SECOND-ORDER RECURRENT NEURAL NETWORKS CAN LEARN REGULAR GRAMMARS FROM NOISY STRINGS

Rafael C. Carrasco and Mikel L. Forcada
{carrasco,mlf}@dtic.ua.es
Departament de Tecnologia Informàtica i Computació
Universitat d'Alacant, E-03071 Alacant (Spain)

Abstract

Recent work has shown that second-order recurrent neural networks (2ORNNs) may be used to infer deterministic finite automata (DFA) when trained with positive and negative string examples. This paper shows that 2ORNN can also learn DFA from samples consisting of pairs (W, μ_W) where W is a noisy string of input vectors describing the degree of resemblance of every input to the symbols in the alphabet, and μ_W is the degree of acceptance of the noisy string, computed with a DFA whose behavior has been extended to deal with noisy strings.

1 Introduction

A number of recent papers have explored the ability of second-order recurrent neural networks (2ORNNs) to learn simple regular languages[1, 2, 3, 4, 5]. As noted in [1, 2], it is possible to obtain a symbolic representation of the language —*i.e.*, the states and transitions of a deterministic finite automaton (DFA) accepting the language— from the trained network. For this purpose, a complete sample consisting of a collection of strings classified as belonging or not to the language is used to train the network. After training, the network behaves as a DFA, that is, a syntactic pattern recognizer.

However, some empirical situations are better described in terms of properties which are difficult to discretize, but are still tractable under the syntactic pattern recognition paradigm. Ideal patterns consist of primitives in the prototype set (for example, strings are made of symbols from an alphabet), but often times real patterns contain primitives which cannot be, in general, clearly identified with a single prototype. Instead, one has (1) a degree of resemblance between each primitive and each of the prototypes (for instance, a probability that the primitive is a noisy version of that prototype), and (2) a degree of acceptance of the whole pattern in a given pattern class (or language). For example, the first part of a speech recognizer could output a *noisy string* of phonemes (or the so-called microphonemes) representing a spoken word, in which each position holds a vector of probabilities with one component for each phoneme. These noisy strings could be then fed into a neural recognition system that classifies the string as being, for instance, a verb or not a verb with a given degree ranging from 0 to 1.

In this paper, we explore the capability of 2ORNNs to reconstruct deterministic finite automata (DFA) from noisy samples. These samples consist of strings not made of ideal, noiseless symbols from an alphabet, but instead, of vectors describing the degree of resemblance of every input to each symbol in the alphabet. The degree (or probability) of acceptance (as computed using a DFA in an extended way) of each string is given instead of a Boolean value indicating whether the string is accepted or not.

2 Preliminaries

Let $\mathcal{A}$ be an alphabet of size $|\mathcal{A}|$. A noiseless string w of length $|w|$ is a sequence of $|w|$ symbols in the alphabet. A noisy string W of length $|W|$, however, consists of $|W|$ input vectors $\mathbf{I}^{[t]}$ ($1 \leq t \leq |W|$) of dimension $|\mathcal{A}|$. Each component is labeled with a symbol from $\mathcal{A}$: component a of $\mathbf{I}^{[t]}$ has a real value $I_a^{[t]} \in [0,1]$ representing the *degree of resemblance* of the t-th input to class or symbol a in $\mathcal{A}$. The concepts of noisy string concatenation and empty noisy string are analogous to their noiseless counterparts.

When the symbols in the alphabet $\mathcal{A}$ represent disjoint classes (*i.e.* a partition), the restriction that all components add to one,

$$\sum_{a\in\mathcal{A}} I_a^{[t]} = 1 \text{ for all } t, \tag{1}$$

may be imposed. Therefore, when the alphabet is $\mathcal{A} = \{0,1\}$, it suffices to give the 1-component of the vector in each position, as $I_0^{[t]} = 1 - I_1^{[t]}$. For instance, the noisy string $W = (0.8)(0.1)$ indicates that the input at the second position of W is a 1 with degree of resemblance 0.1 (or equivalently, a 0 with degree 0.9).

In all of our experiments we assume that the language to be learned is regular, that is, a DFA contains all the information needed in order to evaluate the degree of acceptance of the noisy strings. Indeed, all the experiments reported in this paper are simulations on strings produced by known DFA.

The following straightforward generalization allows a DFA to work with noisy strings. Let $M = (Q, \mathcal{A}, \delta, q_1, F)$ be a DFA, where Q is the set of states, $\mathcal{A}$ the input alphabet, $q_1 \in Q$ the initial state, $F \subseteq Q$ the subset of accepting states, and $\delta : Q \times \mathcal{A} \to Q$ the transition function. The occupancy of state q after reading the noisy string W will be called $n(q, W)$. When W is the empty noisy string Λ,

$$n(q,\Lambda) = \begin{cases} 1 & \text{if } q = q_1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Once the occupancy of each state after reading a noisy string W is known, one may compute the occupancy of state q after reading one more input I, *i.e.*, after reading string WI:

$$n(q, WI) = \sum_{q'\in Q}\sum_{a\in\mathcal{A}} I_a \; n(q', W) \; \kappa(q', a, q), \tag{3}$$

where

$$\kappa(q', a, q) = \begin{cases} 1 & \text{if } \delta(q', a) = q \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The *degree of acceptance* (or simply *acceptance*) μ_W of a noisy string W in the noiseless language $L = L(M)$ accepted by the automaton M is computed then as follows:

$$\mu_W = \sum_{q\in F} n(q, W). \tag{5}$$

It has to be noted that this formulation is very similar to the kind of *fuzzy automata* defined in [6], but with the product as a norm, the sum as a co-norm, and a crisp initial state.

The noisy-string representation of a noiseless string $w = a^{[1]}a^{[2]}...a^{[l]}$ is $W = \mathbf{I}^{[1]}\mathbf{I}^{[2]}...\mathbf{I}^{[l]}$ where $I_a^{[t]} = 1$ if $a = a^{[t]}$ and 0 otherwise. The definition of degree of acceptance has been chosen so that, for these strings, $\mu_W = 1$ if $w \in L(M)$ and 0 otherwise.

For instance, the set of binary strings with an odd number of ones is accepted by the automaton $M = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$ with $\delta(q_1, 0) = q_1$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 0) = q_2$, $\delta(q_2, 1) = q_1$. As the automaton reads the length-two noisy string $W = (0.8)(0.1)$ (remember that only component I_1 is needed in each position), the occupancies of states evolve as follows: $n(q_1, \Lambda) = 1$, $n(q_2, \Lambda) = 0$; $n(q_1, (0.8)) = 0.2$, $n(q_2, (0.8)) = 0.8$; $n(q_1, (0.8)(0.1)) = 0.26$, $n(q_2, (0.8)(0.1)) = 0.74$. The degree of acceptance of the string is then $\mu_{(0.8)(0.1)} = 0.74$. Clearly, the noisy-string representation of a noiseless 011 would be $(0)(1)(1)$ and its acceptance, $\mu_{(0)(1)(1)} = 0$.

A noisy sample over the binary alphabet will consist of pairs (W, μ_W) where μ_W is the degree of acceptance of W in the language, and W is given as a sequence of resemblance vectors.

3 Architecture and training

For our experiments we have used a second-order recurrent neural network as in [1, 2, 3, 7]. The basic features of the architecture are shown in fig. 1. The network has N hidden recurrent neurons plus $|\mathcal{A}|$ input neurons; hidden neuron number 1 is arbitrarily chosen to represent the degree of acceptance. In a way that parallels the function of a DFA, the states $S_i^{[t]}$ of the hidden neurons ($1 \leq i \leq N$) after reading t inputs depend on the states $S_j^{[t-1]}$ ($1 \leq j \leq N$) of the hidden neurons after reading $t-1$ inputs and the values of the $|\mathcal{A}|$ components of the t-th input vector $\mathbf{I}^{[t]}$, as presented through each one of the input neurons. The products (hence the name *second-order*) of the states of each input and each hidden neuron are weighted, summed, and mapped through the sigmoid function $g(x) = 1/(1 + \exp(-x))$ to determine the new states of the hidden neurons:

$$S_i^{[t]} = g(\sigma_i^{[t]}), \tag{6}$$

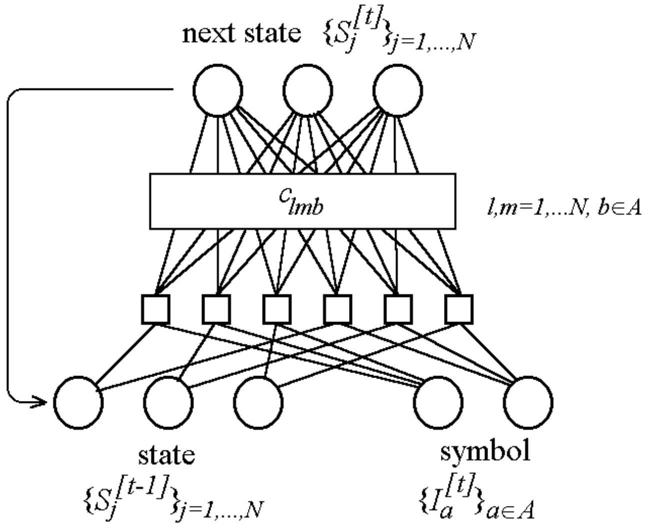

Figure 1: The architecture of the recurrent network

with

$$\sigma_i^{[t]} = \sum_{j=1}^{N} \sum_{a\in\mathcal{A}} C_{ija}\; S_j^{[t-1]} I_a^{[t]}, \tag{7}$$

where C_{ija} $(i,j = 1,\ldots,N,\ a \in \mathcal{A})$ are the real-valued weights. After $|W|$ iterations, a noisy string W has been completely processed and the degree of acceptance is given by $S_1^{[|W|]}$ (recall that, due to the sigmoid function, the values of all states are always in the range $[0,1]$).

The network is completely defined by the set of weights $\{C_{ija}\}$ and the set of initial activations of the hidden neurons, *i.e.*, the set $\{\sigma_i^{[0]}\}$ before any input[1]. Optimization of the $N^2|\mathcal{A}|+N$ parameters characterizing the network is the goal of the learning process, and proceeds basically as described in [1, 2, 7]. At the very beginning, the initial value of every parameter is selected randomly, but this is only done once during the whole learning process. From that point on, the values are changed, trying to minimize the error function

[1]The values of $\sigma_i^{[0]} = g^{-1}(S_i^{[0]})$ are preferred to the actual state values $S_i^{[0]}$ because, unlike the latter, they are not restricted to any range of values. This simplifies the optimization of the whole set of parameters[7].

for the learning sample $\mathcal{S}$

$$E = \frac{1}{2} \sum_{W \in \mathcal{S}} \left(S_1^{[|W|]} - \tilde{\mu}_W \right)^2 \tag{8}$$

where $\tilde{\mu}_W$ is the customary projection of $\mu_W \in [0, 1]$ onto the range $[\epsilon, 1 - \epsilon]$ (we take $\epsilon = 0.1$). The search of the minimum is simply accomplished by means of gradient descent[1, 2, 7] using an adapted version of the RTRL method[8]. The sample $\mathcal{S}$ is repeatedly fed and, after each noisy string is completely processed, weights and initial states are updated:

$$\Delta P = -\alpha \frac{\partial E}{\partial P} + \eta \Delta P \tag{9}$$

where P stands for any of the parameters C_{ija} or $\sigma_i^{[0]}$. On the right-hand side of equation (9), ΔP represents the change computed in the previous iteration; α is the *learning rate* and η is called the *learning momentum*. The derivatives

$$\frac{\partial E}{\partial P} = \left(S_1^{[|W|]} - \mu_w \right) g'(\sigma_1^{[|W|]}) \frac{\partial \sigma_1^{[|W|]}}{\partial P} \tag{10}$$

are recursively calculated:

$$\frac{\partial \sigma_i^{[t]}}{\partial C_{lmb}} = \delta_{il} S_m^{[t-1]} I_b^{[t]} + \sum_{j=1}^{N} \sum_{a \in \mathcal{A}} C_{ija} \, g'(\sigma_j^{[t-1]}) \frac{\partial \sigma_j^{[t-1]}}{\partial C_{lmb}} I_a^{[t]} \tag{11}$$

(where δ_{il} is the Kronecker delta) for $P = C_{lmb}$, and

$$\frac{\partial \sigma_i^{[t]}}{\partial \sigma_l^{[0]}} = \sum_{j=1}^{N} \sum_{a \in \mathcal{A}} C_{ija} I_a^{[t-1]} g'(\sigma_j^{[t-1]}) \frac{\partial \sigma_j^{[t-1]}}{\partial \sigma_l^{[0]}} \tag{12}$$

for $P = \sigma_j^{[0]}$, with the initial values:

$$\frac{\partial \sigma_j^{[0]}}{\partial C_{lmb}} = 0 \tag{13}$$

and

$$\frac{\partial \sigma_i^{[0]}}{\partial \sigma_j^{[0]}} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

for $i, j, l, m = 1, \ldots, N$ and $a, b \in \mathcal{A}$.

4 Experiments

Samples of noisy strings over $\mathcal{A} = \{0, 1\}$ have been generated for three regular languages:

- L_1, the strings having an odd number of ones: $0^*1(0 + 10^*1)^*$;
- L_2, the binary numbers that are multiples of three: $(0 + 1(01^*0)^*1)^*$;
- and L_3, Tomita's[9] fourth language (strings not containing 000 as a substring): $(1 + 01 + 001)^*(\epsilon + 0 + 00)$.

The strings in the samples are generated as follows: first, a length in the range 1 to 10 is randomly selected; then a noisy string of that length with totally random input vectors (a very noisy string) is built, and finally, its degree of acceptance is computed (as described in section 2) for the target automaton.

The network is first trained with a small subset (40 strings) of the sample. Every 100 iterations, a new group (40 more strings) is added to the training set. The process stops when either convergence is reached or the sample is exhausted. The learning rate and momentum are $\alpha = 4.0$ and $\eta = 0.5$. The training process is said to converge when the DFA extracted from the network computes exactly the degree of acceptance of every string in the sample (in practice, we checked that the extracted DFA was equivalent to the one used to generate the sample). DFA extraction is carried out by running noiseless strings through the network to form the set of "pure" states, and then extracting the DFA by a method similar to the one described in [1, 2].

A network having $N = 2$ hidden units was sufficient to learn L_1. Ten runs were performed; a fresh sample of 400 strings and a random set of values for all learnable parameters were used for each run. Convergence was reached after presentation of 40 strings (the initial set) in 3 cases and after 240 strings in 7 cases. That gives an average of 180 strings.

For L_2, a larger network was needed, $N = 3$. Nineteen out of twenty runs converged, using 400-string samples. The number of strings needed were: 80 (in 5 runs), 120 (6 runs), 160 (3 runs), 200 (2 runs), 240 (1 run), 320 (1 run), and 400 (1 run). This shows that a larger number of strings may sometimes be needed to learn this, more complex language. Figure 2 shows two views of the hidden-neuron state space for this network. The points

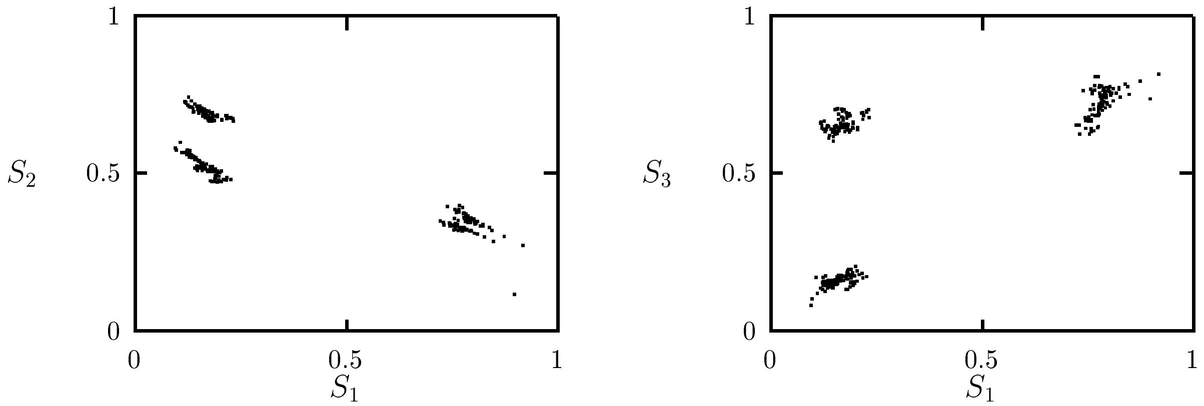

Figure 2: Clusters in the state space of a three-hidden-neuron 2ORNN after training with L_2 (see text).

visited by the network as it processes the 255 noiseless strings of length up to (and including) 7 are clearly shown to form three clusters corresponding to the states of the minimal automaton that accepts the language. Two of the states are non-accepting (low values of S_1 and the other is an accepting state (high value of S_1).

Learning L_3 is harder. A network with $N = 3$ was unable to learn the DFA from 400-string samples in a preliminary suite of ten runs. The training of a $N = 4$ network on 400-string samples converged only in four runs out of twenty; the number of strings needed were 240 in three cases and 280 in the fourth case. The results clearly indicated the need for a larger sample. In a test suite using 800-string sets, the training converged in four out of ten runs, having used 760, 800, 200 and 800 strings.

5 Concluding remarks

We have shown that second-order recurrent neural networks (2ORNN) can learn simple deterministic finite automata from rather small samples consisting of pairs of very noisy strings and their degrees of acceptance, computed according to an extension of the behavior of a DFA to noisy strings. This learning task may be related to many interesting, practical syntactic pattern recognition problems, and happens to be more difficult than the customary task of inferring DFA from noiseless strings. These experiments with recurrent neural networks presented in this paper give a clear indication about the learnability of finite-state recognizers from noisy samples and a motivation to search for other language-theoretical methods (such as [10]) to accomplish the same learning task.

Acknowledgements: The authors wish to thank the Dirección General de Investigación Científica y Técnica of the Government of Spain for support through project CICYT/TIC93-0633-C02.

References

[1] Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., and Lee, Y.C. (1992a) "Learning and extracting finite state automata with second-order recurrent neural networks" *Neural Computation* **4**, 393–405.

[2] Giles, C.L., Miller, C.B., Chen, D., Sun, G.Z., Chen, H.H., and Lee, Y.C. (1992b) "Extracting and learning an unknown grammar with recurrent neural networks", *Advances in Neural Information Processing Systems*, vol. 4 (J. Moody *et al.*, eds; Morgan-Kaufmann, San Mateo, Calif., U.S.A.), 317–324.

[3] Siegelmann, H.T., Sontag, E.D., and Giles, C.L. (1992) "The complexity of language recognition by neural networks" *Information Processing 92*, vol. 1 (Elsevier/North-Holland), p. 329–335.

[4] Watrous, R.L. and Kuhn, G.M. (1992a) "Induction of Finite-State Automata Using Second-Order Recurrent Networks", *Advances in Neural Information Processing Systems*, vol. 4 (J. Moody *et al.*, eds; Morgan-Kaufmann, San Mateo, Calif., U.S.A.), 306–316.

[5] Watrous, R.L. and Kuhn, G.M. (1992b) "Induction of Finite-State Languages Using Second-Order Recurrent Networks", *Neural Computation* **4**, 406–414.

[6] Steimann, F. and Adlassnig, K.-P. (1994) "Clinical monitoring with fuzzy automata", *Fuzzy Sets and Systems* **61**, 37–42.

[7] M.L. Forcada and R.C. Carrasco (1994) "Learning the initial state of a second-order recurrent neural network during regular-language inference", *Neural Computation*, in press.

[8] Williams, R.J. and Zipser, D. (1989) "A learning algorithm for continually running fully recurrent neural networks" *Neural Comp.* **1**, 270.

[9] Tomita, M. (1982) "Dynamic construction of finite-state automata from examples, using hill-climbing" *Proceedings of the Fourth. Annual Cognitive Science Conference*, (Ann Arbor, Mich., U.S.A.) p. 105–108

[10] Carrasco, R.C. and Oncina, J. (1994) "Learning stochastic regular grammars by means of a state merging method", in *Grammatical Inference and Applications, Proc. of the 2nd. Intl. Colloq. on Grammatical Inference ICGI-94 (Alicante, Spain, September 1994)* (Carrasco, R. and Oncina, J., eds.) Lecture Notes in Artificial Intelligence 862 (Springer-Verlag) p. 139–152.