ARTIFICIAL GRAMMAR LEARNING – A CASE STUDY OF THE REBER GRAMMAR

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

[1] Cognitive Neurophysiology Research Group
Department of Clinical Neuroscience, Karolinska Institutet
Stockholm, Sweden

[2] Cognitive Neurology and Memory Research Group
F.C. Donders Centre for Cognitive Neuroimaging
University of Nijmegen, Nijmegen, The Netherlands

Corresponding author
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 the finite state grammars. Artificial grammar learning has been used to study aspects of natural language processing in humans. 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 simple formal analysis and network simulations. We also outline a method for describing network dynamics and suggest that statistical frequency based and rule-based artificial grammar learning in important ways 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

The human capacity to learn and communicate through language is an outstanding scientific challenge to understand and Chomsky, following von Humboldt, has suggested that natural language processing is a paradigmatic example of the 'infinite use of finite means' (Chomsky, 1965). The simplest formal model, incorporating the idea of 'infinite use of finite means' is represented by the family of right-linear phrase structure grammars which can be implemented in the finite state architecture. Furthermore, it has been suggested that the task of learning an artificial grammar is a relevant model for aspects of language learning in infants (Gomez & Gerken, 2000), exploring species differences in learning capacity (Hauser, Chomsky, & Fitch, 2002), and second language learning in adults (Friederici, Steinhauer, & Pfeifer, 2002). Recent FMRI studies indicate that language related brain regions are engaged in artificial grammar processing including Broca's region (e.g., Petersson, Forkstam, & Ingvar, in press). The seminal work of Reber (1967) indicated that humans can learn artificial grammars in an implicit fashion and suggested that relevant information was abstracted from the environmental input. Reber (1967) also suggested that this process represented a mechanism that is intrinsic to natural language learning. Since the 1940s, the classical information processing perspective on cognition has been situated within the framework of Church-Turing computability (Davis, Sigal, & Weyuker, 1994). A complementary perspective is offered by neural network models of language learning and language processing (for a review, see e.g., Christiansen & Chater, 2001).

2. Artificial grammars and generative machines

The Reber grammar and AGs more generally are examples of generative mechanisms for artificial languages (Davis et al., 1994). In the case of artificial languages generated by a FSM it is the case that the corresponding FSM also can be viewed as a language recognizer.

2.1 Elementary analysis

In this section we indicate that the Reber grammar is completely determined by a set of short strings by an elementary analysis. We begin by modifying the finite state machine (FSM) of Reber (Figure 1) by connecting the final state to the initial state with a #-labeled arrow. The modified Reber machine thus outputs an infinite character sequence, by first generating an end-of-string character '#', a string from the Reber language, a '#', and subsequently a new string from the Reber language, and so on indefinitely. It turns out that in order to recognize all possible output sequences from the modified Reber machine, a necessary and sufficient condition is to know a set of 48 trigrams (= TG) and a sequence is generated from the modified Reber machine if and only if the sequence starts with an '#' and only contain trigrams in TG. In order to show this, we first observe that the Reber grammar yields exactly the same sequences as the FSM in Figure 2. Assume that we know the machine in Figure 2 and that we know the latest two characters in an output sequence from this machine. This determines in which internal state the machine is in. The set of all 21 possible

bigrams is {MT, MV, M#, RM, RR, RX, R#, TT, TV, T#, VR, VT, VX, V#, XM, XR, XT, XV, X#, #M, #V}.

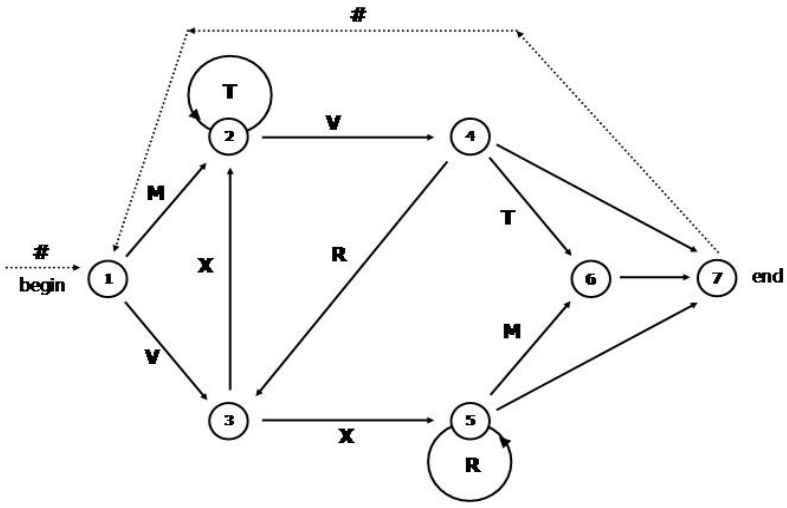

Figure 1. The transition graph representation of the Reber grammar and the modified Reber grammar (modifications dashed).

Any of these determine a unique internal state of the FSM in Figure 2. We obtain the set of possible trigrams from the modified Reber machine if we take these 21 bigrams and extend them according to Figure 2 to yield the 48 trigrams in 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}. If a sequence is an output from the modified Reber machine, it will start with '#' and only contain these trigrams. Conversely, assume that a sequence begins with one of the 48 trigrams, the first character of which is '#'. The only possibilities are '#MT', '#MV', and '#VX', all of which determine unique internal states in Figure 2. Recursively, assume that the last three characters were $C_1C_2C_3$. Then C_2C_3 is a possible bigram, hence determines a unique internal state N in Figure 2. If the next character 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 N in Figure 2. We can thus assign a unique internal state to the machine of Figure 2. In this way, we can traverse the machine in Figure 2 in a way that gives the desired output sequence. It follows that the sequence is a possible output from the modified Reber machine. This indicates one possible reason for the success of fragment-learning methods in acquiring the Reber grammar and similar AGs: a machine that memorizes, in this case 48 trigrams, can in principle recognize the Reber language. Not all grammars studied in the literature may have this property. We can, however, characterize this property by information theoretic tools. In order to do this, we associate a random process with the modified Reber

grammar by setting X[0] = '#', with probability one. Then begin at the start node of the corresponding FSM and randomly select one of the possible transitions with equal probability. Let the value of X[1] be the corresponding character.

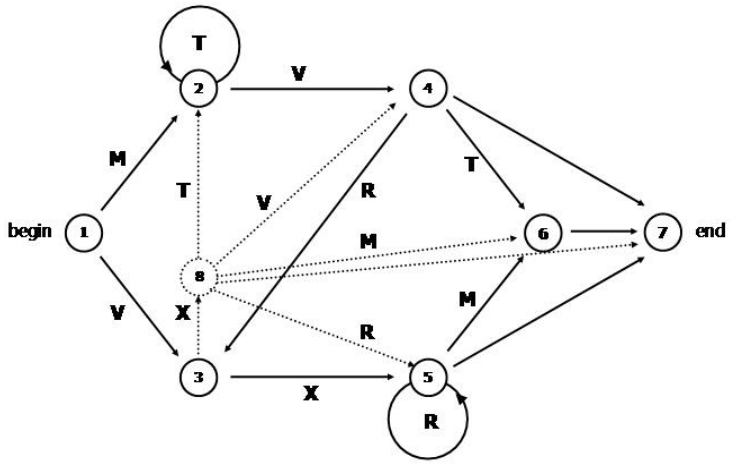

Figure 2. We have added a node 8 with five output transitions to the Reber FSM (modifications dashed). The reason for doing is that the Reber machine has two transitions from the same node both associated with an X. In the new FSM, this X corresponds to only one transition, but we will be back on track after the next move.

Continuing in this manner, randomly traversing the FSM yields a sequence of random character values X[n]. When the end of the FSM is reached, we use the character '#', as indicated above. The process obtained from the modified Reber grammar will be referred to as the Reber process. We note that while the modified Reber grammar and the FSM of Figure 2 generate the same output sequences, the corresponding random processes have different probability distributions.

2.2 Information theoretic analysis and results

In many cases it is possible to define the entropy for a discrete-time, discrete-valued random process, in the present case, a sequence of dependent random variables X[n], $n \geq 0$. This procedure starts with the entropy of X[n] that remains when condition on X[0], … , X[n-1] (cf., Goldie & Pinch, 1991). In our applications, the conditional entropy will converge as $n \rightarrow \infty$. One consequence of the elementary analysis is that, given the two latest characters of a Reber process, the probabilities for the next possible characters 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 blank space. This implies that the Reber process is a second-order Markov process and an important consequence is that we can determine its entropy (Goldie & Pinch, 1991).

We can also measure how much information about the process that is contained in the last and next-to-last out-putted characters etc. Any finite-state-grammar can be analyzed in a similar manner.

Table 1	0	1	2	3	4
Modified Reber	2.46	1.54	1.00	1.00	1.00
FSM of fig. 2	2.25	1.58	1.19	1.14	1.13

Since the Reber process takes its values among six characters, the entropy of a single character = 2.46 ($\cong$ log 6), means that these are approximately equally distributed. If we know the latest character, the entropy of the next decreases to 1.54 bits ($\cong$ log 3), while if given 2 characters the entropy is = 1. If we know the two most recent characters, which for a second-order Markov process amounts to knowing what there is to know, the entropy is 1.00 bits/character. This makes intuitive sense, since most nodes in the Reber FSM have two transitions leaving it. We also see that more of the information is contained in the last character compared to the next-to-last character. This means that one can expect reasonably good performance by recognizing bigrams alone. As might be predicted from the elementary analysis, no further entropy reduction results from conditioning on more than two observations (Table 1). Similar results are observed for the grammar in Figure 2. Again, almost all information is contained in the most recently observed bigram.

3. Neural networks models, acquisition paradigm, and evaluation methods

In this section we describe a series of computer experiments using predictive networks based on the simple recurrent network (SRN) architecture (Elman, 1990). The networks were trained on acquisition data generated from the modified Reber grammar (for details see Grenholm, 2003, and all software is available upon request from the authors). In addition to reproducing previous findings we report some new findings. We also describe a method for extracting representations of grammatical rules from the state space dynamics of the trained networks.

3.1 Performance evaluation

We used four different methods to evaluate network performance: (A) The output of the network is compared with the next character of the test sequence. This difference is precisely what the learning process aims to minimize. Since the next character of the test sequence is a random variable no network will be able to estimate it perfectly and a solution is to compute the theoretically best performance of the network 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 character. This is then compared with the theoretical probability distribution for the next character as obtained from the modified Reber grammar. (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 long grammatical string between strings to be classified, in order to force the network to settle into a start-of-word state. When we use method C1 without this procedure, we denote it by C2, which represents a commonly used evaluation method described in the literature. (D) In the last evaluation method we attempted to measure how far the internal state of the network deviates from what can be considered the 'typical behavior' of the network in terms of its state space dynamics. This is described in greater detail in the last computer experiment (cf., below). Method D is an attempt to analyze the internal behavior of trained network and the basic question is whether the nodes in the hidden layer capture some features of the input 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.

3.2 Methods

In computing entropies we followed the procedures described in section 2.7 of Goldie & Pinch (1991). 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 iteration procedure yielding a numerical limit distribution. The probabilities were rounded and verified to be stationary. The entropy of the Markov process X was then calculated by the formula $H(X) = -\Sigma_{ij}\ \pi_{ij}\log(p_{ij})$, where π_{ij} is the stationary distribution and p_{ij} are the transition probabilities of the Markov process X. From the Reber process, a sequence of 1000 characters was generated, corresponding to 127 strings. In the first series of experiments, five SRNs were defined with the following architecture: 6 input and output nodes in the input and output layers, and 2,4,8,16, and 32 nodes in the hidden layer, respectively. The inverse tangent was used as transfer function in the hidden layer and the output layer was linear. With randomly initialized weights, the mean squared error was used as the objective cost function for the learning process (gradient-descent back-propagation algorithm with adaptable learning rate and momentum). The characters (i.e., MTVRX#) were translated into a vector representation with six elements (0.8 on one position and -0.2 on the other). The networks were trained for 200 epochs using one character from the data as input and the next character as the target. In a second series of experiments, the networks were evaluated using methods A-D. Evaluation by method A and B above 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 characters, with approximately half the strings coming from the Reber process. In method C, we interpreted the network's output as probabilities. We then multiplied the output probabilities for each character in the test word. This

gave a maximum-likelihood score for each string, which was 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 increase the performance, resetting was used in method C1 between each test string by submitting the string MV#MV#MV#MV as input. In this way, the network settles into its start-of-word position. For method D, we first trained a competitive network 10000 epochs with 24 nodes on the activation pattern in the hidden nodes of the SRN, using 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 Haykin (1998). We then measured, for each character, 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 characters 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.

3.3 Network simulation results

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 and 4 hidden nodes) have smaller error in the beginning, while the larger networks perform better in the end (8, 16, and 32). Moreover, there was no gain in performance when the number of hidden nodes is increased from 8 to 16 or 32. In epochs 40-200, the curves continue almost horizontally, replicating previous results. In addition, we evaluated the generalization capacity of the networks on an independent string set, using the methods A-D described above.

Table 2. Evaluation of the trained networks according to the evaluation methods A-D. The results are mean values of 8 networks of each size, reported together with the standard deviation of the mean. We conclude that networks with 8 hidden nodes perform significantly better than smaller networks on evaluation methods A, B and C1. If we use evaluation method C2, grammatical classification as in method C1, but without resetting between strings, the networks with 16 and 32 perform worse than with method C1.

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%

The theoretically best performance was calculated from the full knowledge of the state transitions between internal states of the Reber process (which in this way can be viewed as a Hidden Markov Process). The behavior of the competitive network is illustrated in Figure 3. We see that the competitive network has been able to discriminate between activation states of the hidden nodes corresponding to different states of the modified Reber FSM: to every node corresponds exactly one color.

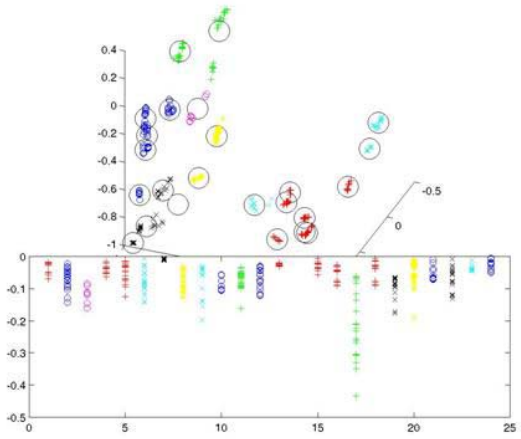

Figure 3. The upper part shows the activation of the hidden nodes of the network, plotted along the first three axes obtained by PCA. The dots are coded in seven colors corresponding to the seven machine states of the modified Reber FSM. The diagram shows that dots with the same color are geometrically close. This means that the dynamics of the hidden nodes reproduces the dynamics of the modified Reber FSM. The 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, with color-coding as above.

4. Discussion

When we trained networks of different sizes on the Reber grammar, there was no substantial increase in performance when the number of hidden nodes increased above eight. The hidden nodes have two main tasks: memory and computation. The information theoretical analysis indicates that the memory required is on the order of 2-3 bits (~2-3 nodes). Hence, the reason that performance increased up to 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 acquisition data. In the present study this appeared not to be of major

concern and is likely related to the close match between the SRN architecture and Markov processes, in general. 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 see e.g., Redington & Chater, preprint). We were able to obtain better than 90% network performance as characterized by method C1. These results are less surprising on theoretical grounds (Casey, 1996). SRNs may be viewed as a time-discrete analog version of the finite state architecture. However, it should also be noted that SRN simulations, using finite precision numbers, effectively is a FSM simulation. We suggest that the lower SRN performance reported in the literature is related to the evaluation method C2 that these studies have employed (i.e., method C1 without re-setting). Especially when large SRNs are tested using method C2, a single incorrect character in a control string can induce an activation pattern over the hidden nodes that is distant from the regions in state space in which they were trained. The state space trajectories can then remain distant from the 'normal' trajectories for several subsequent input characters. In this way, a grammatical string that is processed after an incorrect string can obtain an 'artificially low grammaticality score'. Concerning the dichotomy between rule-learning and statistical fragment-based learning, a closer analysis of the modified Reber grammar showed that its three-character fragments determine it completely. Conversely, an analysis of the activation pattern over the hidden nodes indicated that a trained SRN can be viewed as representing the rules of the grammar, as shown by the competitive ‘pattern classification’ network analysis. This is not surprising since neural networks can be viewed as an universal approximation device, which have been shown to have representational and processing capacities which include the Church-Turing computability framework (Siegelmann, 1999). The use of competitive networks to interpret the state space dynamics of neural networks introduced here suggests one possible avenue for future research.

References

Casey, M. (1996). The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural Computation, 8*, 1135-1178.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. Cambridge, MA.: MIT Press.

Christiansen, M. H., & Chater, N. (Eds.). (2001). *Connectionist Psycholinguistics*. Norwood, NJ: Ablex Publishing.

Davis, M. D., Sigal, R., & Weyuker, E. J. (1994). *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science* (2 ed.). San Diego, CA: Academic Press.

Elman, J. L. (1990). Finding structure in time. *Cogn. Sci., 14*, 179-211.

Friederici, A. D., Steinhauer, K., & Pfeifer, E. (2002). Brain signatures of artificial language processing. *Proc. Natl. Acad. Sci. USA, 99*, 529-534.

Goldie, C. M., & Pinch, R. G. E. (1991). *Communication Theory*. Cambridge, UK: Cambridge University Press.

Gomez, R. L., & Gerken, L. (2000). Infant artificial language learning and language acquisition. *TICS, 4*, 178-186.

Grenholm, P. (2003). *Artificial Grammar Learning - Rules and Statistics*. Stockholm: Cognitive Neurophysiology Research Group, Department of Clinical Neuroscience, Karolinska Institutet.

Hauser, M. D., Chomsky, N., & Fitch, W. T. (2002). The faculty of language: what is it, who has it, and how did it evolve? *Science, 298*(5598), 1569-1579.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation, 2nd ed.,* . Upper Saddle River, NJ: Prentice Hall.

Petersson, K. M., Forkstam, C., & Ingvar, M. (in press). Artificial syntactic violations activates Broca's region. *Cognitive Science*.

Redington, M., & Chater, N. (preprint). Computational models of artificial grammar learning.

Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Basel, Switzerland: Birkhäuser.