Inferring stochastic regular grammars with recurrent neural networks

Rafael C. Carrasco, Mikel L. Forcada, Laureano Santamaría

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante
E-03071 Alicante, Spain.
E-mail: {carrasco, mlf, laureano}@dlsi.ua.es.

Abstract. Recent work has shown that the extraction of symbolic rules improves the generalization performance of recurrent neural networks trained with complete (positive and negative) samples of regular languages. This paper explores the possibility of inferring the rules of the language when the network is trained instead with stochastic, positive-only data. For this purpose, a recurrent network with two layers is used. If instead of using the network itself, an automaton is extracted from the network after training and the transition probabilities of the extracted automaton are estimated from the sample, the relative entropy with respect to the true distribution is reduced.

1 Introduction

A number of papers [1, 2, 3, 4] have explored the ability of second-order recurrent neural networks to infer simple regular grammars from complete (positive and negative) training samples. The generalization performance of a finite state automaton extracted from the trained network is better than that of the network itself [1, 2, 5]. In most cases, they find that once the network has learned to classify all the words in the training set, the states of the hidden neurons visit only small regions (clusters) of the available configuration space during recognition. These clusters can be identified as the states of the inferred deterministic finite automaton (DFA) and the transitions occurring among these clusters when symbols are read determine the transition function of the DFA. Although automaton extraction remains a controversial issue [6], it has recently been has shown [7] that a recurrent neural network can robustly model a DFA if its state space is partitioned into a number of disjoint sets equal to the number of states in the minimum equivalent automaton. An automaton extraction algorithm can be found in [5]. The extraction is based on partitioning the configuration hypercube $[0.0, 1.0]^N$ in smaller hypercubes, such that at most one of the clusters falls into each defined region. An improved algorithm which does not restrict clusters to point-like form is described in [8].

However, complete samples, which include both examples and counterexamples, are scarce and, in most cases, only a source of random (or noisy) examples is available. In such situations, inferring the underlying grammar will clearly improve the estimation of the probabilities, above all, for those strings which do not appear in the sample. In this paper, the possibility of extracting symbolic rules from a recurrent network trained with stochastic samples is explored. The architecture of the network is described in next section, and it can be viewed as an Elman net [9] with a second-order next-state function, as used in [8]. The related work of [10] uses an Elman architecture to predict the probability of the next symbol, but it does not address the question of what symbolic representation is learned by the net. An alternate, algorithmic approach to the inference of stochastic regular languages can be found in [11].

2 Network architecture

Assume that the input alphabet has L symbols. Then, the second-order recurrent neural network consists of N hidden neurons and L input plus $L+1$ output neurons, whose states are labeled S_k, I_k and O_k respectively (see Fig. 1).

The network reads one character per cycle (with $t = 0$ for the first cycle) and the input vector is $I_k = \delta_{kl}$ when a_l (the l-th symbol in the alphabet) is read[1]. Its output $O_k^{[t+1]}$ represents the probability that the string is followed with symbol a_k for $k = 1, ...L$ and $O_{L+1}^{[t+1]}$ represents the probability that the string ends there. The dynamics is given by the equations:

$$O_k^{[t]} = g(\sum_{j=1}^{N} V_{kj} \; S_j^{[t]}) \tag{1}$$

$$S_i^{[t]} = g(\sum_{j=1}^{N} \sum_{k=1}^{L} W_{ijk} \; S_j^{[t-1]} \; I_k^{[t-1]}) \tag{2}$$

where $g(x) = 1/(1 + \exp(-x))$.

The network is trained with a version of the Real Time Recurrent Learning (RTRL) algorithm [12]. The contribution of every string w in the sample to the energy (error function) is given by

$$E_w = \sum_{t=0}^{|w|} E_w^{[t]} \tag{3}$$

where

$$E_w^{[t]} = \frac{1}{2} \sum_{k=1}^{L+1} \left(O_k^{[t]} - I_k^{[t]} \right)^2, \tag{4}$$

$I_k^{[|w|]} = \delta_{k,L+1}$, and $I_{L+1}^{[t]} = 0$ for $t < |w|$. It is not difficult to prove that this energy has a minimum when $O_k^{[t]}$ is the conditional probability of getting symbol a_k after reading the first t symbols in w. Because a probability one (or zero) is difficult to learn, a contractive mapping $f(x) = \epsilon + (1 - 2\epsilon)x$, with ϵ a small

[1] The Kronecker's delta δ_{kl} is 1 if $k = l$ and zero otherwise.

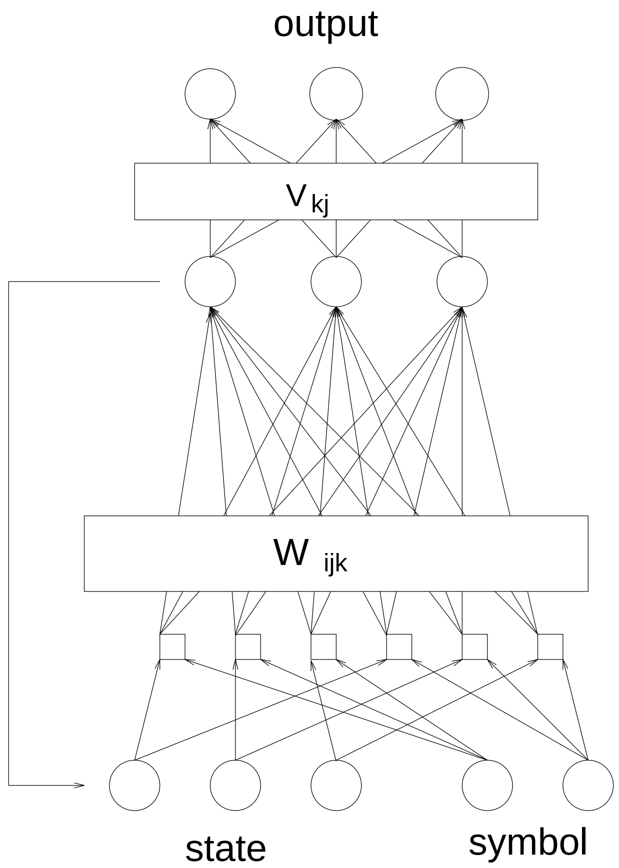

Fig. 1. Second-order recurrent neural network with $N = 3$ and $L = 2$

positive number, is applied to inputs (outputs are correspondingly expanded). This mapping does not affect the essentials of the formalism presented here.

Usually, the weights are set to random values and then learned, but the initial states of the hidden neurons $S_i^{[0]}$ cannot change during training. As remarked in [13], there is no reason to force the initial states to remain fixed, and the behavior of the network improves if they are also learned. This is even more important here, because the predicted probability $O_k^{[t]}$ depends on the states $S_i^{[t]}$, while in a mere classification task it suffices that the accepting states are in the acceptance region and the rest in the non-acceptance region. In order to keep $S_i^{[t]}$ within the range $[0.0, 1.0]$ during gradient descent, one rather defines the net inputs $\sigma_i^{[t]}$ such that $S_i^{[t]} = g(\sigma_i^{[t]})$ and take $\sigma_i^{[0]}$ as the parameters to be optimized.

Therefore, all parameters V_{kj}, W_{ijk} and $\sigma_i^{[0]}$ are initialized to small random values before training, but then learned together. Every parameter P is updated according to gradient descent:

$$\Delta P = -\alpha \frac{\partial E}{\partial P} + \eta \Delta' P \tag{5}$$

with a learning rate α and a momentum term η. The value $\Delta' P$ represents the variation of parameter P at previous iteration.

The contribution of every symbol in w to the energy derivative is:

$$\frac{\partial E_w^{[t]}}{\partial P} = \sum_{b=1}^{L+1} (O_b^{[t]} - I_b^{[t]}) O_b^{[t]} (1 - O_b^{[t]}) \sum_{a=1}^{N} (\frac{\partial V_{ba}}{\partial P} S_a^{[t]} + V_{ba} S_a^{[t]} (1 - S_a^{[t]}) \frac{\partial \sigma_a^{[t]}}{\partial P}) \tag{6}$$

where the derivative of the sigmoid function $g'(x) = g(x)(1 - g(x))$, was used.

The derivatives with respect to V_{kj} may be evaluated in a single step:

$$\frac{\partial E_w^{[t]}}{\partial V_{kj}} = (O_k^{[t]} - I_k^{[t]}) O_k^{[t]} (1 - O_k^{[t]}) S_j^{[t]} \tag{7}$$

However, the derivatives with respect to $\sigma_i^{[0]}$ and W_{ijk} are computed in a recurrent way, using the following equations:

$$\frac{\partial \sigma_m^{[0]}}{\partial W_{ijk}} = 0 \tag{8}$$

$$\frac{\partial \sigma_i^{[0]}}{\partial \sigma_j^{[0]}} = \delta_{ij} \tag{9}$$

$$\frac{\partial \sigma_i^{[t]}}{\partial \sigma_j^{[0]}} = \sum_{a=1}^{N} S_a^{[t-1]} (1 - S_a^{[t-1]}) \sum_{b=1}^{L} W_{iab} \frac{\partial \sigma_a^{[t-1]}}{\partial \sigma_j^{[0]}} I_b^{[t-1]} \tag{10}$$

$$\frac{\partial \sigma_m^{[t]}}{\partial W_{ijk}} = \delta_{im} S_j^{[t-1]} I_k^{[t-1]} + \sum_{a=1}^{N} S_a^{[t-1]} (1 - S_a^{[t-1]}) \sum_{b=1}^{L} W_{mab} \frac{\partial \sigma_a^{[t-1]}}{\partial W_{ijk}} I_b^{[t-1]} \tag{11}$$

3 Results and discussion

The behavior of the network has been checked using the following stochastic deterministic regular grammar:

$$\begin{array}{l} S \to 0S \ (0.2) \\ S \to 1A \ (0.8) \\ A \to 0B \ (0.7) \\ A \to 1S \ (0.3) \\ B \to 0A \ (0.4) \\ B \to 1B \ (0.1) \end{array} \tag{12}$$

where the numbers in parenthesis indicate the probability of the rules. The end-of-string probability for a given variable is the difference between 1 and the sum of the probabilities of all rules with that variable on the left. Two input neurons ($L = 2$) and three output neurons (corresponding to 0, 1 and end-of-string respectively) were used, while the number of hidden neurons in the network was set to $N = 4$. The network was trained with 800 random[2] examples generated by grammar (12) and all of them were used at every iteration of the learning algorithm. The learning rate was $\alpha = 0.004$ and the momentum term $\eta = 0.2$. The training process for a given sample involved 1000 iterations. Figure 2

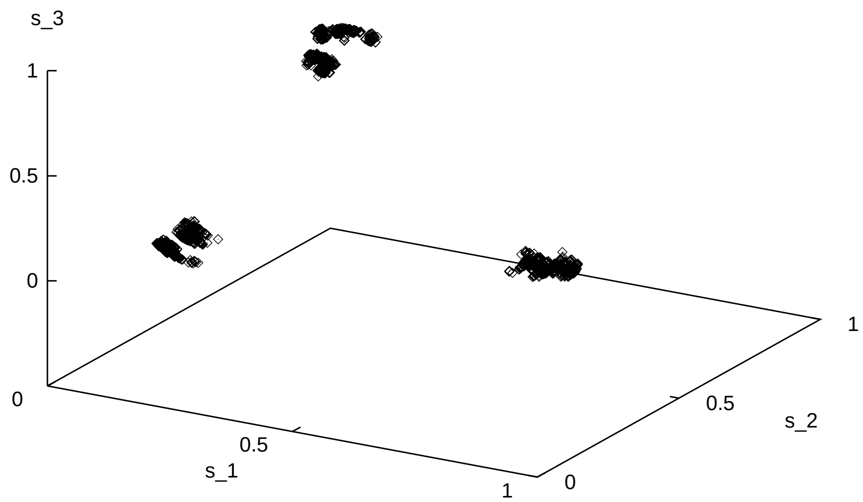

Fig. 2. Configuration space after training with 800 examples.

shows a typical section of the configuration space $[0.0, 1.0]^N$ for a trained net. Every point represents a vector $S^{[t]}$, generated for all words of length up to 10. As seen in the figure, only some regions are visited and three clusters appear corresponding to the three variables in the grammar. Clusters were observed in all the experiments.

[2] The sample contained an average of 200 different strings.

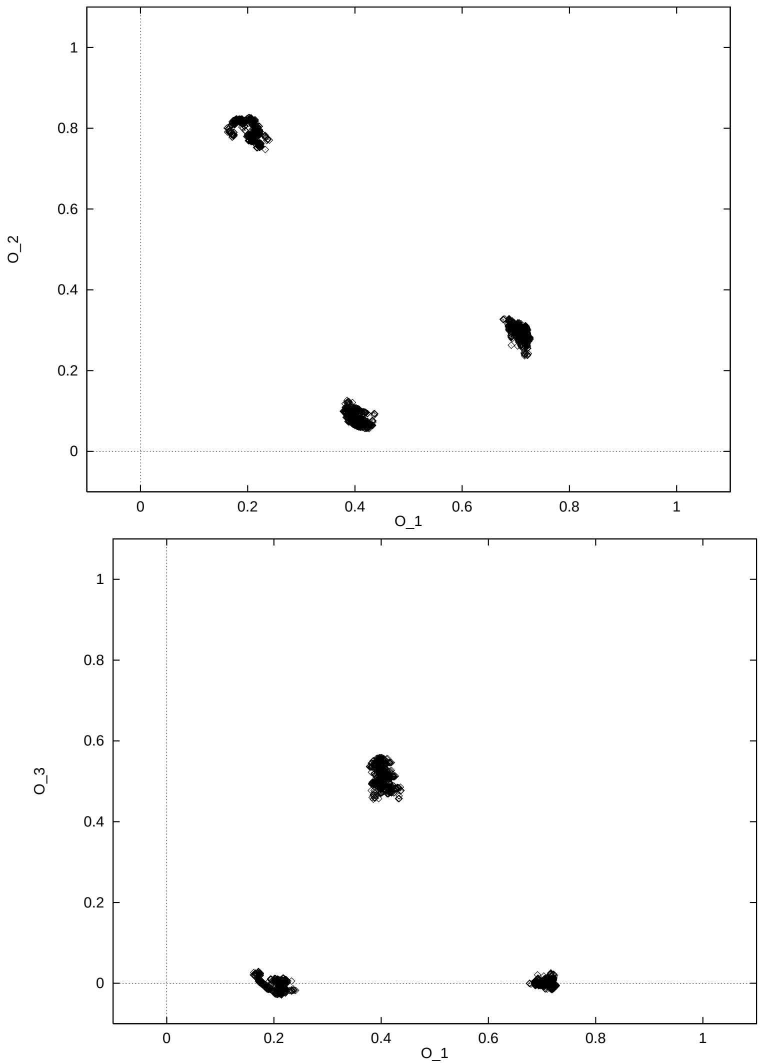

Fig. 3. Output probabilities after training.

In Fig. 3, the projection to the output space (O_1, O_2, O_3) is plotted. The clusters map into the regions around (0.2,0.8,0.0), (0.7,0.3,0.0) and (0.4,0.1,0.5), in agreement with the true probabilities. Note that, even if not required by the model, the sum of all three components is approximately 1. Note also that due to the contractive mapping (we take $\epsilon = 0.1$) one gets some values slightly below zero which should be treated as rounding errors. As observed in the picture, the trained network is able to predict the probability of the next symbol with good accuracy.

A measure of the similarity between the probability distribution $p_{\mathrm{net}}(w)$ estimated with the trained network and the one defined by (12), $p_{\mathrm{grm}}(w)$, is given by the relative entropy or Kullback–Leibler distance [14]:

$$d(p_{\mathrm{grm}}, p_{\mathrm{net}}) = \sum_{w} p_{\mathrm{grm}}(w) \log_2 \frac{p_{\mathrm{grm}}(w)}{p_{\mathrm{net}}(w)} \tag{13}$$

which gave an average value of 0.05 bits in 10 experiments (obtained by summing for all strings w in canonical order until convergence is reached). However, when the transition diagram of the automaton was extracted from the network (using the algorithm in [5]) and its transition probabilities were estimated using the statistics of the sample, the distribution $p_{\mathrm{ext}}(w)$ generated by this stochastic automaton reduced the average distance $d(p_{\mathrm{grm}}, p_{\mathrm{ext}})$ to 0.0015 bits (this distance may be computed exactly using a generalization of the method described in [14] for stationary Markov chains). Therefore, the automaton extracted from the network with experimentally estimated transition probabilities outperforms the network in modeling the stochastic language. It should be emphasized that generalization performance is not assessed here by using a test set, but by comparing the inferred model (either the network or an automaton extracted from it) to the grammar that was used to generate the synthetic sample. The distances obtained above may be shown to be very small when compared to $d(p_{\mathrm{sam}}, p_{\mathrm{grm}})$ (with p_{sam} the actual frequencies observed in the sample), which is 1.2 bits in this case[3]; this gives another indication of the degree of generalization attained by the inference process. The results obtained for other grammars of similar complexity are comparable.

4 Conclusions and future developments

The generalization performance of a second-order recurrent neural network when trained with stochastic samples of regular languages has been studied. It is found that clusters appear in the internal state space which correspond to the states in the finite automaton used to generate the sample. If the probabilities are estimated with the extracted automaton, the distance between the predicted probability distribution and the true one reduces substantially with respect to the distance between the network's estimate and the true distribution; it also has to be noted that both distances are negligible compared to the distance between the sample and the true distribution. This last fact shows by itself the generalization ability of the inference method. The results suggest that second-order recurrent neural networks may become suitable candidates for modeling stochastic processes when grammatical algorithms are not yet developed.

We are currently studying the application of the more rigorous automaton extraction algorithm of [8] to our networks, as well as the effect of having a finite sample that gives an incorrect estimation for under-represented strings on the quality of the internal representation inferred by the network.

---

[3] The distance $d(p_{\mathrm{grm}}, p_{\mathrm{sam}})$ is infinity due to the fact that the sample is finite.

Acknowledgments: We wish to thank the Dirección General de Investigación Científica y Técnica of the Government of Spain for support through projects TIC93-0633-C02 and TIC95-0984-C02-01. We also thank Ramón Ñeco for suggestions regarding the manuscript.

References

1. Giles, C.L., Miller, C.B, Chen, D., Chen, H.H., Sun, G.Z., Lee, Y.C.: "Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks" *Neural Computation* **4** (1992) 393–405.
2. Giles, C.L., Miller, C.B, Chen, D., Sun, G.Z., Chen, H.H., Lee, Y.C.: "Extracting and Learning an Unknown Grammar with Recurrent Neural Networks", in *Advances in Neural Information Processing Systems 4* (J. Moody et al., eds.), Morgan-Kaufmann, San Mateo, CA, 1992, p. 317–324.
3. Watrous, R.L., Kuhn, G.M.: "Induction of Finite-State Automata Using Second-Order Recurrent Networks" *Advances in Neural Information Processing Systems 4* (J. Moody et al., Eds), Morgan-Kaufmann, San Mateo, CA, 1992, p. 306–316.
4. Watrous, R.L., Kuhn, G.M.: "Induction of Finite-State Languages Using Second-Order Recurrent Networks" *Neural Computation* **4** (1992) 406–414.
5. Omlin, C.W., Giles, C.L.: "Extraction of Rules from Discrete-Time Recurrent Neural Networks" *Neural networks* **9**:1 (1996) 41–52.
6. Kolen, J.F.: "Fool's Gold: Extracting Finite-State Automata from Recurrent Network Dynamics" in *Advances in Neural Information Processing Systems 6* (C.L. Giles, S.J. Hanson, J.D. Cowan Eds.), Morgan-Kaufmann, San Mateo, CA, 1994, 501–508.
7. Casey, M.: "The Dynamics of Discrete-Time Computation with Application to Recurrent Neural Networks and Finite State Machine Extraction" *Neural Computation* (1996), to appear.
8. Blair, A.D., Pollack, J.B.: "Precise Analysis of Dynamical Recognizers". Tech. Rep. CS-95-181, Comput. Sci. Dept., Brandeis Univ. (1996).
9. Elman, J.: "Finding Structure in Time" *Cognitive Science* **14** (1990) 179–211.
10. Castaño, M.A., Casacuberta, F., E. Vidal, E: "Simulation of Stochastic Regular Grammars through Simple Recurrent Networks", in *New Trends in Neural Computation* (J. Mira, J. Cabestany and A. Prieto, Eds.). Lecture Notes in Computer Science **686**, Springer-Verlag, Berlin, 1993, p. 210–215.
11. Carrasco, R.C., Oncina, J.: "Learning Stochastic Regular Grammars by Means of a State Merging Method" in *Grammatical Inference and Applications* (R.C. Carrasco and J. Oncina, Eds.), Lecture Notes in Artificial Intelligence **862**, Springer-Verlag, Berlin, 1994, p. 139–152.
12. Williams, R.J., Zipser, D.: "A learning algorithm for continually running fully recurrent neural networks" *Neural Computation* **1** (1989) 270–280.
13. Forcada, M.L., Carrasco, R.C.: "Learning the Initial State of a Second-Order Recurrent Neural Network during Regular-Language Inference" *Neural Computation* **7** (1995) 923–930.
14. Cover, T.M. and Thomas, J.A.: *Elements of Information Theory*, John Wiley and Sons, New York, 1991.

This article was processed using the LaTeX macro package with LLNCS style