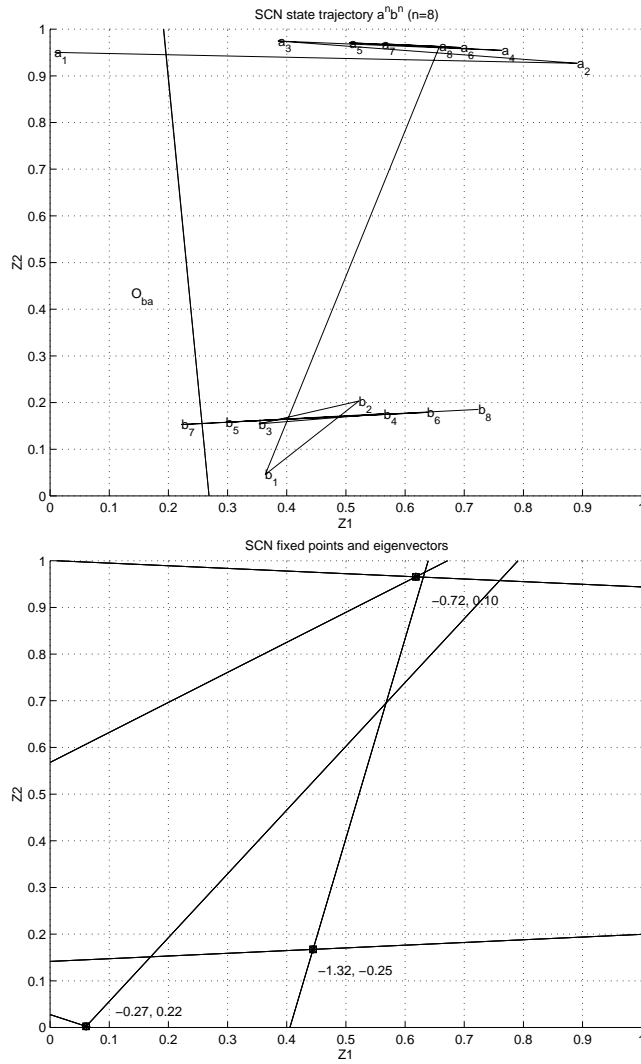


Figure 4: Top: The state trajectory for predicting  $a^8 b^8$  with the output decision boundary (made effective by the input  $[0 \ 1]$  through the outer product) projected into state space. Bottom: The fixed points and eigenvectors for each system in state space.