ARTIFICIAL GRAMMAR LEARNING – A CASE STUDY OF THE REBER GRAMMAR

Peter Grenholm [1] and Karl Magnus Petersson [1, 2, 3]

[1] Cognitive Neurophysiology Research Group, Karolinska Institutet, Stockholm, Sweden

[2] F. C. Donders Centre for Cognitive Neuroimaging, University of Nijmegen, The Netherlands

[3] Centre for Intelligent Systems, Universidade do Algarve, Portugal

Correspondance:

Karl Magnus Petersson

F.C. Donders Centre for Cognitive Neuroimaging

P.O. Box 9101, NL-6500 HB Nijmegen, The Netherlands

Email: karl.magnus.petersson@fcdonders.kun.nl

Abstract

It has been suggested that language processing is an example of the 'infinite use of finite means'. A simple formal model of this idea is represented by finite-state grammars. Recent FMRI studies indicate that language related brain regions are engaged in artificial grammar processing. In the present study we investigate the Reber grammar by means of formal analysis and network simulations. We outline a method for describing network dynamics and suggest that statistical frequency-based and rule-based artificial grammar learning can be viewed as complementary in the case of the Reber grammar and similar artificial grammars.

Keywords: Artificial grammar; Learning; Information theory; Dynamical systems; Artificial neural networks.

1. Introduction

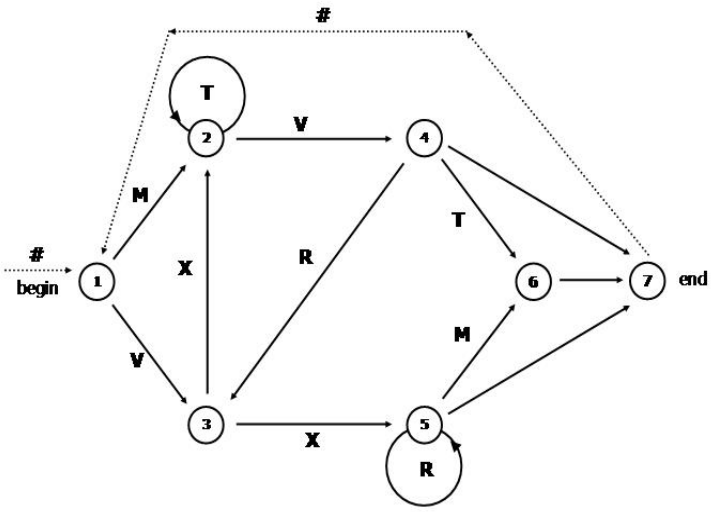
Chomsky, following von Humboldt, suggested that natural language processing is a paradigmatic example of the 'infinite use of finite means' [1]. The simplest formal model, incorporating this idea is represented by the family of right-linear phrase structure grammars which can be implemented in the finite state machines (FSMs), which also serve as language recognizers. It has been suggested that the task of learning an artificial grammar is a relevant model for aspects of language learning in infants [2] and second language learning in adults [3]. Recent FMRI studies indicate that language related brain regions are engaged in artificial grammar processing [4]. The seminal work of Reber [5] indicated that humans can learn artificial grammars in an implicit fashion and suggested that relevant structure was abstracted from the input and suggested that this process is intrinsic to natural language learning.

2. Elementary analysis

We begin by modifying the FSM of Reber by connecting the final to the initial state with a #-labeled transition (Figure 1). The modified Reber machine (RM) thus outputs an infinite symbol string: first an end-of-string symbol '#', then a Reber string, a '#', a new Reber string, and so on indefinitely. It turns out that to recognize all possible output strings of RM, a necessary and sufficient condition is to know a set, TG, of 48 trigrams. A string is generated from RM if and only if the string starts with '#' and only contains trigrams in TG. In order to see this, we first observe that the Reber grammar yields exactly the same strings as the FSM, RM2, in Figure 2. Assume that we know RM and that we know the latest two symbols in an output string from RM2. This determines the internal state of RM (and RM2). The set of possible bigrams is {MT, MV, M#, RM, RR, RX, R#, TT, TV, T#, VR, VT, VX, V#, XM, XR, XT, XV, X#, #M, #V}. We obtain the set of possible trigrams from RM2 if we take these 21 bigrams and extend them according to Figure 2, yielding TG = {MTT, MTV, MVR, MVT, MV#, M#M, M#V, RM#, RRM, RRR, RR#, RXT, RXV, RXR, RXM, RX#, R#M, R#V, TTT, TTV, TVR, TVT, TV#, T#M, T#V, VRX, VT#, VXT, VXV, VXR, VXM, VX#, V#M, V#V, XM#, XRM, XRR, XR#, XTT, XTV, XVR, XVT, XV#, X#M, X#V, #MT, #MV, #VX}. It is clear that a string is an output of RM, if it starts with '#' and only contains trigrams in TG. Conversely, assume that a string begins with one of the 48 trigrams, the first symbol of which is '#'. The only possibilities are '#MT', '#MV', and

'#VX', which determine unique states in RM2. Recursively, assume that the last three symbols were $C_1C_2C_3$. Then C_2C_3 is a possible bigram, hence determines a unique internal state S in RM2. If the next symbol is C_4, by assumption $C_2C_3C_4$ is a possible trigram, but then C_4 must be on one of the transitions from the node S, and thus assigns a unique state to RM2. In this way, we can traverse the state-space of RM2 in a way that yields the desired output string. It follows that the string is a possible output of RM. This suggests one reason for the success of fragment-learning methods in acquiring the Reber grammar and similar AGs: a machine that memorizes TG can in principle recognize the Reber language.

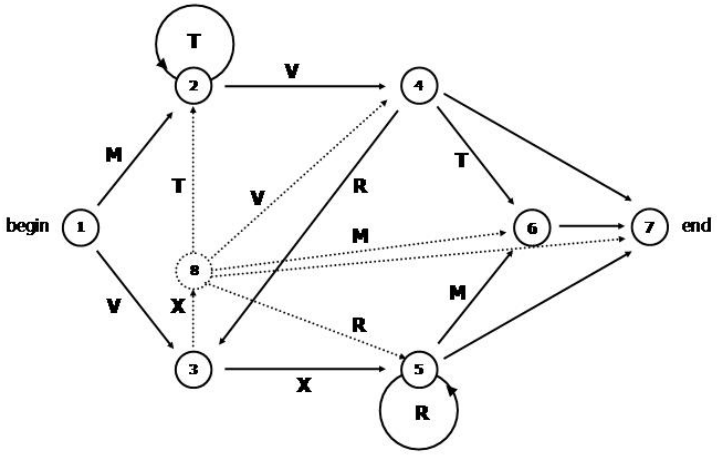
Figure 1. The transition graph representation of the Reber grammar and the modified Reber machine (RM: modifications dashed).



We can further characterize the properties of RM by information theoretic tools. We associate a random process with RM by setting X[0] = '#' with probability one. Then begin at the start state of the RM and randomly select one of the possible transitions with equal probability. Let the value of X[1] be the corresponding symbol. Continuing in this manner, randomly traversing RM yields a sequence of random symbol values X[n]. When the end of RM is reached, we use the symbol '#', as indicated above. The random process obtained will be referred to as the Reber

process. We note that while RM and RM2 generate the same output strings, the corresponding random processes have different probability distributions.

Figure 2. RM2. We added node 8 with five output transitions to the Reber machine (modifications dashed).



3. Information theoretic analysis

In many cases it is possible to define the entropy for a discrete-time random process. This procedure starts with the entropy of X[n] that remains after conditioning on X[0], … , X[n-1], and the conditional entropy will in our application converge as $n \to \infty$. The elementary analysis implies that, given two symbols of a Reber process, the probabilities for the next symbol are determined. In fact, apart from RX and VX, this is clear since we then know our position, and after RX or VX the probabilities are 1/4 for T and V and 1/6 for R, M and #. Thus the Reber process is second-order Markov and a consequence is that we can determine its entropy [cf., 6]. We can also measure how much information that is contained in the last and next-to-last symbol etc.

Table 1	**0**	**1**	**2**	**3**	**4**
RM	2.46	1.54	1.00	1.00	1.00
RM2	2.25	1.58	1.19	1.14	1.13

Since the process takes its values among six symbols, the entropy of a single symbol = 2.46 (≅ log 6), implies that these are approximately equally distributed. If we know the latest symbol, the entropy of the next decreases to 1.54 (≅ log 3), while given 2 symbols, which is all there is to know for a second-order Markov process, the entropy = 1.00 bits/symbol. This makes intuitive sense, since most states of RM have two exit transitions. We also see that more of the information is contained in the last compared to the next-to-last symbol. This means that one can expect reasonably good performance by recognizing bigrams alone. No further entropy reduction results from conditioning on more than two symbols (Table 1). Similar results hold for RM2. Again, almost all information is contained in the most recently observed bigram.

4. Neural network model

In this section we describe a series of computer experiments using predictive networks based on the simple recurrent network (SRN) architecture [7]. The networks were trained on acquisition data generated from RM [for details see 8, all software is available upon request]. We report some new findings and describe a new method for extracting representations of grammar rules from the state-space dynamics of the trained networks.

4.1 Performance evaluation

Four different methods for performance evaluation were used: (A) The output of the network was compared with the next in the test sequence. The difference is precisely what the learning process aims to minimize. Since the next symbol of the test sequence is a random variable it is impossible to predict it perfectly and one approach is to compute the theoretically best performance in terms of generalization performance and compare the actual performance with this. Here the output was interpreted using a winner-take-all competitive architecture in the output layer. (B) It is possible to interpret the normalized output of the network as a probability distribution for the next symbol. This is then compared with the theoretical probability distribution of the RM for the next symbol. (C) The third method mimics human experiments on AGL, in which the network is set to discriminate between correct and incorrect strings in a grammaticality classification task. In method C1 we introduced a grammatical string between strings to be classified (cf. below), in order to force the

network to settle into a start-of-word state. When we use C1 without this procedure, we call it C2, which is commonly used in the literature. (D) Here we attempted to measure how far the internal state of the network deviates from what can be considered the typical behavior in terms of its state-space dynamics. This is described in greater detail below and attempts to analyze the internal behavior of trained network and the basic question is whether the nodes in the hidden layer capture features that are interpretable with respect to underlying generative mechanism of the Reber grammar. To this end, we constructed a competitive network, which was trained to classify the outputs of the hidden nodes.

4.2 Simulation results

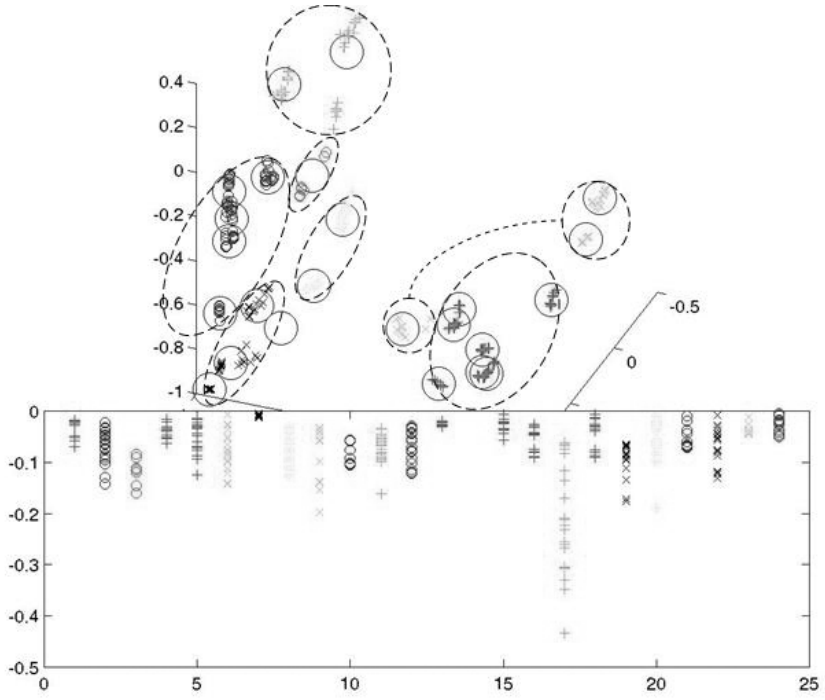
In computing entropies we followed the procedures described in section 2.7 of [6]. Higher-order Markov processes were transformed into Markov processes of order one by considering strings as output of the process instead of individual symbols (e.g., the output MTVRXM is transformed to MT–TV–VR–RX–XM). The limit distribution of the new Markov process was computed by an iterative procedure yielding a numerical limit distribution. The probabilities were rounded and verified to be stationary. The entropy of Markov process X was then calculated according to $H(X) = -\Sigma_{ij}\ \pi_{ij}\log(p_{ij})$, where π_{ij} is the stationary distribution and p_{ij} the transition probabilities of X. From the Reber process, a sequence of 1000 symbols was generated (127 strings). In the first series of experiments, five SRNs were defined with the following architecture: 6 nodes in the input/output layers, and 2,4,8,16,32 nodes in the hidden layer (transfer function in the hidden layer: inverse tangent; output layer: linear). With randomly initialized weights, the mean squared error was used as the objective cost function for the learning process (gradient-descent back-propagation with adaptable learning rate and momentum). The symbols (i.e., MTVRX#) were translated into a vector representation with six elements (0.8 in one position and -0.2 in the rest). The networks were trained for 200 epochs using one symbol of the acquisition sequence as input and the next symbol as target. In a second series of experiments, the networks were evaluated using methods A-D. Evaluation by method A and B was straightforward and in order to evaluate the networks by method C and D we generated a list containing Reber strings and non-Reber strings. The latter were generated from a first-order Markov process with the same bigram statistics as the Reber process.

The list contained 1000 symbols (50% strings from the Reber process). In method C, we interpreted the network's output as probabilities and multiplied these for each symbol in the test string, yielding a maximum-likelihood score for each string. These were normalized by taking the logarithm and dividing by string-length. We measured performance according to how many non-Reber strings that scored better than Reber strings. In order to improve performance, resetting was used in C1 by feeding MV#MV#MV#MV as input between each test string. In this way, the network settles in its start-of-word position. For method D, we first trained a competitive network with 24 nodes for 10000 epochs on the activation pattern of the SRN hidden nodes. The adaptive algorithm used was developed from the competitive networks in Matlab 5.3 Neural Network Toolbox and the algorithms described in section 2.5 of [9]. We then measured, for each symbol, the state-space distance from the activation of the hidden nodes to the closest node in the competitive network. We computed the mean value of these distances for all symbols in a test string. Heuristically, if the mean distance is large, the word is likely to be non-grammatical. The rest of the evaluation was performed as in method C. The results of the first series of experiments (mean learning error on the acquisition data for 10 networks of each size) indicate that the smaller networks (2, 4 hidden nodes) have smaller error in the beginning, while the larger networks perform better in the end (8, 16, 32). There was no gain in performance when the number of hidden nodes increased above 8. In epochs 40-200, the curves continue almost horizontally, replicating previous results. We evaluated the generalization capacity on an independent string set, using the methods A-D described above, see Table 2. The theoretically best performance was calculated from the full knowledge of the state transitions between internal states of RM, a Hidden Markov Process. The behavior of the competitive network is illustrated in Figure 3. We see that the competitive network is able to discriminate between activation states of the hidden nodes corresponding to different states of RM: to every RM-state corresponds one region of state-space.

Table 2. Performance evaluation according to methods A-D. Mean from 8 networks of each size (± std).

Hidden Nodes	A	B	C1	C2	D
2	0.114±0.002	0.037±0.002	60±4%	60±5%	61±2%
4	0.094±0.004	0.017±0.004	79±6%	81±5%	68±6%
8	0.088±0.002	0.010±0.002	91±3%	87±5%	75±5%
16	0.086±0.003	0.009±0.003	93±2%	74±6%	83±5%
32	0.085±0.002	0.008±0.002	94±2%	73±4%	81±5%
Best possible	0.076	0	100%	100%	100%

Figure 3. The upper part shows the activation of the hidden nodes plotted along the first three principal axes obtained by PCA. The dots are coded to correspond to the seven internal states of RM. The diagram shows that dots corresponding to the same internal state are geometrically close (dashed ellipses). Thus the state-space dynamics of the network reproduces the dynamics of the Reber machine. The filled circles in the diagram are centered at the 24 nodes of a trained competitive network. The lower part of the diagram plots all activation points according to their distance from the closest node in the competitive network.



5. Discussion

There was no significant performance increase with more than 8 hidden nodes. These nodes have two main tasks: memory and computation. The information theoretical analysis suggests that the memory required is on the order of 2-3 bits (~2-3 nodes). The reason performance improved with 8 hidden nodes is likely related to requirements on the output representation. In general, there is a risk of over-learning using large networks on small acquisition samples, leading to sub-optimal generalization performance and less efficient acquisition of the underlying structure in the input. In the present study, this was not a major problem, and is likely related to the close match between the SRN architecture and Markov processes. SRNs can be viewed as a time-discrete analog version of the finite state architecture. We evaluated network performance on test data with the same bigram statistics as the Reber process. This is a harder task than commonly employed in AGL experiments on humans or with computers [for a review 10]. We were able to obtain better than 90% network performance as characterized by method C1, perhaps not so surprising on theoretical grounds [11], but suggests that the lower SRN performance reported in the literature is related to the evaluation method C2 (C1 without resetting). Especially when large SRNs are tested, using C2, a single incorrect symbol can induce activation patterns that are distant from the regions of state-space on which they were trained. The state-space trajectories then remain distant from 'normal' trajectories for several subsequent input symbols. A grammatical string that is processed after an incorrect string can thus obtain a low grammaticality score due to transient network behaviors. Concerning the dichotomy between rule-learning and statistical fragment-based learning, the analysis of activation patterns suggests that the SRN can represent the rules of the grammar in its state-space dynamics. Conversely, a finite set of string fragments is sufficient for Reber language recognition. Finally, the use of competitive networks to interpret the state-space dynamics suggests one possible avenue for future research.

References

[1] Chomsky, N., Aspects of the Theory of Syntax. 1965, Cambridge, MA.: MIT Press.

[2] Gomez, R.L. and L. Gerken, Infant artificial language learning and language acquisition. TICS, 2000. 4: 178-186.

[3] Friederici, A.D., K. Steinhauer, and E. Pfeifer, Brain signatures of artificial language processing. Proc. Natl. Acad. Sci. USA, 2002. 99: 529-534.

[4] Petersson, K.M., C. Forkstam, and M. Ingvar, Artificial syntactic violations activate Broca's region. Cogn. Sci., 2004. 28: 383-407.

[5] Reber, A.S., Implicit learning of artificial grammar. J. Verb. Learn. & Verb. Behav., 1967. 6: 855-863.

[6] Goldie, C.M. and R.G.E. Pinch, Communication Theory. 1991, Cambridge, UK: Cambridge University Press.

[7] Elman, J.L., Finding structure in time. Cogn. Sci., 1990. 14: p. 179-211.

[8] Grenholm, P., Artificial Grammar Learning - Rules and Statistics. 2003, Cognitive Neurophysiology Research Group, Karolinska Institutet: Stockholm.

[9] Haykin, S., Neural Networks: A Comprehensive Foundation, 2nd ed. 1998, Upper Saddle River, NJ: Prentice Hall.

[10] Redington, M. and N. Chater, Computational models of artificial grammar learning. Preprint.

[11] Casey, M., The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. Neur. Comp., 1996. 8: 1135-1178.