Journal of Mathematics and Statistics 2 (2): 395-400, 2006
ISSN 1549-3644

Fuzzy Automata Induction using Construction Method

[1,2]Mo Zhi Wen and [2]Wan Min
[1]Center of Intelligent Control and Development, Southwest Jiaotong University, China
[2]College of Mathematics and Software Science, Sichuan Normal University, China

Abstract: Recurrent neural networks have recently been demonstrated to have the ability to learn simple grammars. In particular, networks using second-order units have been successfully at this task. However, it is often difficult to predict the optimal neural network size to induce an unknown automaton from examples. Instead of just adjusting the weights in a network of fixed topology, we adopt the dynamic networks (i.e. the topology and weights can be simultaneously changed during training) for this application. We apply the idea of maximizing correlation in the cascade-correlation algorithm to the second-order single-layer recurrent neural network to generate a new construction algorithm and use it to induce fuzzy finite state automata. The experiment indicates that such a dynamic network performs well.

Key words: Fuzzy automation, construction method, dynamic network

INTRODUCTION

Choosing an appropriate architecture for a learning task is an important issue in training neural networks. Because general methods for determining a "good" network architecture prior to training are generally lacking, algorithms that adapt the network architecture during training have been developed.

Constructive algorithm determines both the architecture of the network in addition to the parameters (weights, thresholds, etc) necessary for learning the data. Compared to algorithms such as back propagation they have the following potential advantages:

* They grow the architecture of the neural network in addition to finding the parameters required.
* They can reduce the training to single-node learning hence substantially simplifying and accelerating the training process. Constructive algorithms such as cascade correlation have at least an order of magnitude improvement in learning speed over back propagation.
* Some constructive algorithms are guaranteed to converge unlike the back propagation algorithm, for example.
* Algorithms such as back propagation can suffer from catastrophic interference when learning new data, that is, storage of new information seriously disrupts the retrieval of previously stored data. The incremental learning strategy of constructive algorithms offers a possible solution to this problem.

Rather than growing neural networks destructive or pruning[1] algorithms remove neurons or weights from large and trained networks and retain the reduced networks in an attempt to improve their generalization performance. Approaches include removing the smallest magnitude or insensitive weights or by adding a penalty term to the energy function to encourage weight decay. However, it is difficult to "guess" the initial network which is bound to solve the problem.

Recurrent neural networks have been demonstrated to have the ability to learn simple grammars[2-6]. However, the problem associated with recurrent networks of fixed topology is apparent. We have to empirically choose the number of hidden neurons, there is no general methods for determining an appropriate topology for specific application. Consequently, the convergence can't be guaranteed. Constructive or destructive methods that add or subtract neurons, layers, connections, etc. might offer a solution to this problem though it is complementary. We propose a construction algorithm for second-order single-layer recurrent neural network which adds a second-order recurrent hidden neuron at a time to adapt the topology of the network in addition to the change of the weights and use it to learn fuzzy finite state machines and experimentally find it effective.

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

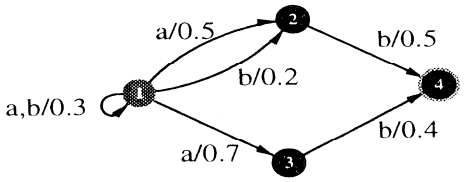
Definition: A fuzzy finite-state automaton (FFA) $\overline{}$ is a 6-tuple $=<\Sigma, Q, Z, q_0, \delta, \omega>$ where Σ is a finite input alphabet and Q is the set of states; Z is a finite output alphabet, q_0 is an initial state, $\delta : \Sigma \times Q \times [0,1] \rightarrow Q$ is the fuzzy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

Corresponding Author: Mo Zhi Wen, [1]Center of Intelligent control and Development,Southwest Jiaotong University

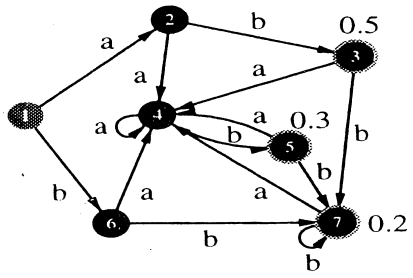
It should be noted that a finite fuzzy automaton is reduced to a conventional (crisp) one when all the transition degrees are equal to 1.

The following result is the basis for mapping FFAs into the corresponding deterministic recurrent neural networks[7]:

Theorem: Given a FFA $\overline{\quad}$, there exists a deterministic finite state automaton (DFA) M with output alphabet Z $\subseteq \{\theta : \theta$ is a production weight$\} \cup \{0\}$ which computes the membership function $\mu : \Sigma^* \rightarrow [0,1]$ of the language L ($\overline{M}$). An example of FFA-to-DFA transformation is shown in Fig. 1a and b[8].



(a)



(b)

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_i 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

Second-order recurrent neural network used to induce FFA: A second-order recurrent neural network linked to an output layer is shown to be a good technique to perform fuzzy automaton induction in[9], it consists of three parts (Fig. 2): input layer, a single recurrent layer with N neurons and two output layers both with M neurons, where M is determined by the level of accuracy desired in the output. If r is the accuracy required(r=0.1 if the accuracy is decimal, r=0.01 if the accuracy is centesimal, etc.), $M=(10^{-\log(r)}+1)$.

The hidden recurrent layer is activated by both the input neurons and the recurrent layer itself. The first output layer receives the values of the recurrent neurons at time m, where m is the length of the input string. The neurons in this layer then compete to have a winner to determine the output of the network. The dynamic of the neural network is as follows:

$$S_i^{t+1}=g(\Xi +) \quad \Xi = \sum_{j,k} W_{ijk} S_j^t I_k^t \quad i=1,\ldots N \qquad (1)$$

$$O_p = g(\sigma_p) \quad \sigma_p = \sum_{i=0}^{N-1} u_{pi} S_i^m \quad p=1,\ldots M \qquad (2)$$

$$out_p = \begin{cases} 1 & \text{if } O_p = \max\{O_0,\ldots, O_{M-1}\} \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

Where g is a sigmoid discriminant function and b is the bias associated with hidden recurrent state neurons S . u_{pi} connected S_i to the neuron O_p of the first output layer, W_{ijk} is the second-order connection weight of the recurrent layer and input layer. The membership degree associated to the input string is determined by the last output layer: $\mu_L(x)=i*r$, where r is the desired accuracy, i is the index of the non-zero neuron of the last output layer. The weights are updated by the pseudo-gradient decent algorithm:

The error of each output neuron is $E_i = \frac{1}{2}(T_i - O_i)^2$, Where T_i is the expected value for the ith output neuron and O_i is the provisional value given by equation (2). The total error for the network is

$$E=\sum_{i=0}^{M-1} E_i \quad (4)$$

When E>ε,modify the weights as follows: $\Delta u_{ij} = -\alpha \frac{\partial E_i}{\partial u_{ij}}$ $\Delta w_{lon} = -\alpha \frac{\partial E}{\partial w_{lon}}$, Whereεis the error tolerance of the neural network, α is the learning rate. Further step:

$$\frac{\partial E_i}{\partial u_{ij}} = (T_i - O_i) \times O_i \times (1-O_i) \times S_j^m$$

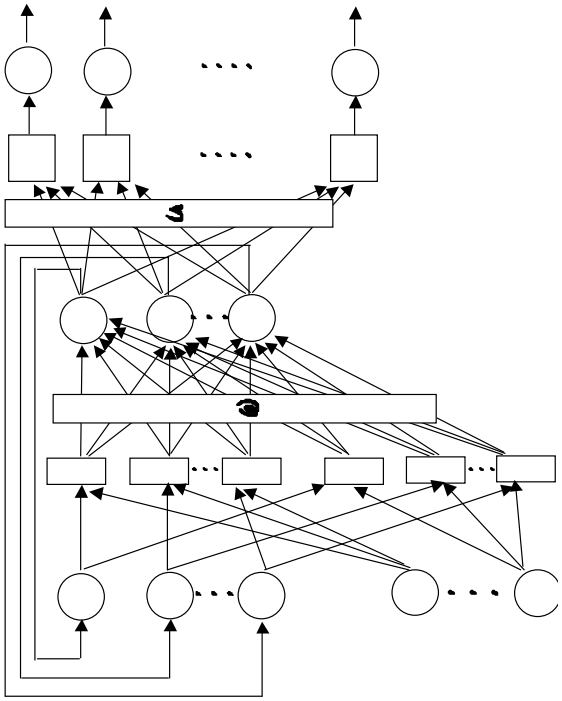


Fig. 2: Neural network used for fuzzy graininar inference

$$\frac{\partial E}{\partial w_{lon}} =$$

$$\sum_{r=0}^{M-1}(T_r - O_r)g'(\sigma_r)\sum_{p=0}^{N-1}u_{rp}g'(\theta_p^{m-1})\times\left[\delta_{pl}S_0^{m-1}I_n^{m-1}+\sum_{j=0}^{N-1}\sum_{k=0}^{L-1}w_{pjk}I_k^{m-1}\frac{\partial S_j^{m-1}}{\partial w_{lon}}\right]\cdot$$

$$\text{Initialize } \frac{\partial S_j^0}{\partial w_{lon}}=0$$

Dynamic adaptation of recurrent neural network architecture: Constructive algorithms dynamically grow the structure during the network's training. Various types of network growing methods have been proposed[10-14]. However, a few of them are for recurrent networks. The approach we propose in this paper is a constructive algorithm for second-order single-layer recurrent neural networks.

In the course of the induction of an unknown fuzzy finite-finite state automata from training samples, our algorithm topologically changes the network in addition to adjusting the weights of the second-order recurrent neural network. We utilize the idea of maximizing correlation in cascade-correlation algorithm to get a prior knowledge at each stage of the dynamic training which is then used to initialize the new generated network for the following learning.

Before the description of the constructive learning, the following criteria need to be kept in mind:

* When the network structure needs to change
* How to connect the newly created neuron to the existing network
* How to assign initial values to the newly added connection weights

We initialize the network with only one hidden unit. The size of the input and output layer are determined by the I\O representation the experimenter has chosen.

We present the whole training examples to the recurrent neural network. When no significant error reduction has occurred or the training epochs have achieved a preset number, we measure the error over the entire training set to get the residual error signal according to (4). Here an epoch is defined as a training cycle in which the network sees all training samples. If the residual error is small enough, we stop; if not, we attempt to add new hidden unit one by one to the single hidden layer using the unit-creation algorithm to maximize the magnitude of the correlation between the residual error and the candidate unit's output.

For the second criterion, the newly added hidden neuron receives second-order connection from the inputs and pre-existing hidden units and then outputs to the output layer and recurrent layer as shown in Fig. 3.

Now, we turn to the unit-creation algorithm in our method, it is different from the case of cascade-correlation due to the different structure we use. We define S as in[10]:

$$S=\sum\left|\sum(\quad-\overline{\quad})(\quad,\quad-\overline{\quad\quad})\right| \qquad (5)$$

where V_p is the candidate unit's output value when the network processing the pth training example and $E_{p,o}$ is the residual error observed at output neuron o when the network structure needs to change. and E_p are the values of V and E_p averaged over the whole training examples.

During the maximization of S, all the pre-exiting weights are frozen other than the input weights of the candidate unit and its connection with the hidden units. The following training algorithm updates the weights at the end of the presentation of the whole training examples. Suppose we are ready to add the hth neuron, p denote the length of the pth example, then V =S , i.e. the output of the hth hidden neuron at time p .

$$\Delta w_{hjk} = \beta\frac{\partial S}{\partial w_{hjk}} = \beta\sum_{,}\sigma\ (E_{\ ,}-\overline{\quad})$$

$$\frac{\partial V_p}{\partial w_{hjk}} \quad j=1,\ldots h,\ k=1,\ldots|\Sigma| \qquad (6)$$

$$\Delta w_{ihk} = \beta\frac{\partial S}{\partial w_{ihk}} = \beta\sum_{,}\sigma\ (E_{\ ,}-\overline{\quad})$$

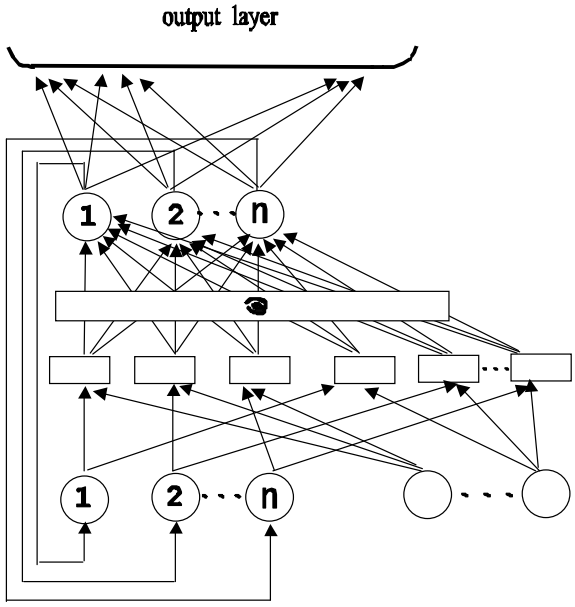


Fig. 3: The network starts with neuron 1 and grows incrementally to n neurons

$$\frac{\partial V_p}{\partial w_{ihk}} \quad i=1,\ldots h-1,\ k=1,\ldots |\Sigma| \qquad (7)$$

where σ_o is the sign of the correlation between the candidate's value and output o due to the modulus in (5), β is the learning rate, w_{hjk} are the trainable input weights to the newly-added neuron h; w_{ihk} connect h to the pre-existing ith hidden unit and $|\Sigma|$ is the number of the input units. According to (1), we further induce:

$$\frac{\partial V_p}{\partial w_{hfm}} = \frac{\partial S_h^{p_l}}{\partial w_{hfm}} = g'xS^{-1} \times I \quad \text{where } f \neq h$$

$$\frac{\partial V_p}{\partial w_{hhm}} = \frac{\partial S_h^{p_l}}{\partial w_{hhm}} = g'x(S^{-1} I + \sum_k w_{hhk} I_k^{p_l} \frac{\partial S_h^{p_l-1}}{\partial W_{hhm}}) \text{ where } f=h \qquad (8)$$

$$\frac{\partial S_h^{p_l}}{\partial w_{fhm}} = g' \times [\sum_k W_{hfk} I^{-1} \times (g'x(S_h^{p_l-2} I_m^{p_l-1} + \sum_k W_{ffk} I^{-1} \frac{\partial S_f^{p_l-2}}{\partial W_{fhm}}))]$$

$$\text{Initialize } \frac{\partial S_f^0}{\partial w_{fhm}} = 0 \qquad (9)$$

Where g' is the derivation for the pth example of the pre-existing or candidate unit 's activation function with respect to the sum of its inputs, α is the learning rate.

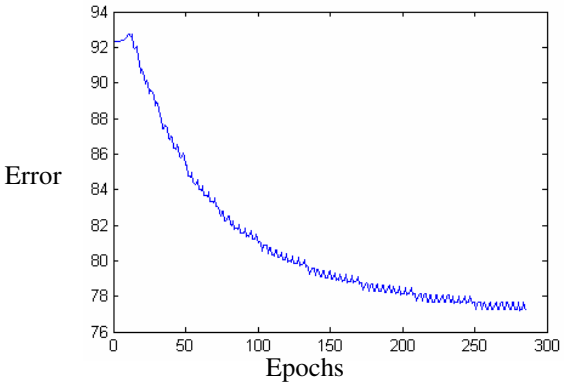


Fig. 4: The evolution of the training process for 1 hidden neuron network

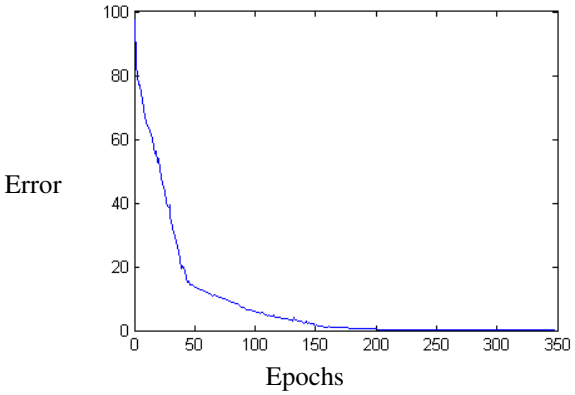


Fig. 5: The evolution of the training process for current network

We present the whole training examples to the current network and train W and W according to the update rules as described in (6)-(9) until we achieve a pre-set number of training epochs or S stop increasing.

At this time, we fix the trained W_{hjk} and W_{jhk} as the initial weights of the newly-added neuron, its connections to the output layer are set to zero or very small random numbers so that they initially have little or no effect on training. The old weights start with their previously trained values. Compared with the cascade-correlation in which only the output weights of the new neuron is trainable, we allow all the weights are trainable. In order to preserve as much knowledge we've learned as possible, we set different learning rate to update the weights we obtained and the output weights of the new neuron connecting to the output layer where the latter is larger than the former. Up to now, the third criterion is resolved.

Instead of a single candidate unit, a pool of candidates is possibly used for the unit-creation algorithm which are trained in parallel from different random initial weights. They all receive the same input signals and see the same residual error for each training example. Whenever we decide that no further progress of S is being made, we install the candidate whose

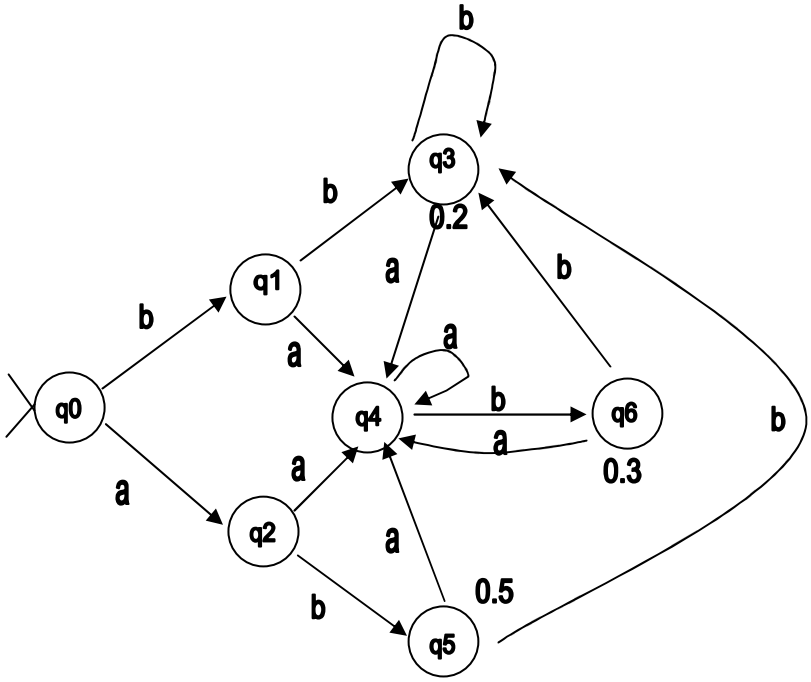


Fig. 6: The extracted automaton from learned network

correlation score is the best with the trained input weights.

Simulation experiment: In this part, we have tested the construction algorithm on fuzzy grammar inference where the training examples are generated from the fuzzy automaton shown in Fig. 1a. The experiment has been designed in the same way as in[9].

1. Generation of examples: We start from a fuzzy automaton M as shown in Fig. 1a, from which we generate a set of 200 examples recognized by M. 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)[9] for detail. The samples strings are ordered according to their length.
2. Forgetting M and train a dynamic network to induce an unknown automaton which recognizes the training examples:

The initial recurrent neural network is composed of 1 recurrent hidden neuron and 11 neurons in the output layer, i.e., a decimal accuracy is required. The parameters for the network learning are: learning rate α =0.3, error tolerance ε =2.5x10^{-5}. Figure 4 shows the evolution of the training process for 1-hidden unite network. After 285 epochs, no significant error reduction has occurred and the residual error signal is 77.3695. It's much larger than ε . So we add a neuron to the hidden layer.

We use 8 candidate units, each with a different set of random initial weights. Learning rate β is set to 0.3. According to (5)-(9) to maximize S in parallel until a preset number of epochs are achieved or S stop increasing, we install the candidate whose correlation score is the best. After 225 epochs, no significant increasing of S has appeared. Initial the new network as described earlier and train it with learning rate 0.15 for the weights connecting the newly-added neuron to the output layer and 0.08 for the rest respectively. The neural network learned all the examples in epochs 348. Figure 5 shows the evolution of the error on the training process. We adopt extraction algorithm 1 in[8]-partitioning of the output space—to extract fuzzy finite automaton from learned network:

CONCLUSION

An appropriate topology made up of a second-order recurrent neural network linked to an output layer is proposed in[8] to perform fuzzy grammatical inference. However, an appropriate number of the hidden unites is difficult to be predicted. Instead of just adjusting the weights in a network of fixed topology, we adopt the construction method for this application. We apply the idea of maximizing correlation of the cascade-correlation algorithm to the second-order recurrent neural network to generate a new construction method and use it to infer fuzzy grammar from examples. At every stage of dynamical training, we obtain a prior knowledge to initialize a newly generated network and train it until no further decrease of the error being made. It recovers the absence of the empiric in the case of the fixed-topology network and generates an optimal topology

automatically. An experiment shows that this algorithm is practical.

REFERENCES

1. Mozer, M.C. and P. Smolensky, 1989. Skeletonization: A technique for trimming the fat from a network via relevance assessment. Connection Sci., 11: 3-26.
2. Giles, C.L., C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun and Y.C. Lee, 1992. Learning and extracting finite state automata with second-order recurrent neural networks. Neural Computation, 4: 393-405.
3. Zeng, Z., R. Goodman and P. Smyth, 1993. Learning finite state machines with self-clustering recurrent networks. Neural Computation, 5: 976-990.
4. Elman, J.L., 1991. Distributed representations, simple recurrent networks and grammatical structure. Machine Learning, 7: 195-225.
5. Cleeremans, A., D. Servan-Schreiber and J.L. McClelland, 1989. Finite state automata and simple recurrent networks. Neural Computation, 1: 372-381.
6. Williams, R.J. and D. Zipser, 1989. A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 1: 270-280.
7. Thomason, M. and P. Marinos, 1974. Deterministic acceptors of regular fuzzy languages. IEEE Trans. Systems, Man and Cybernetics, 3: 228-230.
8. 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.
9. Blanco, A., M. Delgado and M.C. Pegalajar, 2001. Fuzzy automaton induction using neural network. Intl. J. Approximate Reasoning, 27: 1-26.
10. S. Fahlman, “the cascade-correlation learning architecture”, in advances in neural information processing systems 2(D. Touretzky, ed.),(San Mateo, CA), pp. 524-532, Morgan Kaufmann Publishers, 1990.
11. Ash, T., 1989. Dynamic node creation in back-propagation networks. Connection Sci., 1: 365-375.
12. Frean, M., 1990. The upstart algorithm: a method for constructing and training feedforward neural networks. Neural Computation, 2: 198.
13. Gallant, S.I., 1986. Three constructive algorithms for network learning. Proc. 8th Ann. Conf. Cognitive Science Society, pp: 652-660.
14. Hirose, Y., K. Yamashita and S. Hijiya, 1991. Back-propagation algorithm which varies the number if hidden units. Neural Networks, 4: 61-66.