Asynchronous translations with recurrent neural nets

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

In recent years, many researchers have explored the relation between discrete-time recurrent neural networks (DTRNN) and finite-state machines (FSMs) either by showing their computational equivalence or by training them to perform as finite-state recognizers from examples. Most of this work has focussed on the simplest class of deterministic state machines, that is deterministic finite automata and Mealy (or Moore) machines. The class of translations these machines can perform is very limited, mainly because these machines output symbols at the same rate as they input symbols, and therefore, the input and the translation have the same length; one may call these translations synchronous. Real-life translations are more complex: word reorderings, deletions, and insertions are common in natural-language translations; or, in speech-to-phoneme conversion, the number of frames corresponding to each phoneme is different and depends on the particular speaker or word. There are, however, simple deterministic, finite-state machines (extensions of Mealy machines) that may perform these classes of "asynchronous" or "time-warped" translations. A simple DTRNN model with input and output control lines inspired on this class of machines is presented and successfully applied to simple asynchronous translation tasks with interesting results regarding generalization. Training of these nets from input-output pairs is complicated by the fact that the time alignment between the target output sequence and the input sequence is unknown and has to be learned: we propose a new error function to tackle this problem. This approach to the induction of asynchronous translators is discussed in connection with other approaches.

1 Introduction

In recent years, there has been a lot of interest in training discrete-time recurrent neural networks (DTRNN) to behave as finite-state machines [7, 9, 13, 17, 18, 20]. This behavior has recently been formalized [3] in response to hard criticisms [12]. So far, all work has focused on training DTRNN to behave as finite-state acceptors — deterministic finite automata— or, more generally, as very simple translators such as Mealy machines, or, equivalently, as Moore machines [11]. A Mealy machine is a six-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_1)$, where Q is a finite set of states, Σ a finite set of input symbols (*input alphabet*), Δ a finite set of output symbols (*output alphabet*), $\delta : Q \times \Sigma \to Q$ the next-state function, $\lambda : Q \times \Sigma \to \Delta$ the output function, and $q_1 \in Q$ the initial state. The class of translations that this kind of automata may perform is limited: for example, input strings and their translations always have the same length. We are interested in a wider class of translations; for that purpose, Mealy machines may be generalized by modifying the next-state and output functions. One may allow transitions to occur without consuming input: the new next-state function is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q$, where ϵ represents the empty string. For the machine to remain deterministic, states can have transitions defined either on the empty string or on input symbols but not on both; in addition, for the translator to halt always, it is required that no state be reachable from itself using only ϵ–transitions. The output function may be further generalized for transitions to occur without output: $\lambda : Q \times (\Sigma \cup \{\epsilon\}) \to \Delta \cup \{\epsilon\}$. The class of translations that these *extended Mealy machines* may perform is wider, and include —using a suitable end-of-input symbol— the class of *subsequential* translations whose algorithmic inference, based on a state-merging method, was studied by [15]. Subsequential translators have recently been used to infer rather complex natural-language translations in restricted domains [4].

This paper shows how an augmented architecture based on a second-order DTRNN may be used to learn an extended Mealy machine from a sample of translations, that is, of input string–output string pairs. It should be mentioned that an alternative neural approach to asynchronous translation exists [5, 6] which uses RAAMs (recursive auto-associative memories, [16]); there, a RAAM, which may be seen as a recurrent neural net, is specially trained to build unique, reversible state-space representations of each input string while, at the same time, another RAAM net is trained to unravel each representation and produce an output string. More recently [8] we have devised a related approach which we call *recursive heteroassociative memories* (RHAM), which is computationally equivalent to the RAAM translators of [5] and [6] but simpler because it does not require learning a complete autoassociator for the strings in both languages. The emerging results of this approach are not reported here due to lack of space. The main difference between the RAAM/RHAM approach and the one presented here is that a RAAM/RHAM needs to see the whole string before translating it while the DTRNN approach reported here tends to output partial translations as soon as possible by representing families of input strings as states. The work presented here extends the one reported preliminarily by us in an earlier paper [14].

2 Network architecture

DTRNNs used so far in the literature behave as Mealy or Moore machines: input is always consumed, output is always present even if not used. A single-layer or a multilayer feedforward neural net is used to compute the next state and the output from the previous state and the input symbol. We will build an extended Mealy machine by augmenting the customary architecture with two control

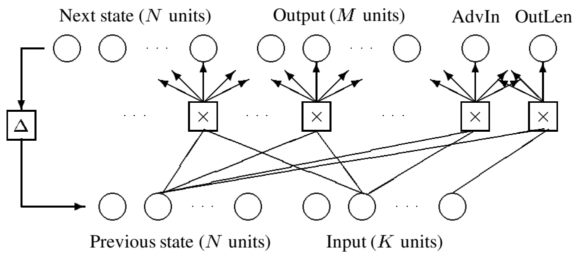

Figure 1: The architecture of the second-order DTRNN.

lines. We build around a second-order, single-layer network [17, 9, 20, 2, 7]. The network will have $N + K$ input units: N for the previous state, and $K = |\Sigma| + 1$ units for "one-hot" (unary) input of symbols from alphabet Σ plus one unit for a special symbol representing the empty string (*i.e.*, no input). There will be $N + M + 2$ output units: N for the next-state function, $M = |\Delta|$ for output of symbols from alphabet Δ, one for an "advance input" (AdvIn) control signal, one for an "output length" (OutLen) control signal. The architecture is shown in Figure 1.

Here is the sequence of steps the network takes each cycle: If the AdvIn signal was set to "high" (≥ 0.5) in the previous cycle, input is advanced and the symbol just read is presented to the network; if it is "low" (< 0.5), the special input corresponding to the empty string is presented to the network, without advancing the input. Then, the current input and the state computed in the previous cycle are combined by the neural net to compute a new state, a candidate for output symbol, and the new states of the AdvIn and OutLen lines. The output of the network is interpreted as a special kind of strings that we will call *noisy strings* [2]. A noisy string W is a pair $W = (\mathbf{I}, L)$, where $\mathbf{I}$ is a sequence of m output vectors $\mathbf{I}^{[t]}$, $(1 \leq t \leq m)$ and each component of $\mathbf{I}^{[t]}$ is associated with a symbol of the output alphabet and has a real value in the range $[0, 1]$ representing the degree of resemblance of the t-th vector of W to that symbol. L is a sequence of m values (output lengths) in the range $[0, 1]$, each one representing the *length* of the t-th symbol of W. The total length of the noisy string is $\sum_{t=1}^{m} L^{[t]}$. When noisy strings have values of $L^{[t]}$ of either 0 or 1 and the vectors $\mathbf{I}^{[t]}$ contain zeros in all but one component, they may be interpreted as regular (clean) strings. Since values 0 and 1 may not be output by neurons with a sigmoid activation function, we use a semi-linear mapping, as in [2]; when the output is less than or equal to a small value ξ it is taken to be "0", if it is higher than or equal to $1-\xi$ it is taken to be "1". Intermediate values are linearly mapped.

3 Training

Training of neural nets usually proceeds by minimization, with respect to the parameters of the net, of a cost function defined for a training sample. Minimization may be performed by gradient descent when the cost function is well behaved and derivatives are available in closed form. When input and output strings have the same length, it is quite easy to define a cost function based on the departure of output signals from the correct translation, because the alignment of the output and target strings is unambiguous. This is not the case with translations produced by extended Mealy machines, because there is no way to know in which cycle a given input symbol should be read or when a given output symbol should be produced (some cycles consume no input or produce no output). Error information is, however, available when the whole input has been consumed: the output string may then be compared to the target string to compute a cost.

We have chosen to use a cost function based on a measure of dissimilarity between output strings (noisy strings) and target strings (clean strings). The contribution of each training example to the cost function has the form

$$E(w, W) = E_s + E_l + \alpha(E_o + E_i),$$

where w is a clean string; W is a noisy string; E_s is the substitution term (which will be described below); E_l is the absolute difference between the length of W and the length of w; E_o is the "output length" term and E_i is the "input advance" term, which penalize control signals that are not 0 or 1, and help the network further to behave as an extended Mealy machine. These are sums of terms, one for each time the control signal is output; maximum penalization —equivalent to the cost of a wrong symbol in the output— occurs for a value of 0.5 and minimum penalization (0) for ξ and $1-\xi$ ($\xi = 0.08$ in the experiments). The coefficient α (1.0 in the experiments) is used to weight the penalty terms.

The substitution term E_s is computed as follows: (1) the target string is converted to a noisy string; (2) the target and the output strings are left-aligned; (3) since one of the strings may be longer, the tail of the longest one is discarded; (4) the resulting alignment is divided in frames by using the symbol boundaries of both the noisy and clean strings; (5) the squared difference between the clean and noisy vectors in each frame is computed; and (6) the distance in each frame is weighted with the length of the frame and summed over all frames. For example, the substitution term in the distance between the target string $w = 01$ and the output string $W = (\mathbf{I}, L)$ with $\mathbf{I} = (0.8, 0.2)(0.1, 0.9)(0.15, 0.85)$ and $L = (0.7\ 0.8\ 0.1)$ (total length 1.6) would be computed as follows. First, w is converted to a noisy string $W' = (\mathbf{I}', L')$ with $\mathbf{I}' = (1, 0)(0, 1)$ and $L' = (1.0\ 1.0)$. Then the strings are aligned, and 4 frames result, with lengths 0.7, 0.3, 0.5, and 0.1 respectively. Then the squared differences between both vectors in each frame are computed: 0.08, 1.62, 0.02, 0.045. Finally, they are length-weighted and summed to yield a dissimilarity of 0.5565.

Instead of minimizing the cost function by means of error gradient descent, we use Alopex [19], a stochastic algorithm —related to simulated annealing— which uses local correlations between changes in individual weights and changes in the global cost function[1]. The cost function is minimized both with respect to all the $NK(N+M+2)$ second-order weights and with respect to the $N+1$ initial values of the state units and the initial value of the "input advance" line[2].

4 Results

We present results for four different translation tasks T_1, T_2, T_3 and T_4, all with $\Sigma = \Delta = \{0, 1\}$. Task T_1 may be performed by a Mealy machine; the other three correspond to extended Mealy machines. The automata used to generate the training sets (M_1, M_2, M_3 and M_4) are presented in Figure 2, where the labels on the edges indicate in-

[1] We have recently derived a gradient-based training algorithm for these networks, which will be described elsewhere; the algorithm is based on a continuous interpretation of the AdvIn line, which is used as an "input length".

[2] For most meaningful translations, it would be enough to set the initial value of AdvIn to "high". Setting it to "low" makes sense only when all translations share a common prefix.

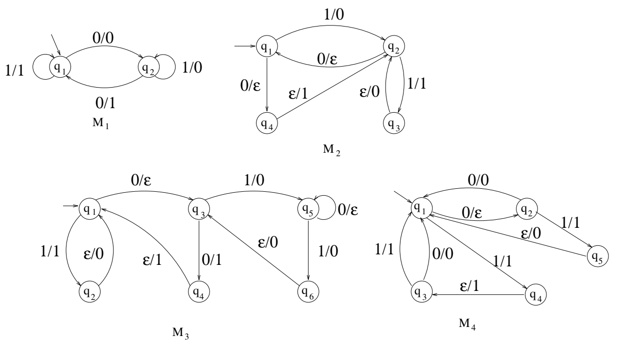

Figure 2: Automata used in the experiments.

Table 1: Results of the experiments (40 runs)

Task	N	failed jobs	k-epochs	generalization (tight/loose)
T_1	2	5	4.8 ± 1	83% / 98%
T_2	4	4	5.3 ± 1	77% / 95%
T_3	5	6	12.7 ± 3	79% / 96%
T_4	5	6	11 ± 4	80% / 93%

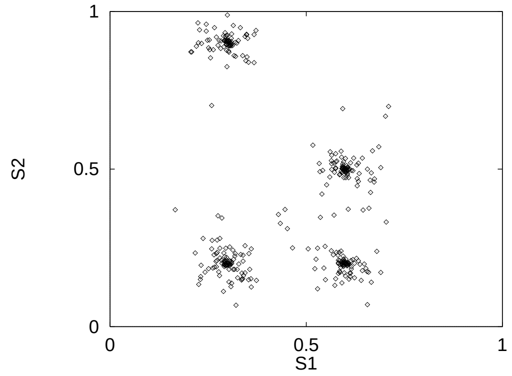

Figure 3: Points visited by a random set of strings with lengths 1 to 11 for task T_2.

put symbol/output symbol. In all cases, the training sets contained the translations of all strings from length 1 to 11; the maximum number of Alopex epochs[3] was set to 100 000. Following [9], training starts with a working set of 30 randomly-selected examples; 15 incorrectly translated examples are added either after 5 000 epochs without success or when the working set is correctly translated. The working set is increased whenever incorrectly translated strings remain in the training set. A successful translation is reported when (1) no unmapped OutLen or AdvIn signal departs more than ξ from the targets ξ or $1-\xi$; (2) the translation has the same length as the target translation; and (3) all the unmapped components of output vectors depart less than ξ from those in the unmapped correct (clean) target symbols, which consist of ξ's and $1-\xi$'s.

After training, we check the generalization ability of the networks with the set of all possible input strings with length up to 15, using a tight criterion (no unmapped output departs more than ξ from the targets ξ and $1-\xi$) and a loose criterion (the range of each neuron is split in half: values below 0.5 are taken to be equivalent to ξ and those above 0.5 equivalent to $1-\xi$). The averaged results for 40 runs on each translation task are presented in Table 1, where the number of state neurons, failed jobs, average number of epochs (in thousands) and the two generalization percentages are shown. The number of neurons was chosen as the minimum number for which consistent (larger than 85%) convergence on the training set was observed, in order to avoid overfitting.

[3] An *epoch* corresponds to a complete presentation of the current example set.

Figures 3 and 4 show a two-dimensional view of the hidden-neuron state space for the network obtained after training a 4-state-neuron DTRNN on task T_2. The points shown in figure 3 are those visited by a random set of 1700 strings with lengths 1 to 15. Those shown in figure 4 correspond to those strings in the random set having length 1–11. The four clusters corresponding to the states of the extended Mealy machine used in the task are clearly visible in the state space of the trained DTRNN. For longer strings, states become blurred and that accounts for errors in generalization. A study of (a) the extraction of extended Mealy machines from the trained DTRNN and (b) the conditions under which the augmented DTRNN presented here may "robustly model" [3] an extended Mealy machine is currently under way.

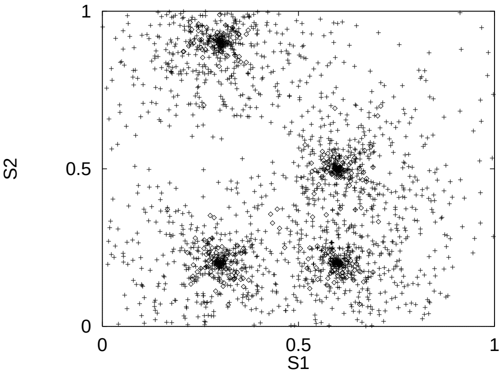

Figure 4: Points visited by a random set of 1700 strings with lengths 1 to 15 for task T_2.

5 Concluding remarks

The results reported in this paper show that an augmented DTRNN may be trained, using examples, to perform translations that Mealy machines cannot represent. This work is, in that sense, parallel to the algorithmic procedures described in [15], although the use of neural networks may yield simpler translators by ignoring or filtering exceptions that may present in the training set (see [10]). We are currently studying the nature of the learned representations and working on methods to extract automata from the networks in order to improve generalization [9, 1]. In addition, we plan to study the use of the augmented DTRNN architecture to model more complex translation tasks (such as the conversion of text to speech or speech to phonemes), where the ability of the DTRNNs presented here to discover the "asynchronous" or "time warped" correspondences between sequences (in some cases, not necessarily symbolic in nature) may be advantageously used.

Acknowledgements

We thank Rafael C. Carrasco for his suggestions during the preparation of this manuscript. This work has been supported through grant TIC95-0984-C02-01 of the Spanish Comisión Interministerial de Ciencia y Tecnología (CICyT). Ramón P. Ñeco is supported by the Generalitat Valenciana (Spain).

References

[1] Blair, A.D., Pollack, J.B. (1996) "Precise Analysis of Dynamical Recognizers", Technical Report CS–95–181, Computer Science Department, Volen Center for Complex Systems, Brandeis University.

[2] Carrasco, R.C., Forcada, M.L. (1995) "Second-order recurrent neural networks can learn regular grammars from noisy strings" in *From Natural to Artificial Neural Computation* (Proc. IWANN'95, Málaga, Spain, June 1995; Mira, J., Sandoval, F., eds.), *Lecture Notes in Computer Science* **930**, 605–610 (Heidelberg, Germany: Springer-Verlag).

[3] Casey, M. (1996) "The Dynamics of Discrete-Time Computation, with Applications to Recurrent Neural Networks and Finite State Machine Extraction", *Neural Computation* **8**:6, 1135–1178.

[4] Castellanos, A., Galiano, I., Vidal, E. (1994) "Application of OSTIA to Machine Translation Tasks", *Grammatical Inference and Applications* (Proc. ICGI'94, Alicante, Spain, September 1994; Carrasco, R.C., and Oncina, J., eds.), *Lecture Notes in Artificial Intelligence* **862**, 93–105. (Heidelberg, Germany: Springer-Verlag).

[5] Chalmers, D.J. (1990) "Syntactic Transformations on Distributed Representations", *Connection Science* **2**:1–2, 53–62.

[6] Chrisman, L. (1991) "Learning Recursive Distributed Representations for Holistic Computation", *Connection Science* **3**:4, 345–366.

[7] Forcada, M.L., Carrasco, R.C. (1995) "Learning the initial state of a second-order recurrent neural network during regular language inference", *Neural Computation* **7**, 923–930.

[8] Forcada, M.L., Ñeco, R.P. (1997) "Recursive Hetero-Associative Memories for Translation", submitted to *IWANN'97* (Lanzarote, Canary Islands, June 6–8, 1997).

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

[10] Gori, M., Maggini, M., Soda, G. (1995) "Learning Regular Grammars from Noisy Examples Using Recurrent Neural Networks", Proceedings of NEURAP 95, p. 207–214 (Marseille, France, 20–22 March 1996).

[11] Hopcroft, J.E., Ullman, J.D. (1979) *Introduction to Automata theory, Languages and Computation*, Reading, Massachusetts: Addison-Wesley.

[12] Kolen, J.F. (1994) "Fool's Gold: Extracting Finite-State Automata from Recurrent Network Dynamics", *Advances in Neural Information Processing Systems* **6**, 501–508.

[13] Manolios, P., Fanelli, R. (1994) "First-Order Recurrent Neural Networks and Deterministic Finite State Automata", *Neural Computation*, **6**, 1155–1173.

[14] Ñeco, R.P., Forcada, M.L. (1996) "Beyond Mealy Machines: Learning Translators with Recurrent Neural Networks", *Proc. World Congress on Neural Networks* (San Diego, Calif., Sept. 1996), p. 408–411.

[15] Oncina, J., García, P., Vidal, E. (1993) "Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**, 448–458.

[16] Pollack, J.B. (1990) "Recursive Distributed Representations", *Artificial Intelligence* **46**, 77–105.

[17] Pollack, J.B. (1991) "The induction of dynamical recognizers", *Machine Learning* **7**, 227–252.

[18] Tiňo, P., Sajda, J. (1995) "Learning and Extracting Initial Mealy Automata with a Modular Neural Network Model", *Neural Computation* **7**, 822–844.

[19] Unnikrishnan, K.P., Venugopal, K.P. (1994) "Alopex: A Correlation-Based Algorithm for Feed-forward and Recurrent Neural Networks", *Neural Computation* **6**, 469–490.

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