Learning Finite State Machines With Self-Clustering Recurrent Networks

Zheng Zeng, Rodney M. Goodman
Department of Electrical Engineering, 116-81
California Institute of Technology
Pasadena, CA 91125

Padhraic Smyth
Jet Propulsion Laboratory, 238-420
California Institute of Technology
Pasadena, CA 91109

March 11, 1993

Abstract

Recent work has shown that recurrent neural networks have the ability to learn finite state automata from examples. In particular, networks using 2nd-order units have been successful at this task. In studying the performance and learning behavior of such networks we have found that the 2nd-order network model attempts to form clusters in activation space as its internal representation of states. However, these learned states become unstable as longer and longer test input strings are presented to the network. In essence, the network "forgets" where the individual states are in activation space. In this paper we propose a new method to force such a network to learn stable states by introducing discretization into the network and using a pseudo-gradient learning rule to perform training. The essense of the learning rule is that in doing gradient decent, it makes use of the gradient of a sigmoid function as a heuristic hint in place of that of the hard-limiting function, while still using the discretized value in the feedback update path. The new structure uses isolated points in activation space instead of vague clusters as its internal representation of states. It is shown to have similar capabilities in learning finite state automata as the original network, but without the instability problem. The proposed pseudo-gradient learning rule may also be used as a basis for training other types of networks which have hard-limiting threshold activation functions.

1 Introduction

Theoretical aspects of grammatical inference have been studied extensively in the past [1, 2, 12, 13]. A variety of direct search algorithms have been proposed for learning grammars from positive and negative examples (strings) [3, 9, 17, 21]. More recently recurrent neural networks have been investigated as an alternative method for learning simple grammars [5, 6, 7, 8, 10, 11, 15, 18, 19, 22]. A variety of network architectures and learning rules have been proposed. All have shown the capability of recurrent networks to learn different types of simple grammars from examples.

In this paper we restrict the focus to studying a recurrent network's behavior in learning regular grammars, which are the simplest type of grammar in the Chomsky hierarchy, and have a one-to-one correspondence to finite state machines [14]. A regular language can be defined as the language accepted by its corresponding finite state acceptor: $< \Sigma, T, t_0, \delta, F >$, where

- Σ is the input alphabet.
- T is a finite nonempty set of states.
- t_0 is the start (or initial) state, an element of T.
- δ is the state transition function; $\delta : T \times \Sigma \rightarrow T$.
- F is the set of final (or accepting) states, a (possibly empty) subset of T.

The purpose of the study is to obtain a better understanding of recurrent neural networks, their behavior in learning, and their internal representations, which in turn may give us more insight into their capability for fulfilling other more complicated tasks.

Giles et al. have proposed a "2nd-order" recurrent network structure to learn regular languages [10, 11]. Henceforth, all references to 2nd-order recurrent networks imply the network structure described in [10] and [11]. Our independent experiments have confirmed their results that 2nd-order nets can learn various grammars well. In addition, we found that this structure learns these grammars more easily than the simple recurrent network structure (or the Elman structure) [7] which does not use 2nd-order units. However, a stability problem emerges with trained networks as longer and longer input strings are presented (similar behavior in recurrent networks has been found in different contexts [20, 18]). In our experiments, the problem appears in 14 out of 15 of the trained networks on nontrivial machines for long strings. A string can be misclassified as early as when its length is only 30% longer than the maximum training length in some of the experiments. The stability problem led us to look deeper into the internal representation of states in such a network and the following interesting behavior was observed: during learning, the network attempts to form clusters in hidden unit space as its representation of states. This behavior occurred in all the learning experiments we performed. Once formed, the clusters are stable for short strings, i.e., strings with lengths not much longer than the maximum length of training strings. However, in 14 out of 15 learned networks, when sufficiently long strings are presented for testing, the clusters (states) start to merge and ultimately become indistinguishable. (Details of these experiments will be explained in Section 2.), To solve this problem we propose a discretized combined network structure, as well as a pseudo-gradient learning method, which can be shown to successfully learn stable state representations. In the proposed network, instead of clusters, the states of the network are actually isolated points in hidden unit activation space.

2 The "Unstable State" Behavior of a Learned 2nd-Order Net

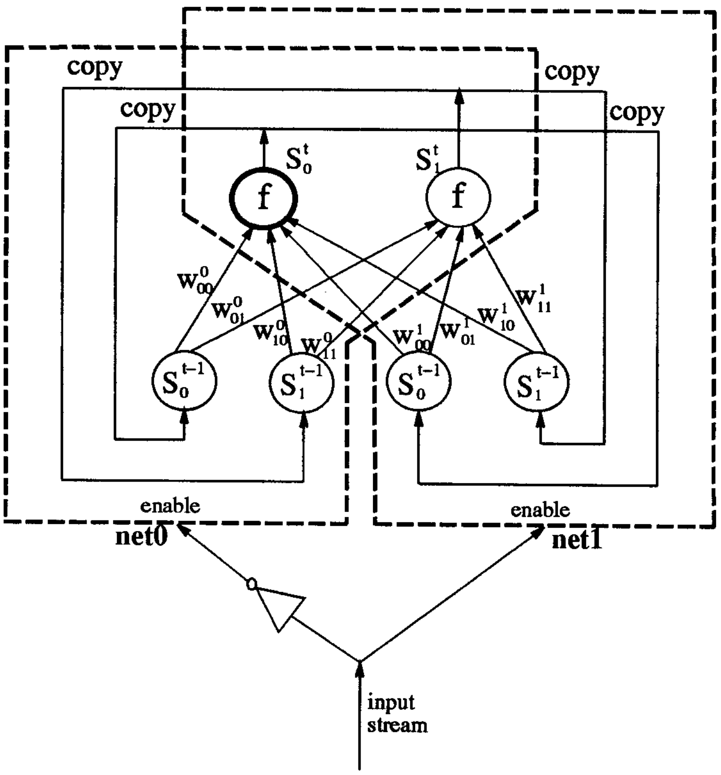


Figure 1: Equivalent 1st order structure of 2nd-order network

We found that the 2nd-order network can be represented as two separate networks controlled by a gating switch (Figure 1) as follows: the network consists of two 1st order networks with shared hidden units. The common hidden unit values are copied back to both **net0** and **net1** after each time step, and the input stream acts like a switching control to enable or disable one of the two nets. For example, when the current input is 0, **net0** is enabled while **net1** is disabled. The hidden unit values are then decided by the hidden unit values from the previous time step weighted by the weights in **net0**. The hidden unit activation function is the standard sigmoid function, $f(x) = \frac{1}{1+e^{-x}}$. Note that this representation of a 2nd-order network, as two networks with a gating function, provides insight into the nature of 2nd-order nets, i.e., clearly they have greater representational power than a single simple recurrent network, given the same number of hidden units. This structure was used in our initial experiments.

We use S_i^t to denote the activation value of hidden unit number i at time step t. w_{ij}^n is the weight from layer 1 node j to layer 2 node i in **net**n. $n = 0$ or 1 in the case of binary inputs. Hidden node S_0^t is chosen to be a special indicator node, whose activation should be close to 1 at the end of a legal string, or close to 0 otherwise. At time $t = 0$, initialize S_0^0 to be 1 and all other S_i^0's to be 0, i.e., assume that the null string is a legal string. The network weights are initialized randomly with a uniform distribution between -1 and 1.

In the experiments described here we used the following grammars:

- Tomita grammars [21]:
  - #1 — 1*.
  - #4 — any string not containing "000" as a substring.
  - #5 — even number of 0's and even number of 1's.
  - #7 — 0*1*0*1*.
- Simple vending machine [4]: The machine takes in 3 types of coins: nickel, dime and quarter. Starting from empty, a string of coins is entered into the machine. The machine "accepts", i.e., a candy bar may be selected, only if the total amount of money entered exceeds 30 cents.

A training set consists of randomly chosen variable length strings with length uniformly distributed between 1 and L_{max}, where L_{max} is the maximum training string length. Each string is marked as "legal" or "illegal" according to the underlying grammar. The learning procedure is a gradient descent method in weight space (similar to that proposed by Williams and Zipser [22]) to minimize the error at the indicator node for each training string [11].

In a manner different from that described in [11], we present the whole training set (which consists of 100 to 300 strings with L_{max} in the range of 10 to 20), all at once to the network for learning, instead of presenting a portion of it in the beginning and gradually augmenting it as training proceeds. Also, we did not add any *end* symbol to the alphabet as in [11]. We found that the network can successfully learn the machines (2–10 states) we tested on, with a small number of hidden units (4–5) and less than 500 epochs, agreeing with the results described in [11].

To examine how the network forms its internal representation of states, we recorded the hidden unit activations at every time step of every training string in different training epochs. As a typical example, shown in Figure 2(a)-(e) are the $S_0 - S_3$ activation-space records of the learning process of a 4-hidden-unit network. The underlying grammar was Tomita #4, and the training set consisted of 100 random strings with $L_{max} = 15$. Note that here the dimension S_0 is chosen because of it being the important "indicator node", and S_3 is chosen arbitrarily. The observations that follow can be made from any of the 2-D plots from any run in learning any of the grammars in the experiments. Each point corresponds to the activation pattern of a certain time step in a certain string. Each plot contains the activation points of *all* time steps for *all* training strings in a certain training epoch as described in the caption. The following behavior can be observed:

1. As learning takes place, the activation points seem to be pulled in several different directions, and distinct clusters gradually appear (Figure 2(a)-(e)).

2. After learning is complete, i.e., when the error on each of the training strings is below a certain tolerance level, the activation points form distinct clusters, which consist of segments of curves (Figure 2(e)).

3. Note in particular that there exists a clear gap between the clusters in the S_0 (indicator) dimension, which means that the network is making unambiguous decisions for all the training strings and each of their prefix strings (Figure 2(e)).

4. When given a string, the activation point of the network jumps from cluster to cluster as input bits are read in one by one. Hence, the behavior of the network looks just like a state machine's behavior.

It is clear that the network attempts to form clusters in activation space as its own representation of states and is successful in doing so. Motivated by these observations, we applied the k-means clustering algorithm to the activation record in activation space of the trained network to extract the states (instead of simply dividing up the space evenly as in [11]). In choosing the parameter k, we found that if k was chosen too small, the extracted machine sometimes could not classify all the training strings correctly, while a large k always guaranteed perfect performance on training data. Hence, k was chosen to be a large number, for example, 20.

The initial seeds were chosen randomly. We then defined each cluster found by the k-means algorithm to be a "state" of the network and used the center of each cluster as a representative of the state. The transition rules for the resulting state machine are calculated by setting the S_i^{t-1} nodes equal to a cluster center, then applying an input bit (0 or 1 in binary alphabet case), and calculating the value of the S_i^t nodes. The transition from the current state given the input bit is then to the state that has a center closest in Euclidean distance to the obtained S_i^t values. In all our experiments, the resulting machines were several states larger than the correct underlying minimal machines. Moore's state machine reduction algorithm was then applied to the originally extracted machine to get an equivalent minimal machine which accepts the same language but with the fewest possible number of states. Similar to the results in [11], we were able to extract machines that are equivalent to the minimal machines corresponding to the underlying grammars from which the data was generated.

These trained networks perform well in classifying unseen short strings (not much longer than L_{max}). However, as longer and longer strings are presented to the network, the percentage of strings correctly classified drops substantially. Shown in Figure 2(f) is the recorded activation points for S_0-S_3 of the same trained network from Figure 2(e) when long strings are presented. The original net was trained on 100 strings with $L_{max} = 15$, whereas the maximum length of the test strings in Figure 2(e) was 50. Activation points at all time steps for all test strings are shown.

Several observations can be made from Figure 2(e):

1. The well-separated clusters formed during training begin to merge together for longer and longer strings and eventually become indistinguishable. These points in the center of Figure 2(e) correspond to activations at time steps longer than $L_{max} = 15$.

2. The gap in the S_0 dimension disappears, which means that the network could not make hard decisions on long strings.

3. The activation points of a string stay in the original clusters for short strings and start to diverge from them when strings become longer and longer. The diverging trajectories of the points form curves with sigmoidal shape.

Similar behavior was observed for 14 out of 15 of the networks successfully trained on different machines, excluding the vending machine model. Some of the networks started to misclassify as early as when the input strings were only 30% longer than L_{max}. Each of these 14 trained networks made classification errors on randomly generated test sets with maximum string length no longer than $5L_{max}$. The remaining one network was able to maintain a stable representation of states for very long strings (up to length 1000). Note that the vending machine was excluded because it's a trivial case for long strings, i.e., all the long strings are legal strings so there's no need to distinguish between them. This is not the case for the other machines.

3 A Network That Can Form Stable States

From the above experiments it is clear that even though the network is successful in forming clusters as its state representation during training, it often has difficulty in creating *stable* clusters, i.e., to form clusters such that the activation points for long strings converge to certain centers of each cluster, instead of diverging as observed in our experiments. The problem can be considered as inherent to the structure of the network where it uses analog values to represent states, while the states in the underlying state machine are actually discrete. One intuitive suggestion to fix the problem is to replace the analog sigmoid activation function in the hidden units with a threshold function:

$$D(x) = \begin{cases} 1.0 & \text{if } x \geq 0.5 \\ 0.0 & \text{if } x < 0.5. \end{cases}$$

In this manner, once the network is trained, its representation of states (i.e., activation pattern of hidden units) will be stable and the activation points won't diverge from these state representations once they are formed. However, there is no known method to train such a network, since one can not take the gradient of such activation functions.

An alternative approach would be to train the original 2nd-order network as described earlier, but to add the discretization function $D(x)$ on the copy back links during testing. The problem with this method is that one does not know *a priori* where the formed clusters from training will be. Hence, one does not have good discretization values to threshold the analog values in order for the discretized activations to be reset to a cluster center. Experimental results have confirmed this prediction. For example, after adding the discretization, the modified network can not even correctly classify the training set which it has successfully learned in training. As in the previous example, after training and without the discretization, the network's classification rate on the training set was 100%, while with the discretization added, the rate became 85%. For test sets of longer strings, the rates with discretization were even worse.

We propose that the discretization be included in *both* training and testing in the following manner: Figure 3 shows the structure of the network with discretization added.

From the formulae below, one can clearly see that in operational mode, i.e. when *testing*, the network is equivalent to a network with discretization only:

$$h_i^t = f(\sum_j w_{ij}^{x^t} S_j^{t-1}), \qquad \forall i, t,$$

$$
\begin{aligned}
S_i^t &= D(h_i^t), & where\ \ D(x) = \begin{cases} 0.8 & \text{if } x \geq 0.5 \\ 0.2 & \text{if } x < 0.5, \end{cases} \\
\Rightarrow S_i^t &= D(\ f(\sum_j w_{ij}^{x^t} S_j^{t-1})\ ) & \\
&\equiv D_0(\sum_j w_{ij}^{x^t} S_j^{t-1}), & where\ \ D_0(x) = \begin{cases} 0.8 & \text{if } x \geq 0.0 \\ 0.2 & \text{if } x < 0.0. \end{cases}
\end{aligned}
$$

(Here x^t is the input bit at time step t. We use h_i^t to denote the analog value of hidden unit i at time step t, and S_i^t the discretized value of hidden unit i at time step t.)

Hence, the sigmoid nodes can be eliminated in testing to simplify computation.

During training, however, the gradient of the soft sigmoid function is made use of in a pseudo-gradient method for updating the weights. The next section explains the method in more detail.

By adding these discretizations into the network, one might argue that the capacity of the net is greatly reduced, since each node can now take on only 2 distinct values, as opposed to infinitely many values (at least in theory) in the case of the undiscretized networks. However, in the case of learning discrete state machines, the argument depends on the definition of the capacity of the analog network. In our experiments, 14 out of 15 of the learned networks have unstable behavior for nontrivial long strings, so one can say that the capabilities of such networks to distinguish different states may start high, but deteriorate over time, and would eventually become zero.

4 The Pseudo-Gradient Learning Method

During training, at the end of each string: $x^0, x^1, ..., x^L$, the mean squared error is calculated as follows (note that L is the string length, h_0^L is the analog indicator value at the end of the string):

$$E = \frac{1}{2}(h_0^L - T)^2,$$

where

$$T = target = \begin{cases} 1 & \text{if ``legal''} \\ 0 & \text{if ``illegal''}. \end{cases}$$

Update w_{ij}^n, the weight from node j to node i in net$n$, at the end of each string presentation:

$$w_{ij}^n = w_{ij}^n - \alpha \frac{\widetilde{\partial E}}{\partial w_{ij}^n}, \qquad\qquad \forall n, i, j,$$

$$\frac{\widetilde{\partial E}}{\partial w_{ij}^n} = (h_0^L - T)\frac{\widetilde{\partial h_0}^L}{\partial w_{ij}^n}, \qquad\qquad \forall n, i, j,$$

where $\frac{\widetilde{\partial}}{\partial w_{ij}^n}$ is what we call the "pseudo-gradient" with respect to w_{ij}^n.

To get the pseudo-gradient $\frac{\widetilde{\partial h_0}^L}{\partial w_{ij}^n}$, pseudo-gradients $\frac{\widetilde{\partial h_k}^t}{\partial w_{ij}^n}$ for all t, k need to be calculated forward in time at each time step:

$$\frac{\widetilde{\partial h_k}^t}{\partial w_{ij}^n} = f' \cdot (\sum_l w_{kl}^{x^t} \frac{\widetilde{\partial h_l}^{t-1}}{\partial w_{ij}^n} + \delta_{ki}\delta_{nx^t} S_j^{t-1}), \quad \forall i, j, n, k, t.$$

(Initially, set: $\frac{\widetilde{\partial h_k}^0}{\partial w_{ij}^n} = 0, \qquad \forall i, j, n, k.$)

As can be seen clearly, in carrying out the chain rule for the gradient we replace the real gradient $\frac{\partial S_l^{t-1}}{\partial w_{ij}^n}$, which is zero almost everywhere, by the pseudo-gradient $\frac{\widetilde{\partial h_l}^{t-1}}{\partial w_{ij}^n}$. The justification of the use of the pseudo-gradient is as follows: suppose we are standing on one side of the hard threshold function $S(x)$, at point $x_0 > 0$, and we wish to go downhill. The real gradient of $S(x)$ would not give us any information, since it is zero at x_0. If instead we look at the gradient of the function $f(x)$, which is positive at x_0 and increases as $x_0 \to 0$, it tells us that the downhill direction is to decrease x_0, which is also the case in $S(x)$. In addition, the magnitude of the gradient tells us how close we are to a step down in $S(x)$. Therefore, we can use that gradient as a heuristic hint as to which direction (and how close) a step down would be. This heuristic hint is what we used as the pseudo-gradient in our gradient update calculation.

5 Experimental Results

Shown in Figure 4(a),(b),(c) are the $h_0 - h_3$ activation-space records of the learning process of a discretized network (h values are the undiscretized values from the sigmoids). The underlying grammar is again the Tomita Grammar #4. The parameters of the network and the training set are the same as in previous case. Again, any of the other 2-D plots from any run in learning any of the grammar in the experiments could have been used here.

Figure 4(c) is the final result after learning, where the weights are fixed. Notice that there are only a finite number of points in the final plot in the analog activation h-space due to the discretization. Figure 4(d) shows the discretized value plot in $S_0 - S_3$, where only 3 points can be seen. Each point in the discretized activation S-space is automatically *defined* as a distinct state, no point is shared by any of the states. The transition rules are calculated as before, and an internal state machine in the network is thus constructed. In this manner, the network performs self-clustering. For this example, 6 points are found in S-space, so a 6-state-machine is constructed as shown in Figure 5(a). Not surprisingly this machine reduces by Moore's algorithm to a minimum machine with 4 states which is exactly the Tomita Grammar #4 (Figure 5(b)). Similar results were observed for all the other grammars in the experiments.

There are several advantages in introducing discretization into the network:

1. Once the network has successfully learned the state machine from the training set, it's internal states are stable. The network will always classify input strings correctly, independent of the length of these strings.

2. No clustering is needed to extract out the state machine, since instead of using vague clusters as its states, the network has formed distinct, isolated points as states. Each point in activation space is a *distinct* state. The network behaves *exactly* like a state machine.

3. Experimental results show that the size of the state machines extracted out in this approach, which need not be decided manually (no need to choose k for k-means) as in the previous undiscretized case, are of a much smaller size than found previously by the clustering method.

It should be noted that convergence has a different meaning in the case of training discrete networks as opposed to the case of training analog networks. In the analog networks' case, learning is considered to have converged when the error for each sample is below a certain *error tolerance level.* In the case of discrete networks, however, learning is only stopped and considered to have converged when *zero error* is obtained on all samples in the training set. In the experiments reported in this paper the analog tolerance level was set to 0.2. The discretized networks took on average 30% longer to train in terms of learning epochs compared to the analog networks for this specific error tolerance level.

6 Conclusion

In this paper we explored the formation of clusters in hidden unit activation space as an internal state representation for 2nd-order recurrent networks which learn regular grammars.

These states formed by such a network during learning are not a stable representation, i.e., when long strings are seen by the network the states merge into each other and eventually become indistinguishable.

We suggested introducing hard-limiting threshold discretization into the network and presented a pseudo-gradient learning method to train such a network. The method is heuristically plausible and experimental results show that the network has similar capabilities in learning finite state machines as the original 2nd-order network, but is stable regardless of string length since the internal representation of states in this network consists of isolated points in activation space.

The proposed pseudo-gradient learning method suggests a general approach for training networks with threshold activation functions.

Acknowledgments

The research described in this paper was supported in part by DARPA under grant number AFOSR–90–0199. In addition this work was carried out in part by the Jet Propulsion Laboratories, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] D. Angluin, "Inference of reversible languages," *Journal of the Association for Computing Machinery,* Vol.29, No.3, pp.741-765, 1972.

[2] D. Angluin, "On the complexity of minimum inference of regular sets," *Information and Control,* Vol.39, pp.337-350, 1978.

[3] D. Angluin, C. H. Smith, "Inductive inference: theory and methods," *ACM Computing Survey,* Vol.15, No.3, p.237, 1983.

[4] J. Carroll, D. Long, *Theory of Finite Automata*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[5] A. Cleeremans, D. Servan-Schreiber, J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation,* Vol.1, pp.372-381, 1989.

[6] J. L. Elman, "Finding structure in time," *Cognitive Science,* Vol.14, pp.179-211, 1990.

[7] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning,* Vol.7, No.s 2/3, pp.195-225, 1991.

[8] S. E. Fahlman, "The recurrent cascade-correlation architecture," *Advances in Neural Information Processing Systems*, pp.190-196, 1990.

[9] K. S. Fu, *Syntactic Pattern Recognition and Applications,* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[10] C. L. Giles, G. Z. Sun, H. H. Chen, Y. C. Lee, D. Chen, "Higher order recurrent networks and grammatical inference," *Advances in Neural Information Processing Systems,* pp.380-387, 1990.

[11] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, Y. C. Lee, "Second-order recurrent neural networks," *Neural Computation,* Vol.4, No.3, pp.393-405, 1992.

[12] E. M. Gold, "System identification via state characterization," *Automatica,* Vol.8, pp.621-636, 1972.

[13] E. M. Gold, "Complexity of automaton identification from given data," *Information and Control,* Vol.37, pp.302-320, 1978.

[14] J. E. Hopfcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation,* Addison-Wesley, Reading Mass., 1979.

[15] M. I. Jordan, *Serial Order: A Parallel Distributed Processing Approach,* Tech. Rep. No.8604, San Diego: University of California, Institute for Cognitive Science, 1986.

[16] M. Kudo, M. Shimbo, "Efficient regular grammatical inference techniques by the use of partial similarities and their logical relationships," *Pattern Recognition,* Vol.21, No.4, pp.401-409, 1988.

[17] S. Muggleton, *Grammatical Induction Theory,* Addison-Wesley, Turing Institute Press, 1990.

[18] J. B. Pollack, "The induction of dynamical recognizers," *Machine Learning,* Vol.7, No.s 2/3, pp.227-252, 1991.

[19] D. E. Rumelhart,J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing,* pp.354-361, The MIT Press, 1986.

[20] D. Servan-Schreiber, A. Cleeremans, J. L. McClelland, "Graded state machines: the representation of temporal contingencies in simple recurrent networks," *Machine Learning,* Vol.7, No.s 2/3, pp.161-193, 1991.

[21] M. Tomita, "Dynamic construction of finite-state automata from examples using hill-climbing," *Proceedings of the Fourth Annual Cognitive Science Conference,* pp.105, 1982.

[22] R. J. Williams, D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation,* Vol.1, No.2, pp.270-280, 1989.

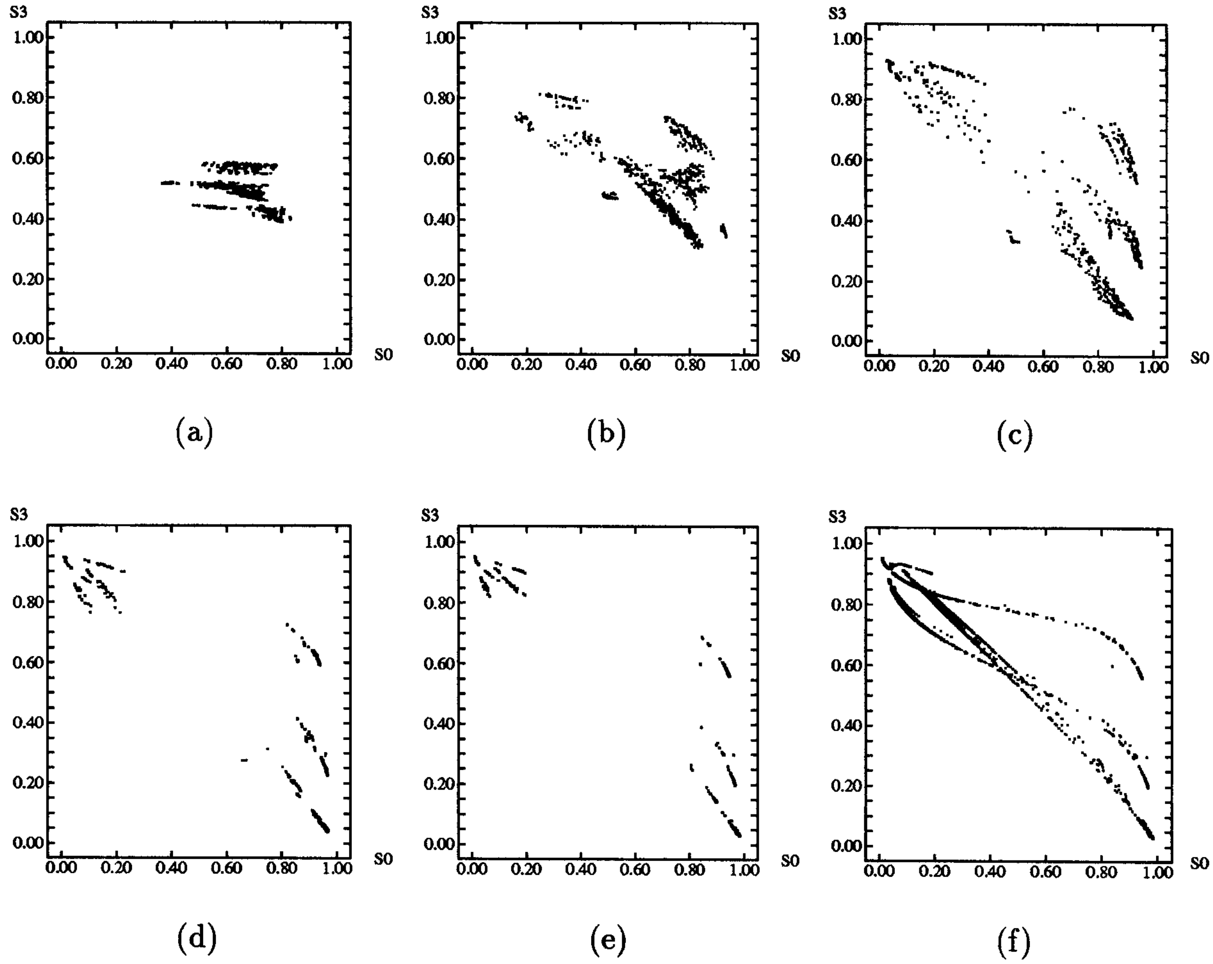


Figure 2: Hidden unit activation plot $S_0 - S_3$ in learning Tomita grammar #4. (S_0 is the x axis.) (a)-(e) are plots of all activations on the training data set. (a)During 1st epoch of training. (b)During 16th epoch of training. (c)During 21st epoch of training. (d)During 31st epoch of training. (e)After 52 epochs, training succeeds, weights are fixed. (f)After training, when tested on a set of maximum length 50.

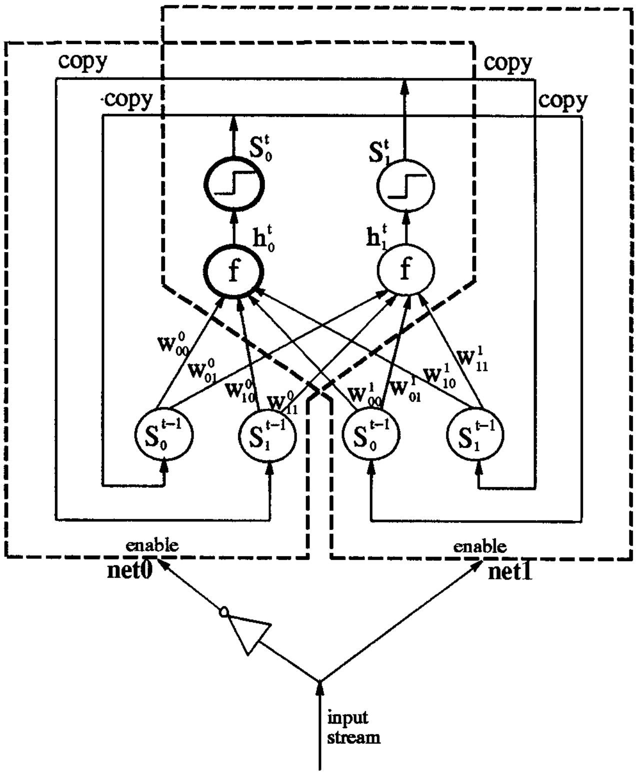
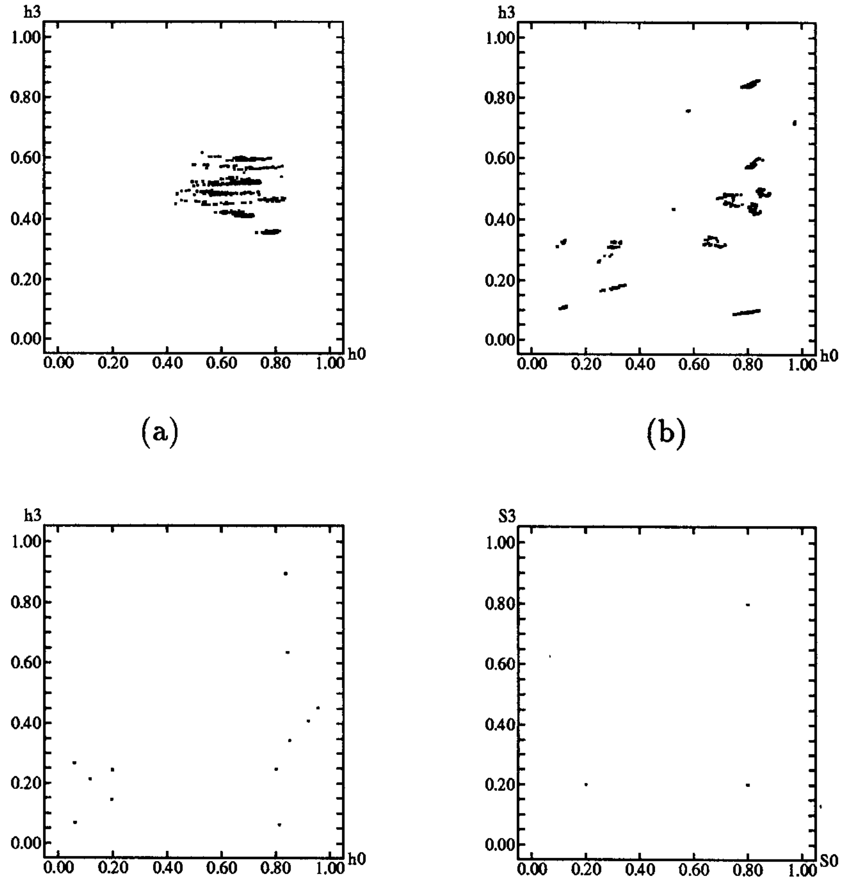


Figure 3: A combined network with discretizations



(a)

(b)

(c)

(d)

Figure 4: Discretized network learning Tomita grammar #4. (a)$h_0 - h_3$ during 1st epoch of training. (b)$h_0 - h_3$ during 15th epoch of training. (c)$h_0 - h_3$ after 27 epochs when training succeeds, weights are fixed. (d)$S_0 - S_3$, the discretized copy of $h_0 - h_3$ in (c).

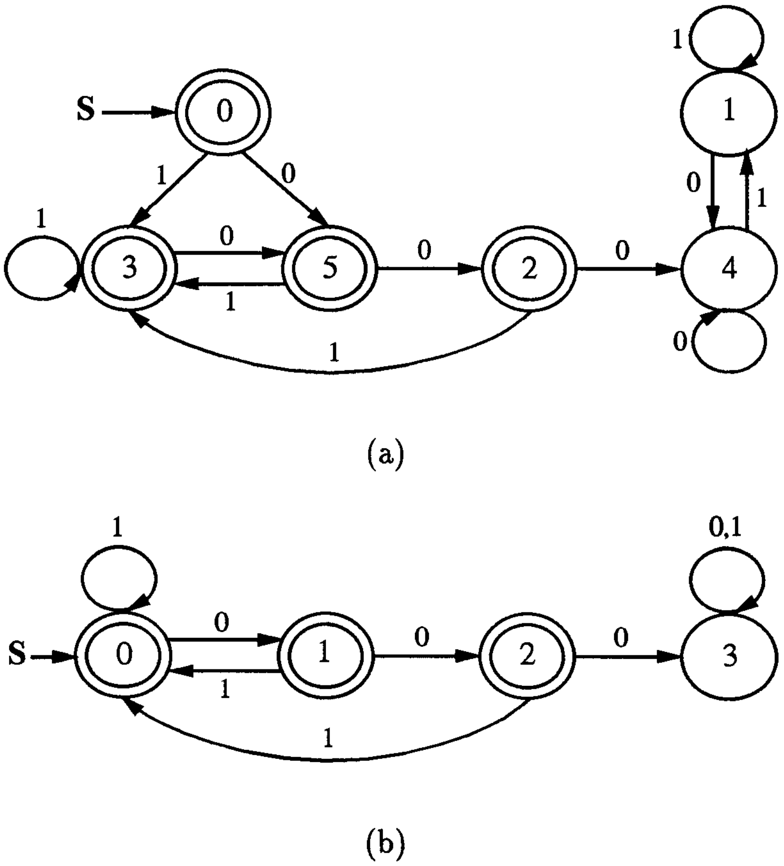


Figure 5: Extracted state machine from the discretized network after learning Tomita grammar #4. (double circle means "accept" state, single circle means "reject" state.) (a)6-state machine extracted directly from the discrete activation space. (b)Equivalent minimal machine of (a).