Learning to predict a context-free language: Analysis of dynamics in recurrent hidden units

Mikael Bodén*, Janet Wiles*†, Bradley Tonkes* and Alan Blair*
*Department of Computer Science and Electrical Engineering
†School of Psychology
University of Queensland, 4072
Australia

Abstract

Recurrent neural network processing of regular languages is reasonably well understood. Recent work has examined the less familiar question of context-free languages. Previous results regarding the language $a^n b^n$ suggest that while it is possible for a small recurrent network to process context-free languages, learning them is difficult. This paper considers the reasons underlying this difficulty by considering the relationship between the dynamics of the network and weightspace. We are able to show that the dynamics required for the solution lie in a region of weightspace close to a bifurcation point where small changes in weights may result in radically different network behaviour. Furthermore, we show that the error gradient information in this region is highly irregular. We conclude that any gradient-based learning method will experience difficulty in learning the language due to the nature of the space, and that a more promising approach to improving learning performance may be to make weight changes in a non-independent manner.

1 Introduction

Recurrent neural networks (RNN) can be trained to recognize regular languages from examples (Cleeremans et al., 1989; Elman, 1990; Pollack, 1991; Giles et al., 1992). The operation of such RNNs has commonly been understood in terms of finite-state automata (FSA). States organize in activation space as distinct clusters and weights establish transformations between them reflecting the operation of the associated FSA (Casey, 1996). It has been argued that this discretization is misleading and that the operation of RNNs is better understood in terms of iterated function systems (Kolen, 1994) or, more generally, continuous dynamical systems (Rodriguez et al., 1999).

This work considers RNNs trained with a context-free language (CFL). CFLs cannot be processed with FSA and thus any solution requires a different understanding of RNN dynamics. The conventional extension is a push-down automaton (PDA) which adds a stack and a counting mechanism to the FSA. Previous work has demonstrated that an RNN, can be successfully trained on a simple CFL, without making use of an explicit counter or stack. Instead, hidden units develop oscillating dynamics which provide means for a potentially infinite number of states (Wiles and Elman, 1995; Rodriguez et al., 1999; Tonkes and Wiles, in press). However, learning does not always result in a solution and when a solution is found the network is prone to losing it with further training (Tonkes and Wiles, in press).

This paper extends previous work by investigating two particular aspects of network performance in light of simulations using the context-free language $a^n b^n$:

- What constitutes the learned (or learnable) solution? What are the constraints, variations and limits of the network learning?
- Why is learning difficult and unstable?

2 Experiments

All networks consisted of 2 input units (one for each token), 2 hidden units and 2 output units (one for each token). The network was fully connected and the hidden units were recurrent, as shown in Figure 1.

A variety of networks were trained using backpropagation through time (BPTT) on the context-free language $a^n b^n$, e.g. *aaabbb*, *ab*, *aaaaaabbbbbb*. The language was presented as a continuous stream of strings with varying lengths up to $n = 10$. The target output was the next token in the string or, at the last token, the first token of the next string. Since strings were presented in random order, this prediction task (originally

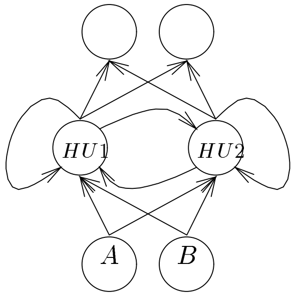

Figure 1: The network used in all experiments. Each token has its designated input and output unit. The hidden units are recurrent.

used by Elman, 1990) is non-deterministic. However, the network can develop mechanisms for deterministically predicting the next token whenever the b token is presented. The network weights were updated after each completely presented string. Generalization was tested up to $n = 12$. Generalization requires that the network has established a means for counting the number of a's to predict the same number of b's. The two tokens, a and b, were represented with [1 0] and [0 1] respectively.

Each network was unique and had either different initial weights or was configured with different learning parameters including learning rate (fixed at 0.3 (FLR) or an adaptive strategy (ALR) described by Lawrence et al., 1998), number of activation copies saved for BPTT (ranging from 5 to 12), target codes (binary (BT): [1 0][0 1] or soft (ST): [0.9 0.1][0.1 0.9]). These variations allowed us to study the impact of prior constraints. The logistic output function was used for all networks. No momentum was used.

The percentage of networks finding a solution (correctly handling all strings up to $n = 12$) within the presentation of 20000 strings was 60% for the optimal parameter settings and around 20% on average. The success rate was considerably worse when the number of activation copies for BPTT was kept low (5 or below). The data distribution was biased towards shorter strings with the highest frequency for $n = 2$. Some alternative learning and data presentation strategies – a smaller learning rate for the hidden layer weights (SLRH), a presentation scheme where longer strings were introduced after some learning period (StS), and a presentation scheme which only contained strings with maximum length equal to the level of BPTT unfolding (ShS) were tested, but demonstrated no significant performance advantage. Table 1 summarizes the results.

Config.:	BPTT unfolding:							
	5	6	7	8	9	10	11	12
BT/FLR	9	13	38	15	23	21	34	12
ST/FLR	0	4	0	9	19	53	36	6
BT/ALR	0	0	0	2	23	11	23	0
ST/ALR	0	0	0	6	0	13	6	0
SLRH						15		
StS						28		
ShS	0	6	28	60	43	21		

Table 1: Success rates (percentage) for network learning with different configurations. Each configuration was tested with a population of 47 networks.

3 Solution

About 200 successful weightsets (from different networks) were saved for further analysis. All successfully generalizing networks made use of oscillating hidden units to keep track of the level of embedding. Cluster analysis (in which each weight was compared to all other weights in the same position) revealed eight major clusters. As it turned out, these clusters corresponded to the eight symmetries of the dihedral group acting on weight-space. Consequently, each network was transformed to a canonical form described below.

The main solution, which has been described in previous work by Wiles and Elman (1995), relies on one hidden unit (HU1 in the canonical representation) to oscillate in synchrony with presentation of the a token and the other hidden unit (HU2) to oscillate in synchrony with the b token. The first hidden unit implements a 2-periodic oscillator, which slowly *converges* to a fixed point in activation space. The second hidden unit implements a 2-periodic oscillator, which *diverges* from an unstable fixed point to a fixed cycle in activation space. The number of oscillations performed by the first hidden unit effectively determines the starting point and the stepsize for the second oscillation. The second hidden unit approaches a constant threshold value (an activation which basically marks the end of the string) from different starting points and with different stepsizes. Figure 2 shows a hidden activation trajectory and decision thresholds of the output units for a standard solution in canonical form.

4 Analysis

Observation of the hidden unit activations reveals the dynamics of the network. During continuous presentation of a single token, most change occurs in one hidden unit, and

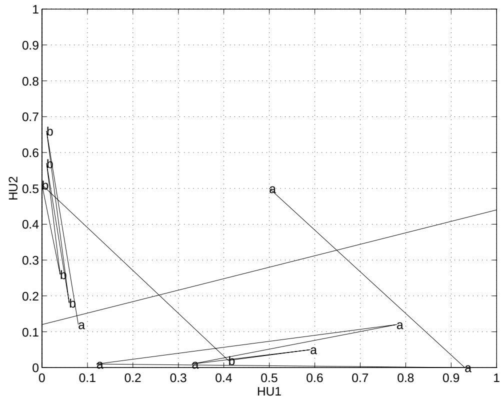

Figure 2: A typical hidden activation trajectory for processing the string *aaaaaabbbbbb* ($n = 6$, starting point [0.5 0.5]). The line forms the output hyperplane. Note when the last b is presented the activation ends up in the "predicting a" region of the decision thresholds (at 0.5 for the logistic function) implemented by the output units.

the other remains largely inactive. Thus, we will analyse the network behaviour by considering the simplified case of each hidden unit in isolation with only a bias and a selfweight. If we consider this single unit under constant input, then we can subsume any inputs under the bias term. However, it should be noted that some communication between the hidden units is necessary to set the starting point, x_0, for each phase of processing the continuous stream of strings.

4.1 Dynamical Behaviour

There are four basic behaviours exhibited by the single recurrent unit (Hölldobler et al., 1997). We assume the logistic activation function resulting in the iterated map, $f(x) = 1/(1 + e^{-wx-b})$ (selfweight w and bias b) which has at most 3 fixed points (where $f(x) = x$). Let x_i be the fixed point which has the largest output gradient $f'(x_i)$.

1. The selfweight is positive.
   (a) If $0 < f'(x_i) < 1$, there is one attractive fixed point to which the unit output eventually converges.
   (b) If $f'(x_i) > 1$, then two attractors and one repeller result. The outcome depends on the initial point, x_0.
2. The selfweight is negative.
   (a) If $-1 < f'(x_i) < 0$, there is one attractive fixed point to which the unit output converges by damped oscillations.
   (b) If $f'(x_i) < -1$, then the activations converge towards a fixed 2-periodic cycle.

The standard solution outlined earlier makes exclusive use of behaviour 2(a) in HU1 and behaviour 2(b) in HU2. Solutions exist using behaviours 1(a) and 1(b) (Hölldobler et al., 1997) but such networks have not been observed to learn and successfully generalize as a result of training with BPTT. To illustrate the difficulties for the learning algorithm we focus on behaviours 2(a) and 2(b).

To process longer strings, the network must fit as many oscillations as possible into the hidden unit space before converging to an attractive point or cycle. Figure 3 depicts the number of iterations, k, of the single hidden unit iterated map before $|f^{k-t}(x) - f^k(x)| < \epsilon, \epsilon = 0.001$ for varying bias and selfweight when $x_0 = 0.5$. The figure shows an ellipsoidal ridge where many oscillations can be made before convergence. Importantly, this ridge also forms a border between behaviours 2(a) (outside) and 2(b) (inside) above. Crossing the border results in a bifurcation in the dynamics of the network and a radically different outcome.

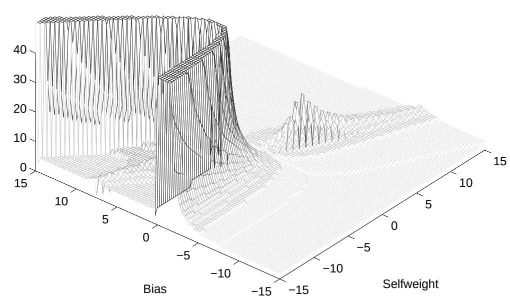

Figure 3: Number of oscillations before convergence for a self-recurrent single hidden unit. The number of oscillations was cut off at 50 for clearer visualization. Behaviour 2(a) is found outside the ridge, behaviour 2(b) is found inside the ridge.

In terms of the network's solution for $a^n b^n$ for $n \leq 12$, HU1 (which oscillates in synchrony with a input) must be close to this ridge and on the outside for a input, and further from it for b input. Conversely, HU2 must be close to the ridge and on the inside for b input, and further from it for a input. The external signals from the a and b inputs are able to facilitate this change in oscillation performance by shifting the network along the bias axis.[1] Due to the nature of the surface in figure 3 the translation of the effective bias must be performed with substantial precision. The unit must be moved to a region closer to the border to achieve the required oscillation performance, but not so far as to send it over the border which would result in crossing the bifurcation boundary of the unit's dynamics.

The situation is further complicated when we consider the recurrent connections between the hidden units. These connections allow the network finer grained control over the transition between the a and b phases by setting the starting conditions for HU2.

Figure 4 shows where the hidden units of successful networks fall in terms of the landscape in figure 3. The absolute weight values have been modified to incorporate the influence of the corresponding input weight for the two hidden units relative to the two input cases (a input for HU1, b for HU2). The weights for the first hidden unit are found outside the bifurcation border and the weights for the second hidden unit are found inside the border. The figure is an idealization of the condition where HU1 and HU2 are independent and outliers in the figure are weight sets that violate this assumption.

5 The Error Surface and Learning

It is clear that the representation requires some degree of precision, but what makes learning so difficult and unstable?

Weight changes were traced during learning for a number of trials. Again the network was analysed by considering two separate self recurrent units, with their respective biases accounting for the appropriate input condition. Typically, the network weights evolve in three main phases. Initially, weights smoothly migrate towards the region of good oscillation performance. When the weights reach a region close to the bifurcation border, updates become highly irregular and weights tend to fluctuate. Finally, at some point the weights are changed to such a degree that the network moves out of the desirable regions. In all the studied cases, large concurrent changes in the bias and the input weight caused the problem. The selfweight appeared reasonably stable. The same behaviour was observed for runs when the BPTT unfolding memory was kept above 8. For networks trained with more copies (up to 12) the network managed to stay in the proximity of solution space longer. For networks trained with fewer copies, the second phase showed more consistent weight changes but found solutions less frequently.

[1] Recall that under constant input it is safe to subsume the input values under the bias term.

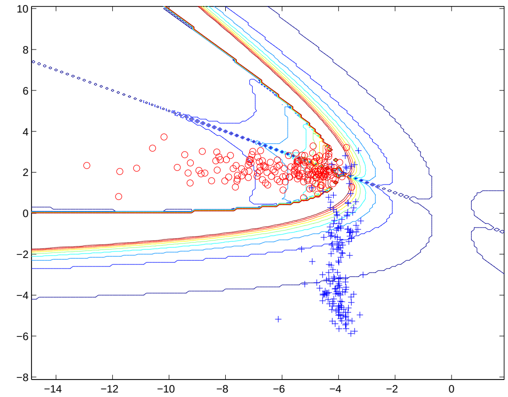

Figure 4: Plot of weights for the two hidden units combined with a contour plot for the oscillation performance. Weights for the first hidden unit are found outside the bifurcation border. The weights for the second unit are found within. Weight values are calculated on the basis of networks in canonical form: the selfweight for each hidden unit is unchanged, the bias is the sum of the original bias plus the input weight from the active unit (the a input for HU1, the b input for HU2).

The error that the learning algorithm minimizes is based on the difference between the presented strings and what is predicted by the network. Since weights are updated after each presented string and since strings of different lengths impose different requirements on the weight sets, the error may fluctuate as a result of presenting consecutive strings of dramatically varying lengths. However, in a separate analysis the observed weight changes did not correlate with difference in length for consecutive strings.

To investigate the nature of the error surface, we considered the error gradient computed by BPTT for a family of weights. To reflect the unstable region in weightspace around the solution, weights were taken from a successfully generalizing network. We then considered for each hidden unit separately, the error gradient for varying values of selfweight and (effective) bias. To ensure that any possible influence from string length did not affect the result, the error was calculated on the basis of the entire range of strings in the training set ($n = 1..10$). A

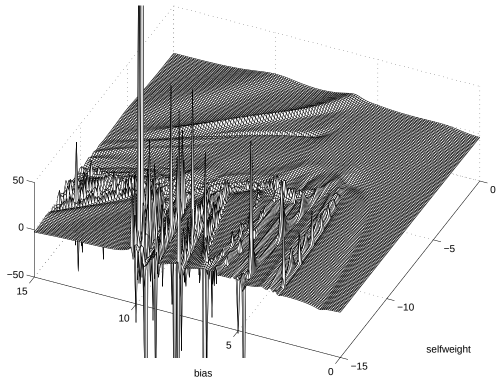

Figure 5: Error gradients for the second hidden bias in a successful network when the selfweight and bias of the second hidden unit are varied. In the region of interest, the error gradient is extremely unstable.

representative sample of the gradients can be seen in Figure 5. The gradients indicate that the error surface is littered with deep narrow potholes (in terms of both magnitude and direction) close to the bifurcation border. Thus, if the weight changes are proportional to the magnitude of the gradient (as in backpropagation) extreme weight changes occur. We also noted by computing gradients for different sets of strings that the complexity of the surface is higher when longer strings are used. This difference may be a result of the proximity of the solution to the bifurcation border.

A more specific reason for the instability can be found in the recurrent weight from the first hidden unit to the second. This connection is largely responsible for the transition between the a phase and the b phase. A correlation analysis of the set of successful networks revealed a strong positive relationship between the weight values found on the connection from the a input unit to HU1 and on the connection from HU1 to HU2 and, negatively, from HU2 to HU1. By studies of weight changes and experimentation we found that by only slightly changing the weight from the first to the second hidden unit the starting conditions for the second oscillation were greatly affected. We traced the effect of learning signals on a successfully generalizing network in the proposed canonical form. The impact of the learning signal affects mainly the stepsize, not the actual starting point for the second oscillation. By only adjusting the weight on the connection from the first hidden unit to the second the oscillation stepsize (not the starting point) for the second hidden unit was affected. A positive change led to smaller oscillations for the odd-numbered strings, and larger for the even-numbered. A negative change led to the opposite. The observation is related to the correlation we found. In fact, by manually adjusting these three weights according to their relationship (a positive a Input-HU1, requires a positive HU1-HU2 weight and a negative HU2-HU1 weight, and vice versa) the learning instability was greatly reduced in a test network.

6 Conclusions

Compared to regular languages, context-free languages put radically different requirements on recurrent neural networks. It is no longer sufficient to support representation of a finite set of states in which all inputs can be grouped. Instead mechanisms for supporting representation of infinitely many states are required. Classical systems and some neural network systems resort to external counters and stacks. This work investigates a learning approach which requires no such manually designed modules. Instead a simple recurrent neural network establishes oscillating dynamics which have the potential to represent and process infinite states.

By extensive experimentation we have shown that, empirically, all successfully generalizing networks implement essentially the same solution. Furthermore, we were able to demonstrate that the difficulties experienced by BPTT in finding and keeping this solution were largely consistent across a wide variety of training conditions. We observed that performance deteriorated when we only unfolded the network for a few time steps. Optimal performance was achieved when we unfolded the network for about as many time steps as there were levels of embedding. It seems reasonable to believe that BPTT can only find the oscillating solution when the network is sufficiently unfolded. Thus, a simple recurrent network as originally employed by Elman (1990) should not be capable of learning the oscillating solution for predicting $a^n b^n$ without additional constraints.

The oscillating dynamics found by all generalizing networks can only be found in certain weight regions. One way of understanding these weight regions has been to consider the number of oscillations by the decoupled recurrent units before convergence. For the logistic function the map distinguishes between convergent and divergent oscillatory behaviour by an infinitely thin border. The standard solution requires that one hidden unit employs convergent behaviour and that the other employs matching divergent behaviour. During learning the network dynamics undergo a bifurcation when the border is crossed, making gradient-based learning difficult. The network output becomes radically different which greatly affects the error when weights cross the border. In addition, we have shown that the error landscape, which controls the network weight changes, is extremely complex (steep and irregular) close to the bifurcation border. The oscillation map also demonstrates that the desired dynamics are only found close to the border.

The problem with learning then, does not appear to be of finding a better learning algorithm that works in the same weight space. Figure 5 highlights the complex nature of the error surface on which, it would appear, any gradient based method seems likely to experience difficulty. A more promising approach, and one which we are currently investigating, is to consider an alternative search space. This study provides the basis for developing a learning scheme which takes into account the observed dependencies between critical weights responsible for the unstable learning dynamics.

Acknowledgments

This work was partly funded by an ARC grant to Mikael Bodén and Janet Wiles, a UQ Postdoctoral Fellowship to Alan Blair and an APA to Bradley Tonkes.

References

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.

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

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

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z. and Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):380.

Hölldobler, S., Kalinke, Y., and Lehmann, H. (1997). Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of KI-97: Advances in Artificial Intelligence*, pages 313–324. Springer.

Kolen, J. (1994). Fool's gold: Extracting finite state machines from recurrent network dynamics. In *Advances in Neural Information Processing Systems 6*.

Lawrence, S., Giles, C. L. and Fong, S. (1998) Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering* (to appear).

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

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

Tonkes, B., and Wiles, J. (in press) Learning a context-free task with a recurrent neural network: An analysis of stability In *Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*.

Wiles, J. and Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the Seventeenth Annual Meeting of the Cognitive Science Society*, pages 482–487. Lawrence Erlbaum.