Constrained Second-Order Recurrent Networks for Finite-State Automata Induction

Stefan C. Kremer
Computing and Information Science Department,
University of Guelph, Guelph, Ontario, N1G 2W1, Canada
E-mail: `skremer@snowhite.cis.uoguelph.ca`

Ramón P. Ñeco, Mikel L. Forcada
Departament de Llenguatges i Sistemes Informàtics,
Universitat d'Alacant, E-03071 Alacant, Spain.
E-mail: {`neco,mlf`}`@dlsi.ua.es`

Abstract

This paper presents an improved training algorithm for second-order dynamical recurrent networks applied to the problem of finite-state automata (FSA) induction. Second-order networks allow for a natural encoding of finite-state automata in which each second-order connection weight corresponds to one transition in a finite-state automaton. In practice, however, when trained using gradient descent, these networks seldom assume this type of encoding and sophisticated algorithms must be used to extract the encoded automata. This paper suggests a simple modification to the standard error function for second-order dynamical recurrent networks which encourages these networks to assume natural FSA encodings when trained using gradient descent. This obviates the need for cluster-based extraction techniques and provides a simple method for guaranteeing the stability of the network for arbitrarily long sequences. Initial results also suggest that fewer training strings must be presented to achieve convergence using the modified error.

1 Introduction

A number of researchers have trained discrete-time recurrent neural networks (DTRNN) to behave like deterministic finite-state automata (DFA) (see, e.g., [1, 2]. The internal representation of learned DFA states can deteriorate for long strings, due to the use of real-valued sigmoid functions which never reach the bounds[3]. In this case, we say that the network is *unstable*. For this reason, it is difficult to make predictions about the generalization performance of trained DTRNN. As a partial solution to this problem, some authors force learning to occur such that state values form clusters [3, 4]. In this paper we show a slightly modified learning algorithm with respect to that used in [1] to learn DFA using second-order DTRNN, which obviates the need for cluster-based extraction of the DFA, and obtain a more stable behavior for long strings.

2 Encoding DFA in DTRNN

2.1 Definitions

Deterministic finite-state automata (DFA): Formally a DFA M is a 5-tuple, $M = (Q, \Sigma, \delta, q_I, F)$, where Q is a set of $|Q|$ states labelled $q_1, ..., q_{|Q|}$; Σ is a set of input symbols labelled $\sigma_1, ..., \sigma_{|\Sigma|}$; δ is the state transition function, $\delta : Q \times \Sigma \to Q$; q_I is the initial state, and F is the set of accepting states. A string is accepted by the DFA M if an accepting state is reached; otherwise, the string is rejected.

Discrete-time recurrent neural networks (DTRNN): A DTRNN [5] can be defined in a way that is parallel to the above definition of DFA [2]. A DTRNN is a 6-tuple $N = (X, U, Y, \mathbf{f}, \mathbf{h}, \mathbf{x}_0)$, where $X = [S_0, S_1]^{n_X}$ is the state space, with S_0 and S_1 the values defining the range of outputs of the transfer functions, and n_X the number of state units; $U = \mathcal{R}^{n_U}$ is the set of possible input vectors, with n_U the number of input lines; $Y = [S_0, S_1]$ is the output space of the network (one output unit); $\mathbf{f} : X \times U \to X$ is the *next-state function*, which computes a new state $\mathbf{x}[t]$ from the previous state $\mathbf{x}[t-1]$ and the input just read $\mathbf{u}[t]$; $\mathbf{h} : X \times U \to Y$ is the output function; and finally $\mathbf{x}_0$ is the initial state. In this paper we use a second-order DTRNN (2ODTRNN) [1, 2] such that the i-th coordinate of the next-state function is : $\mathbf{f}_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{ijk}^{xxu} x_j[t-1] u_k[t]\right)$, and the output is: $y = \mathbf{h}(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{jk}^{yxu} x_j[t-1] u_k[t]\right) \in Y$ where $g(x) = 1/(1+e^{-x})$. All weights in the above equation are real numbers.

2.2 Encoding

An advantage of 2ODTRNN over their first-order counterparts is that the former allow a one-to-one mapping between the states of an automaton and the processing units in the network (see [6]) whereas the latter do not (see [6]). To encode a given DFA, a 2ODTRNN is created which takes input vectors equal in dimensionality to the number of input symbols. In particular, we assume that component $u_k[t]$ of each input vector is equal to 1 if the input symbol at time t is σ_k and 0 for all other input symbols (*one-hot* encoding). We will use a similar encoding to map automaton states to DTRNN unit activations. We say that, if after presentation of a string, the activation value of the output unit y is close to 1, then the string is accepted, while if it is close to 0, the string is rejected. We use the *state units*, to represent the state of the automaton using a one-hot encoding; that is, $\mathbf{x}_i[t]$ is high if the state of the automaton at time t is q_i and low otherwise. Thus, the DTRNN will have $n_X = |Q|$ state units. The initial state is represented by unit 1. The objective of the encoding scheme used in this paper is to ensure that when the current input symbol and state are encoded in the network as defined above, the operation of the network guarantees that the next activation vector corresponds to the next state of the automaton. To ensure this operation by the network, we require that

$$W_{ijk}^{xxu} = \begin{cases} +H & \text{if } \delta(q_j, \sigma_k) = q_i \\ -H & \text{otherwise} \end{cases} \qquad W_{jk}^{Yxu} = \begin{cases} +H & \text{if } \delta(q_j, \sigma_k) \in F \\ -H & \text{otherwise} \end{cases}$$

where $H > 0$. It is easy to prove that a second-order DTRNN of size $n_X = |Q|$ may easily emulate a DFA of size $|Q|$ using the encoding defined above for certain values of H. The proof presented here is a special case of a more general scheme described in [8] and [9]. To our knowledge, this construction using no biases and weights that are either $-H$ or H has never been considered before. We define that a state vector $\mathbf{x}$ of the network is a an ϵ-*valid* representation of state q_i for some ϵ in $(0, 0.5)$ if: (1) the state x_i of the unit representing state q_i is *high*, that is, within the interval $[1-\epsilon, 1]$; (2) the state of all the other units representing state is *low*, that is, within the interval $[0, \epsilon]$; and (3) the state of the output unit is high if $q_i \in F$ and low otherwise. As can be seen, there is a symmetric forbidden interval $(\epsilon, 1-\epsilon)$ which is inaccessible to ϵ-valid representations.

Then, given the following conditions: (1) the initial state of the network is an ϵ-*valid representation* of the initial state of the DFA; (2) the activation function is $g(x) = 1/(1+\exp(-x))$; (4) the weights are chosen as defined above; then, for every automaton of size $|Q|$ there is always a set of pairs (H, ϵ) such that the DTRNN always assumes valid representations of state regardless of the length of the input string, and it is always possible to find the minimum value of H by varying ϵ.

The proof proceeds by induction on the length of strings. The base case (zero length) is guaranteed (independently ϵ) by the choice of initial state made above. And for the induction step, we want to find conditions on the pair (H, ϵ) such that the induction step holds, that is, that a network in an ϵ-valid state moves to the corresponding ϵ-valid state after reading any input symbol. Let us assume, without loss of generality, that q_j is the state before reading symbol σ_k and that $\delta(q_j, \sigma_k) = q_i$. That is, the state before reading the symbol is an ϵ-valid representation of q_j, in particular the following worst (weakest) case, when $\mathbf{x}_j[t-1]$ is $1-\epsilon$ and all other $\mathbf{x}_l[t-1]$ with $l \neq j$ are all ϵ. After reading symbol σ_k we have to make sure that: (1) $\mathbf{x}_i[t]$ is high; (2) all $\mathbf{x}_n[t]$ with $n \neq i$ are low; (3) the state of the acceptance neuron is high if $q_i \in F$ and low otherwise.

Ensuring a high value for $\mathbf{x}_i[t]$ in the worst case leads to the following inequality: $1 - \epsilon \leq g\left(H - |Q|H\epsilon\right)$. Ensuring a low value for $\mathbf{x}_n[t]$ $(i \neq n)$ is most difficult in the following special case: when, for all, $l \neq j$ $\delta(q_l, \sigma_k) = q_n$. In that case, the following inequality has to hold: $\epsilon \geq g\left(-H + |Q|H\epsilon\right)$. Both inequalities are equivalent, since $g(x) = 1 - g(-x)$ and $g(x)$ is a strictly growing function. Identical inequalities are obtained by considering the behavior of the state of the acceptance unit. The corresponding equation relating H and ϵ, $\epsilon = g(-H + H|Q|\epsilon)$ ensures a minimum H for a given ϵ, which can be found by taking $dH/d\epsilon = 0$ and solving iteratively the resulting equation.

The resulting pair (H, ϵ) ensures the validity of the induction step, and so, the correct behavior (validly represented states) of the DTRNN for strings of any length. Example values of H are 2, 3.113, 3.589, 3.922, 6.224 for $|Q| = 2, 3, 4, 5, 30$.

3 Training Algorithm

We use a modification to the error used in the gradient descent algorithm which encourages the DTRNN to adopt the above encoding. More precisely, we augment the standard performance error measure with an encoding error measure that evaluates the difference between the network's current weights and the above encoding. This causes the gradient descent algorithm to favor these desirable (stable and easily interpretable) encodings over less favorable encodings while simultaneously attempting to match the desired outputs.

We identify two desired criteria for our encodings: (1) all weights in the network should approach $+H$ or $-H$ and, since we are interested only in DFA, (2) only one state unit should be high for a given input and current state. The second criterion can also be specified by saying that for each state unit i, input unit pair, (j, k), for each value of i, only one of the weights W_{ijk} should be positive. We can measure the degree to which a network satisfies these criteria by two simple metrics that measure an encoding error.

We define the error of the network as $E = \alpha E_{\text{perf}} + \beta E_H + \gamma E_1$, where E_{perf} represents the standard RMS performance error measuring the difference between the actual and desired output. $E_H \equiv \frac{1}{4}\sum_{ik}\sum_{j\neq 0}(W_{ijk}^2 - H^2)^2$ is minimized when all connection weights assume a value $-H$ or $+H$. $E_{1+} \equiv \frac{1}{2}\sum_{j\neq 0}\sum_k(\sum_{i\neq 0} W_{ijk} - H(3 - N_{\text{SODRN}}))^2$ is minimized when only one of the connection weights leading away from a given state and input unit is $+H$ and the others are all $-H$ and is zero when exactly one of the $|Q|$ weights leading out of each state-input pair, (j, k) is H, and all of the other $|Q| - 1$ weights are $-H$. The parameters α, β and γ are used to control the effect of each error term[1].

We use an incremental technique which presents strings to the network in order of increasing length. First we present to the network the strings in the learning set up to length L. We say that the network has learned all these strings when two conditions are satisfied: (1) the output is correct, and (2) for each j and k, only one weight W_{ijk} is positive. When both conditions are satisfied, we add to the learning set strings of length $L + 1$. We repeat the process until all the strings in the training set are correctly classified (we used all strings up to length 9 for these experiments). Training the net using this method, we find that the network incrementally learns the grammar rules or state transitions. This incremental learning technique can be viewed as an alternative to inserting symbolic knowledge covering these rules as in [10].

In our simulations, we use a value of H that is different for each set of weights $C_{jk} = \{W_{ijk} | i = 1, 2, ..., |Q|\}$, so we have $|\Sigma||Q|$ distinct values of H,

[1]In all the experiments reported here we used $\alpha = 1.1$, $\beta = 0.1$ and $\gamma = 0.1$.

one value for each pair (j, k), H_{jk}, the mean value of the set C_{jk} for each training epoch[2].

4 Experiments

We examined the performance of the learning algorithm on a class of benchmark problems for grammar induction: The Tomita grammars [1] which were also used to evaluate the automaton extraction techniques described earlier [1]. These grammars are regular grammars over a binary alphabet ($\Sigma = \{0, 1\}$).

In all the experiments, the presentation of training samples is performed as follows. For each given string length, L, we use the same training algorithm used in [1] and [10]. The network only gets to see some small randomly-selected fraction of the learning set (we have used 2 strings). When the network classifies the current learning set correctly or when it reaches a maximum number of epochs (50 in our experiments), 2 more strings are added to the current learning set. In Table 1 we show some preliminary results using the technique described. In this table we show, for each Tomita grammar tested, the maximum length of the strings presented, the number of strings presented during learning, epochs required and the value of the tolerance used (τ). If we compare these results with those reported in [1], it reveals that while our method offers a simpler extraction technique, we require many more epochs to achieve convergence. This, however, is an unfair comparison, since we present significantly fewer training strings per epoch (in particular, the largest numbers of epochs are often devoted to the shortest string lengths). This makes our approach more suitable for problems in which training data is sparse. It should further be noted that our average string length (especially for early epochs) is much less than that in [1].

Tomita grammar	Maximum length	Strings presented	Epochs required	τ
2	4	30	46	0.2
4	6	126	133	0.3
6	4	30	53	0.1
7	7	254	186	0.1

Table 1: Results obtained learning Tomita grammars.

[2] This means that we are in fact not computing a true gradient since, in our equations, we assumed that the value of H did not depend on the current weights (future work may consider using a true gradient).

5 Concluding Remarks

In this paper, we have suggested a simple modification to the standard error definition for second-order DTRNN which encourages these networks to assume natural encodings of DFA when trained using gradient descent. The automata learned using the modified error are strongly biased towards having $|Q|$ states if the network has $|Q|$ state units. Future work will examine what happens when there is a mismatch between the number of states and processing units. It is guessed that these networks might be able to infer simple automata that are reasonable approximations to the desired automata even when exact automata compatible with the training sample would be much bigger.

Acknowledgements: We acknowledge funding by the Spanish Comisión Interministerial de Ciencia y Tecnología (CICyT) under project TIC97-0941. R.P. Ñeco has been supported by the Spanish Generalitat Valenciana. The authors would like to thank Dr. Lee Giles and NEC Research Institute (Princeton, NJ) for providing software to simulate 2ODTRNN.

References

[1] Giles, C.L., Miller, C.B., Chen, D. *et al.* (1992) *Neural Computation* 4(3):393–405.

[2] Pollack, J.B. (1991) *Machine Learning* 7:227–252.

[3] Zeng, Z., Goodman, R., Smyth, P. (1993) *Neural Computation* 5(6):976–990.

[4] Das, S., Mozer, M. (1998) *Neural Networks* 11(1):53–64.

[5] Haykin, S. (1994) *Neural Networks, A Comprehensive Foundation*, New York, NY: Macmillan.

[6] Goudreau, M.W., Giles, C.L., Chakradhar, S.T. *et al.* (1994) *IEEE Transactions on Neural Networks*, 5(3):511–513.

[7] Kremer, S.C. (1995) *IEEE Transactions on Neural Networks* 6(4):1000–1004.

[8] Carrasco, R.C., Forcada, M.L., Valdés, M.A., Ñeco, R.P. (1998) A stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units, to be published.

[9] Ñeco, R.P., Forcada, M.L., Carrasco, R.C. *et al.* (1998) Encoding of Sequential Translators in Discrete-Time Recurrent Neural Nets, submitted to *International Conference on Artificial Neural Networks 1998* (Skövde, Sweden).

[10] Omlin, C.W., Giles, C.L. (1996) *Neural Computation* 8:675–696.