Beyond Mealy machines: Learning translators with recurrent neural networks

Ramón Ñeco, Mikel L. Forcada

Departament de Llenguatges i Sistemes Informàtics,
Universitat d'Alacant,
E-03071 Alacant, Spain.
E-mail: {neco,mlf}@dlsi.ua.es
URL: http://www.dlsi.ua.es/lsi/componentes/mlf/mlfeng.html

Abstract

Recent work has shown that recurrent neural networks can be trained to behave as finite-state automata from samples of input strings and their corresponding outputs. However, most of the work has focused on training simple networks to behave as the simplest class of deterministic machines, Mealy (or Moore) machines. The class of translations that can be performed by these machines are very limited. For example, input and output strings have the same length. However, deterministic state machines can perform more complex translation tasks, and it has been recently shown that they can be inferred from input–output pairs. In this paper we show how a slightly augmented architecture based on a second-order recurrent neural network may be trained to behave as an instance of the most powerful class of deterministic sequential translator.

1 Introduction

In recent years, there has been a lot of interest in training recurrent neural networks to behave as finite-state machines [3, 4, 7, 9]. So far, all work has focused on training recurrent neural networks to behave as finite-state acceptors —deterministic finite automata— or, more generally, very simple translators: such as Mealy machines, or, equivalently, as Moore machines[5]. 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 corresponding output strings 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. 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 an end-of-input symbol— the class of *subsequential* translations defined in [6], which have recently been used to model rather complex natural-language translations in restricted domains[2].

This paper shows how an augmented architecture based on a second-order recurrent neural network may be used to learn an extended Mealy machine from a sample of translations, that is, of input string–output string pairs.

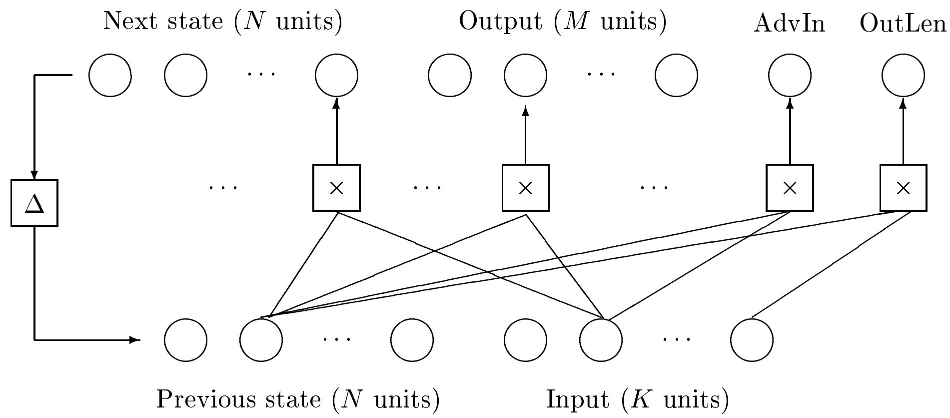

Figure 1: The architecture of the second-order recurrent neural net

2 Architecture

Recurrent neural networks used so far in the literature behave as Mealy machines: input is always consumed, output is always present. 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 augmenting the customary architecture with two control lines. We build around a second-order, single-layer network[1, 3, 4]. The network will have $N+K$ input units: N for the previous state, and $K=|\Sigma|$ for "one-hot" input of symbols from alphabet Σ. 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 fig. 1. This is the sequence of steps the new architecture takes each cycle: If the "advance input" 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 same symbol as in the previous cycle is presented without advancing the input. Then, the 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 "advance input" and "output length" lines. The output of the network is interpreted as a special kind of strings that we will call *noisy strings*[1]. 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. We impose the restriction that the M components of each $\mathbf{I}^{[t]}$ add to one. 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[1].

3 Training

Training of neural nets usually proceeds by minimization of a cost function for a training sample. The cost function depends on the choice of network parameters. The best set of parameters may be determined by minimizing the cost function by means of a gradient descent method, because the function is usually 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 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 step a given output symbol has been produced (some steps produce no output).

[1] Obviously, values 0 and 1 may not be output by neurons with a sigmoid activation function; to overcome this, we use a mapping, as in [1].

$M_1 : \delta/\lambda$	0	1
q_1	$q_2/0$	$q_1/1$
q_2	$q_1/1$	$q_2/0$

$M_2 : \delta/\lambda$	0	1	ϵ
q_1	q_4/ϵ	$q_2/0$	
q_2	q_1/ϵ	$q_3/1$	
q_3			$q_2/0$
q_4			$q_2/1$

$M_3 : \delta/\lambda$	0	1	ϵ
q_1	q_3/ϵ	$q_2/1$	
q_2			$q_1/0$
q_3	$q_4/1$	$q_5/0$	
q_4			$q_1/1$
q_5	q_5/ϵ	$q_6/0$	
q_6			$q_3/0$

Table 1: Automata learned by the network

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 $E(w, W) = s + l + o + i$, where w is a clean string, W a noisy string, s is the substitution term (which will be described below), l the absolute difference between the length of W and the length of w, o the "output length" term, i the "input advance" term, which penalize control signals that are not 0 or 1, and force the network to behave as an extended Mealy machine. The substitution term 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 will 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 Euclidean distance 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 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 Euclidean distances 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 0.5625.

Instead of minimizing the cost function by means of error gradient, we use Alopex[8], a stochastic algorithm —related to simulated annealing— which uses local correlations between changes in individual weights and changes in the global cost function. The cost function is minimized both with respect to all the $NK(N + M + 2)$ second-order weights and with respect to the $N + K + 1$ initial values of the state units, the input vector and the "input advance" line.

4 Results

We present results for three different translation tasks T_1, T_2 and T_3, all with $\Sigma = \Delta = \{0, 1\}$. Task T_1 may be performed by a Mealy machine; the other two correspond to extended Mealy machines. The transition and output tables of the automata used to generate the training sets (M_1, M_2 and M_3) are presented in Table 1. In all cases, the training sets contained the translations of all 510 strings from length 1 to 8; the maximum number of Alopex epochs was 100 000. Training starts with a working set of 30 randomly-selected examples; 15 incorrectly classified examples are added either after 5 000 epochs without success or when the working set is translated correctly. The working set is increased whenever incorrectly translated strings remain in the training set. A successful translation is reported when (1) no "output length" signal departs more than $\epsilon = 0.08$ from 0 or 1; (2) the translation has the same length as the target translation; and (3) output vectors have all their components within $\epsilon = 0.08$ of the correct (clean) target symbols. After training, we check the generalization ability of the networks with the set of all possible input strings (4094) with length up to 10, using the same stringent criterion ($\epsilon = 0.08$) and two weaker criteria ($\epsilon = 0.2$ and $\epsilon = 0.5$). The average correct generalization percentages will be called $G_{0.08}$, $G_{0.2}$ and $G_{0.5}$ respectively.

For task T_1, we used $N = 2$ state neurons. Out of 10 runs, 5 converged, all of them in less than 40 000 epochs. For higher values of N, convergence was slower but a higher rate of success was observed. Generalization results ($N = 2$) were $G_{0.08} = 75\%$, $G_{0.2} = 85\%$ and $G_{0.5} = 96\%$.

For task T_2, with $N = 4$, 8 out of 10 runs succeeded in an average of 13 000 epochs and always in less than 24 000 epochs. Success was also obtained with smaller values of N, but it was not as consistent. With $N = 3$, 5 out of 10 runs succeeded in an average of 21 000 epochs; with $N = 2$, only 1 out of 10 runs succeeded. Generalization results ($N = 4$) were $G_{0.08} = 58\%$, $G_{0.2} = 69\%$ and $G_{0.5} = 92\%$.

Task T_3 is more complex (machine M_3 has more states). Using $N = 5$, success was obtained in 7 runs out of 10, in an average of 19 000 epochs, and always in less than 33 000 epochs. Generalization results ($N = 5$) were $G_{0.08} = 58\%$, $G_{0.2} = 69\%$ and $G_{0.5} = 92\%$.

5 Concluding remarks

The preliminary results reported in this paper show that an augmented architecture based on a recurrent neural network 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 [6], although the use of neural networks may yield simpler translators by ignoring or filtering exceptions that may present in the training set. 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[4]. In addition, we plan to study the use of the augmented recurrent architecture to model more complex translation tasks (such as voice to voice translation), where neural networks may outperform discrete algorithmic procedures.

Acknowledgements: The authors wish to 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).

References

[1] 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).

[2] 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).

[3] 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.

[4] 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.

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

[6] 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.

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

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

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