Learning the initial state of a second-order recurrent neural network during regular-language inference

Mikel L. Forcada, Rafael C. Carrasco

Departament de Tecnologia Informàtica i Computació,
Universitat d'Alacant,
E-03071 Alacant, Spain.
E-mail: {mlf,carrasco}@vm.cpd.ua.es

Abstract

Recent work has shown that second-order recurrent neural networks (2ORNNs) may be used to infer regular languages. This letter presents a modified version of the real-time recurrent learning (RTRL) algorithm used to train 2ORNNs, that learns the initial state in addition to the weights. The results of this modification, which adds extra flexibility at a negligible cost in time complexity, suggest that it may be used to improve the learning of regular languages when the size of the network is small.

To appear in *Neural Computation*, vol. 7 (1995).

Introduction: A number of recent papers (Giles *et al.* 1992a,b; Siegelmann, Sontag and Giles, 1992; Watrous and Kuhn 1992a,b) have explored the ability of second-order recurrent neural networks (2ORNNs) to learn simple regular grammars (that is, with languages accepted by small automata) from positive and negative training word samples. This letter presents a modified version of the real-time recurrent learning (RTRL) algorithm used by these authors (except Watrous and Kuhn 1992a,b, where the backpropagation through time (BPTT) method was used), which in turn is based in a training method by Williams and Zipser (1989). Instead of using a randomly-selected or fixed initial state, the new model learns the initial state in addition to the weights. This adds flexibility to the learning process at a negligible cost in time, as shown by the preliminary results presented.

We are currently working on the extension of this modification to other recurrent network architectures.

The original model: A summary of the model used by Giles *et al.* (1992a,b) follows. The architecture used is that of a second-order recurrent neural network, with N hidden, recurrent neurons (with states labeled S_j) and L input, nonrecurrent neurons (with states labeled I_k) for character input (see figure 1). The network reads one character per cycle. The product of each hidden neuron state with each input neuron state, $S_j I_k$ is fed, modified by weight W_{ijk} to hidden neuron i in each cycle. This represents a transition of the underlying deterministic automaton: the network goes from state q ($\{S_j\}_{j=1,\ldots,N}$) to state q' ($\{S'_j\}_{j=1,\ldots,N}$) after reading symbol a ($\{I_k\}_{k=1,\ldots,L}$). The dynamics is given by the equation:

$$S_i^{(t)} = g(\Xi_i^{(t-1)}), \tag{1}$$

where

$$\Xi_i^{(t)} = \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)}. \tag{2}$$

Function g is the sigmoid $1/(1+\exp(-x))$. Characters are usually (but not necessarily) encoded in a *one-hot* or *unary* fashion: each input neuron corresponds to a character.

Giles *et al.* (1992a,b) train their network using a second-order version of Zipser and Williams' (1989) Real Time Recurrent Learning (RTRL) algorithm. The state of a preselected hidden neuron, call it $S_{\rm acc}$, is chosen to represent acceptance ($S_{\rm acc} \in [1-\tau, 1]$) or rejection ($S_{\rm acc} \in [0, \tau]$). The target values $\hat{S}_{\rm acc}$ are chosen to be $1-\tau/2$ and $\tau/2$ respectively; the usual choice is $\tau = 0.2$. The error in the state of the acceptance neuron, $E_{\rm acc} = (S_{\rm acc}^{(f)} - \hat{S}_{\rm acc})^2/2$ after reading a word of length f is minimized by varying the weights W_{ijk} using gradient descent:

$$\Delta W_{lmn} = -\alpha(S_{\rm acc}^{(f)} - \hat{S}_{\rm acc})(\partial S_{\rm acc}^{(f)}/\partial W_{lmn}) + \eta \Delta W'_{lmn}\,, \tag{3}$$

with a learning rate α and a *momentum* term η; $\Delta W'_{lmn}$ stands for the previous value of ΔW_{lmn}. This requires the evaluation of all the $\partial S_i^{(t)}/\partial W_{lmn}$ derivatives, which is computationally intensive, and is carried out in a recurrent way in each symbol step t, based on the values obtained in symbol step $t-1$, with t varying from 1 to f, where f is the length of the word.

The recurrent formula is

$$\frac{\partial S_i^{(t)}}{\partial W_{lmn}} = g'(\Xi_i^{(t-1)}) \left[\delta_{il} S_m^{(t-1)} I_n^{(t-1)} + \sum_{j,k} W_{ijk} \frac{\partial S_j^{(t-1)}}{\partial W_{lmn}} I_k^{(t-1)} \right], \qquad (4)$$

where g' is the derivative of the sigmoid function, δ_{il} is Kronecker's delta ($\delta_{il} = 1$ if $i = l$ and zero otherwise), and

$$\partial S_i^{(0)} / \partial W_{lmn} = 0. \qquad (5)$$

In the original model of Giles *et al.* (1992a,b) the initial states of the hidden neurons S_i^0 are fixed (either randomly chosen from the interval [0.0,1.0] or taking $S_i^0 = \delta_{i0}$) for each learning run. The initial weight set is also a set of small random values around 0.0 (both positive and negative), unique for each job.

Each update of the whole set of derivatives in real time has a time complexity of $O(N^4L^2)$; however, due to the *one-hot* codification of inputs, it reduces to $O(N^4L)$. The space needed to store the derivative information is $O(N^3L)$.

In most cases, Giles *et al.* (1992a,b) 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, and the transitions ocurring among these clusters as symbols are read determine the transition function of the DFA. The automaton extraction algorithm uses a partition of the configuration hypercube $[0.0, 1.0]^N$ to locate these clusters.

The modified model: The new model learns the initial state of all hidden neurons in addition to the weights, at almost no extra cost. The gradient-descent formulas for learning the initial state are then

$$\Delta S_j^{(0)} = -\alpha (S_{\mathrm{acc}}^{(f)} - \hat{S}_{\mathrm{acc}}) \frac{\partial S_{\mathrm{acc}}^{(f)}}{\partial S_j^{(0)}} + \eta \Delta S'^{(0)}_j, \qquad (6)$$

with $\Delta S'^{(0)}_j$ being the previous value of $\Delta S_j^{(0)}$, and

$$\frac{\partial S_i^{(t)}}{\partial S_l^{(0)}} = g'(\Xi_i^{(t-1)}) \sum_{j,k} W_{ijk} \frac{\partial S_j^{(t-1)}}{\partial S_l^{(0)}} I_k^{(t-1)}; \qquad (7)$$

the parameters α and η have the same meaning as in the weight updating formulas (3-5); their values may be different in the initial-state updating formulas, but, to avoid parameter proliferation, we have chosen to use the same values for the learning of weights and initial states[1]. The initial values of the derivatives are

$$\frac{\partial S_i^{(0)}}{\partial S_l^{(0)}} = \delta_{il}. \tag{8}$$

The initial values of the initial states $S_i^{(0)}$, before any learning takes place, are randomly chosen and unique for each run. If the empty word is in the training set, instead of applying the update rule in equation (6), which would only modify the value of $S_{\rm acc}^{(0)}$, we choose to use more efficiently this information to accelerate convergence by updating directly the initial state of the acceptance neuron: each time the empty word is presented for learning, we simply set the state of $S_{\rm acc}^{(0)}$ to $1-\tau$ if the word is in the positive example set or to τ if it is in the negative example set, and leave all the other $S_j^{(0)}$ unchanged. For non-empty words, whenever the value of an updated $S_j^{(0)}$ falls outside the $[0.0, 1.0]$ interval, it is automatically *clipped* so that it falls within the valid range[2].

The increase in space complexity ($O(N^2)$ is added to $O(N^3L)$) and time complexity ($O(N^3L)$ is added to $O(N^4L^2)$ in the general case, and for *one-hot* encoding, only $O(N^3)$ is added to $O(N^4L)$) due to this modification are both negligible, and therefore impose no burden on the learning process while allowing for increased flexibility. The automaton extraction used by Giles *et al.* (1992a,b) may be used without modification in our new model, since the clustering of states is not dependent on the fact that the initial state is also learned.

Learning the *odd-number-of-ones* language: Tables 1 and 2 show preliminary results of the new model compared with the original model; the grammar chosen for these experiments is the *odd-number-of-ones* language on $\Sigma = \{0, 1\}$, which is recognized by a simple two-state DFA. The training set (positive and negative examples) consisted of the 1023 words of Σ^* that

[1] The optimum (fastest learning) values may actually be different

[2] As one of the referees has pointed out, this clipping is not needed if the initial state is treated as the output of a sigmoid and the input of this sigmoid (in the range $]-\infty, \infty[$) is learned instead. This approach is conceptually more elegant, may be implemented at no extra computational cost, and has been found to give equivalent results in test runs. We plan to use it from now on.

have length 9 or less. The learning process starts with 63 randomly chosen words, and adds 16 words to the set each time the current learning set is completely classified.

The maximum number of epochs (an *epoch* is the basic learning unit, corresponding to a one pass with the current learning set) was set to 1000, the learning rate α to 2.0, and the learning momentum η to 0.2. The network was trained with $\tau = 0.1$, that is, to accept with a target value $\hat{S}_{\rm acc}$ of 0.95 and to reject with a target value of 0.05; errors below τ were taken as successes. According to the prescriptions explained above, the initial values for the initial states of the hidden neurons and for the weights were random and unique for each run and model. Note that the value used for τ, 0.1, is substantially smaller than the one used by Giles *et al.* (1992a,b): $\tau = 0.4$, target values, 0.8 for acceptance, 0.2 for rejection. We have found that, for this particular grammar, we either had to use a very small value of τ or had to update the weights after *all* words in the training set, misclassified or not, contrary to the "update only when misclassified" rule used by Giles *et al.* (1992a,b) in order to achieve learning in a reasonable number of epochs. This behavior has not been observed for other grammars.

Table 1 shows convergence results for a network with 2 hidden neurons and 2 or 3 input neurons. The third neuron is for an end-of-word symbol, used in the original model to compensate for the fact that an inadequate choice of the fixed initial state may preclude a correct classification of the empty word and make the network too rigid. This way we also avoid giving the new model an unfair advantage in the comparison. As may be seen, while the original model fails to learn the grammar in 1000 epochs in eleven out of eleven runs, our modified model learns it succesfully in seven out of eleven runs. This is mainly due to the fact that, in the new model, the initial state may move around in state space and reach a favorable position, whereas in the old model, the initial state is fixed and additional transitions are needed take care of the end-of-word symbol. When the number of neurons (and therefore, the dimensionality of the state space) is limited, the additional adaptiveness of the new model improves the chances that the grammar is learned.

Table 2 shows results for 3 hidden neurons. In this case, the old model reaches convergence in 1000 epochs only in two out of eleven runs, and the new model in three out of eleven. Apparently, the effect of the modification in the model is much smaller in this case. Here, three neurons is enough to accomodate both kinds of automata, and the effect of learning the initial state is almost negligible (an indication of this is given by the fact that some of the automata extracted using the method described by Giles *et al.* (1992a,b) were not the minimal, two state automaton, as was the case for all the runs made with 2 hidden neurons).

Learning Tomita's 4th grammar: Tomita's (Tomita 1982) 4th grammar (for strings on $\{0,1\}$ not containing "000" as a substring) is a typical test grammar used in studies of regular grammar inference. The minimal DFA has four states. Giles *et al.* (1992) used this grammar to test the capability of 2ORNNs to learn regular grammars. Their experiments included networks with 3, 4, and 5 hidden neurons.

Table 3 shows the results of learning this grammar with a network with three hidden neurons, which seems to be the minimum number needed. The test set consists of the 1023 strings having length 9 or less, with the same word addition scheme; the maximum number of epochs is set to 1000, the value of τ is 0.4, corresponding to the target values 0.2 and 0.8 for rejection and acceptance respectively, and the learning parameters are $\alpha = 2.0$ and $\eta = 0.2$, the same as in the previous runs. The results show that the new model, where the initial state is also learned, reaches convergence more easily than the original model, which failed to converge in 1000 epochs in nine out of twenty-two runs as compared to three out of twenty-two for the new model. Even after eliminating the runs that failed in both cases, the average number of epochs is somewhat smaller for the new model (355 versus 447).

It may be argued that, again, the network is too small to accomodate easily the possible additional DFA transitions and states that may be due to the presence of an end-of-string symbol, but easily accomodates the smaller DFA's inferred by the new model.

Table 4 shows the results of learning Tomita's 4th grammar with a network with four hidden neurons, one more than in the previous test set. All the other parameters are the same. As could be expected for a larger network where both kinds of automata may be accomodated easily (non-minimal automata are sometimes inferred), the new model does not make a big difference. The original model fails to learn the grammar in 1000 epochs in three out of twenty runs, whereas the new model fails in five runs. The average number of epochs, not taking the failed jobs into account, is: old, 375; new, 235, a fact that somehow compensates for the higher number of failures by the new model.

Concluding remarks: The results presented show that, for the new model introduced in this letter, the learning process is faster when the recurrent network is small for a given grammar (when the number of neurons is larger, the speed is not appreciably affected by the improvement). Indeed, the modification may sometimes be critical to achieve learning in a reasonable number of epochs. This occurs at a negligible increase in time complexity in most cases. An extension of this modification to other recurrent architectures and a more thorough experimental suite to assess its effect on the learning process are under way.

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

- Giles, C.L.; Miller, C.B; Chen, D.; Chen, H.H.; Sun, G.Z.; Lee, Y.C. (1992a) "Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks" *Neural Computation* **4**, 393–405.

- Giles, C.L.; Miller, C.B; Chen, D.; Sun, G.Z.; Chen, H.H.; 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.), 317–324.

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

- Tomita, M. (1982) "Dynamic construction of finite-state automata from examples, using hill-climbing" *Proc. Fourth. Annu. Cogn. Sci. Conf*, p. 105.

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

- Watrous, R.L.; 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.), 306–316.

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

Initial state	Number of epochs for 11 consecutive runs
Learned	925, (failed), 109, 122, (2 failed), 314, 863, 136, 109
Fixed	(all jobs failed)

Table 1: Learning the *odd-number-of-ones* grammar with a 2ORNN having 2 hidden neurons, with and without optimization of the initial state.

Initial state	Number of epochs for 11 consecutive runs
Learned	(failed), 128, (3 failed), 107, (3 failed), 737, (failed)
Fixed	(4 failed), 214, 363, (5 failed)

Table 2: Learning the *odd-number-of-ones* grammar with a 2ORNN having 3 hidden neurons, with and without optimization of the initial state.

Initial state	Number of epochs for 22 consecutive runs
Learned	221, 501, (failed), 237, 247, 224, 520, 399, (failed), 349, 896 576, 180, (failed), 293, 540, 205, 181, 179, 573, 195, 227
Fixed	873, 455, 302, 360, (failed), 396, 231, 764, 587, (failed), 321 (4 failed), 370, (failed), 297, (failed), 238, 621, (failed)

Table 3: Learning the Tomita-4 grammar with a 2ORNN having 3 hidden neurons, with and without optimization of the initial state.

Initial state	Number of epochs for 22 consecutive runs
Learned	196, (failed), 137, 291, 169, (failed), 340, 286, 163, 276, (failed) 341, 151, 208, (failed), 296, (failed), 266, 290, 258, 346, 377
Fixed	937, 240, 211, (failed), 270,, 406, 314, 254, 281, 576, 183, 178, 322, 320, 320, 908, 525, (2 failed), 278, 476, 169

Table 4: Learning the Tomita-4 grammar with a 2ORNN having 4 hidden neurons, with and without optimization of the initial state.

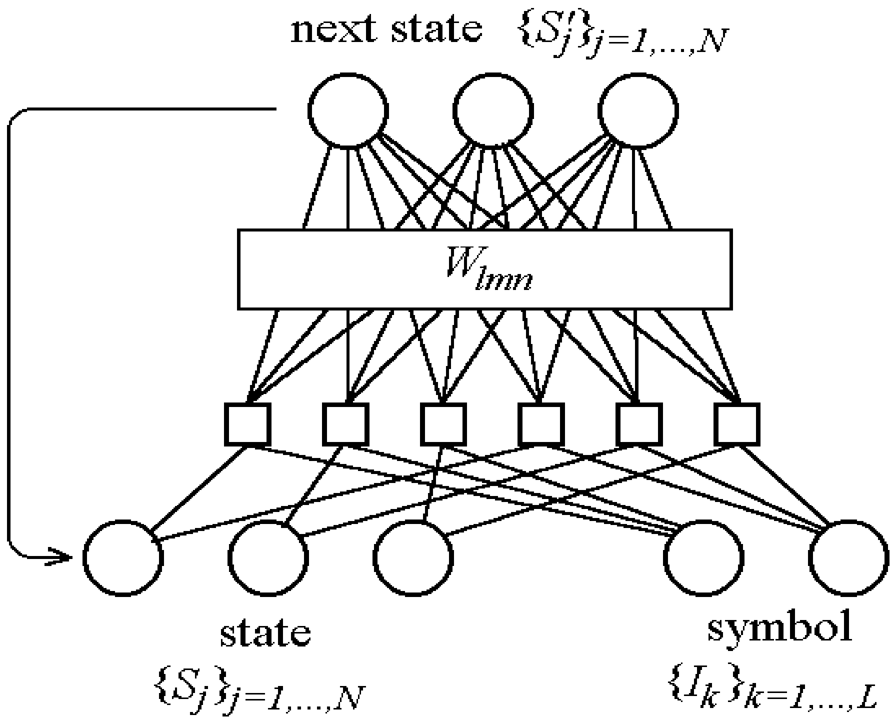

Figure 1: The architecture of the second-order recurrent neural network of Giles *et al.* (1992)