
Evolving context-free language predictors

Mikael Bodén

mikael@csee.uq.edu.au

Department of Computer Science
and Electrical Engineering
University of Queensland
QLD 4072, Australia.

Henrik Jacobsson

henrikj@ida.his.se

Department of Computer Science
University of Skövde
Box 408, Skövde, Sweden.

Tom Ziemke

tom@ida.his.se

Department of Computer Science
University of Skövde
Box 408, Skövde, Sweden.

Abstract

Recurrent neural networks can represent and process simple context-free languages. However, the difficulty of finding with gradient-based learning appropriate weights for context-free language prediction motivates an investigation on the applicability of evolutionary algorithms. By empirical studies, an evolutionary algorithm proves to be more reliable in finding prediction solutions to a simple CFL. Moreover, the evolutionary algorithm demonstrates greater diversity by making use of a larger repertoire of dynamical behaviors for solving the problem.

1 INTRODUCTION

A series of investigations (Wiles and Elman, 1995; Tonkes et al., 1998; Rodriguez et al., 1999; Bodén et al., 1999) has established that the one-step look-ahead prediction task for simple context-free languages (CFLs) is difficult to learn for a Simple Recurrent Network (SRN; Elman, 1990) using a gradient-based learning algorithm. A simple CFL is 0^n1^n which allows strings starting with any number of 0s followed by exactly the same number of 1s, e.g. 000111 ($n = 3$), 01 ($n = 1$), and 0000000011111111 ($n = 8$). The most successful solution type for predicting 0^n1^n makes use of contracting oscillation for counting up 0s and expanding oscillation for counting down 1s. From a dynamical systems perspective, training an SRN to the oscillating solutions involves going through at least one critical bifurcation in which the system is non-hyperbolic. In the vicinity of the bifurcation border the error gradient becomes chaotic (Bodén et al., 1999) – making gradient-based weight adaptation inappropriate. To that end, evolutionary algorithms (EAs) have shown promising results of reliably generating

suitable weight sets in the same domain (Batali, 1994; Tonkes et al., 1998).

This paper reports on some simulations which evolve weight sets for both first-order and second-order recurrent networks to predict the CFL 0^n1^n . Moreover, we address the question if the solutions found by the EA are qualitatively different compared to those found by Back-Propagation Through Time (BPTT).

2 EXPERIMENTS

Two network types were used: The SRN, which is the first-order recurrent network most commonly used for formal language recognition/prediction, and the Sequential Cascaded Network (SCN; Pollack, 1991), a second-order recurrent network. Both networks made use of the same number of processing units (1 input, 1 output and 2 state units) but due to the connection topology the SCN had more weights (see Figure 1).

The process for calculating the output of each network, $o_k^t = F(i^t)$, varies with the network type. The output of the SRN (for an arbitrary number of units) is defined as

$$o_k^t = f\left(\sum_j w_{kj}h_j^t + w_{k\theta}\right) \quad (1)$$

$$h_j^t = f\left(\sum_i v_{ji}i_i^t + \sum_m v_{jm}h_m^{t-1} + v_{j\theta}\right) \quad (2)$$

The index k is used for identifying the output units, j and m are used for the hidden units (at time t and $t - 1$ respectively), and i is used for the input. Biases are introduced as additional elements in the weight matrices, w and v (indexed with θ).

The output of the SCN (with arbitrary number of units) is defined as

$$o_k^t = f\left(\sum_{i,j} w_{kji}z_j^{t-1}i_i^t + \sum_j w_{kj\theta}z_j^{t-1} + \right)$$

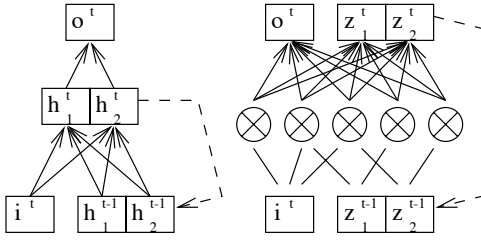


Figure 1: The network architectures: (1) The SRN with 1 input (i), 2 hidden units (h_1 and h_2) and 1 output unit (o). The hidden activation is copied back to the input layer and used in the next presentation. Biases for the hidden units and the output unit are not shown. (2) The SCN with 1 input (i), 2 state units (z_1 and z_2) and 1 output unit (o). The state from the previous time step is used together with the input. The input and state activations are multiplied as indicated by the lines connecting with the crossed circles. The products are fed through a fully connected weight layer. Biases for the output and the state units are not shown.

$$\sum_i w_{k\theta i} i_i^t + w_{k\theta\theta} \quad (3)$$

$$z_k^t = f\left(\sum_{i,j} v_{kji} z_j^{t-1} i_i^t + \sum_j v_{kj\theta} z_j^{t-1} + \sum_i v_{k\theta i} i_i^t + v_{k\theta\theta}\right) \quad (4)$$

The index k is used for identifying individual output and state units, j is used for the state of the previous time step, and i is used for the current input. w is the weight set for the output units and v is the weight set for the state units. As an example of notation, w_{kji} is a second-order weight (used for connecting the product of the j th state unit and the i th input with the k th output unit), $w_{kj\theta}$ and $w_{k\theta i}$ are first-order weights (feeding the k th output with either the j th state unit or the i th input, and $w_{k\theta\theta}$ is a first-order bias (associated with the k th output unit). The logistic activation function, $f(x) = 1/(1 + e^{-x})$, is used.

All strings from the language $s_n \in \mathcal{O}^{n1^n}$ ($1 \leq n \leq 10$) were presented to the network in random order, one letter at a time, $s_n = s_n^1, s_n^2, \dots, s_n^{2n}$, concatenated into one continuous string. The network was evaluated according to its ability to predict the next letter in each string. Each output, o , is compared to the desired output, d , and if the output is correctly predicted and it was in fact a predictable output ($i = 1$) the network

receives an ‘award’. The payoff is defined as

$$c(i, o, d) = \begin{cases} 1 & \text{if } |o - d| < 0.5 \wedge i = 1 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

The network’s fitness is proportional to

$$\sum_{i \in R} \frac{\sum_{\tau=1}^{|s_i|} c(s_i^\tau, F(s_i^\tau), s_i^{\tau+1})}{|s_i|} \quad (6)$$

where R is a randomly ordered set of integers¹ between 1 and 10 with three instances of each number. When $\tau = |s_i|$, $s_i^{\tau+1}$ is 0. This means that strings of different lengths are weighted equally and the unpredictable symbols are ignored in the fitness measure.

The EA operated on real-valued genotypes which was mapped on to the weights of the networks, i.e. an individual is simply a vector $\mathbf{u} \in \mathbb{R}^m$ where m is the number of weights in the corresponding network. The population size was set to 100 individuals of which the 20 best (according to their prediction performance, stated in Equation 6) were selected as the ‘elite’ of the current generation. The elite was copied unchanged into the next generation while the remaining 80 individuals were replaced by ‘mutated’ copies of selected members of the elite. The selection, of which individuals to be reproduced in each generation, were biased towards the best members of the elite. The mutation corresponded to adding a Gaussian-distributed value to each weight. The mutation-rate σ remained fixed during the whole evolution. No crossover was used due to the so called ‘permutation’ problem (Belew et al., 1991). The population of networks evolved for a maximum of 20000 generations. The populations were initialized with weights and biases uniformly distributed in the interval $[-1, 1]$.

We refer to a network as ‘successful’ if it scores maximum fitness according to Equation 6. To evaluate the networks quantitatively we focus on three measures partly adapted from Tonkes et al. (1998): *Efficiency* – how many generations must be evaluated before a successful weight set is found, *Reliability* – how many of the populations find at least one successful weight set within k generations (here $k = 20000$), and *Generalization* – how well does the successful weight set perform on longer strings of \mathcal{O}^{n1^n} .

As can be seen in Table 1 the EA finds solutions to \mathcal{O}^{n1^n} for both the SRN and SCN reliably but does so very inefficiently due to the population size. As a crude comparison, BPTT on a similar task finds solu-

¹The set employs a new random order for each individual and generation.

Mut.	SRN	SCN	SRN	SCN	SRN	SCN
	$\sigma = 1.0$		$\sigma = 0.5$		$\sigma = 0.1$	
Rel.	95	86	95	78	31	70
Eff.	1513	1920	968	728	2661	2456
Gen.	13.9	12.3	14.9	13.5	17.4	14.6
BG	28	32	36	32	38	50

Table 1: Performance based on 100 evolutionary runs for each network type and mutation setting (Mut.). Reliability (Rel.) indicates how many of the networks that were deemed successful within 20000 generations. Efficiency (Eff.) shows the average number of generations before success. Generalization (Gen.) shows the average maximum string length managed by a successful network. The maximum string length managed by the best network (BG) is also shown.

tions within the presentation of around 3000 strings² for the SRN and 5000 strings for the SCN but with a reliability around 50% for the SRN and 20% for the SCN when the networks were “unfolded” 10 time steps. A BPTT run was deemed successful if within 10000 strings it found a weight set that solved the task, that is, achieved maximum fitness points according to Equation 6.

3 ANALYSIS

Discrete time recurrent neural networks can be understood as discrete time dynamical systems (Casey, 1996; Rodriguez et al., 1999). To characterize solutions qualitatively and to compare the solution types generated, each successful network was examined from a dynamical systems perspective.

The activation over the hidden units of the SRN and the state units of the SCN corresponds to the system’s state, X . The weights and the inputs to the network define the function, G , which transform the system from a state to the next state, $X^t = G(X^{t-1}, I^t)$, where I is the externally set input to the system. In our particular case the input can be discarded by distinguishing between two systems, G_0 and G_1 , by incorporating (into the definition of the function) the influence of the two possible inputs, 0 and 1, respectively.

In the non-linear case the behavior of the dynamical system can be determined by calculating the Jacobian

²3000 strings would translate into the equivalent of 1 generation, since each generation is evaluated by presenting $\text{population size} * \max(n) * 3 = 3000$ strings. However, tests have indicated that a fitness measure which is based on a single round of each stringlength per individual is almost as effective ($100 * 10 * 1 = 1000$ strings/generation).

(the partial derivative matrix) of the state over time (Devaney, 1989). The behavior is particularly relevant in the vicinity of the system’s fixed points. A fixed point is a point, $X = GX$, for which the state does not change over time.

The eigenvalues of the Jacobians around the fixed points indicate rates of contraction or expansion of system states (over time, along the principal axes defined by the corresponding eigenvectors) and supply us with an understanding of how the task is performed (Rodriguez et al., 1999).

To that end, we calculated the eigenvalues around the fixed points of each successful network. For comparison a large set of similar networks were adapted by BPTT and similarly analysed. To compensate for the lack of effective learning signals when the input 0 is used,³ BPTT networks were each equipped with an additional input and output unit which was set to 1 when the letter was 0 and vice versa. The state space remained 2-dimensional, providing both the EA and BPTT networks with similar resources for implementing dynamical behaviors.

Each fixed point was classified according to its eigenvalues. The classification criterion is based on critical boundaries between dynamical behaviors (bifurcations) effectively distinguishing between attractive and repelling behaviors, and 1- or 2-periodic or even rotating behaviors around the fixed point. For a two-dimensional state space there are two eigenvalues, each expressing the change of the system along one principal axis (given by the corresponding eigenvector). In the codes used below (see Table 2), the first two letters signify the behavior of the smaller eigenvalue: either attracting (A) or repelling (R), and rotating (c) or, 1- or 2-periodic. The next two letters similarly indicate the behavior of the larger eigenvalue (see Table 2).

It should be noted that if one eigenvalue is complex (c), first, it is always accompanied by its conjugate thus we only state the behavior of the first eigenvalue. Second, it suggests a rotation around the fixed point. Tables 3 and 4 contain statistics on classifications of the dynamical behaviors employed by the networks.

An analysis of trained SRNs shows that they implement the task in two different ways (cf. Bodén et al., 1999; Rodriguez et al., 1999): (1) By first oscillating towards a fixed point (often using A2A2) while 0 is presented and then, after a shift to the part of the state

³Weight adjustments in backpropagation are proportional to the product of the error of the receiving unit and the output of the sending unit. Thus, when 0 is used as input the associated weight is not adjusted.

Code	Eigenvalues	EV 1	EV 2
R2A2	$\lambda_1 < -1 < \lambda_2 < 0$	Repelling	Attracting
R2A1	$\lambda_1 < -1 < 0 < \lambda_2 < 1$	Repelling	Attracting
R2R1	$\lambda_1 < -1 < 1 < \lambda_2$	Repelling	Repelling
A2A2	$-1 < \lambda_1, \lambda_2 < 0$	Attracting	Attracting
A2A1	$-1 < \lambda_1 < 0 < \lambda_2 < 1$	Attracting	Attracting
A2R1	$-1 < \lambda_1 < 0 < 1 < \lambda_2$	Attracting	Repelling
A1A1	$0 < \lambda_1, \lambda_2 < 1$	Attracting	Attracting
A1R1	$0 < \lambda_1 < 1 < \lambda_2$	Attracting	Repelling
R1R1	$1 < \lambda_1, \lambda_2$	Repelling	Repelling
Ac	$imag(\lambda_{1,2}) \neq 0, \lambda_{1,2} < 1$	Attracting	Attracting
Rc	$imag(\lambda_{1,2}) \neq 0, \lambda_{1,2} > 1$	Repelling	Repelling

Table 2: Observable behaviors along two eigenvectors (EV 1 and EV 2) defined by the eigenvalues (λ_1 and λ_2) to which they belong.

	BPTT		EA	
	SRN	SCN	SRN	SCN
R2A2	0 (0)	0 (0)	0 (0)	0 (1)
R2A1	0 (0)	0 (0)	0 (0)	0 (1)
A2A2	96 (96)	26 (20)	84 (82)	39 (34)
A2A1	0 (0)	58 (69)	6 (7)	53 (58)
A1A1	0 (0)	2 (2)	8 (9)	6 (5)
A1R1	0 (0)	0 (0)	0 (0)	0 (0)
Ac	3 (3)	9 (6)	2 (3)	2 (1)
Rc	0 (0)	4 (2)	0 (0)	0 (0)

Table 3: The distribution (in percent) of dynamical behaviors around the main fixed point of each G_0 system (constant 0-input). The classification for all fixed points found are given within parentheses. At least 150 successful networks were used to calculate each value.

	BPTT		EA	
	SRN	SCN	SRN	SCN
R2A2	100 (100)	62 (53)	84 (79)	29 (21)
R2A1	0 (0)	23 (17)	6 (6)	43 (33)
R2R1	0 (0)	0 (0)	0 (0)	2 (1)
A2A2	0 (0)	0 (7)	0 (1)	0 (4)
A2A1	0 (0)	0 (11)	0 (2)	12 (26)
A2R1	0 (0)	0 (0)	0 (0)	5 (5)
A1A1	0 (0)	2 (3)	9 (10)	9 (7)
A1R1	0 (0)	0 (0)	0 (1)	1 (2)
Ac	0 (0)	0 (0)	0 (0)	0 (0)
Rc	0 (0)	13 (9)	0 (0)	0 (0)

Table 4: The distribution (in percent) of dynamical behaviors for the networks for G_1 (1 is presented continuously). The classification for all fixed points found are given within parentheses. At least 150 successful networks were used to calculate each value.

space predicting 1, oscillating from a fixed point (often using R2A2), with a starting point determined by the shift, until the state reaches the part of the state space which predicts 0 again (see Figure 2). (2) By monotonic state-changes for each presented 0 relative to a fixed point (typically A1A1) and then monotonic state-changes in the opposite direction for each presented 1 relative a fixed point (typically A1A1) (for a slightly untypical example see Figure 6).

An analysis of trained SCNs revealed that some of them employ a third group of behaviors namely entangled spiralling: By rotating towards a fixed point (typically Ac) while 0 is presented and while 1 is presented rotating from another fixed point (typically Rc, close to the first fixed point) in the reversed direction, until the prediction of 0 is triggered (see Figure 3).

Bodén et al. (1999) confirmed that with BPTT, SRNs made exclusive use of the oscillating solution (1) if the networks were required to generalize beyond the training set. Rodriguez et al. (1999) identified conditions, in terms of the eigenvalues and eigenvectors, for this particular solution. The first condition is that 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 G_0 matches the rate of expansion around the saddle point of G_1 (Rodriguez et al., 1999, p. 23).

The second condition is that the fixed point of G_0 should lie on the axis given by the eigenvector corresponding to the smallest absolute eigenvalue of G_1 (the direction in which the saddle point attracts) when projected through the G_1 fixed point (Rodriguez et al., 1999, p. 23). This configuration basically entails that the first thing that happens in G_1 is a long-distance state shift along its eigenvector to a part in state space close to the fixed point of G_1 . The positioning of the eigenvector ensures that the final state of G_0 (which identifies the 0-count) correctly affects the starting point for the expansion in G_1 . The small eigenvalue ensures an instantaneous transition (see Figure 2).

The weight sets adapted by the EA are more varied than those adapted by BPTT. With BPTT almost all SRNs made use of A2A2 for counting up *as* (see Figure 2). With EA a majority of the SRNs employed A2A2 but also A2A1 and A1A1 were used, exploiting the attracting direction to implement solution (2) above. For counting down the 1s, BPTT adapted SRNs to use R2A2 exclusively. EA managed to employ R2A1 and A1A1 as well for the same task. For SCNs BPTT predominantly used R2A2 and R2A1 for

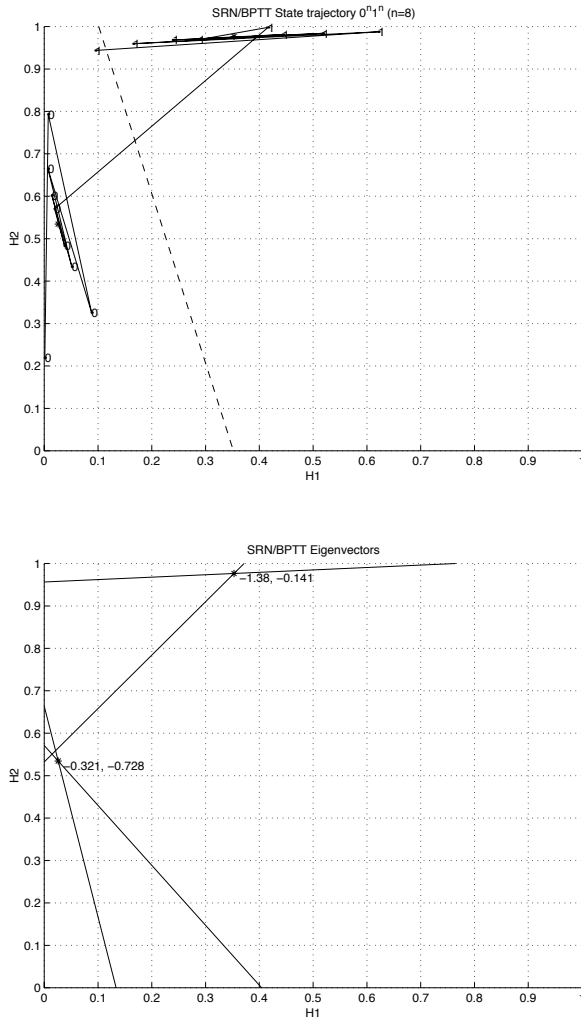


Figure 2: (1) A typical state trajectory of SRN/BPTT utilizing A2A2 and R2A2 fixed points. The dashed line is the decision boundary implemented by the output weights separating the states which predict 0 from those that predict 1. (2) The eigenvectors projected through the corresponding fixed points.

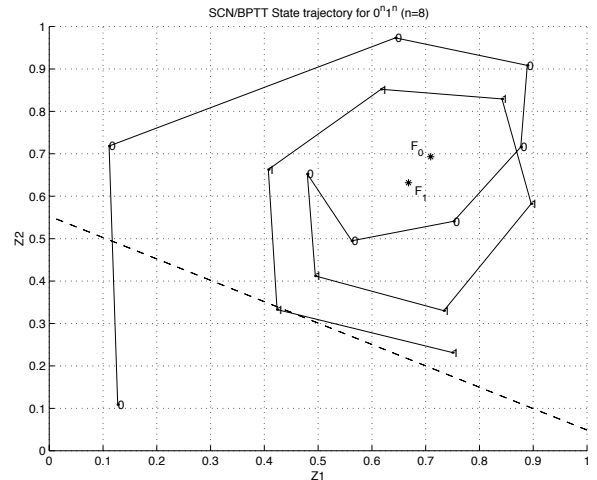


Figure 3: A spiralling state trajectory of SCN/BPTT utilizing Ac and Rc fixed points ($G_0: \lambda_0 = 0.52 \pm 0.85i$ and $G_1: \lambda_1 = 0.58 \pm 1.01i$). The dashed line is the decision boundary implemented by the output weights in state space separating the states that predict 0 from those that predict 1. Note that according to Equation 3 the state is delayed in the SCN (in contrast to the SRN), thus, it is the second to last 1-state (found at $(0.42, 0.33)$) that predicts the first 0 of the next string.

counting down 1s, whereas EA found A2A1, A2R1 and A1A1 fixed points as well. However, the entangled spiralling counting behavior, which requires that both G_0 and G_1 implement rotation, was only found with BPTT (see Figure 3). The few Ac fixed points found by the EA only implemented slight rotation (similar to A2A2 oscillation) around the main fixed point of G_0 .⁴

Judging from the eigenvalues a majority of the SRNs found with the EA implement damped oscillation using A2A2 and R2A2. However, in a number of cases the state trajectories appear to visit an intermediate state (“an elbow”) (cf. Tonkes et al., 1998) before beginning the outward oscillation (see Figure 4). At a closer look it turns out that the eigenvectors are not aligned in accordance with the conditions identified by Rodriguez et al. (1999). Still, the variation allows for proper generalization.

Notably, EA makes use of more fixed points in the solutions (for an example see Figure 6). For SRNs adapted by BPTT only one fixed point was found in each system. With EA 6% additional fixed points were

⁴The main fixed point is the fixed point which is primarily responsible for the state changes in either G_0 or G_1 when processing $0^n 1^n$. It is identified by visual inspection.

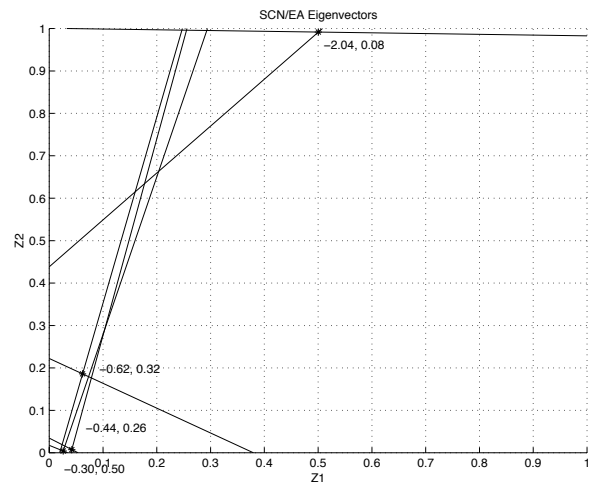
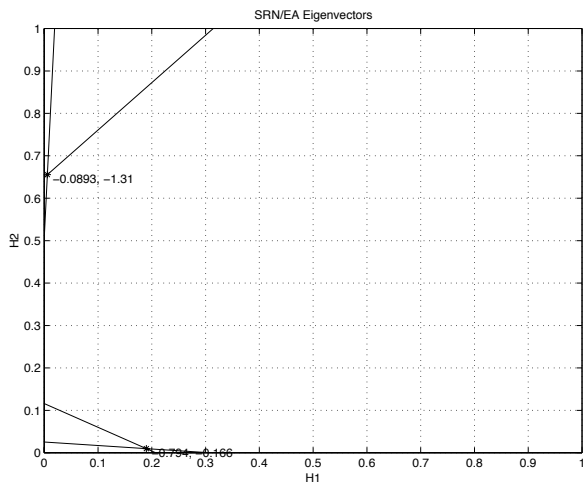
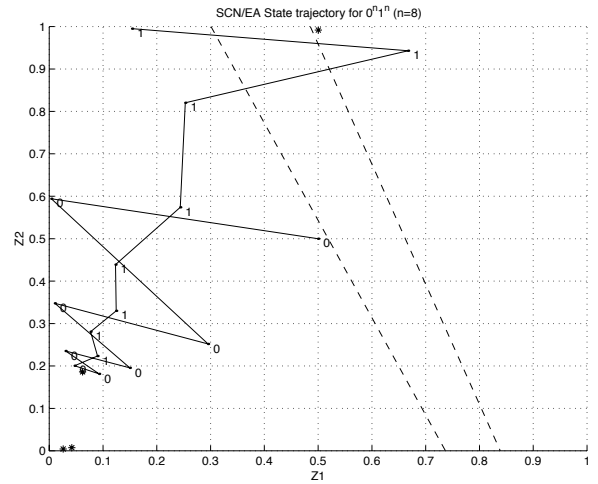
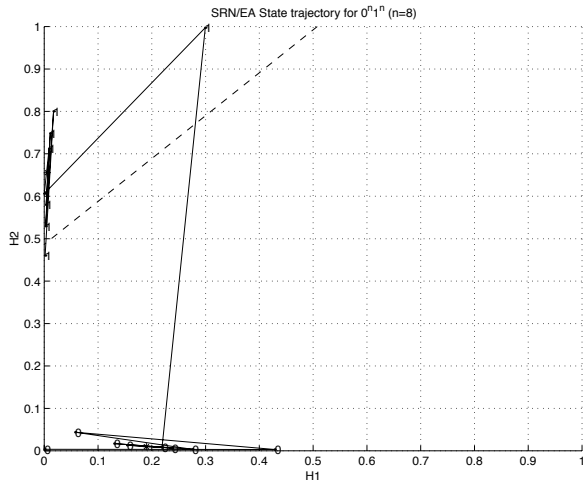


Figure 4: (1) A SRN/EA state trajectory utilizing an A2A2 and an R2A2 fixed point with an “elbow”. (2) The eigenvectors projected through the corresponding fixed points.

Figure 5: (1) A SCN/EA state trajectory utilizing multiple fixed points, two A2A1’s in G_0 and an R2A1 and an A2A1 in G_1 . (2) The eigenvectors projected through the corresponding fixed points.

found each with its own behavior. For SCNs adapted with BPTT 57% additional fixed points were used. With EA the additional fixed points increased to 67%. The multitude of fixed point enforces hybridization between solution types (see Figure 6).

The EA finds solutions which are less compact in state space, and even though they strongly resemble solutions found by BPTT the networks are less conformant to the conditions identified for them. For example, the damped oscillation implemented by the network in Figure 5 is stretched out over state space and the eigenvalues and eigenvectors for the main fixed points do not obey the conditions identified by Rodriguez et al. (1999). Instead there is often an additional group of fixed points (see lower left corner in Figure 5) which support the trajectory.

A fourth, previously not identified, solution type was found by the EA in a few cases (see Figure 7). The solution resembles the oscillating solution and works exactly the same during G_0 . G_1 , however, has a fixed point close to the G_0 fixed point with an R2R1 behavior and “pushes” (by repelling) the states towards the opposite limit border ($z_2 = 1$) while oscillating outwards.

4 CONCLUSIONS

It is difficult for neural networks to learn CFLs. However, natural language does contain CFL-constructs, suggesting that humans have indeed the necessary prerequisites but with limited precision (cf. Christiansen and Chater, 1999). This work is one step towards resolving if there can be a simple language mechanism for handling CFLs (based on neural networks) and, if so, how, and how well it can be established through evolutionary algorithms compared to conventional learning.

It has been demonstrated in this paper that EAs can, in contrast to BPTT, reliably adapt small recurrent networks to predict a simple CFL. The efficiency by which this is done is considerably lower than with BPTT. However, the EA is not restricted to follow an error gradient and attempts, sometimes successfully, to solve the problem in ways which are seemingly hidden by error gradients. The more varied repertoire of dynamical behaviors (confirmed by greater diversity of dynamics around fixed points, observed differences in state trajectories, and through the violation of some conditions known to hold for BPTT networks) and more available fixed points are two indicators of this that have been put forward here. The results suggest that EAs may supply better means for diversification

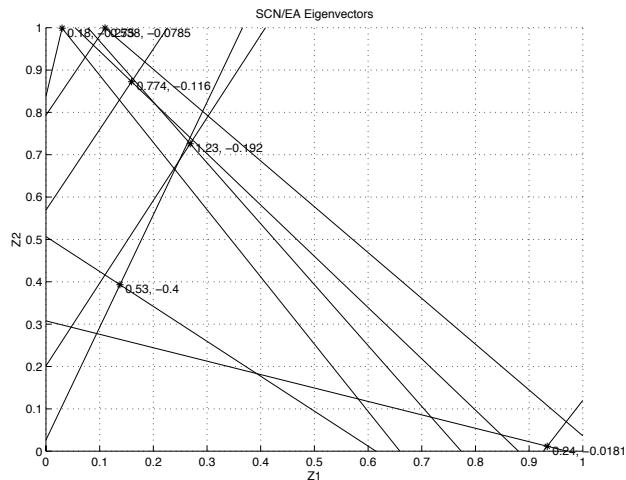
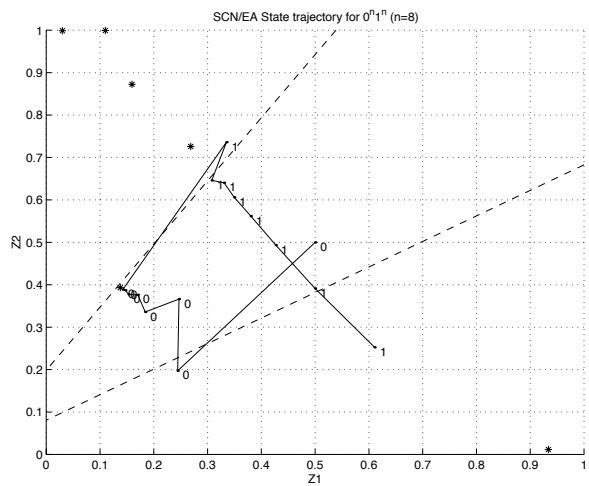


Figure 6: (1) A SCN/EA state trajectory utilizing multiple and hybrid fixed points to “steer” activations so as to count “monotonically” with a slight tendency of oscillation. (2) The eigenvectors projected through the corresponding fixed points.

in multi-net systems compared with BPTT.

Acknowledgments

This work was supported by an ARC grant to MB and a KK-Foundation grant to HJ and TZ.

References

- Batali, J. (1994). Innate biases and critical periods: Combining evolution and learning in the acquisition of syntax. In *Proceedings of the Alife IV Workshop*. Cambridge, MA: MIT Press.
- Belew, R. K., McInerney, J., Schraudolph, N. (1991). Evolving networks: using genetic algorithms with connectionist learning. In *Proceedings of Second Artificial Life Conference*, 511–547. New York: Addison-Wesley.
- 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 Int. Conference on Artificial Neural Networks*, 359–364. 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.
- Christiansen, M. and Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157–205.
- 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.
- Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227.
- Rodriguez, P., Wiles, J., and Elman, J. L. (1999). A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40.
- Tonkes, B., Blair, A., and Wiles, J. (1998). Inductive bias in context-free language learning. In *Proceedings of the 9th Australian Conference on Neural Networks*, 52–56.
- 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 17th Annual Meeting of the Cognitive Science Society*, 482–487. Hillsdale, NJ: Lawrence Erlbaum.

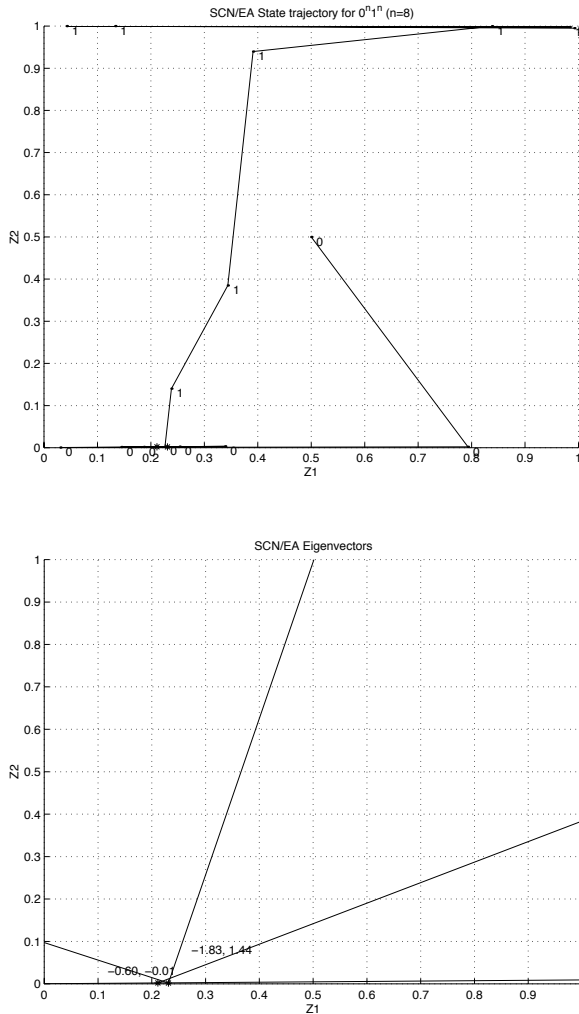


Figure 7: (1) A SCN/EA state trajectory utilizing an A2A2 fixed point in G_0 and an R2R1 fixed point in G_1 to “push” the activation towards an axis (along which oscillation occurs). (2) The eigenvectors projected through the corresponding fixed points.