Journal of Mathematics and Statistics 2 (2): 386-390, 2006
ISSN 1549-3644
© Science Publications, 2006

An Evolution Strategy for the Induction of Fuzzy Finite-state Automata

[1,2]Mozhiwen and [1]Wanmin
[1]College of Math.& Software Sci., Sichuan Normal University, Chengdu, 610066, P.R. China
[2]Department of Applied Math., Southwest Jiaotong University, Chengdu, 610031, P.R. China

Abstract: This study presents an evolution strategy used to infer fuzzy finite-state automata from examples of a fuzzy language. We describe the fitness function of an generated automata with respect to a set of examples of a fuzzy language, the representation of the transition of the automata as well as the output of the states in the evolution strategy and the simple mutation operators that work on these representations. Results are reported on the inference of a fuzzy language.

Key words: Evolution strategy, fuzzy finite state automata, mutation, fitness, generalization

INTRODUCTION

Grammar inference (GI) encompasses theory and methods for the inference of any structure that can recognize a language-be it a grammar, an automaton, or some other procedure-from training data. It has many applications, including syntactic pattern recognition[1], speech and natural language processing, gene analysis, image processing, sequence prediction, information retrieval, cryptography and many more. Gold[2] presented the most classical formalization of GI, known as Language Identification in the Limit. Valiant[3] propose the probably approximately correct (PAC) identification in which a learning method is asked to find models which, with arbitrarily high probability, are arbitrarily precise approximations to the unknown language. Artificial neural networks have also been developed to carry out the GI which focus on obtaining the automaton that recognizes the example set[4,5] since the problems of inferring a grammar or the associated automaton are equivalent[6].

Relative fewer efforts have been made to induce automaton using evolution strategies (ES). We believe that ES is suitable for the inference of a language due to their direct coding scheme and their simple way of handling constraints. We encode the state transitions into a matrix with some constraints due to the completeness and deterministic of the generated automaton and consequently the special mutation operators.

Fuzzy finite state automata (FFA): We begin by the class of fuzzy automata which we are interested in learning:

Definition 2.1[7]: A fuzzy finite-state automaton (FFA) $\overline{M}$ is a 6-tuple $\overline{M} = <\Sigma, Q, Z, q_0, \delta, \omega>$ where Σ is a finite input alphabet, Q is a finite set of states, Z is a finite output alphabet, q_0 is an initial state, $\delta: \Sigma \times Q \times [0,1] \rightarrow Q$ is the fuzzy transition map and $\omega: Q \rightarrow Z$ is the output map.

It should be noted that a regular fuzzy grammar as well as a finite fuzzy automaton is reduced to a conventional (crisp) one when all the production and transition degrees are equal to 1.

The following result gives a transformation from a fuzzy finite automata to the corresponding crisp one[7]:

Theorem: Given a FFA $\overline{M}$, there exists a deterministic finite state automaton (DFA) M with output alphabet Z $\subseteq$ { θ : θ is (a membership degree)} $\cup$ {0} that computes the membership function $\mu: \Sigma^* \rightarrow [0,1]$ of the language L ($\overline{M}$) in the output of its states. An example of FFA-to-DFA transformation is shown in Fig. 1a and b.

From the automata theory, it's easy to transform an incomplete DFA to a complete one. So in the following discussion we suppose the DFA in theorem 2.1 is a complete one.

Generation of FFA based on evolution strategy: According to theorem 2.1, the induction of FFA can be transformed to the induction of a complete and deterministic finite state automata with outputs. So, we only need to consider the case of the complete DFA when using evolution strategy in the rest part of this section.

Algorithm of evolution strategies: Evolution strategies are specially suited for difficult search and optimization problems where the problem space is large, complex and contains possible difficulties like high dimensionality, multimodality, discontinuity and noise.

Corresponding Author: Mozhiwen, College of Math.& Software Sci., Sichuan Normal University.E-mail:mozhiwen@263.net

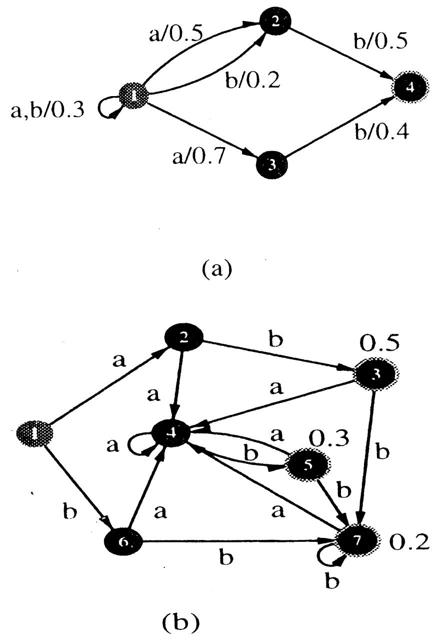

Fig. 1: Example of a transformation of a specific FFA into its corresponding DFA. (a) A fuzzy finite-state automaton with weighted state transitions. State 1 is the automaton's start state; accepting states are drawn with double circle. A transition from state q_j to q_i on input symbol a_k with weight θ is represented as a direct arc from q_j to q_j labeled a_k/θ . (b) corresponding deterministic finite-state automata which computes the membership function strings. The accepting states are labeled with the degree of membership. Notice that all transitions in the DFA have weight one

Currently, several evolution strategies(**ES**) are available. The two most widely used algorithm are noted as $(\mu+\lambda)$-**ES** and (μ,λ)-**ES**[25]. The former selects the best μ individuals from both of the μ parents and λ off-springs as the parents of the next generation, while the latter selects the best μ individuals only from the λ off-springs. It is believed that the (μ,λ)-**ES** outperforms the $(\mu+\lambda)$-**ES** because (μ,λ)-**ES** is less likely to get stuck in local optima. The numbers of parents and off-springs are recommended to be at a ratio of $\mu/\lambda \approx 1/7$ [8].

An ES algorithm that is capable of dealing with optimization can be described by the following notation[9]:

(μ,λ)-**ES**=(I, μ, λ ; m, s, σ ; f, g) (1)

where I is a string of real or integer numbers representing an individual in the population, μ and λ are the numbers of the parents and off-springs respectively; σ is the parameter to control the step size, m represents the mutation operator, which is the main operator in the mechanism of ES. In ES algorithm, not only the variables, but also the step-size control parameter σ are mutated. In (1), parameter s stands for the selection method and in this case, the parents will be selected only from the λ descendants; f is the objective function to be minimized and g is the constraining function to which the variables are subject.

In our case, the parameters representing the states transitions and the corresponding state outputs are encoded into matrixes and we adopt the (μ,λ)-**ES**. A lot of strategy parameters which have great influence on the performance of the algorithm, must be fixed manually. We'll see them later. The optimization problems in the real world normally have a lot of sub-optima, in which a standard evolution strategy can get trapped. To acquire a solution as good as possible, it's desired to improve the performance in specific application domain. In practice, we control the step-size from large to small according to the distance between the current fitness generated by the best individual which has been found so far and the expected fitness: when the fitness is far away from the expected one, we enlarge the step-size, otherwise, it's decreased.

Representing of the FFA: The coding of a complete DFA is straitforward. It consists of two parts: the state transition and the outputs of the states.

The state transition coding is important because it determined the structure of the automata. Suppose the automata has n states, its state transitions can be encoded with the following matrix:

$q_1, q_2 \cdots q_n$

$$\text{Trans}=\begin{bmatrix} a_{11}, a_{12} \dots a_{1n} \\ a_{21}, a_{22} \dots a_{2n} \\ \dots\dots\dots \\ a_{n1}, a_{n2} \dots a_{nn} \end{bmatrix} \quad (2)$$

Where q_1 is the initial state of the automata; $a_{ij} \in \Sigma$, Σ is the finite input alphabet. Non-zero character a_{ij} corresponds the state transition $q_j = \delta(q_i, a_{ij})$, while $a_{ij}=0$ indicates that there is no transition between the two states.

The automata we want to obtain is complete and deterministic and consequently there exist constrains on the matrix. A valid transition matrix must satisfy the following conditions: the non-zero characters are different from each other and run over the input alphabet Σ in a same row.

The output of an n-sate automata is directly represented by a 1-by-n matrix whose elements are initially selected from 0 to 1 with random:

Output=$[c_1, c_2, \dots, c_n]$ (3)

Where c_j is the output of the jth state of the automata. Thus, any n states FFA can be represented by an nxn transition matrix (2) and a $1\times$n output matrix (3).

Mutation: The n states automata is deterministic and complete, so the non-zero characters in each row of the transition matrix form an non-repeated permutation of the input alphabet. Due to this specification, the mutation in our case is concerned with arrangement. The step-size control parameter of the evolving of transition is mutated proportional to the distance between the fitness (denoted by $fit_{current}$) generated by the best individual up to now and the expected error tolerance (denoted by fit_{expect}):

$$\delta = \frac{fit_{expect} - fit_{current}}{fit_{current}} * \tau$$

Where τ is a constant.

For the evolution of the transition matrix, there are lots of words, but a very simple technique.

Randomly generate an integer: h=round(δ *rand(1,1)), repeat it until it is such that $0 \le h \le n$; it represents the number of the rows that are to be changed.

Where rand(M,N) is an M-by-N matrix with random entries chosen from a uniform distribution on the interval (0,1), round(x) rounds the elements of x to the nearest integers. The parameter δ control the step size of the mutation of transition matrix: when the current fitness is far away from the expected one, δ is large and consequently the mutation step of the matrix. To avoid the mutation stopping too early, we set h= round(δ *rand(1,1))+1 as e1< $fit_{current}$ <e2 and the evolution finally enters a fine-tuning process when $fit_{current}$ <e1.

* Randomly generate a 1xh integer matrix B=(b_{1j}) where b_{1j} ($1 \le b_{1j} \le n$) represent the index of the row to be mutated at the jth step.
* Randomly generate a hx1 integer matrix C=(c_{i1}), the jth element c_{j1} ($1 \le c_{j1} \le n$) indicates that the element in column c_{j1} and row b_{1j} is to commute with the first nonzero element in the same row.
* This step guarantees the difference between the parent and its off-springs as h is larger than 0.
* Commute the corresponding elements in the transition matrix according to the indexes obtained from step 1-3.

We give an example to illustrate the mutation operator: suppose Trans=(a_{ij}), n=5, h=2, B=[2 4], C= $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$, then the mutation includes two steps: a_{23} exchange with the first nonzero character in row 2 and a_{42} commute with the first nonzero character in row 4. The output matrix is evolved as follows:

$$c_i' = c_i - (\text{or } +)\, \eta * \text{rand}(1,1),\ i=1,2,\ldots,n \qquad (3)$$

Where the sign in (3) is selected randomly and η is the step-size control parameter. If the mutated value is out of the expected range, repeat the following process until $0 \le c_i' \le 1$:

If $c_i' > 1$ $c_i' = c_i - \eta$ *rand(1,1)

elseif $c_i' < 0$ $c_i' = c_i + \eta$ *rand(1,1)

Besides the above-mentioned mutations, the size of the off-springs (denoted by **size**$_{child}$) is also adaptive. It's randomly chosen from a range [**size**$_{parent}$**-m**, **size**$_{parent}$**+m**], where **size**$_{parent}$is the size of the parent, m is an invariable integer. If **size**$_{child}$**>** **size**$_{parent}$, the mutated parent is copied to the off-spring and the remainder of the matrix is generated randomly. On the other hand, the mutated parent is copied to the child up to the size of the off-spring when **size**$_{child} \le$ **size**$_{parent}$. The size of the parent is initialized by the number of different outputs seen from the training examples multiplying a small integer m_p.

Selection of the objective function: The fitness of automata is evaluated according to 2 fitness criteria that assess (a) consistency of the language recognized by the automata with the training examples f_E and (b) generalization capacity of the generated automata f_{gener}:

$$f_E = \sum_{i=1}^{N} (T_i - O_i)^2 \qquad (6)$$

where T_i is the desired output of the ith sample while O_i is the actual output when the ith sample is presented to the generated automata. N is the size of the training examples.

$$f_{gener} = \frac{K}{V} \quad (7)$$

where K is the number of the strings in the test samples that the automata can correctly recognize, V is the length of the test examples.

The quality of a generated automata is evaluated with the following objective function:

$$\text{fit} = \alpha \times f_E - \beta \times f_{gener} \quad (8)$$

Where α is the proportional coefficient for the consistency and β for the generalization capacity in the fitness function.

Application example: In this part, we will induce the FFA from training examples using our evolution strategy.

Generation of examples: We start from a fuzzy automata M as shown in Fig. 1a, we generate a set of 400 examples recognized by M from Fig.1b. Any example consist of a pair (P_i, μ_i) where $P_i$is a random sequence formed by symbols of the alphabet Σ ={a, b} and μ_i is the membership degree to fuzzy language L (M)[8]. The samples strings are ordered according to their length. The first 200 examples are used to train and the rest are for generalization.

"Forgetting" the initial automaton M: Our objective was to find a complete DFA to correctly recognize the examples using ES. The parameters for the evolution strategy are: the initial size of the parent is set to 4*2=8,i.e. m_p=2; the number of the parentsμ=4, each parent generates 7 off-springs, the constant τ is set to 0.2, the control parameters e1=0.5, e2=5.3, the step-size control for the mutation of the output η =0.08, the proportional coefficient α=1, β =0.3 and the expected fitness is set to 2.5$\times$ 10^{-5}-0.3. After the evolving phase, the size of the final automata is 12 and the evolution of the fitness is plotted in Fig. 2. The generated automata recognize all the training examples and test examples in generation 63. Figure 3 shows the evolution of the generalization capacity. The final obtained automata is shown in Fig. 4, it's not difficult to see that it's equivalent to the automata in Fig. 1 b:

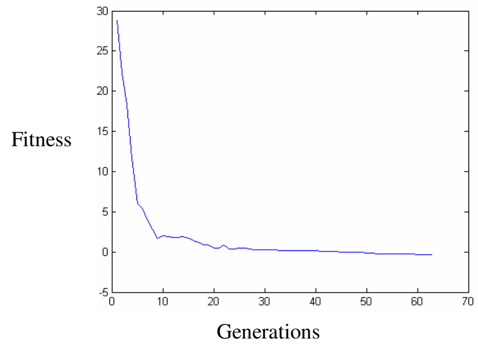

Fig. 2: Evolution of the fitness

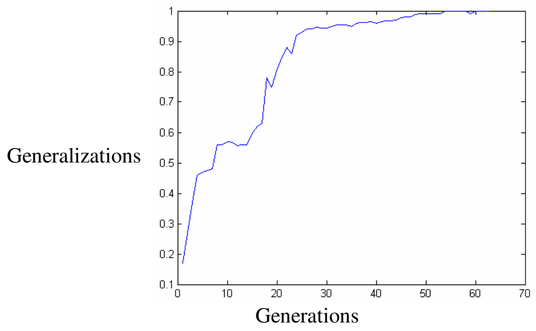

Fig. 3: The evolution of the generalization in the fitness

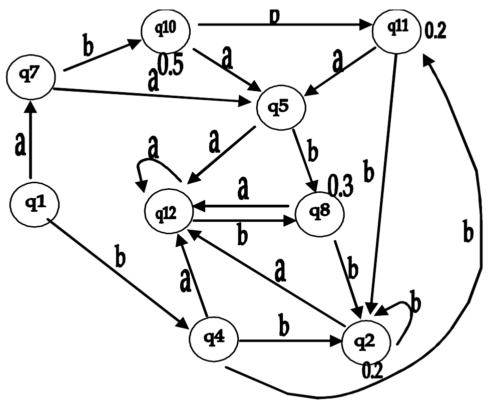

Fig. 4: Automata induced by the evolution strategy

CONCLUSION

We present a simple technique for the induction of an unknown fuzzy finite-state automata from the training data. The result indicates that the evolution strategy can successfully be applied to search for an optimal automata in the structure space to match the data and generalize well.

ACKNOWLEDGEMENT

This research has been supported by National Natural Science Foundation of china (No.60474022).

REFERENCES

1. Fu, K. and T. Booth, 1986. Grammatical inference: Introduction and survey. IEEE Trans. Pattern Analysis and Machine Intelligence, 8: 343-375.
2. Gold, E., 1967. Language identification in the limit. Information and Control, 10: 447-474.
3. Valiant, L., 1984. A theory of the learnable. Communications of the ACM, 27: 1134-1142.
4. Giles, C.L. C.B. Miller and D. Chen, 1992. Learning and extracting finite state automata with second-order recurrent neural networks. Neural Computation, 4: 393-405.
5. Zeng, Z., R. Goodman and P. Smyth, 1994. Discrete recurrent neural networks for grammatical inference. IEEE Trans. Neural Networks, 5: 320-330.
6. Kohonen, T., 1990. The self-organizing map. Proc. IEEE, 9: 1464-1480.
7. Christian, W.O., K.K. Thornber and C.L. Giles, 1998. Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks. IEEE Trans. Fuzzy Systems, 6: 1.
8. Schwefel, H.P., 1995. Evolution and Optimum Seeking. New York, Wiley.
9. Back, T., 1994. Parallel Optimization of Evolutionary Algorithms. Parallel Problem Solving from Nature. Berlin, Germany, Spring-Verlag, pp: 418-427.