Continuous Time Recurrent Neural Networks for Grammatical Induction

Joseph Chen
Computer Science Department, University of Hamburg
Vogt-Kölln-Straße 30, D-22527 Hamburg, Germany
joseph@nats.informatik.uni-hamburg.de

Stefan Wermter
School of Computing & Information Systems
University of Sunderland
St Peter's Way, Sunderland SR6 0DD, United Kingdom
stefan.wermter@sunderland.ac.uk

Abstract

In this paper we explore continuous time recurrent networks for grammatical induction. A higher-level generating/processing scheme can be used to tackle the grammar induction problem. Experiments are performed on several types of grammars, including a family of languages known as Tomita languages and a context-free language.The system and the experiments demonstrate that continuous time recurrent networks can learn certain grammatical induction tasks.

1 Introduction

Recently research has paid a lot of attention to the development of a connectionist framework for natural language processing [1, 2, 3, 4, 5, 6]. However, these approaches are often based on discrete time neural networks. In contrast we suggest that a *continuous time recurrent network with time delay* – CTRN [7, 8], equipped with rich temporal dynamics, is suitable for such tasks.

A discrete time network encodes the information in discrete time and no internal states exist in between two *ticks*. We believe this is a shortcoming when applying discrete time networks to highly structured processing. In contrast, a continuous time neural network can "wrap" the internal structure of a symbolic sequence in continuous time.

Grammar induction is the problem to learn to classify whether a sequence of symbols should be accepted or rejected according to a specific grammar, given a finite set of positive and negative training examples. In this paper, we focus on a set of regular languages known as Tomita languages [9] and a context-free language defined on $\{0,1\}^*$. For examining this continuous approach, we show that a higher level connectionist processing module together with a continuous time recurrent network is able to perform some grammatical induction tasks.

2 Continuous Time Recurrent Neural Networks

A continuous time recurrent neural network (CTRN) [7, 8] consists of n neurons, each of which is connected with the other neurons with a time delay and whose activation evolves continuously in time. For the i-th neuron, the activation y_i can be modeled by the following equation:

$$k_i \frac{dy_i}{dt} = -y_i + \sigma\left(\sum_j w_{ij} y_j \left(t - \tau_{ij}\right) - \theta_i\right)$$

where t is time; $\sigma(\cdot)$ is a sigmoidal function; w_{ij} is the connection weight from the j-th neuron to the i-th neuron; τ_{ij} is the time delay from the j-th neuron to the i-th neuron; k_i is the time constant associated with the i-th neuron. The sigmoidal function is,

$$\tanh(x) = \frac{e^{\lambda x} - e^{-\lambda x}}{e^{\lambda x} + e^{-\lambda x}}$$

where λ is a shaping constant. Since the range of $tanh(\cdot)$ is $(-1,1)$, the activation of each neuron in this network is confined to $(-1,1)$. Now we can take a snapshot at each time t=integer, and choose a symbol mapping function $S : (-1,1)^n \rightarrow \Sigma$, where Σ is a finite set of symbols. Along the evolution of the system, a symbolic sequence is generated. In fact, a CTRN is a dynamical system, which could have very rich dynamics. Notice that the activation space $(-1,1)^n$ can be partitioned into a finite number of *symbol-partitions* through S, each of which corresponds to a unique symbol in Σ.

3 Higher Level Connectionist Processing of CTRN

Here we propose an architecture which exploits the rich dynamics of a CTRN using a *generator-processor* scheme.That is, at first the input symbol sequence is fed in a CTRN sequentially and the CTRN is trained independently to faithfully reproduce the input symbol sequence. The output of the generator is the normalized network parameter vector (that is, the normalized concatenation of weights, delays, thresholds, and time constants) of the CTRN with respect to a specific input sequence. Then the output of the CTRN can be subject to further connectionist processing.

3.1 CTRN Generator

First, the neurons in the CTRN are assigned the activation of $+1$. Then, the CTRN is set free to evolve once the input sequence has been given. The parameters of the CTRN are adjusted in such a way that a minimal difference between the output of the CTRN and the input sequence can be achieved. Specifically, given a sequence of input vector activations, each of which corresponds to a symbol in Σ, the training goal is to minimize,

$$E(\vec{p}) = \sum_{r=1}^{n} \int_{0}^{T} (o_r(t) - i_r(t))^2 \, dt$$

where $\vec{p}$ is a point in the parameter domain; n is the dimension of the CTRN; T is the length of the input sequence; $o_r(t)$ is the output of the CTRN, and $i_r(t)$ is the given input sequence. Our sampling is done by extracting the activation states whenever t is integer and ignore the intermediate states. That is,

$$E(\vec{p}) = \sum_{r=1}^{n} \int_{0}^{T} ((o_r(t) - i_r(t)) \cdot s_w(t))^2 \, dt$$

where $s_w(t)$ is a windowing-function with the observation window width w,

$$s_w(t) = \begin{cases} 1 \; if \;\; |t - floor(t)| \leq w \\ \quad 0 \; otherwise. \end{cases}$$

where $floor(t)$ is the largest integer less than or equal to t. The evolution of a CTRN can be numerically depicted with the Runge-Kutta algorithm. Since each parameter in a CTRN can have a different upper and lower bound and the evolution of the dynamic system is continuous, the training of a CTRN is difficult using gradient descend algorithms. Therefore, we use the "Adaptive Simulated Annealing" (ASA) algorithm [10, 8] to train CTRN. For further details of ASA, see Ingber 1989[10]. In our implementation, we linearly normalize the output parameter vector according to pre-specified bounds of search space for each parameter. Using this scheme, we can compress a sequence of an arbitrary length to a fixed dimensional activation.

3.2 Higher Level Grammaticality Processor

In our preliminary experiment, we use a feedforward[11] network to identify whether a given input sequence should be rejected or accepted by a specific grammar. The architecture is illustrated in figure 1.

In this architecture, the representation of a given sequence is a parameter vector of the sequence with which the CTRN traverses each symbol-partition of the symbols in the sequence. The output of the feedforward network is the decision whether the sequence is grammatical or not.

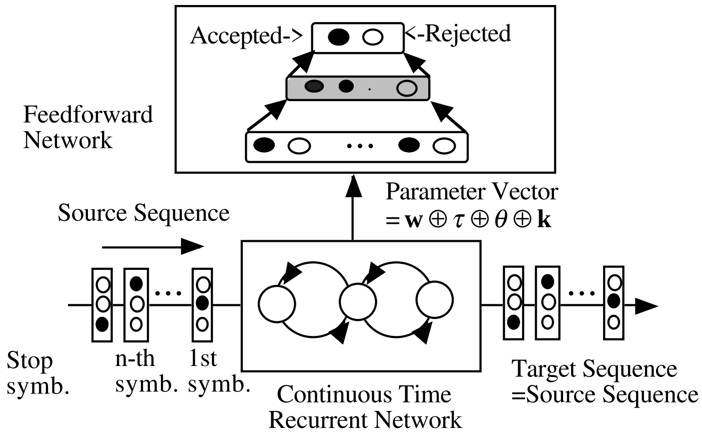

Figure 1: CTRN Grammaticality Checker.

4 Experiments

Tomita languages are a set of finite state languages over 0,1* proposed by Tomita [9] which were selected as benchmark problems by many researchers.The seven regular languages are,

L1. 1*
L2. (10)*
L3. no odd-length 0-string anywhere after an odd-length 1-string
L4. no more than two 0s in a row
L5. bit pairs, #01s + #10s = 0 mod 2
L6. abs(#1s-#0s) = 0 mod 3
L7. 0*1*0*1*

Additionally, we tested our setup on a context-free grammar on $\{0,1\}^* - 1^n0^n$ (labeled as CF below).

In our experiment, we generated each of the languages up to length 5 (62 sequences). Since CTRN encoding is a stochastic (non-deterministic) process, each sequence is presented to the CTRN ten times to construct a reservoir of parameter vectors. That is, there are totally 620 parameter vectors. These 62 sequences were then labeled as "accepted" or "rejected" according to each language. In one of the experiments, these vectors were divided into 70% (training set) and 30% (labeled as "Test1" below). In another experiment setup, the training set contains all the patterns of length up to 4 (30 sequences), and the test set consists of the sequences of length 5 (32 sequences). This setup is labeled as "Test2" below.

The activation representation for symbol "1" is $< 1, -1, -1 >$, and for "0" is $< -1, 1, -1 >$. An additional activation for the stop symbol is $< -1, -1, 1 >$. The value +1 denotes an activation of the neuron and -1, deactivation. The search space of parameters is chosen as $W_{ij} \in [-10, 10]$, $k_i \in [0.2, 5]$, $\tau_{ij} \in [0.0, 5]$, $\theta_i \in [-10, 10]$. The shaping constant of the sigmoidal function λ is 1.5. The observation window is 0.2.

The feedforward network (FF) is trained with conjugate gradient method [12]. The number of hidden units in the FF is chosen incrementally until all patterns in the training set can be learned successfully.

A typical experiment result is summarized in table 1. The base for the accuracy is the collection of all parameter vectors of a sequence. That is, if the accuracy is less than 100%, *some* parameter vectors associated with a specific sequence are not classified correctly. For example, in our first experiment setup (Test1), there are only three sequences (2 in L6, 1 in CF) whose parameter vectors (all ten) are classified incorrectly in the 70%-30% configuration.

To examine the incorrectly classified sequence further, we setup a "voting" process. That is, if more than half of the tested parameter vectors for a sequence are classified incorrectly, we label it as "incorrect", otherwise, we label it as "correct." This accuracy is shown in table 1 under the label "voting."

Upon examining the poorer performance of CTRN-FF classifier (on L5,L6, and L7), we found that all three languages are actually quite "difficult" for human subjects as well, especially when the number of training examples is not very large. However, various other grammars can be learned perfectly and do generalization quite well.

Lang.	# H. Units	Training	Test1 (Plain)	Test1 (Voting)	# H. Units	Test2 (Plain)	Test2 (Voting)
L1	1	100%	91.42%	96.43%	1	84.69%	90.62%
L2	1	100%	100.00%	100.00%	2	88.13%	100.00%
L3	3	100%	96.51%	100.00%	3	89.69%	100.00%
L4	8	100%	100.00%	100.00%	2	63.44%	56.25%
L5	9	100%	45.00%	43.75%	8	53.13%	40.63%
L6	8	100%	45.21%	60.87%	6	51.88%	56.25%
L7	4	100%	76.87%	87.50%	1	79.06%	81.25%
CF	2	100%	92.94%	94.11%	3	88.75%	96.88%

Table 1: A Typical Result of CTRN-FF Language Induction.

5 Discussion and Conclusion

Our underlying motivation in the CTRN architecture was that a CTRN can encode/generate a compressed and fixed dimensional *representation* of sequences. We expect that if the sequences have some systematic internal structure (e.g. all are grammatical according to a specific grammar), their corresponding parameter vectors in a CTRN should also have some kind of systematic characteristics.

It would be interesting to compare a CTRN architecture with a discrete time recurrent network, e.g. recurrent auto-associative memory (RAAM) [13]. In contrast to RAAM, whose "systematic" encoding/decoding ability relies solely on the inter-connection weights and has a first-in-last-out decoding scheme, CTRN uses additional parameters (time delay and time constant) in encoding a sequence and puts the results in another connectionist module. The generation phase in a CTRN is not reversed as in a RAAM but in a *phenomenally* left to right forward manner. The learning of a RAAM is carried out by the mechanism of the network itself (adjusting its weights in a distributive manner.) In contrast, the training in a CTRN can be guided and mediated by other connectionist modules.

References

[1] S. Wermter and J. Chen, "Cautious steps towards hybrid connectionist bilingual phrase alignment," in *Recent Advances in Natural Language Processing 97 (RANLP97)* (R. Mitkov, ed.), (Tsigov Chark, Bulgaria), 1997.

[2] S. Wermter, *Hybrid Connectionist Natural Language Processing.* Chapman & Hall Neural Computing Series, London: Chapman & Hall, 1995.

[3] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[4] D. Chalmers, "Syntactic transformations on distributed representations," *Connection Science*, vol. 2, no. 1&2, pp. 53–62, 1990.

[5] M. F. S. John and J. L. McClelland, "Learning and applying contextual constraints in sentence comprehension," *Artificial Intelligence*, vol. 46, pp. 217–257, 1990.

[6] L. Niklasson and T. van Gelder, "Can connectionist models exhibit non-classical structure sensitivity?," in *Proceedings of the 16^{th} Annual Conference of the Cognitive Science Society*, 1994.

[7] B. A. Pearlmutter, "Gradient calculations for dynamic recurrent neural networks: A survey," *IEEE Trans. on Neural Networks*, vol. 6, no. 5, pp. 1212–1228, 1995.

[8] B. Cohen, D. Saad, and E. Marom, "Efficient training of recurrent neural network with time delays," *Neural Networks*, vol. 10, no. 1, 1997.

[9] M. Tomita, "Dynamic construction of finite automata from exapmles using hill-climbing," in *Proceedings of Fourth Int. Cog. Sci. Conf.*, pp. 106–108, 1982.

[10] L. Ingber, "Very fast simulated re-annealing," *Mathematical and Computational Modeling*, vol. 12, pp. 967–973, 1989.

[11] D. E. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing* (Rumelhart, D. E., McClelland, and J. L., eds.), vol. 1, Foundation, pp. 318–362, MIT Press, 1986.

[12] B. Kalman and S. C. Kwasny, "Trainrec: A system for training feedforward & simple recurrent networks efficently and correctly," Technical Report, Washington University, May 1993.

[13] J. B. Pollack, "Recursive distributed representations," *Artificial Intelligence*, vol. 46, pp. 77–105, 1990.