DEPARTMENT OF COMPUTER SCIENCE
AND SOFTWARE ENGINEERING,
THE UNIVERSITY OF NEWCASTLE

First Order Recurrent Neural Networks Learn To Predict A Mildly Context-Sensitive Language

Stephan K. Chalup
Department of Computer Science
and Software Engineering
The University of Newcastle
Callaghan NSW 2308, Australia
chalup@cs.newcastle.edu.au

Alan D. Blair
Department of Computer Science
and Software Engineering
The University of Melbourne
Parkville VIC 3010, Australia
blair@cs.mu.oz.au

Technical Report 2000-06

26 October 2000

ISBN: 0-7259-1109-3

First Order Recurrent Neural Networks Learn To Predict A Mildly Context-Sensitive Language

Stephan K. Chalup [1] & Alan D. Blair [2]

Abstract

This study shows experimentally how first order recurrent neural networks learn to predict the mildly context-sensitive language $\mathcal{MA}_3 = \{a^n b^n c^n;\ n \geq 1\}$. The training algorithm is evolutionary hill climbing with an incremental learning strategy. The experiments explore generalisation abilities of the networks emerging in the process of evolution. Generalisation with respect to depth n of the language strings and regarding their ordering in the training sequence is evaluated. Experiments without an incremental learning strategy indicate that the latter accelerates or facilitates training. Interestingly networks trained incrementally and non-incrementally show a qualitatively different pattern of hidden unit activity. Training on the context-free $a^n b^{2n}$ language and the mildly context-sensitive $a^n b^n c^n$ language is compared. The results show that even when the networks are identical and the data sets have the same size, learning of the context-free language is easier than learning of the mildly context-sensitive language.

1 Introduction

The aim to understand human language ability and thinking in the context of evolution, development of the central nervous system and learning processes during childhood is motivation for research at the intersection of several disciplines including psychology, linguistics, formal languages and automata, artificial intelligence, cognitive neuroscience, brain research and others.

Noam Chomsky drew a distinction between semantics and syntax of language and proposed a hierarchy of formal languages - the classical Chomsky hierarchy [16] - with the aim to investigate the syntax of human language. The hierarchy is strict and has four complexity classes: regular, context-free, context-sensitive and recursively enumerable languages (see Table 1). To each language class belongs a class of grammars and a class of automata. Regular grammars and the associated finite state automata (FSA) are at the most restricted level while the other three classes are more general, with unrestricted grammars and Turing machines at the most general level [32].

It has become widely accepted that natural language does not fit into the classical Chomsky hierarchy but has common features with context-free and context-sensitive languages. The *family of mildly context-sensitive languages* (MCSL) [3] is slightly more powerful than that of context-free languages but preserves their essential properties. MCSLs are closely related to tree-adjoint grammars (TAG) (see [33]) and purport to contain the most significant languages that appear in the study of natural languages [34]. The language of *(three) multiple agreements* $\mathcal{MA}_3 = \{a^n b^n c^n; n \geq 1\}$ is mildly context-sensitive and is in fact part of the definition of this language class, see Appendix A or [37].

The primary goal of this report is to show how first order recurrent neural networks can learn to predict sequences of strings from $\mathcal{MA}_3$ and to explore generalisation abilities of the trained neural networks. While first experimental results were announced in [14] we now provide a detailed exposition of our investigation. In a more general context this study might have established a missing link in a chain of arguments and findings which can help us to connect results obtained via artificial neural network modeling to biological and psycho-linguistic phenomena of natural language processing.

Before talking about learning issues it is good to know what types of languages artificial neural networks can theoretically process and whether models of such networks exist.

[1] Address: Department of Computer Science and Software Engineering, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, phone: +61 (0)2 4921 6080, fax +61 (0)2 4921 6929, email: chalup@cs.newcastle.edu.au
Funding: Parts of this research was conducted while the first author was supported by a PhD scholarship from the Faculty of Information Technology at Queensland University of Technology in Brisbane, Australia
[2] Address: Department of Computer Science and Software Engineering, The University of Melbourne, 221 Bouverie Street, Parkville VIC 3010, Australia, email: blair@cs.mu.oz.au
Funding: This research was partially supported by the ARC Small Grant S4005295.

[3] A definition of MCSLs was provided in [37] and is included in Appendix A

Theoretical language processing capability depends on the type of activation function used by the network, e.g. a simple recurrent network [22] with threshold units can represent any deterministic finite state automaton [36]. Assuming more general continuous activation functions the existence of an artificial neural network which can simulate a universal Turing machine can be proven [43, 52]. But if we demand robustness to noise neural networks are limited to regular languages [13, 39].

Different fixed network models have been constructed for predicting or recognizing regular [25, 53, 12], context-free [31, 23] and context-sensitive [54] languages.

However, the question of how far neural networks are able to *learn* language tasks of different complexity levels is still being explored.

On the complexity level of regular languages there are quite a number of studies which show how first and second order recurrent networks learn regular languages from examples or how finite state automata can be extracted from the trained networks, see e.g. [18, 42, 25, 56, 26].

Wiles and Elman [62] showed how simple recurrent networks can be trained by backpropagation to predict the context-free language $\mathcal{MA}_2 = \{a^n b^n; n \geq 1\}$. Following on from this several papers analysed hidden unit activity [46] and stability issues. It was repeatedly reported that $a^n b^n$ is a hard task and training is unstable. Evolutionary hill climbing was used in [57] hoping to overcome some of the instabilities encountered with backpropagation.

Pioneering experiments [14] on learning to predict the language $\mathcal{MA}_3$ employed evolutionary hill climbing in combination with data incremental learning. It was later shown that second order sequential cascaded networks could be successfully trained by backpropagation on a task related to $\mathcal{MA}_3$, although training of first order networks by backpropagation remained unsuccessful on this task, see [8].

Our current study further refines the incremental learning combined with evolutionary hill climbing approach and shows how to train first order recurrent networks to obtain networks which are able to predict sequences of strings from $\mathcal{MA}_3 = \{a^n b^n c^n; n \geq 1\}$ and to generalise to larger values of n and to different orderings of the strings in the data sequence. The development of each candidate network during the course of evolution is monitored and some of the emerging solutions are tested on their generalisation ability.

The main achievements of this study are a refined incremental learning scheme and network architecture as well as experimental results which provide answers to the questions:

- Can first order recurrent neural networks learn to predict sequences of strings from the language $\mathcal{MA}_3$ and can they generalise beyond the training data ?
- How efficient is data incremental learning when compared with non-incremental learning on the $\mathcal{MA}_2$ and $\mathcal{MA}_3$ tasks ?

Key information for answering these questions will later be summarised in the overview in Table 9 and results of a successfully trained SRN are displayed in Figure 3 and Table 12.

Additional achievements are comparative experiments on the $a^n b^{2n}$ language prediction task and observations obtained from a qualitative analysis of the dynamics of hidden unit activity in a movie-type animation of several thousand networks which were generated for this experiment.

The report is structured as follows: The next section provides some background information on recurrent neural networks. The prediction task is explained in section 3 and some definitions from formal language theory are provided in Appendix A. Incremental evolutionary hill climbing for recurrent networks is discussed in section 4. Its special adjustment to the context-sensitive language task called 'Data Juggling', including the incremental learning strategy and possible forms of generalisation, are the topic of section 4. Simulations, experiments and their evaluation are covered in section 5.

2 Recurrent Neural Networks

In recent years a large variety of different recurrent neural network architectures for supervised learning have been investigated, see e.g. [30], [29] or [60] and [59] for overviews of the discrete and the continuous case.

First order recurrent neural networks work similarly to feed-forward networks but have recurrent connections. Second order networks additionally have multiplicative connections. Many different second order architectures have been developed.

For language processing tasks with regular and context-free languages both first and second order networks have been successfully trained or hand crafted, see e.g. [25, 64, 62]. While the present study uses first order networks, two other recent studies have employed second order networks for processing context-sensitive languages: In [54], a second order network was hand-crafted for predicting the context-sensitive language $\{(ba^k)^n; k \geq 0, n > 0\}$ while in [8] sequential cascaded networks with two hidden units were trained on a context-sensitive language task.

In this study we have used the well-known *simple recurrent neural network* (SRN) architecture with three units in each layer and trained it using supervised learning on a one-step look-ahead prediction task, basically following the concept of Elman [22]. The SRN is displayed in Figure 1 with input units I1-I3, hidden units H1-H3, output units O1-O3 and state units S1-S3. Bent arrows are fixed one-to-one copy connections and the connections from the state units S1-S3 to the hidden units H1-H3 operate with a delay of one time step. The state units S1-S3 are therefore copies of H1-H3 and establish a memory for the network.

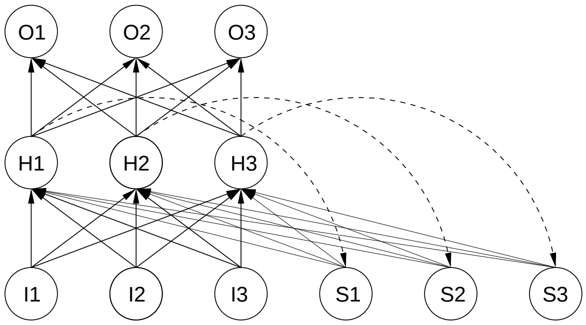


Figure 1: Simple recurrent network (SRN).

In addition to the SRN we used SRNs with additional shortcut connections (SRNSC), see Figure 2. The shortcuts [4] connect the three input units I1-I3 directly to the output units O1-O3. Consequently SRNSCs have more weights than SRNs. These two additional features turned out to be helpful in learning $\mathcal{MA}_3$.

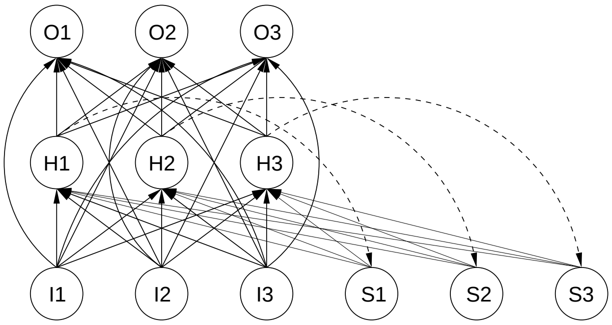


Figure 2: SRN with shortcut connections (SRNSC).

As activation function a hyperbolic tangent function was employed.

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{1}$$

[4] Shortcut connections had shown advantges for feed-forward networks when applied to classification tasks in [63, 15].

It has the same shape as the standard sigmoid function

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

but is stretched and translated so that it is point symmetric about the origin.

The network topology can be defined by a connectivity matrix, where the columns represent I1-I3 and H1-H3 and the rows represent H1-H3 and O1-O3.

$$\mathbf{C} = (c_{ij}) = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{3}$$

Matrix **C** determines the topology of a standard SRN. If the nine zero entries are replaced by '1' it determines a SRNSC.

Let $\delta : \mathbb{Z} \times \mathbb{Z} \longrightarrow \{1, 0\}$ be the Kronecker delta function, defined by

$$\delta(a,b) := \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{4}$$

then given a weight matrix $\mathbf{W} = (w_{ij})$ the neural network is determined by the equations:

$$\begin{aligned} y_{it} &= \sum_{j=0}^{U} \delta(1, c_{ij}) \cdot w_{ijt} \cdot x_{jt} + \sum_{j=1}^{U} \delta(2, c_{ij}) \cdot w_{ijt} \cdot x_{j(t-1)} && (5) \\ x_{it} &= \tanh(y_{it}), \quad i > IP && (6) \\ x_{0t} &= 1 && (7) \\ x_{it} &= input_{it}, \quad i = 1, ..., IP && (8) \end{aligned}$$

Depending on the type of training task recurrent neural networks have been interpreted in different ways:

1. *Interpretation as Mealy–Moore Machines:* Discrete time recurrent neural networks have been interpreted as neural state machines in analogy to a type of finite state machines called Mealy-Moore Machines , see [12] for more details and compare [42, 32].

2. *Interpretation as Dynamical Systems:* A recurrent neural network can be regarded as a dynamical system and while processing an input sequence each of its units' acitvation moves along a trajectory in the corresponding activation space. Several recent studies analysed trajectories of hidden unit activation while processing language tasks, see e.g. [46, 9, 10, 62, 35].

3 Prediction Task

One aspect of language processing is to predict a symbol sequence which is formed according to syntactical rules determined by a grammar.

A recurrent neural network can transform an input sequence of symbols to an output sequence using a stepwise prediction based on the input information. In the *one step look ahead prediction task* [22] symbols of the input sequence are presented to the network one by one and the network output is taken as prediction of the next symbol. Input and output sequences are supposed to be the same once the network has learned the task correctly. Hence a perfectly trained network could theoretically predict the whole sequence if the output is fed back to the input units. However, in some situations not all symbols of the sequence are uniquely determined; e.g. in case of the $a^n b^n c^n$ task, the first b of each string cannot be predicted.

The most well-known system of grammars is the *Classical Chomsky Hierarchy* – a strict hierarchy having four levels corresponding to four language complexity classes [16]. Table 1 displays the four classes starting with the most general level at the top. To each language class belongs a type of grammar and an associated automaton. The last column contains typical examples. Grammars provide the formal system for generating strings over an alphabet of a language using a set of rules. Automata can recognise strings of the corresponding language mechanically. Many other formal languages classes have been defined and not all of them fit into the classical hierarchy. MCSLs [5] are one of the newer additional classes and would be in between the two middle classes, the context-free (CF) and the context-sensitive (CS) languages.

Language	*Grammar*	*Automaton*	*Example*
Recursively enumerable	Unrestricted	Turing machine	Any computable function
Context-sensitive	Context-sensitive	Linear-bounded automaton	$a^n b^n c^n$
Context-free	Context-free	Non-deterministic pushdown automaton	$a^n b^n$
Regular	Regular	Finite-state acceptor	a^*

Table 1: The Classical Chomsky Hierarchy.

In aid of investigating learnability of formal languages closely related to natural language, i.e. context-free, context-sensitive and in particular mildly context-sensitive languages, we performed experiments using training sequences formed by strings of one of the following types, where always $n \geq 1$:

$$q_n = a^n b^n \tag{9}$$
$$r_n = a^n b^{2n} \tag{10}$$
$$s_n = a^n b^n c^n \tag{11}$$

In all cases the neural network is presented with a sequence of these strings for varying values of n.

Since our interest is focused on the language of three agreements $\mathcal{MA}_3 = \{s_n; n \geq 1\}$ we will from now on use strings $s_n = a^n b^n c^n$ from this language as examples in our explanations, which can be adjusted to the other languages straightforwardly.

The training sequence is made of strings of different depths and the symbols of each string are represented by vectors

$$\begin{aligned} a &= (+\mathbf{1}, -1, -1), \\ b &= (-1, +\mathbf{1}, -1), \\ c &= (-1, -1, +\mathbf{1}). \end{aligned}$$

Each symbol of the input sequence is fed one at a time into the three dimensional input layer (I1-I3) of the neural network (Figure 1 or 2). Each output unit (O1-O3) is assigned to one of the three symbols (a, b, c) and simply the unit with the highest activation determines the predicted symbol.

Since the depth n is not known to the network at the start of a string it cannot predict when the first b will occur. Once the network has processed the first b, it is required to predict $n-1$ additional b's, followed by n c's, followed by an indeterminate (but nonzero) number of a's (which form the beginning of the subsequent string).

Example. In the sequence of three strings

$$s_3 s_2 s_4 = aaa\mathbf{b}bbccc|aa\mathbf{b}bcc|aaaa\mathbf{b}bbbcccc$$

[5] A definition of MCSLs is included in Appendix A.

all 27 symbols except the three emphasised **b**'s have to be correctly predicted.

Remarks.

1. A variation which could be called the $a^1 *^n b^{n-1} c^n$ task requires the network to do the same prediction as in the $a^n b^n c^n$ task [6] with the only difference that it has free choice for all n symbols following the first a of each string. With this task only 18 instead of 24 symbols in the above example sequence are required to be correctly predicted:

$$s_3 s_2 s_4 = \mathbf{aaab}bbccc|a\mathbf{ab}bcc|a\mathbf{aaab}bbbcccc$$

   This variation of the $a^n b^n c^n$ task was employed in [8].

2. Another variation is due to different interpretations of the output unit activation used to determine which of the three symbols is predicted. In our studies simply the unit with the highest activation determines the symbol. However, other studies employed a more rigid scheme requiring that the unit which determines the correct symbol have an activation above zero while the activations of the other two units were required to be below zero.

4 Incremental Evolutionary Hill Climbing

Several variants of the standard feed-forward backpropagation algorithm [50] for supervised training of recurrent neural networks have been developed, see [21] for an overview. In contrast to feed-forward neural networks, the smoothness of the error surface during training is not well-controlled in the recurrent case. Successful training depends on finding a way around bifurcation problems, see [20]. As pointed out by [5] learning long-term dependencies with gradient descent methods is difficult.

Despite these difficulties, many recurrent neural networks have been trained by backpropagation methods. For example [62] and [46] employed backpropagation through time (BPTT) for training a simple recurrent network on the $a^n b^n$ task. They observed instabilities during training and declared $a^n b^n$ to be a hard task for SRNs.

It was suggested in [57] that evolutionary hill-climbing might overcome some of the instabilities observed using gradient descent on the $a^n b^n$ task. Comparative experiments between evolutionary and gradient descent training were conducted in [7].

In the current study SRNs and SRNSCs were trained using a special version of evolutionary hill climbing called *Data Juggling* which will be introduced later in section 4.3. It is based on the standard version of evolutionary hill climbing which is displayed in Table 2.

We call the current weight matrix $W_{champ} = (w_{ij})$ and the corresponding neural network the *champion*. A weight w_{ij} connects unit j to unit i. In the experiments of this project the initial weights were all generated from a N(0; 0.05) normal distribution. Given the champion W_{champ} the hill climber endeavors to find a 'better' weight matrix by generating a *mutant* matrix $W_{mutant} \leftarrow W_{champ} + \Delta W_{mutant}$ and comparing the champion and the mutant by evaluating the corresponding networks on the training set. The coefficients of the matrix ΔW_{mutant} are randomly generated from a normal distribution with a standard deviation which is selected, for each new matrix, from a N(0; 0.01) normal distribution. This two-step generation of the random weights was inspired by [2].

Training with evolutionary hill climbing is a form of epoch learning and requires a training set for updating neural network weights depending on the accumulated training error at the end of each epoch. Instead of using a fixed data set we generated new training sets during the evolutionary training process. The training sets were classified according to d_{Stage} which denotes the depth n of the longest string $a^n b^n c^n$ which occurs in the sequence of strings representing the training set. Hence each training set belonged to one of 18 difficulty classes for $d_{Stage} = 3, ..., 20$.

Another study which used incremental learning for training a recurrent network on FSAs is [25].

The purpose of employing an incremental learning scheme is to facilitate or enable learning and to guide the network to generalise beyond the training sets in two directions:

[6] To be precise, the $a^n b^n c^n$ task actually is a $a^n * b^{n-1} c^n$ task because the first b is not determined.

```
Evaluate network with initial weights W_champ
==> error_champ

WHILE((error_goal <= error_champ) and
    (counter <= nEpochs))

    stepsize <- 0.01 · Gaussian noise

    FOR all weights of the neural net
        ΔW_mutant <- stepsize · Gaussian noise
        W_mutant <- W_champ + ΔW_mutant
    end FOR

    Evaluate network given by W_mutant
    ==> error_mutant

    IF(error_mutant < error_champ)
        W_champ <- W_mutant
        error_champ <- error_mutant
    end IF

    counter <- counter + 1
end WHILE

RETURN error_champ
```

Table 2: Evolutionary Hill Climbing

1. Generalisation to sequences with different orderings of the strings s_n.
2. Generalisation to sequences which contain strings for larger values of n than contained in the sequences which were used for training. This form of generalisation splits into further subclasses as we will explain later in section 5.5.1.

Both types of generalisation together reflect the general structure of the language to be learnt.

The following paragraphs introduce an incremental learning scheme for epoch learning which instead of using a fixed data set modifies it during the evolutionary training process. First we describe the training data. Then we explain the evolutionary training procedure which employs a special fitness calculation and selection, a new version of evolutionary hill climbing and a method to evaluate the whole training process and all the networks which have been evolved.

4.1 Training Data

Let $1 \leq d_{Stage} \leq 20$ be an integer number and $v(d_{Stage}) \in \{1, ..., d_{Stage}\}^{30}$ be a sequence of length 30 such that d_{Stage} appears exactly once and the other 29 numbers are randomly selected from $\{1, ..., d_{Stage} - 1\}$ with a distribution biased towards the lower numbers and with each number guaranteed to occur at least once. A *training set of stage* d_{Stage} is a concatenation of 30 strings s_n and is uniquely determined by identifying the index set of the string sequence with $v(d_{Stage})$. Hence in each training sequence the string $s_{d_{Stage}}$ appears only once and all strings of lower depth are contained at least once.

4.2 Fitness Evaluation

The error calculation takes all symbols of the training sequence except the first **b** of each string into consideration (compare section 3).

At each stage d the mean squared error is calculated for all strings of depth $\leq d_{Stage}$ (mse_{Low}) and separately for the string of depth $= d_{Stage}$ (mse_{High}), which is the longest string in the sequence. The fitness of the network at the end of each epoch is then calculated as the sum of squares of both errors:

$$\text{fitness} = mse_{Low} \cdot mse_{Low} + mse_{High} \cdot mse_{High},$$

This fitness sum is an adaptive version of the linear combination used in [14]. It automatically reinforces the influence of any small change of either the low or the high error and thus damps oscillations between both parts of the error. Similarly the accuracy of correct prediction is separately calculated for the low and the high part of the string.

Selection follows the rule:

Accept new mutant if either:
1.) Mutant's accuracy on the low part of the sequence is equal to 1.0 and its mean squared error on the high part of the sequence is less than that of the champ.
or
2.) Champ's accuracy on the low part of the string is less than 1.0 *and mutant's fitness is higher than that of the champ.*

4.3 Data Juggling Algorithm

The idea of the *Data Juggling Algorithm* whose pseudo code is listed in Table 3 is to use the function 'juggle' to permute the order of the symbol strings in the training sequence and to continue training on the modified training set. The function 'juggle' can be called at any time depending on some conditions, e.g. as soon as the champion has reached a new stage or after a fixed number (nEpochs) of iterations (i.e. when the algorithm 'got stuck'). While epoch learning methods like evolutionary hill climbing use a fixed training set, data juggling can be regarded as a step towards pattern learning.

Three parameters control the juggling process:

- *Juggle factor*: determines how many strings change positions in the sequence during one permutation. After a series of tests with different juggle factors we decided to use juggle factor 4 as standard for our experiments.
- *Maximum of Juggles*: determines how many permutations the network is trained on before a new stage is approached. In our experiments we increased the maximum of juggles depending on the stage.
- *Maximum of Epochs*: determines the maximum of epochs which should be spent on one training sequence.

4.4 Evolutionary Training Process

Starting with a randomly generated set of weights the network evolves, following the combination of incremental hill climbing and permutation of the training sequence which is implemented in the data juggling algorithm.

The epochs are counted from the beginning and each time a new solution network evolves which is able to process a newly generated training sequence with 100% accuracy its weights and the associated epoch are written into the 'epochfile'. Finally the epochfile contains a list of epochs and associated solution networks which emerged in the course of evolution.

Hence each training process results not in a single solution but in a series of solution networks with typically several solutions for each stage.

```
WHILE(d_stage ≤ d_MAX)
    Evaluate champ
    FOR ce = 1 to nEpochs
        Generate mutant
        Evaluate mutant
        IF(mutant better than champ)
          champ ← mutant
          IF(d_champ > d_stage)
            d_stage ← d_champ
            Generate new training
              sequence up to d_stage
            break
          end IF
        end IF
    end FOR
    IF(juggling conditions)
      juggle()
    ELSE
      Generate new training
        sequence up to d_stage
    end IF
end WHILE
```

Table 3: The Data Juggling Algorithm.

5 Experiments and Evaluation

The experimental investigation employed three types of evaluation methods which will be described in more detail later:

1. Evaluation of the efficiency of training up to stage 8.
2. Characterisation of the evolution of hidden unit dynamics.
3. Refined evaluation of generalisation ability, see section 5.5.1.

For investigating how first order networks can learn $\mathcal{MA}_3$ we have trained SRNs and SRNSCs with and without incremental learning. These experiments at the same time answer the question of whether incremental learning is more efficient. We will refer to networks which begin their training at stage 3 as being trained *with incremental learning*, while networks which begin training at stage 8 will be considered to be trained *without incremental learning* because they miss out on the crucial 'incremental' period between stages 3 and 8.

Notational Convention We introduce a notational convention to distinguish experiments according to the following categories:

1. *Data type*: '*ab*', '*abb*', '*abc*' as short forms for a^nb^n, a^nb^{2n} or $a^nb^nc^n$, respectively.
2. *Stage at which incremental learning started*:
    - '*I3*' for incremental learning starting at stage 3.
    - '*I8*' for non-incremental learning of stage 8, but incremental learning from stage 8 upwards.

3. *Network type*: Simple recurrent network '*srn*' or simple recurrent network with shortcuts '*srnsc*'.
4. *Number of experiment or initial condition*: Usually between 1 and 30 or 40.
5. *Epoch number*: Only specified if a single network is evaluated, e.g. for analysis of hidden unit dynamics.

An example is

abcI3srn17_1,144,641

which is a simple recurrent neural network incrementally trained from stage 3 on the language $a^n b^n c^n$, which emerged at global epoch number 1,144,641 from the evolutionary process starting from initial condition 17 [7].

5.1 Efficiency of Incremental Learning

To investigate the influence of the incremental training scheme 30 networks with different random weights from a $N(0; 0.5)$ Gaussian distribution were generated. Each of them was trained first with incremental learning and then again using the same initial weights without incremental learning, directly starting from stage 8.

Experiments were conducted for both the the $a^n b^n$ task and the $a^n b^n c^n$ task.

The tables in the next two subsections display the number of training epochs which were required until the first network was evolved which could process a symbol sequence of stage 8 correctly.

A 'C' in the table means that the training process was cut after 1,000,000 epochs (or a higher number of epochs if specified) and the network did not reach a solution for stage 8 within this number of epochs.

In most cases we terminated training after 1,000,000 epochs for the $a^n b^n$ task or 2,000,000 for the $a^n b^n c^n$ task, but in some cases where we trained longer than 2,000,000 epochs the tables display higher epochs. The meaning of 'C' remains the same.

5.1.1 Incremental and Non-Incremental Learning of $a^n b^n$ with SRNSCs

Results of the twenty comparative experiments are displayed in Table 4. Twenty four solutions were found with and without incremental learning. The epoch numbers indicate that on average incremental learning was faster. To confirm this statistically we calculated confidence intervals. For this calculation we considered only experiments where both incremental learning and non-incremental learning found a solution starting from the same initial condition. Twenty such solution pairs were found below 1,000,000 epochs. This number increases to 22 solution pairs for training up to 2,000,000 epochs. The statistical results are displayed in Table 5.

Since we found disjoint confidence intervals we can now state:

- Training up to 1 million epochs with incremental learning reaches stage 8 significantly faster, with a confidence level of 99%, than training without incremental learning.
- Training up to 2 million epochs with incremental learning reaches stage 8 significantly faster, with a confidence level of 95%, than training without incremental learning.

[7] This network by the way is the one in Figure 3 and Table 12 below.

Experimental Results for the $a^n b^n$ Task Displaying where Stage 8 is Reached with and without Incremental Learning					
Exp. no.	1	2	3	4	5
with	37,141	26,006	12,479	14,630	325,891
without	288,054	195,842	150,000	162,138	169,930
Exp. no.	6	7	8	9	10
with	C 2,000,000	31,120	17,821	28,663	396,119
without	C 2,000,000	147,719	262,074	1,712,458	138,878
Exp. no.	11	12	13	14	15
with	76,553	8,148	96,870	10,728	58,103
without	236,235	256,002	43,971	C 2,000,000	303,568
Exp. no.	16	17	18	19	20
with	23,260	62,364	C 2,000,000	44,531	18,746
without	30,825	1,860,000	107,788	360,007	389,129
Exp. no.	21	22	23	24	25
with	5,618,848	10,001	17,621	10,244,631	4,054,440
without	C 2,000,000	C 2,000,000	134,449	391,342	212,297
Exp. no.	26	27	28	29	30
with	31,112	7,380	49,955	6,947,211	39,876
without	397,722	135,832	442,407	288,513	195,006
Evaluation					
with	24 solutions below 1,000,000 epochs				
without	24 solutions below 1,000,000 epochs				

Table 4: Incremental vs. non-incremental learning of $a^n b^n$ with SRNSCs.

Statistical Comparison of $a^n b^n$ Experiments with and without Incremental Learning		
30 experiments with cutoff after 10^6 epochs resulted in 20 solution pairs		
Incremental learning	with	without
Minimum	7,380	30,825
Maximum	396,119	442,407
Mean	66,668	221,989
Median	31,116	195,424
Standard deviation	103,851	114,494
Confidence interval for $\gamma = 0.99$	[253; 133,083]	[148,768; 295,210]
30 experiments with cutoff after 2×10^6 epochs resulted in 22 solution pairs		
Incremental learning	with	without
Minimum	7,380	30,825
Maximum	396,119	1,860,000
Mean	64,745	364,192
Median	31,116	216,038
Standard deviation	99,114	473,526
Confidence interval for $\gamma = 0.99$	[4,521; 124,969]	[76,468; 651,918]
Confidence interval for $\gamma = 0.95$	[20,581; 108,909]	[153,195; 575,191]

Table 5: Statistical evaluation of the efficiency of incremental learning on the $a^n b^n$ task.

5.1.2 Incremental and Non-Incremental Learning of $a^n b^n c^n$ with SRNSCs

We observed that the training process in the case of the $a^n b^n c^n$ task is more likely to 'get stuck' than in the case of $a^n b^n$.

Experimental results for the $a^n b^n c^n$ task displaying where stage 8 is reached with and without incremental learning					
Exp. no.	1	2	3	4	5
with	C 2,000,000	986,473	C 2,000,000	C 2,000,000	836,386
without	1,710,473	C 15,000,000	C 3,000,000	509,997	2,617,540
Exp. no.	6	7	8	9	10
with	C 2,000,000	C 2,000,000	579,171	844,385	1,374,642
without	3,240,956	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000
Exp. no.	11	12	13	14	15
with	3,487,326	C 7,000,000	839,451	242,541	222,301
without	C 2,000,000	C 2,000,000	645,000	C 2,000,000	C 2,000,000
Exp. no.	16	17	18	19	20
with	1,436,349	C 5,000,000	238,964	6,217,057	2,255,428
without	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000	761,227
Exp. no.	21	22	23	24	25
with	11,150,498	3,641,066	289,382	1,392,481	357,858
without	C 2,000,000	1,473,267	C 2,000,000	C 2,000,000	C 2,000,000
Exp. no.	26	27	28	29	30
with	1,251,682	8,653,672	2,638,069	814,510	665,046
without	C 2,000,000	1,021,226	1,637,845	C 2,000,000	C 2,000,000
Exp. no.	31	32	33	34	35
with	774,506	C 3,000,000	709,869	183,086	717,068
without	C 3,000,000	990,373	1,085,609	830,983	C 3,000,000
Exp. no.	36	37	38	39	40
with	1,703,701	2,171,513	774,506	C 3,000,000	860,413
without	2,162,407	C 3,000,000	C 3,000,000	C 3,000,000	C 3,000,000
Evaluation					
with	23 (17) solutions below 2,000,000 (1,000,000) epochs				
without	10 (5) solutions below 2,000,000 (1,000,000) epochs				

Table 6: Comparison of incremental and non-incremental learning of $a^n b^n c^n$ with SRNSCs.

These results indicate that incremental learning facilitates the training process in the case of the language $a^n b^n c^n$ even more than in the case of $a^n b^n$.

5.1.3 SRNs Learn $a^n b^n c^n$ With and Without Incremental Learning

For comparison we include some results on experiments using standard SRNs without shortcut connections as we had used them in [14]. SRNs are able learn the task and are also able to generalise as we shall see later (e.g. network number *abcI3srn17* in Table 12 and Figure 3). However, our experiments indicate that solutions with SRNSCs are easier to obtain than with SRNs (see Table 9). Comparative experiments with and without incremental learning in Table 7 show again that incremental learning is more efficient.

Experimental results with SRNs for the $a^n b^n c^n$ task displaying where stage 8 is reached with and without incremental learning					
Exp. no.	1	2	3	4	5
with	C 2,000,000	1,571,338	58,746	C 2,000,000	195,565
without	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000
Exp. no.	6	7	8	9	10
with	C 2,000,000	422,252	C 2,000,000	1,674,318	C 2,000,000
without	C 2,000,000	C 2,000,000	261,071	C 2,000,000	C 2,000,000
Exp. no.	11	12	13	14	15
with	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000
without	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000
Exp. no.	16	17	18	19	20
with	C 2,000,000	1,144,641	C 2,000,000	C 2,000,000	C 2,000,000
without	375,107	1,269,795	C 2,000,000	211,102	C 2,000,000
Exp. no.	21	22	23	24	25
with	C 2,000,000	902,129	C 2,000,000	972,448	C 2,000,000
without	1,995,119	C 2,000,000	C 2,000,000	C 2,000,000	C 2,000,000
Exp. no.	26	27	28	29	30
with	C 2,000,000	1,470,594	C 2,000,000	C 2,000,000	218,694
without	C 2,000,000	604,972	C 2,000,000	C 2,000,000	C 2,000,000
Evaluation					
with	10 (6) solutions below 2,000,000 (1,000,000) epochs				
without	6 (4) solutions below 2,000,000 (1,000,000) epochs				

Table 7: Comparison of incremental and non-incremental learning of $a^n b^n c^n$ with SRNs.

5.2 From Context-Free to Context-Sensitive

The question arises as to whether the improved performance for the a^nb^n task relative to the $a^nb^nc^n$ task can be explained by the smaller size of the network and training data. To investigate this question, we trained 30 networks on the context-free a^nb^{2n} language using the same network which was used for the $a^nb^nc^n$ experiments. The results in Table 8 show that even if network and data size are the same, learning the context-free a^nb^{2n} language is easier than learning the context-sensitive $a^nb^nc^n$ language.

Experimental results of the $a^nb^nc^n$ and a^nb^{2n} task displaying where stage 8 is reached with incremental learning					
Exp. no.	1	2	3	4	5
$a^nb^nc^n$	C 2,000,000	986,473	C 2,000,000	C 2,000,000	836,386
a^nb^{2n}	C 2,000,000	61,957	C 2,000,000	C 2,000,000	43,663
Exp. no.	6	7	8	9	10
$a^nb^nc^n$	C 2,000,000	C 2,000,000	579,171	844,385	1,374,642
a^nb^{2n}	C 2,000,000	153,713	14,746	32,473	36,350
Exp. no.	11	12	13	14	15
$a^nb^nc^n$	3,487,326	C 7,000,000	839,451	242,541	222,301
a^nb^{2n}	22,102	C 2,000,000	C 2,000,000	136,284	23,503
Exp. no.	16	17	18	19	20
$a^nb^nc^n$	1,436,349	C 5,000,000	238,964	6,217,057	2,255,428
a^nb^{2n}	C 2,000,000	65,719	825,079	70,088	18,861
Exp. no.	21	22	23	24	25
$a^nb^nc^n$	11,150,498	3,641,066	289,382	1,392,481	357,858
a^nb^{2n}	141,252	41,261	39,231	C 2,000,000	255,235
Exp. no.	26	27	28	29	30
$a^nb^nc^n$	1,251,682	8,653,672	2,638,069	814,510	665,046
a^nb^{2n}	C 2,000,000	C 2,000,000	20,234	28,492	97,003
Evaluation					
$a^nb^nc^n$	16 solutions below 2,000,000 epochs				
a^nb^{2n}	20 solutions below 2,000,000 epochs				

Table 8: Comparison of training on $a^nb^nc^n$ and a^nb^{2n}.

5.3 Overview of Training Results

Table 9 below gives an overview of all experiments of the previous sections which compared the number of epochs required to reach stage 8 for different versions of the algorithms, tasks and networks.

A *solution for stage d* is a network which is able to process at least one sequence of strings at stage d correctly.

The first column of the table specifies the task, the second column tells the network type and the third column indicates whether incremental learning (IL) was used. Every experiment consisted of either 30 or 40 runs (see 4th column) started from different initial weights. Columns 5 and 6 contain the number of solutions obtained below 1 million epochs and 2 million epochs, respectively. The numbers in brackets are the percentage values with respect to the number of networks in column four.

Overview of experiments on testing efficiency of learning up to stage 8					
Task	Network type	IL	#Networks	#Solutions below 10^6	#Solutions below $2 \cdot 10^6$
$a^nb^nc^n$	SRNSC	yes	40	17 (42.5%)	23 (57.5%)
$a^nb^nc^n$	SRNSC	no	40	5 (12.5%)	10 (25.0%)
$a^nb^nc^n$	SRN	yes	30	6 (20.0%)	10 (33.3%)
$a^nb^nc^n$	SRN	no	30	4 (13.3%)	6 (20.0%)
a^nb^{2n}	SRNSC	yes	30	20 (66.6%)	20 (66.6%)
a^nb^n	SRNSC	yes	30	24 (80.0%)	–
a^nb^n	SRNSC	no	30	24 (80.0%)	–

Table 9: Overview of the experimental results

5.4 Dynamics of Hidden Unit Activation

In addition to the experiments on the efficiency of incremental learning we have visualised the dynamics of hidden unit activation of some example solutions to obtain insight in their qualitative characteristics.

To explain the concepts more systematically the first example in Figure 3 displays the solution in a different way than the other examples (Figures 4, 6, 5 and 7) which are explained further below.

The trajectory of hidden unit dynamics in Figure 3 belongs to a SRN which has been trained on the $a^nb^nc^n$ task in experiment *abcI3srn17*. Displayed are the activations of the hidden units H1-H3 of the network in Figure 1for the predicted and the input symbols while processing the string $a^8b^8c^8$. Starting at the activation point corresponding to the first input symbol 'a1' the trajectory progresses through the state space, visiting the activation points for all symbols of the string until it reaches the activation obtained with the final input 'c8' which predicts again an a represented by a '•'. Generalisation tests for this particular example were conducted with respect to depth and different orderings of the strings in the symbol sequence. These results are displayed in Table 12 below.

For all other examples (including those displayed in Figures 4, 6, 5 and 7) the activation of units H1-H3 of the network in Figure 2 was recorded while processing the sequence $a^2b^2c^2|a^3b^3c^3|...|a^8b^8c^8$. Displayed are the resulting 105 states in a 3-dimensional graph and its 2-dimensional projections to the three coordinate planes. To detect and analyse cluster formation of states representing the same symbol the activation for each *predicted* symbol a, b and c is displayed as '•', '×' and '∘', respectively. The activation following the input of the last a in each string is represented by an asterisc '∗'. The 24 states of the final string $a^8b^8c^8$ are connected by lines (in the same way as in 3) to show qualitatively how the state trajectory steps through the symbol clusters.

For a global analysis of hidden unit activation dynamics of the whole evolutionary process we produced several movies which show how the dynamics evolves during training. We recorded the weights of every network that reached a new stage or was able to process a new ordering of the data sequence with 100% accuracy. All these weights together with the epoch at which they emerged were stored in an *epochfile*.

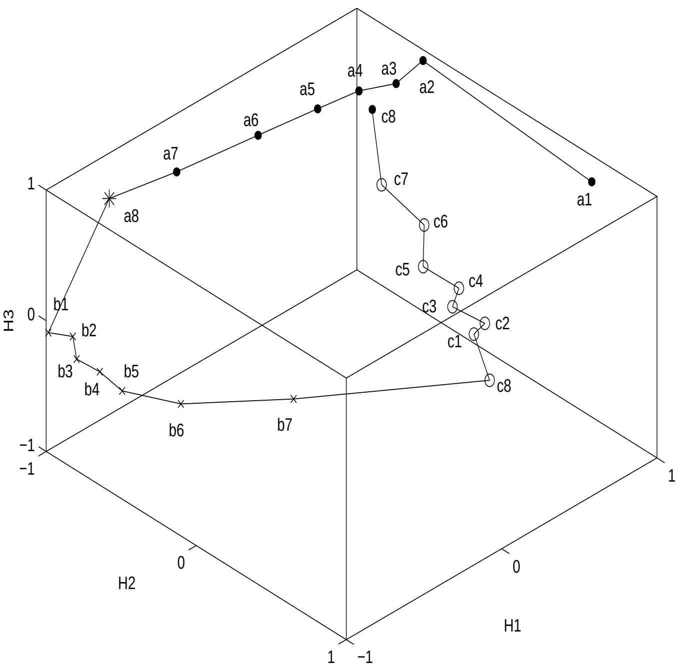


Figure 3: A SRN has partially learned the language $a^n b^n c^n$. Shown is the trajectory of hidden unit activations while the trained network processes the string $a^8 b^8 c^8$. The symbols '•', '×' and '∘' correspond to the *predicted* symbols 'a', 'b' and 'c', respectively. The first 'b' of the string cannot be predicted which is indicated by the asterisc '∗'. In addition to the predicted symbols, each state has assigned its corresponding input symbol 'a1–a8', 'b1–b8' and 'c1–c8'.

After training, each of these networks was evaluated on the data sequence $a^2b^2c^2|a^3b^3c^3|...|a^8b^8c^8$ and the activation of the hidden units (H1-H3) was plotted in a three dimensional graph as shown in the example Figures 4, 6, 5 and 7 and explained above. This way we obtained for each evolutionary process (corresponding to one initial condition) a sequence of graphs displaying the hidden unit activation dynamics of all solution networks of all stages. The movies helped to analyse the way evolution has found solutions to the language task. Figures 4, 6, 5 and 7 show four snapshots from four different movies.

Our analysis focused on a comparison of the incremental and the non-incremental approach and analysed visually using the movies seven pairings of incremental and non-incremental experiments (initial conditions: 1, 4, 6, 13, 20, 22, 27) where each pair started from the same initial condition. These pairings were selected from Table 6 above.

Surprisingly networks trained by incremental learning and non-incremental learning showed a qualitatively different pattern of hidden unit activation dynamics.

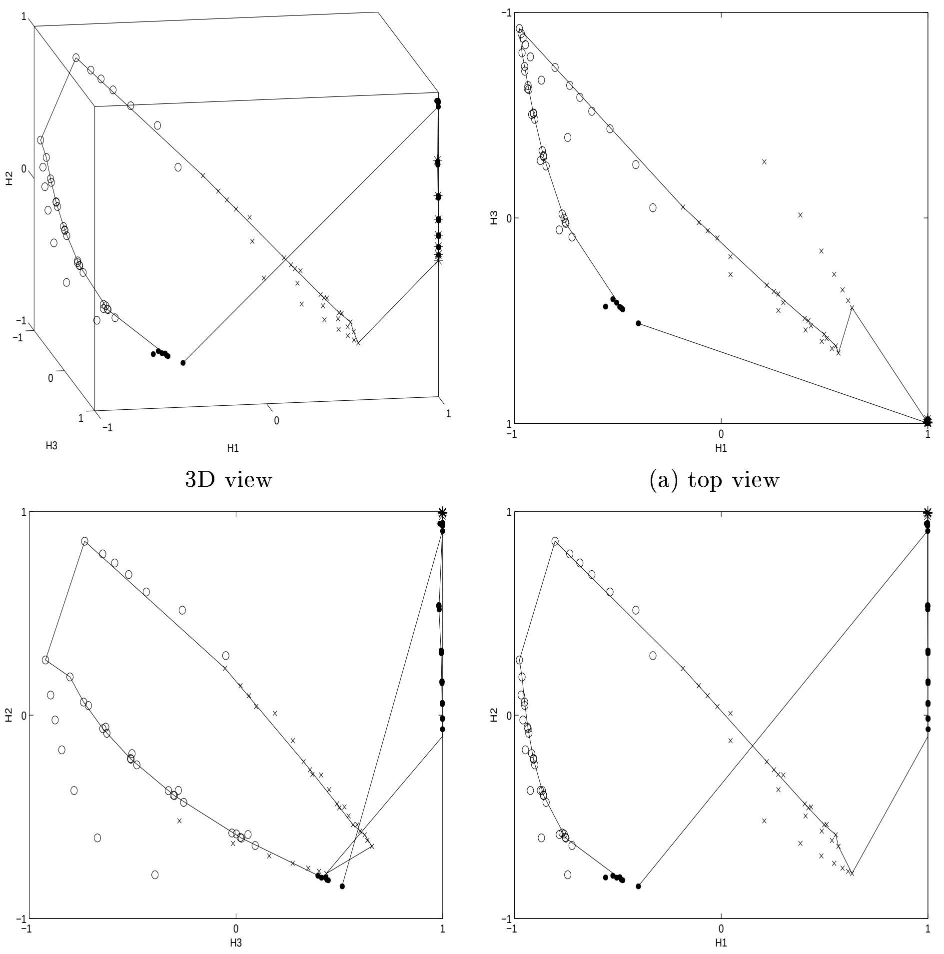
A trajectory is said to be *monotonic* if its projections to the coordinate axes develop monotonically in time, otherwise it is called *oscillating*. A similar classification of dynamics for networks trained on counting tasks was used in [46].

Most of the incremental processes exhibited a solution which was *monotonic* or almost monotonic within the clusters of points for each symbol a, b or c. These clusters typically were long and thin. The processes which were directly trained on stage 8 showed a different picture. The points in state space clustered in more compact groups and the trajectory *oscillated* within the clusters before it moved to the next cluster.

We display and discuss two representative solution pairings:

1. Solutions *abcI3srnsc4* and *abcI8srnsc4* started from the same initial condition (number 4) and were trained incrementally ($I3$) and non-incrementally ($I8$), respectively.

2. Solutions *abcI3srnsc20* and *abcI8srnsc20* started from the same initial condition (number 20) and



3D view

(a) top view

(b) side view

(c) front view

Figure 4: Trajectory of *abcI3srnsc4* at epoch 1,747,349. Incremental learning results in an almost monotonic solution.

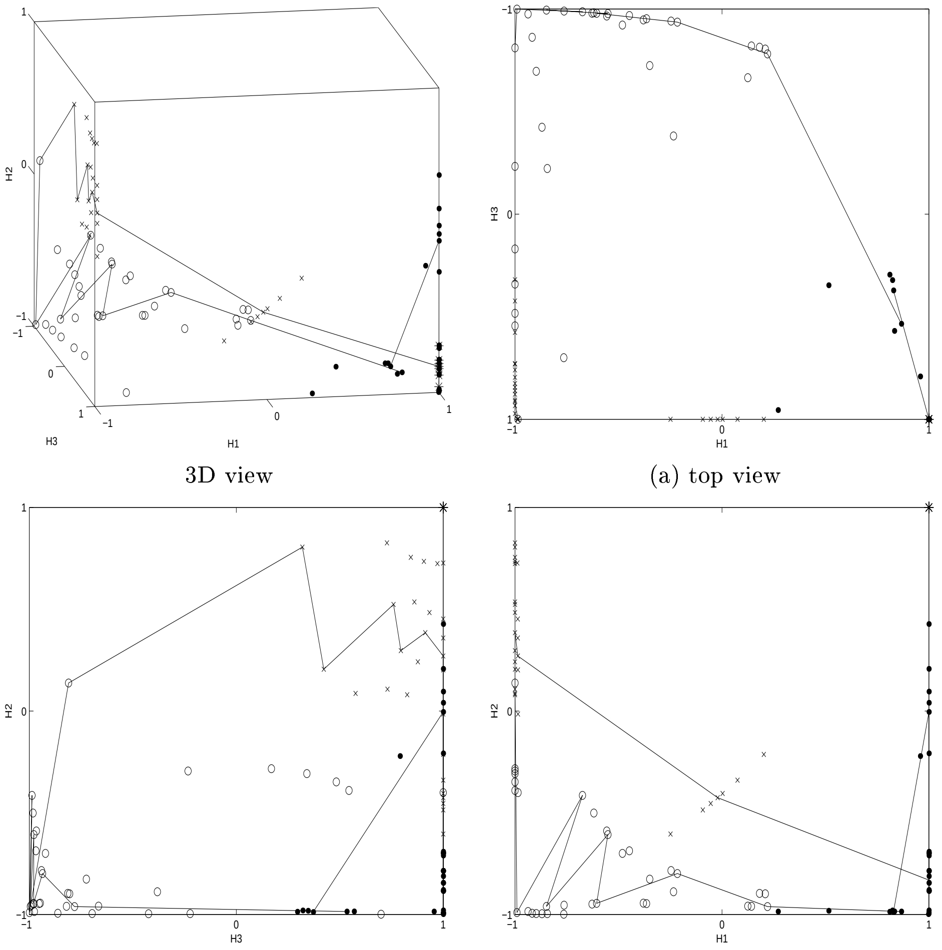
were trained incrementally ($I3$) and non-incrementally ($I8$), respectively.

The representative examples *abcI3srnsc4* and *abcI3srnsc20* (Figures 4 and 6) show a *monotonic* solution which is typical for almost all networks which were obtained using the incremental approach, while *abcI8srnsc4* and *abcI8srnsc20* (Figures 5 and 7) show *oscillations* within the symbol clusters which is typical for almost all networks which were trained directly on stage 8 without incremental learning.

The generalisation ability of these four examples will be evaluated below in section 5.5 in Tables 13, 16, 14 and 17 .

In contrast to the very unstable picture of generalisation ability which we discuss in section 5.5, the characteristics of the hidden unit activity dynamics during the course of evolution were very stable. When viewing the trajectories and state clusters of all 2000 – 3000 evolved networks almost all of them showed the described behaviour.

In both cases – the incrementally trained and the non-incrementally trained examples – it was remarkable that the solutions' qualitative picture of hidden unit dynamics hardly ever changed during the whole course of evolution. Once a solution was found the following development and subsequent solutions were locked into the same behaviour.



3D view (a) top view

(b) side view (c) front view

Figure 5: Trajectory of *abcI8srnsc4* at epoch 1,138,034. Non-incremental learning of stage 8 results in an oscillating solution.

5.5 Refined Generalisation Evaluation

Since it can happen that the networks unlearn and degenerate during training not only the latest network is evaluated but the whole sequence of networks which emerged during the evolutionary process.

We classify the networks in two categories depending on their generalisation ability regarding different orderings of the language strings s_n in the data sequence $v(d)$:

- *Weak solutions for stage d:* Networks which are able to process at least one sequence $v(d)$ at depth d.
- *Strong solutions for stage d:* Networks which are able to process all sample permutations [8] of the data sequence with 100% accuracy up to depth d.

We further distinguish between:

- *Weak jumps:* A network which is trained for stage d and which is able to process at least one permutation of stage $d+1$ is said to be performing a *weak jump* between the two stages.

[8] We tested 200 sample permutations for each stage and network.

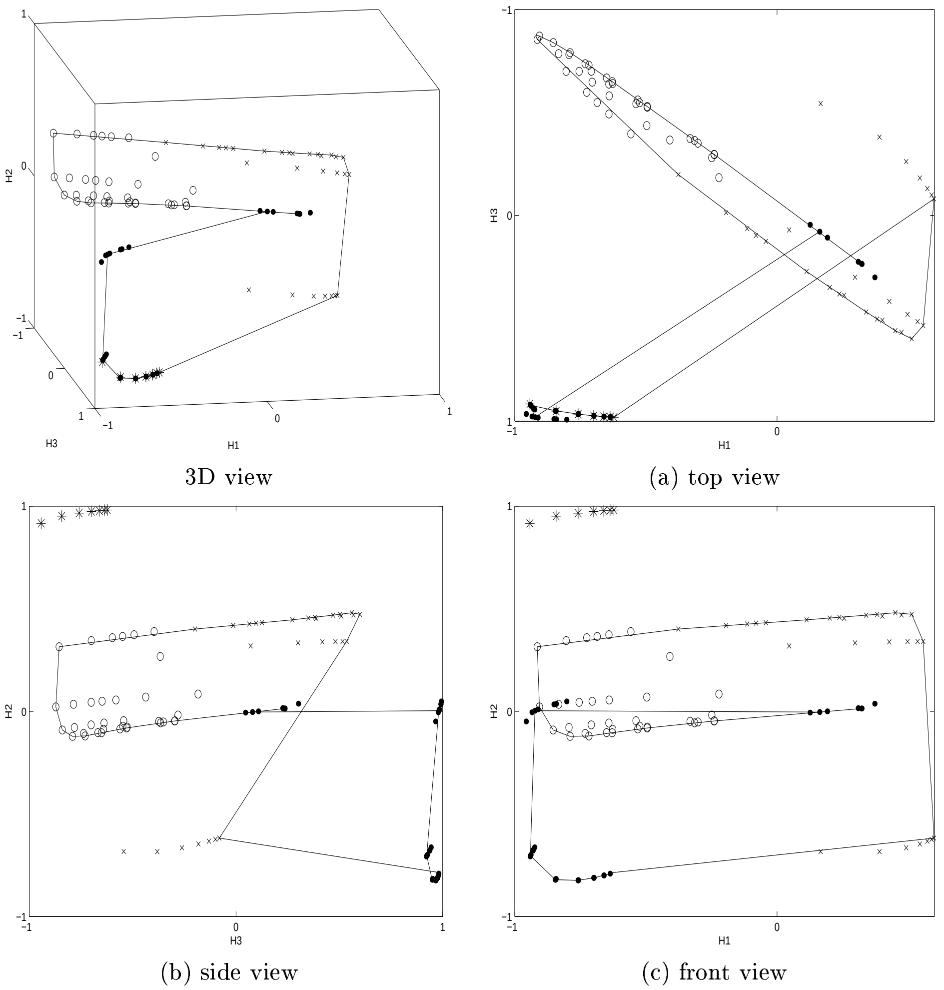
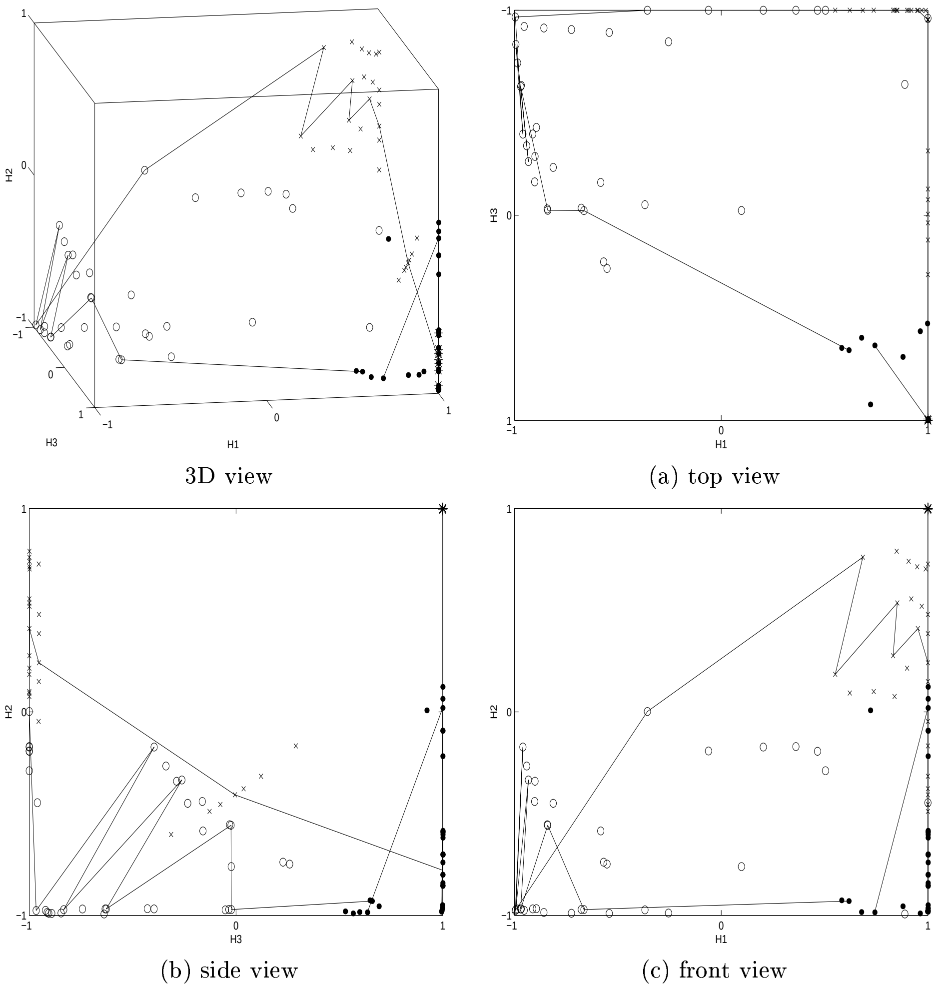


Figure 6: Trajectory of *abcI3srnsc20* at epoch 3,545,544. Incremental learning results in an almost monotonic solution.



3D view (a) top view

(b) side view (c) front view

Figure 7: Trajectory of *abcI8srnsc20* at epoch 764,587. Non-incremental learning of stage 8 results in an oscillating solution.

- *Strong jump:* A network which is a strong solution for the stage it has been trained for *and* also for the next higher stage is said to be performing a *strong jump* between the two stages.

Our method and terminology for testing generalisation ability of the trained recurrent networks is a refinement and extension of the approach taken by other studies of this kind where the network was only tested on a single string of the next stage.

The network's activation is reset at the end of each epoch, i.e. the hidden unit activation is transferred from the preceding to the following string within each training sequence. Therefore, a network which is able to process a sequence of strings including a string of the next higher stage might not be able to process this string in isolation. Only if the higher order string is at the beginning of the sequence is it obvious that the network would accept the isolated string, too. Otherwise it can happen that the network requires some of the other lower order strings of the sequence to be processed first to move to a state from which it accepts the higher order string.

On the other hand a network which has been tested only on a single higher order string might not be able to process all random order sequences of strings which contain that string.

Of course we must distinguish our training method which is epoch learning without reset of the state units within a sequence from other studies which might use reset of the state units after processing a single string or no resets at all. Networks using other reset schedules might have different generalisation behaviour. For networks with reset after each string, testing on a longer sequence of strings would not provide any advantage over testing only on a single string.

The following section contains refined evaluations of some example processes, including those which correspond to the initial conditions which were discussed in the previous sections. Generalisation was very unstable and networks generalising well were exceptional. Hence we have restricted our evaluation to a few characteristic example processes.

5.5.1 Explanation of the Generalisation Results

In the tables below the generalisation ability of a selection of networks is evaluated using our refined evaluation method.

In the first column of each of the following tables (e.g. Table 10) the depth of the longest string in the sequence which corresponds to the stage of training is displayed. In the horizontal direction each column corresponds to a neural network which was written to the epochfile. The number in the first row of each column is the epoch at which this happened. The other numbers in the table are the percentage values for correctly predicted permutations of the data sequence.

The number in bold corresponds to the stage the network has been trained for. If the entries below the bold number are not zero then the network shows jumping behaviour.

The tables display some of the networks which were generated during the evolutionary training. In most cases for each stage several networks were stored and the tables display the best network with respect to generalisation ability for each stage. In some cases two networks (e.g. if two interesting solutions were found) or no network (e.g. if no network for this stage was generated) is displayed. The best network for each stage is selected by taking into account its jumping ability in the first instance. The network with the longest jump is the best. If two networks show the same jumping behaviour they are compared regarding their ability to process sequences of different orderings correctly which is reflected in their mean accuracy (average of the numbers in one column). If two networks are the same with respect to jumps and mean accuracy the network whose mean accuracy has a lower standard deviation is regarded as better. In practice the best network could easily be identified.

5.5.2 Generalisation Results for $a^n b^n$

Two from thirty runs with SRNSCs are displayed in the two Tables below. While the first example *abI3srnsc21* (in Table 10) showed long weak jumps the second example *abI3srnsc28* (in Table 11) contained shorter but strong jumps.

Percentage of correctly predicted sequences (experiment *abI3srnsc21*)						
Stage	Epochs of successful networks					
	3,323	2,138,535	5,618,291	5,618,294	5,618,580	5,619,193
3	**100.0**	2.8	100.0	49.0	100.0	100.0
4	0.0	**27.2**	100.0	31.6	100.0	100.0
5	0.0	0.0	**97.0**	34.4	100.0	100.0
6	0.0	0.0	2.8	**19.4**	100.0	100.0
7	0.0	0.0	0.0	28.2	**100.0**	100.0
8	0.0	0.0	0.0	0.0	0.0	**99.8**
9	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0
	Epochs of successful networks					
	5,620,876	5,620,948	5,622,191	5,623,629	5,624,489	5,624,597
3	100.0	100.0	100.0	100.0	100.0	100.0
4	100.0	100.0	100.0	100.0	100.0	100.0
5	100.0	100.0	100.0	100.0	100.0	100.0
6	100.0	100.0	100.0	100.0	100.0	100.0
7	100.0	100.0	100.0	100.0	100.0	100.0
8	94.8	100.0	100.0	100.0	100.0	94.8
9	**88.8**	100.0	100.0	100.0	97.0	88.8
10	71.0	100.0	100.0	100.0	87.2	71.0
11	82.4	**100.0**	100.0	100.0	88.4	81.2
12	76.2	95.0	**100.0**	100.0	83.8	68.8
13	66.4	3.4	97.8	**100.0**	59.4	40.6
14	5.2	0.0	2.6	20.8	**76.2**	42.6
15	0.0	0.0	0.0	0.0	58.8	**17.2**
16	0.0	0.0	0.0	0.0	56.4	10.6
17	0.0	0.0	0.0	0.0	64.4	14.2
18	0.0	0.0	0.0	0.0	53.6	14.8
19	0.0	0.0	0.0	0.0	22.0	16.0
20	0.0	0.0	0.0	0.0	0.0	9.8

Table 10: Generalisation of *abI3srnsc21*.

5.5.3 Generalisation Results for $a^n b^n c^n$

The results were similar to the $a^n b^n$ task in the previous section but jumps were shorter and strong jumps were not observed. Some of the networks jumped weakly two stages ahead, e.g. the SRN in Table 12. Table 15 shows the generalisation results obtained for an SRNSC which was incrementally trained starting from initial condition no.8. Jumping ability and generalisation to different orderings change during the course of evolution which reflects the typical unstable learning behaviour of these networks. The last three columns in Table 15 seem to indicate a tradeoff between good generalisation regrding the ordering (column 6) and good generalisation regarding depth (column 5 and 7). Tables 13, 14, 16 and 17 correspond to the experiments with initial condition no.4 and no.20 for which we discussed the trajectories of hidden unit activation in section 5.4 above. We speculate that the non-incremental learning approach produces better generalising networks than the incremental learning approach. But this question is still open.

Percentage of correctly predicted sequences (experiment *abI3srnsc28*)										
Stage	Epochs									
	5,155	49,083	49,367	49,729	49,821	50,054	50,355	50,575	50,605	50,685
3	**100.0**	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
4	0.0	**100.0**	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
5	0.0	0.0	**100.0**	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	0.0	0.0	0.0	**100.0**	100.0	100.0	100.0	100.0	100.0	100.0
7	0.0	0.0	0.0	0.0	**100.0**	100.0	100.0	100.0	100.0	100.0
8	0.0	0.0	0.0	0.0	0.0	**100.0**	100.0	100.0	100.0	100.0
9	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	100.0	100.0	100.0
10	0.0	0.0	0.0	0.0	0.0	0.0	100.0	100.0	100.0	100.0
11	0.0	0.0	0.0	0.0	0.0	0.0	7.6	**100.0**	100.0	100.0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**100.0**	100.0
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	48.8	**100.0**
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	99.4
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 11: Generalisation of *abI3srnsc28*.

Percentage of correctly predicted sequences (experiment abcI3srn17)						
Stage	Epochs of successful networks					
	2,225	2,796	632,801	1,107,652	1,144,349	3,545,544
3	**47.0**	3,500	100.0	100.0	100.0	100.0
4	0.0	**11.5**	97.0	100.0	100.0	100.0
5	0.0	0.0	**17.0**	86.0	100.0	94.0
6	0.0	0.0	0.0	**26.5**	100.0	57.5
7	0.0	0.0	0.0	1.5	**60.5**	0.0
8	0.0	0.0	0.0	0.0	2.5	**0.0**
9	0.0	0.0	0.0	0.0	15.5	1.0

Table 12: Generalisation of *abcI3srn17*.

6 Discussion and Significance

The fact that artificial neural networks are able to learn a mildly-context sensitive grammar [14] appears to establish a missing link in a sequence of findings and arguments which can help us to connect results obtained via artificial neural network modeling to phenomena observed in natural language acquisition.

To explain this connection the following paragraphs briefly review some of the links which were obtained or proposed by other researchers from different disciplines.

The language of multiple agreements $\mathcal{MA}_3 = \{a^n b^n c^n; n \geq 1\}$ belongs by definition to the family of mildly-context sensitive languages (MCSLs) (see definition in Appendix A) which contains the most significant languages that appear in the study of natural languages [34], [37].

MCSLs are only slightly more powerful than CFLs and preserve their essential properties. Extended push down automata (EPDAs) are related to tree adjoining grammars. While pushdown automata, which correspond to CFLs, consist of a simple stack, EPDAs can consist of a sequence of stacks. TAGs use adjunction as their basic composition operation. Hence TALs are mildly context-sensitive and contain CFLs. EPDAs can process crossed dependencies as they arise in certain languages like Dutch.

EPDAs also can recognise strings of $\{a^n b^n c^n; n \geq 1\}$ or $\{a^n b^n c^n d^n; n \geq 1\}$.

Incremental natural language processing was studied from a psycholinguistic perspective in [38] and schema-tree adjoining grammars (Schema-TAGs) [61] were proposed in [28] as a corresponding formal representation.

Mechanisms of language processing in the brain have been a topic of intense research, see e.g. [27, 11, 19,

Percentage of correctly predicted sequences (experiment *abcI3srnsc4*)						
Stage	Epochs of successful networks					
	278,252	554,543	888,058	983,680	1,747,349	1,864,580
3	**81.0**	0.0	100.0	100.0	100.0	100.0
4	0.0	**1.5**	100.0	100.6	52.0	100.0
5	0.0	0.0	**98.5**	71.0	60.0	61.0
6	0.0	0.0	0.0	**59.5**	17.5	67.5
7	0.0	0.0	0.0	0.0	**28.0**	**18.0**
8	0.0	0.0	0.0	0.0	4.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0

Table 13: Generalisation of *abcI3srnsc4*.

41, 44, 45].

The use of discrete recurrent neural networks for modeling aspects of language processing in the brain is built on several assumptions which all have a speculative component and are a topic of ongoing discussions in the neural network community.

While some people argue that all processes in biological brains are continuous, several studies proposed that higher order brain processes related to language or consciousness are of symbolic and hence discrete nature, see e.g. [47, 40].

Artificial recurrent neural networks have been proposed as models for recurrent neural configurations as found in the CA3 region of the hippocampus [48]. But the cortex has many recurrent connections [1] and artificial recurrent neural networks might also be abstractions of neural circuits in other brain areas which are involved in language processing. With respect to syntax several areas beyond the classical Broca area are involved, see [27].

In summary we have now cited a list of studies which connect biological mechanisms in the brain to classical discrete connectionist models, and dicussed the ability of these models to process formal languages (see also [58]). With our current study we extend this chain of arguments to the point where connectionist models start to be able to learn a kind of language complexity which is according to some current linguistic opinions closer to natural language than the classical language complexity classes of the Chomsky hierarchy.

Pending on validity of these assumptions our study might shed additional light on questions of language acquisition by incremental, evolutionary and learning processes in the brain.

Apart from more linguistic and biological aspects a lengthy, controversial and often speculative debate about neural networks and natural language is going on in the field of connectionist psycholinguistics, see e.g. [17] to get an overview.

Percentage of correctly predicted sequences (experiment *abcI3srnsc20*)						
Stage	Epochs of successful networks					
	9,393	46,988	399,131	559,277	1,923,200	3,545,544
3	**100.0**	92.0	98.0	96.0	80.0	0.0
4	0.0	**100.0**	95.0	97.0	99.0	59.0
5	0.0	0.0	**85.0**	94.0	59.0	97.0
6	0.0	0.0	0.0	**83.0**	47.0	94.0
7	0.0	0.0	0.0	0.0	**24.0**	94.0
8	0.0	0.0	0.0	0.0	0.0	**82.0**
9	0.0	0.0	0.0	0.0	0.0	0.0

Table 14: Generalisation of *abcI3srnsc20*.

Percentage of correctly predicted sequences (experiment *abcI3srnsc8*)							
Stage	Epochs						
	106,839	133,404	340,542	1,206,191	1,289,798	1,485,278	1,596,517
3	**0.2**	100.0	93.9	98.6	1.6	100.0	79.7
4	0.0	**99.9**	0.9	73.3	26.9	100.0	27.5
5	0.0	0.0	0.0	78.9	38.6	100.0	22.1
6	0.0	0.0	**0.8**	47.5	79.0	100.0	45.7
7	0.0	0.0	0.0	3.8	46.0	100.0	30.2
8	0.0	0.0	0.0	**49.9**	44.7	100.0	17.4
9	0.0	0.0	0.0	13.3	**23.1**	100.0	13.7
10	0.0	0.0	0.0	2.3	38.1	**100.0**	5.5
11	0.0	0.0	0.0	0.0	49.3	0.0	**16.0**
12	0.0	0.0	0.0	0.0	0.0	0.0	13.8
13	0.0	0.0	0.0	0.0	0.0	0.0	10.7
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 15: Generalisation of *abcI3srnsc8*.

Percentage of correctly predicted sequences (experiment *abcI8srnsc4*)							
Stage	Epochs of successful networks						
	1,138,034	1,181,893	1,615,017	1,956,311	2,504,541	2,693,464	2,949,268
3	100.0	100.0	100.0	100.0	100.0	100.0	100.0
4	100.0	100.0	100.0	100.0	100.0	100.0	100.0
5	100.0	100.0	100.0	100.0	100.0	98.5	100.0
6	100.0	100.0	100.0	100.0	100.0	86.5	100.0
7	100.0	100.0	99.5	100.0	100.0	64.5	100.0
8	**100.0**	100.0	99.5	100.0	100.0	77.0	94.5
9	0.0	**100.0**	99.0	100.0	100.0	59.5	61.5
10	0.0	0.0	**100.0**	99.5	100.0	87.5	36.0
11	0.0	0.0	0.0	**100.0**	99.0	79.0	53.0
12	0.0	0.0	0.0	0.0	**96.0**	63.5	35.0
13	0.0	0.0	0.0	0.0	0.0	**52.0**	21.5
14	0.0	0.0	0.0	0.0	0.0	10.0	**30.0**
15	0.0	0.0	0.0	0.0	0.0	0.0	23.5
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 16: Generalisation of *abcI8srnsc4*.

7 Summary

This study has shown how to use evolutionary hill climbing to train first order recurrent neural networks on the $a^nb^nc^n$ task. Solutions were evolved which generalise in depth and to different orderings of the string sequence for both SRNs and SRNSCs. The latter network architecture was developed specifically for this task.

Therefore first order networks are sufficient for learning this type of context-sensitive language and second order networks as they were suggested in [8] (or [54]) are not obligatory but may have other training and generalisation properties. Due to different language tasks and training schemes the studies cannot directly be compared.

The question of whether incremental learning is more efficient than non-incremental learning has been investigated on the a^nb^n and the $a^nb^nc^n$ task. In almost all experiments incremental learning produced more solutions than non-incremental learning and incremental learning produced its solutions faster. In the case of a^nb^n enough solutions were obtained to verify the significance of the speed difference. The solutions for $a^nb^nc^n$ indicate at least the same speed difference.

Percentage of correctly predicted sequences (experiment *abcI8srnsc20*)						
Stage	Epochs of successful networks					
	761,227	764,587	771,069	781,299	785,075	794,584
3	0.0	100.0	100.0	100.0	100.0	100.0
4	10.0	100.0	100.0	100.0	100.0	100.0
5	27.0	100.0	100.0	100.0	100.0	100.0
6	15.0	100.0	100.0	100.0	100.0	100.0
7	67.0	100.0	100.0	100.0	100.0	100.0
8	**10.0**	**100.0**	77.0	100.0	100.0	100.0
9	7.0	0.0	**36.0**	100.0	60.0	100.0
10	0.0	0.0	2.0	**100.0**	64.0	100.0
11	0.0	0.0	0.0	1.0	**35.0**	100.0
12	0.0	0.0	0.0	0.0	2.0	**100.0**
13	0.0	0.0	0.0	0.0	0.0	15.0
14	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0
	Epochs of successful networks					
	799,197	803,702	901,590	925,777	1,033,597	1,159,221
3	100.0	100.0	100.0	100.0	100.0	100.0
4	100.0	100.0	100.0	100.0	100.0	100.0
5	100.0	88.0	100.0	100.0	100.0	100.0
6	100.0	93.0	100.0	100.0	99.0	100.0
7	100.0	92.0	100.0	100.0	89.0	100.0
8	100.0	72.0	100.0	100.0	96.0	100.8
9	100.0	87.0	100.0	100.0	97.0	100.8
10	100.0	63.0	100.0	100.0	100.0	100.0
11	100.0	70.0	100.0	99.0	97.0	99.0
12	100.0	61.0	100.0	95.0	99.0	79.0
13	**100.0**	79.0	100.0	98.0	99.0	83.0
14	0.0	**59.0**	100.0	96.0	100.0	52.0
15	0.0	27.0	**100.0**	90.0	93.0	74.0
16	0.0	0.0	0.0	**99.0**	100.0	86.0
17	0.0	0.0	0.0	6.0	**99.0**	84.0
18	0.0	0.0	0.0	0.0	21.0	**70.0**
19	0.0	0.0	0.0	0.0	0.0	2.0
20	0.0	0.0	0.0	0.0	0.0	0.0

Table 17: Generalisation of *abcI8srnsc20*.

Training on the context-sensitive language task $a^nb^nc^n$ is harder than on the context-free language tasks a^nb^n or a^nb^{2n} .

The experiments show that training is unstable and very dependent on the initial weights. Symbol sequences which have been learned by the network might get unlearned at a later stage of the evolutionary training process and generalisation ability can go up and down.

A visualisation and brief analysis of hidden unit dynamics indicates that incremental learning produces more monotonic solutions while non-incremental learning results in solutions with trajectories which tend to oscillate within the state clusters.

In contrast to the abovementioned instability regarding generalisation ability of the networks during training, the qualitative picture of hidden unit dynamics remained stable during the whole process in almost all of the roughly 200 evaluated evolutionary processes. Once a network had learned to process the first stage it was trained on, the characteristics of hidden unit dynamics of this network did not change during further training.

A Formal Language Theory

In books on formal languages, e.g. [32, 55] we find the following definitions:

An *unrestricted grammar* is a quadruple (V, Σ, P, S) where V is a finite set of variables, Σ (the alphabet) is a finite set of terminal symbols, P is a set of rules and S is a distinguished element of V, called the start symbol. The rules have the form $u \to v$, where $u \in (V \cup \Sigma)^+$, $v \in (V \cup \Sigma)^*$ and $V \cap \Sigma = \emptyset$.

The other three classes of grammars are restrictions of the unrestricted grammar by allowing only certain rules:

A *context-sensitive grammar* has rules of the form $u \to v$, where $u \in (V \cup \Sigma)^+$, $v \in (U \cup \Sigma)^+$ and $length(u) \leq length(v)$.

In a *context-free grammar* the rules are of the form $A \to \alpha$, $A \in V$ and α is a string of variables from $(V \cup \Sigma)^*$. The left-hand side of each rule is a single variable.

The rules for a *regular grammar* are: $A \to aB$, $A \to a$ or $A \to \lambda$, where $A, B \in V$ and $a \in \Sigma$.

Remarks on context-sensitive languages:

1. The name 'context-sensitive' stems from an old definition where each rule had to be of the form $uAv \to uwv$, where $A \in V$, $w \in (V \cup \Sigma)^+$ and $u, v \in (V \cup \Sigma)^*$, i.e. the variable A can be replaced by w only when it is in the context of being preceded by u and followed by v [55].
2. $a^n b^n$ is context-free (CF) but not regular and $a^n b^n c^n$ is context-sensitive (CS) but not context-free.
3. A context-sensitive grammar for $\{a^n b^n c^n;\, n \geq 0\}$ is given by:
   $S \to aAbc|abc$
   $A \to aAbC|abC$
   $Cb \to bC$
   $Cc \to cc$

The Family of Mildly Context-Sensitive Languages A *mildly context-sensitive family of languages (MCSL)* is a family of languages which satisfies the following conditions:

1. Each language in MCSL is semilinear [37].
2. For each language in MCSL the membership problem can be solved in deterministic polynomial time.
3. MCSLs contain the following three non-context-free languages:
   (a) Language of multiple agreements $\mathcal{MA}_3 = \{a^n b^n c^n;\, n \geq 0\}$
   (b) Language of crossed arguments $\mathcal{CA} = \{a^n b^m c^n d^m;\, n, m \geq 0\}$
   (c) Language of duplication $\mathcal{D} = \{ww;\, w \in \{a, b\}^*\}$

This definition was taken from [37].

References

[1] M. Abeles. *Corticonics. Neural Circuits of the Cerebral Cortex.* Cambridge University Press, 1991.

[2] P.J. Angeline, G.M. Saunders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5:54–65, 1994.

[3] M.A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks.* Cambridge, Mass : MIT Press, 1995.

[4] M. Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[6] A.D. Blair and J.B. Pollack. Analysis of dynamical recognizers. *Neural Computation*, 9(55):1127–1142, 1997.

[7] M. Boden, H. Jacobson, and T. Ziemke. Evolving context-free language predictors. In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, 2000.

[8] M. Boden and J. Wiles. Context-free and context-sensitive dynamics in recurrent neural networks. *submitted to Connection Science*, 2000.

[9] M. Boden, J. Wiles, B. Tonkes, and A. Blair. Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units. In *Proceedings of ICANN 99*, pages 359–364, Edinburgh, 1999. IEE.

[10] M. Boden, J. Wiles, B. Tonkes, and A. Blair. On the ability of recurrent nets to learn deeply embedded structures. In *presented at the IJCAI 99 workshop on Sequence learning*, Stockholm, 2000.

[11] Brain and Language. Journal, 1993-2000. ISSN 0093-934X.

[12] R.C. Carrasco and M.L. Forcada. Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, 1999.

[13] M. Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135–1178, 1996.

[14] S. Chalup and A.D. Blair. Hill climbing in recurrent neural networks for learning the $a^n b^n c^n$ language. In T. Gedeon et al., editor, *Proceedings, 6th International Conference on Neural Information Processing (ICONIP'99), 16-20 November 1999, Perth, Australia*, volume 2, pages 508–513, 1999. ISBN 0-7803-5871-6.

[15] S. Chalup and F. Maire. A study on hill climbing for neural network training. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99), July 6-9, 1999, Washington, D.C., USA*, pages 2014–2021, 1999. ISBN 0-7803-5536-9.

[16] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137–167, 1959.

[17] M.H. Christiansen and N. Chater. Connectionist natural language processing: The state of the art. *Cognitive Science*, 23(4):417–437, 1999.

[18] A. Cleeremans, D. Servan-Schreiber, and J.L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[19] T.W. Deacon. *The Symbolic Species. The Co-Evolution of Language and the Brain.* W. W. Northon and Company, 1997.

[20] K. Doya. Bifurcations in the learning of recurrent neural networks. In *Proceedings of 1992 IEEE International Symposium on Circuits and Systems*, pages 2777–2780, 1992.

[21] K. Doya. Recurrent networks: Supervised learning. In Arbib [3], pages 796–800.

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

[23] G.D. Finn. A recurrent neural network for the language $a^n b^n$: Encoding and decoding binary strings in a neuron's activation function. Technical report, Machine Learning Research Centre, School of Computing Science, FIT, Queensland University of Technology, 1998. unpublished.

[24] C.L. Giles and M. Gori, editors. *Adaptive Processing of Sequences and Data Structures.* Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998.

[25] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[26] M. Gori, M. Maggini, E. Martinelli, and G. Soda. Inductive inference from noisy examples using the hybrid finite state filter. *IEEE Transactions on Neural Networks*, 9(3):571–575, 1998.

[27] Y. Grodzinsky. The neurology of syntax: Language use without broca's area. *Behavioral and Brain Sciences*, 23(1), 2000.

[28] K. Harbusch. Incremental natural-language processing with schema-tree adjoining grammars. Technical Report 41, Deutsches Forschungszentrum für künstliche Intelligenz (DFKI) GmbH, 1994. Verbmobil Verbundvorhaben.

[29] S. Haykin. *Neural Networks. A Comprehensive Foundation.* Prentice Hall, 2 edition, 1999.

[30] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, 1991.

[31] S. Hölldobler, Y. Kalinke, and H. Lehmann. Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of 21st German Conference on Artificial Intelligence*, LNAI 1301, pages 313–324. Springer-Verlag, 1997.

[32] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company, 1979.

[33] A.K. Joshi and Y. Schabes. *Tree-Adjoining Grammars*, pages 69–123. Volume 3 of Rozenberg and Salomaa [49], 1997.

[34] A.K. Joshi, K. Vijay-Shanker, and D. Weir. *The convergence of mildly context-sensitive grammatical formalisms*, pages 31–81. In Sells et al. [51], 1991.

[35] J.F. Kolen. Recurrent networks: State machines or iterated function systems? In M.C. Mozer, P. Smolensky, D.S. Touretzky, and J. L. Elman, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 203–210. Erlbaum Associates, 1994.

[36] S. Kremer. On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.

[37] M. Kudlek, C. Martin-Vide, and A. Mateescu. An infinite hierarchy of mildly context-sensitive families of languages. Technical Report 163, Turku Centre for Computer Science (TUCS), February 1998. ISBN 952-12-0163-0.

[38] W.J.M. Levelt. *Speaking: From Intention to Articulation.* The MIT Press, Cambridge, MA, 1989.

[39] W. Maass and P. Orponen. On the effect of analog noise in discrete-time analog computation. In *Proceedings Neural Information Processing Systems*, pages 218–224, 1996.

[40] T. Omori, A. Mochizuki, K. Mizutani, and M. Nishizaki. Emergence of symbolic behaviour from brain like memory with dynamic attention. *Neural Networks*, 12(7-8):1157–1172, 1999. Special Issue: Organisation of Computation in Brain-like Systems.

[41] S. Pinker. *The language instinct: How the mind creates language.* New York: William Morrow, 1994.

[42] J. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[43] J.B. Pollack. *On Connectionist Models of Natural Language Processing.* PhD thesis, Computer Science Department, University of Illinois, Urbana, IL, 1987.

[44] M.I. Posner and M.E. Raichle. *Images of Mind.* Scientific American Library, 1994.

[45] F. Pulvermueller. Words in the brain's language. *Behavioral and Brain Sciences*, 1998.

[46] P. Rodriguez, J. Wiles, and J.L. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

[47] E.T. Rolls. Consciousness in neural networks ? *Neural Networks*, 10(7):1227–1240, 1997.

[48] E.T. Rolls and A. Treves. *Neural Networks And Brain Function.* Oxford University Press, 1998.

[49] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, 1997.

[50] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, pages 533–536, 1986.

[51] P. Sells, S.M. Shieber, and T. Wasow, editors. *Foundations Issues in Natural Language Processing*. MIT Press, 1991.

[52] H.T. Siegelmann. *Neural Networks and Analog Computation - Beyond the Turing Limit*. Progress in Theoretical Computer Science. Birkhäuser Boston, 1999.

[53] E. Sontag. *Automata and Neural Networks*, pages 119–123. In Arbib [3], 1995.

[54] M. Steijvers and P. Grünwald. A recurrent network that performs a context-sensitive prediction task. In *Proceedings of the 18th Annual Conference of the Cognitive Science Society*, pages 335–339. Morgan Kauffman, 1996.

[55] T.A. Sudkamp. *Language and Machines: An Introduction to the Theory of Computer Science*. Addison Wesley, 2nd edition, 1997.

[56] P. Tino and J. Sajda. Learning and extracting initial mealy automata with a modular neural network model. *Neural Computation*, 7(4), 1995.

[57] B. Tonkes, A. Blair, and J. Wiles. Inductive bias in context-free language learning. In *Proceedings of the Ninth Australian Conference on Neural Networks (ACNN'98), Brisbane*, pages 52–56. University of Queensland, 1998.

[58] D. Touretzky, editor. *Connectionist Approaches to Language Learning*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1991. A Special Issue of Machine Learning on Connectionist Approaches to Language Learning.

[59] A.C. Tsoi. *Recurrent Neural Network Architectures: An Overview*, pages 1–26. In Giles and Gori [24], 1998.

[60] A.C. Tsoi and A. Back. Discrete time recurrent neural network architectures: A unifying review. *Neurocomputing*, 15:183–233, 1997.

[61] D. Weir. Characterizing mildly context-sensitive grammar formalisms. Ph.D. Proposal, 1987.

[62] J. Wiles and J.L. Elman. 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*, pages 482–487. MIT Press, 1995.

[63] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, May 1997.

[64] Z. Zeng, R.M. Goodman, and P. Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 5(2):320–330, 1994.