Software for Analysing Recurrent Neural Nets That Learn to Predict Non-regular Languages

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

[1] School of Electrical Engineering & Computer Science
The University of Newcastle, Callaghan, 2308, Australia
phone: +61 2 4921 6034, fax: +61 2 4921 6929
`www.cs.newcastle.edu.au/~chalup`
`chalup@cs.newcastle.edu.au`

[2] School of Computer Science & Engineering
The University of New South Wales, Sydney, 2052, Australia
`blair@cse.unsw.edu.au`

Abstract. Training first-order recurrent neural networks to predict symbol sequences from context-free or context-sensitive languages is known as a hard task. A prototype software system has been implemented that can train these networks and evaluate performance after training. A special version of the (1+1)–ES algorithm is employed that allows both incremental and non-incremental training. The system provides advanced analysis tools that take not only the final solution but the whole sequence of intermediate solutions into account. For each of these solutions a qualitative analysis of hidden unit activity and a quantitative evaluation of generalisation ability can be performed.

1 Introduction

Most studies on training first order recurrent nets to predict non-regular languages report long training times and a low success rate (cf., [Wiles et al., 2001]). The software system described in the present paper is based on the code that was written for the study of [Chalup & Blair, 1999] in which for the first time first-order recurrent neural networks were able to learn to predict subsets of a context-sensitive language. More details about the algorithms and refined experimental results that were obtained with the extended, current version of the system have been included in [Chalup & Blair, 2002]. Compared with feed-forward neural networks, recurrent neural networks typically take longer to train and they are also prone to losing a solution again once it has been found. Recurrent neural networks can be analysed within the framework of dynamical systems.

2 Description of the Software System

The system consists of three main components for the following tasks: (1) training of the networks, (2) evaluation of the trained networks' generalisation ability,

(3) analysis of hidden unit activity. The first two tasks have a very high time complexity and the software has been implemented in the programming language *C* which can be compiled and run on most supercomputers. The third component does not have such high speed requirements and is implemented in MATLAB where it employs visualisation and animation methods. We now address the three components of the system in more detail.

2.1 Training

The system allows for a large variety of network topologies. Each net is represented by a set of connectivity matrices. For example, the well-known simple recurrent networks (SRNs) of [Elman, 1990] have feed-forward and copy-back connections which are active at two consecutive time steps. The system represents a SRN by two connectivity matrices, one for each time step. Networks active on k time steps would be represented by k connectivity matrices. The coefficients of the connectivity matrices are integers which determine whether a link is learnable, frozen or not existing. Associated with each connectivity matrix is a separate weight matrix. The weights were initialised with small random numbers.

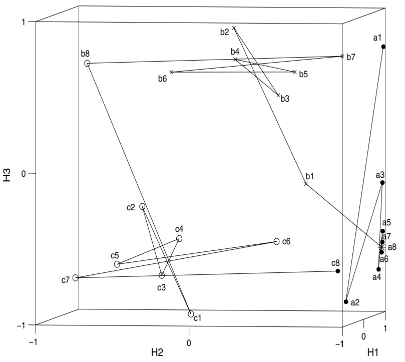
The data consisted of sequences of 30 randomly concatenated strings from one of the languages $\{a^n b^n;\, n \geq 1\}$ (context-free) or $\{a^n b^n c^n;\, n \geq 1\}$ (context-sensitive). Each symbol was encoded as a vector $a =$ (-1,1,1), $b =$ (1,-1,1) or $c =$ (1,1,-1), respectively. The task was a one-step look-ahead prediction task, that is, the symbols of the sequence are fed into the network one after the other and the network has to predict for each symbol the next symbol in the sequence. As training algorithm the (1+1)–ES of [Rechenberg, 1965] and [Schwefel, 1965] was implemented. It was combined with *data juggling* which is a method that randomly changes the order of the strings in the training sequence after each epoch during training (cf., [Chalup & Blair, 2002]).

2.2 Generalisation Tests

A neural network can generalise if it is able to process data that was not used for training. In the project of [Chalup & Blair, 2002] the networks had two different ways to generalise. One was to generalise to sequences that contained strings of larger depth n. The other possibility was to generalise to sequences in which the strings are concatenated in a different order. The software for the generalisation evaluation takes both aspects into account. Starting with a sequence consisting of low order strings the network's accuracy in processing this sequence is tested on samples of 200 permutations of the test sequence. These tests are repeated and evaluated for sequences containing strings of increasingly larger depth n up to and beyond the maximum depth that was used during training.

2.3 Analysis of Hidden Unit Activity

This component of the software can be used to analyse qualitatively how the network learns. Each network that during the process of weight evolution is able to process the training sequence with 100% accuracy is recorded in a separate file. Then for each of these networks a graph is generated that shows the trajectory of the activity of the three hidden units while the network is processing the string $a^8b^8c^8$. An example graph is plotted below, where the input symbols are encoded as $a1$-$a8$, $b1$-$b8$, $c1$-$c8$ and the output symbols are encoded as $a = \bullet$, $b = \times$, $c = \circ$ and the first b of the string is encoded as $*$.



Each training run produces 40–200 of these graphs. Each of them is a frame in one of the movie animations of our software demo. This method can be used to observe bifurcations during training and whether incremental and non-incremental training have different characteristics.

Acknowledgements

The project is supported by the APAC supercomputer grant *Large-Scale Training of Recurrent Neural Networks* and the University of Newcastle ECR grant *Machines Learn via Biologically Motivated Incremental Algorithms.*

References

[Chalup & Blair, 2002] Chalup, S. K. & Blair, A. D.: Incremental Training of First Order Recurrent Neural Networks to Predict a Context-Sensitive Language. Submitted

[Chalup & Blair, 1999] Chalup, S. & Blair, A. D.: Hill Climbing in Recurrent Neural Networks for Learning the $a^nb^nc^n$ Language, Proceedings, 6th International Conference on Neural Information Processing (ICONIP'99),(1999) 508–513

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

[Rechenberg, 1965] Rechenberg, I.: Cybernetic Solution Path of an Experimental Problem, Royal Aircraft Establishment, Library Translation No. 1122, (1965)

[Schwefel, 1965] Schwefel, H.-P.: Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungsmechanik, Diplomarbeit, Technische Universität Berlin, Hermann Föttinger Institut für Hydrodynamik, (1965)

[Wiles et al., 2001] Wiles, J., Blair, A. and Boden, M.: Representation Beyond Finite States: Alternatives to Push-Down Automata, in A Field Guide to Dynamical Recurrent Networks, Kolen, J. F. and Kremer, S. C. (eds.), , IEEE Press, (2001) 129-142