Discrete Recurrent Neural Networks for Grammatical Inference

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
4800 oak Grove Drive
Pasadena, CA 91109

Abstract

Recurrent neural networks have recently been shown to have the ability to learn regular and context-free grammars from examples. We show that while conventional analog recurrent networks try to form clusters in activation space to represent discrete states of the grammars during learning, and can be successful in doing so, the clusters so formed tend to become unstable as longer and longer test input strings are presented to the network. In this paper, by discretizing both the internal feedback signals and the external stack, we propose a new method to force recurrent networks to learn stable states. For training such discrete networks we propose a pseudo-gradient learning rule. The essence of the learning rule is that in doing gradient descent it makes use of the gradients of a sigmoid function as heuristic hints in place of those of the hard-limiting function, while still using the discretized values in the feedback update path and in the operations on the external stack. The new network structure uses isolated discrete points instead of "cluster clouds" as its internal representation of states in activation space. It is shown to have similar capabilities in learning regular and context-free grammars as the conventional analog recurrent 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

Recurrent neural networks have recently been investigated in terms of their ability for learning simple grammars [2, 4, 5, 6, 8, 10, 11, 12, 15]. In addition, particular types of recurrent

networks have been shown to have the ability to learn context-free grammars by using an external "continuous stack" [3]. A variety of different network architectures and learning rules have been proposed. In general, each has been shown empirically to have the capability to learn different types of simple grammars from examples. in this paper, we consider the "second-order" recurrent network structure proposed by Giles et al. in [8] -- henceforth, this particular model is referred to as the analog second-order network.

A typical analog second-order network is shown in Fig. 1. We have found that higher order networks are particularly adept at learning grammars when compared to the simple recurrent network structure (also known as the Elman structure [4, 5]) which do not use product units -- this confirms earlier work reported in [8].

We have also found that in learning a regular grammar, the analog second-order network attempts to form clusters of points in hidden unit activation space as its representation of the states of the grammar. Once formed, these clusters are stable for short strings (strings with lengths not much longer than the maximum length of training strings) in the sense that the hidden unit activations move from one distinct cluster to another as the network follows a trajectory in hidden unit space. However, it was found that for most of the learned networks, when sufficiently long strings are presented for testing, the hidden unit activations start to converge to a single cluster and the original clusters ultimately become indistinguishable[16]. Similar behavior of recurrent networks with different structures has been found in different, contexts[13, 11].

As a typical example of such behavior. Fig. 2(a)-(c) show the evolution of hidden unit activations (by way of two dimensional cross-section plots for a particular analog second-order network) *during the learning phase* for '1'omits Grammar #4(the Tomita grammar is a commonly used grammar in learning tests). This learning experiment consisted of a network with 4 hidden units being trained on a data set of 100 labeled (and randomly chosen) strings with maximum length 15. So is the "indicator" unit, whose desired activation is close to 1 at the end of legal strings, and 0 otherwise. The formation of clusters of points as learning proceeds can clearly be seen. 'The subsequent extraction of equivalent finite state machines (by means of clustering methods described in [S, 16]) demonstrates that these clusters indeed correspond to the network's internal state representation.

The stability problem emerges as the network is presented with strings much longer than the maximum length of training strings. Fig. 2(f) is a plot of tile same ('.1'0ss section of hidden unit activation space of tile trained network when unlabeled strings as long as length 50 are processed for testing. The initially well-separated clusters eventually become indistinguishable. It is clear that even though the network is successful in forming c.lusters as its state representation during training, it often has difficulty in creating *stable* clusters, i.e.,

in forming clusters such that the activation points for long strings converge to the centers of each cluster, instead of diverging as observed in our experiments. The problem can be considered as inherent to the internal representation of a network which uses analog values to represent states, while the states in the underlying state machine are actually discrete.

To achieve stability for long strings, we propose a discrete recurrent network structure which uses discretization in its feedback links. A pseudo-gradient training method is used to train the network. For context-free grammars, we use the same structure with an external discrete stack. In the proposed network, instead of clusters, the states of the network consist of isolated points in hidden unit activation space, Hence, once formed, the internal state representation is stable in a manner independent of string length.

The remaining part of the paper is organized as follows: Section 2 introduces the basic structure of second-order discrete recurrent networks, and Section 3 discusses the pseudo-gradient training method. Section 4 describes experimental results in learning regular grammars using the discrete recurrent network structure. Section 5 introduces discrete recurrent networks which use external stacks, and the pseudo-gradient training algorithm necessary for such models. Section 6 presents experimental results in learning context-free grammars using the discrete stack model. Section 7 is a brief discussion and Section 8 concludes *the* paper.

2 Basic Structure of Discrete Recurrent Networks

A discrete recurrent network can be constructed by simply taking an analog recurrent network and adding threshold units to all the feedback links. In the case of second-order networks, one can represent the structure as two separate networks controlled by a gating switch (Fig. 3) as follows: the network consists of two 1st order networks with shard hidden units. The common hidden unit values are discretized and 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 "subnetworks. " For example, when the current input is O, net O 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, $f(x) = \frac{1}{1+e^{-x}}$. The discretization function is defined to be:

$$D(x) = \begin{cases} 0.8 & \text{if } x \geq 0.5 \\ 0.2 & \text{if } x < 0.5. \end{cases} \quad (1)$$

The values 0.2 and 0.8 arc chosen instead of () and 1 here in order to give some power of influence to each of the current hidden unit values over the next time step. A unit with value

0 would eliminate any influence of that unit over the next time step. This is the general network structure which was used in our first set of experiments.

Note that this representation of a 2nd-order network, as two subnetworks with a gating function, provides insight into the nature of 2nd-order networks in general, i.e., clear] y they have greater representational power than a single simple recurrent network, given the same number of hidden units.

We use h_i^t to denote the analog value of hidden unit i at time step t, and S_i^t to denote the discretized value of hidden unit i at time step t. w_{ij}^n is the weight from layer 1, unit j to layer 2, unit i in **net**n. n = 0 or 1 in the case Of binary inputs. Hidden unit h_0^t is chosen to be a special indicator unit whose activation should be greater than 0.5 at the end of a legal string, or smaller than 0,5 otherwise. At time t = O, initialize S_0^0 to be 0.8 and all other S_i^0's to be 0.2, 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.

One intuitive suggestion to fix the stability problem is to replace the analog sigmoid activation function in each of the hidden units with the threshold function of Equation (1). 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 analog secon(l-order network as usual, but to add the discretization function $D(x)$ in the feedback links during testing. The problem with this method is that one dots 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 classily the training set which it has successfully learned in training. This was verified using the Tomita grammar described earlier: 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 during *both* training and testing using the formulae below. Note that from the formulae one can clearly see that in operational mode, i.e. when *testing*, the network is equivalent to a network with discretization only:

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

$$S_i^t = D(h_i^t), \qquad where\ D(x) = \begin{cases} 0.8 & \text{if } x \geq 0.5 \\ 0.2 & \text{if } x < 0.5, \end{cases}$$

$$\Rightarrow S_i^t = D(\ f(\sum_j w_{ij}^{x^t} S_j^{t-1})\ )$$

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

(Here x^t is the input bit at time step t.)

Hence, the sigmoid units can be eliminated during 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 this discretization into the network, one might argue that the capacity of the net is greatly reduced, since each unit can now take on only 2 distinct values. as opposed to infinitely many values (at least in theory) in the case of the undiscretized (analog) networks. However, in the case of learning discrete state machines, the argument depends on tile definition of the capacity of the analog network. Since in our experiments in [16], 14 out of 15 of the learned networks had unstable behavior for nontrivial long strings, it is difficult to define what capacity means for such unstable networks. Hence, although *in theory* a discretized network (as proposed here) may have a lower capacity than a stable analog network, a stable discretized network may be more preferable than an unstable analog network independent of their respective capacities.

3 The Pseudo-Gradient Learning Method

The question now remains as to how to train the discretized networks proposed in the last section. To this end we propose an approximation to gradient descent which wc call the pseudo-gradient learning rule. During training, at the end of each string {x", x^1, ,... x^L} the mean squared error is calculated as follows (note that L is tile string length and that h_0^L is the analog indicator value at the end of the string):

$$E = \frac{1}{2}(h_0^L - T)^2, \qquad where\ '1'\ =\ target\ =\ \begin{cases} 1 \text{ if "legal"} \\ 0 \text{ if "illegal'}). \end{cases}$$

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

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

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

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:

$$\frac{\widetilde{\partial h_k^t}}{\partial w_{ij}^n} = f' \cdot \left(\sum_l w_{kl}^{x^t} \frac{\widetilde{\partial h_l^{t-1}}}{\partial w_{ij}^n} + \delta_{ki}\delta_{nx^t} S_j^{t-1}\right), \quad \forall i, j, n, k, t. \tag{2}$$

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

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 $\frac{\partial h_l^{t-1}}{\partial w_{ij}^n}$. The justification of the use of the pseudo-gradient is as follows: suppose wc 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 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 $x_0 \rightarrow$ O, it indicates 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 c1osc we are to a step down in $S(x)$. Therefore, we can use that gradient as a heuristic hint to indicate which direction and how close a step down would be, This heuristic hint is what we usc as the pseudo-gradient in our gradient update calculation in Equation (2)

4 Experimental Results on Learning Regular Grammars

A regular language can be defined as the language accepted by its corresponding finite state acceptor: < Σ, T, t_0, δ, F >, where

- Σ is the input alphabet.
- T is a finite nonempty set of st ates.
- t_0 is the start (or initial) state, an element 01'7'.
- δ is the state transition function; δ : 'f' x $\Sigma \rightarrow$ 'f'.
- F is the set of final (or accept ing) states, a (possibly empty) subset of '/'.

The following grammars were used in the learning experiments described in this section:

- Tomita grammars [14]:
  - o #1 — 1".
  - o #4 — any string not containing "000" as a substring.
  - o #5 — an even number of O's and an even number of 1 's.
  - o #7 — 0*1*0*1*.
- Simple vending machine [1]: 'The machine takes in 3 types of coins: nickels, dimes and quarters. 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 10-state machine shown in Fig. 6(b).

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.

Table 1 shows the experimental results of training the discrete recurrent network by the pseudo-gradient learning method on these grammars. An epoch is one presentation of the whole training set to the network. The total number of characters processed is the cumulative count of all characters in all strings presented to the network in all training epochs.

As a typical example, Fig. 4(a),(h),(c) show 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 as listed in Table 1.

Fig. 4(c) is a run on an unlabeled test string set 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. Fig. 4(d) shows the discretized value plot in SO – S_3, where only 3 points can be seen. Each point in the discretized activation S-space is automatically defined as a distinct state and no point is shared by any of the states. The transition rules are calculated by setting the S_i^{t-1} units equal to one state, then applying an input bit (O or 1 in the binary alphabet case) and calculating the values of the S_i^t units. This value corresponds to a point in S-space and thus is the next state given that particular input. An internal state machine in the network is thus constructed. For this example, 6 points are found in S-space, so a 6-state-machine is constructed as shown in Fig. 5(a). Not surprisingly this machine reduces by Moore's algorithm to an equivalent minimum machine

with 4 states which is exactly the Tomita Grammar #4 (Fig. 5(b)). Similar results were observed for all the other grammars in the experiments.

Fig. 6(a) shows the extracted automaton from the network trained on the grammar for the 10-state machine. Again, Moore's algorithm reduces this 15-state automaton to the correct 10-state machine shown in Fig. 6(b).

5 Discrete Recurrent Networks with External Stacks

Regular grammars are the simplest type of grammar in the Chomsky language hierarchy[9], and have a one-to-one correspondence with finite state machines. So a network that can represent any finite state machine is sufficient for representing regular grammars. The next class of grammars in the hierarchy are called context-free or type 2 grammars. They represent a much wider class of languages than do regular grammars --- finite state machines are not sufficient enough to represent all such grammars.

The theory of finite automata and formal languages states that there exists a one-to-one correspondence between context-free languages and pushdown automata. That is, one needs to have an external stack to operate on besides the finite state machine in order to represent context-free grammars. By training the network to behave like a pushdown automaton we equivalently obtain a finite-state machine with an external stack that accepts the corresponding context-free grammar.

As in [3], we restrict the scope of context-free grammars by the following: the alphabet of the stack symbol is set to be the same as the input alphabet, only the current input symbol can be pushed onto the stack, and epsilon transitions (which can make state transitions or stack actions without reading in a new input symbol) are not allowed.

Shown in Fig. 7 is the structure of a discrete recurrent network with an external stack for the case of binary input and stack symbols. The primary differences between this structure and the one proposed in [3] arc that wc have a discrete stack as well as discretized units.

In Fig. 7 we have in effect four' 1st-order networks with shared hidden units. In addition to the input symbol which acts as control to enable or disable **net0** or netl, the current top-of-stack symbol also acts as a second gating control which enables or disables net2 or net3. Note that if the stack is empty, then both net2 and net3 are disabled, a situation that does not happen to the **net0**-**net1** pair.

As before, the unit h_0 is defined Lo be the "indicator" unit, whose activation sl ould be greater than ().5 at the end of a legal string and smaller than 0.5 otherwise. The last unit, in this case h_2, is singled out to be the "action" unit, whose activation decides what stack action to take, However, the value of this activation does not get copied back to the next

time step. If h_2 is greater than a certain value (for the experiments reported here it is set to 0.6) then the current input symbol is pushed to the stack. If it is smaller than a certain value (0.4 in our case), then a symbol is popped out of the stack. Otherwise no action is taken.

The activation functions of the h units and the discretization function of the S units are the same as defined in Section 2.

The error functions for training networks with stacks to learn context-free grammars are more complicated than for the simple grammars discussed in Section 4. Several situations can be encountered during learning, each requiring the use of a different error function. We start by basing our error functions on those proposed in [3], but there are some significant differences.

Let $h_0, h_1, ..., h_N$ be the hidden units of the network, where h_0 is the "indicator" unit and h_N is the "action" unit. Assume the current string being processed is $x^0, x^1, \ldots . x^L$, where L is the string length, and let d^t denote the depth of the stack at time step t. The different error functions are as follows:

- If the string is legal and the end of string is reached (without any attempt to pop an empty stack),

$$E = \frac{1}{2}((1 - h_0^L)^2 + (d^L)^2). \tag{3}$$

  This means that for legal strings we want both the indicator unit to be on and the stack to be empty.

- If the string is illegal and the end of string is reached (without any attempt to pop an empty stack),

$$E = \begin{cases} h_0^L - d^L & \text{if } h_0^L - d^L > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

  This means that for illegal strings we want either the stack to be nonempty, or the indicator unit to be off.

- If the network attempts to pop an empty stack at time step t,

$$E = \begin{cases} \frac{1}{2}(1 - h_N^t)^2 - d^t & \text{if the string is legal} \\ 0 & \text{if the string is illegal.} \end{cases} \tag{5}$$

  This means that for legal strings we want to correct the error of attempting to pop an empty stack by forcing the action unit value away from O, i.e., avoid the "pop stack"

action and at the same time encourage the stack to become nonempty. On the other hand, for illegal strings, we do nothing because the attempt to pop an empty stack is considered an indication that the string is illegal.

Das et al have suggested in [3] that by providing the network with a "teacher" or an "oracle" to give hints, the learning can be sped up significantly. The teacher or oracle works as follows: there are certain illegal strings which are not prefixes to any legal strings, i.e., any symbols that follow such strings do not provide any further information. Henceforth, we will call these strings dead strings. The teacher is assumed to have the ability to identify such strings. Whenever a point is reached in the input string such that no further processing of the remaining string is necessary, the teacher produces a signal and the learning is halted. The network is then trained to have another special hidden unit, designated as the "dead unit", turn on. After the network has been trained in this way, a string is considered to be classified as illegal whenever the dead unit is turned on during testing. The error functions have to be modified accordingly.

We found that it is not sufficient to add an error function only for the dead strings and to keep the other error functions (3)-(5) the same. For strings other than the dead strings, the network needs to be trained to have the dead unit turn off to avoid confusion. More specifically, letting h_1^t be the dead unit, we have:

- If the string is legal and the end of string is reached (without any attempt to pop an empty stack),

$$E = \frac{1}{2}((1 - h_0^L)^2 + (d^L)^2 + (h_1^L)^2),$$

  i.e., we want the indicator unit to be on, the stack to be empty *and the dead unit to be off.*

- If the string is illegal but not a dead string, and the end of string is reached (without any attempt to pop an empty stack),

$$E = \begin{cases} h; - d^L + \frac{1}{2}h_1^2 & \text{if } h_0^L - d^L > 0 \\ 0 & \text{otherwise,} \end{cases}$$

  i.e., we want either the stack to be nonempty, or the indicator unit to be off, *and for both cases, the dead unit to be off.* The dead unit should not be on for such strings because they could be prefixes to certain legal strings.

- If the string up to time step t is a dead string,

$$E = \begin{cases} \frac{1}{2}((1-h_1^t)^2 + (h_0^t)^2) & \text{if stack is empty} \\ \frac{1}{2}(1 - h_1^t)^2 & \text{if stack is nonempty.} \end{cases}$$

This means we want the dead unit to turn on *and either the indicator unit to turn off or the stack to be nonempty.*

- If the dead unit turns on at time step t before any possible signal for a dead string,

$$E = \frac{1}{2}(h_1^t)^2.$$

We do not want the dead unit to turn on too early since the string up to thus point could still be a prefix to certain legal strings.

- If the network attempts to pop an empty stack at time step t, before any possible signal for a dead string,

$$E = \begin{cases} \frac{1}{2}(1-h_N^t)^2 - d^t & \text{if the string is legal} \\ 0 & \text{if the string is illegal.} \end{cases}$$

Here we do not try to force the dead unit to turn on or off because it has been behaving, as desired so far.

As in Section 3, for the case with non-stack networks, the pseudo-gradient method is again used for training. The pseudo-gradients of' error functions in weight space concern both $\frac{\widetilde{\partial h_k^t}}{\partial w_{ij}^n}$ for all t, k, n, i, j, and $\frac{\widetilde{\partial d^t}}{\partial w_{ij}^n}$ for all t, n, i, j. The former is calculated the same way as before. To calculate the latter, i.e., the pseudo-gradient of the depth of the stack, we use the iterative operational equation:

$$d^t = d^{t-1} + D_1(h_N^t), \qquad \text{Where } D_1(x) = \begin{cases} 1 \text{ if' } x > 0.1 \\ -1 \text{ if } x < 0.6 \\ 0 \text{ otherwise.} \end{cases}$$

Initially, set $\frac{\widetilde{\partial d^0}}{\partial w_{ij}^n}$ = O for all n, i, j.

After each time step, update:

$$\frac{\widetilde{\partial d^t}}{\partial w_{ij}^n} = \frac{\widetilde{\partial d^{t-1}}}{\partial w_{ij}^n} + \frac{\widetilde{\partial h_N^t}}{\partial w_{ij}^n}, \qquad \forall n, i, j.$$

Here, in place of the gradient of the piece-wise step function D_1, we still use the pseudo-gradient of the action unit h_N. Although the value of the action unit dots not get discretized and copied back after each time step, its pseudo-gradient can still be calculated by utilizing the pseudo-gradients of other hidden units:

$$\frac{\widetilde{\partial h_N^t}}{\partial w_{ij}^n} = f' \cdot \left(\sum_{l=0}^{N-1} w_{Nl}^{x^t} \frac{\widetilde{\partial h_l^{t-1}}}{\partial w_{ij}^n}\right), \quad \forall i, j, n, t$$

6 Experimental Results on Learning Context-Free Grammars

We'experimented with the same grammars as in [3], i.e.,

- The parenthesis matching grammar.
- The postfix grammar.
- $a^n b^n$.
- $a^{m+n} b^m c^n$.
- $a^n\, b^n c b^m a^m$.

As in [3], a training set consists of all strings up to a certain length, with repeated legal strings so that there are about half as many legal strings as illegal ones.

Table 2(a) and (b) show the detailed results for experiments with and without hints, respectively. The numbers in each row arc averages over the successful runs (out of 10 possible successful runs) with different initial conditions --- a successful run is taken to mean that the network generalizes perfectly for ail string lengths. The number of overfitting runs indicates the number of times in the 10 runs that the network overfits the data by using too many internal states and did not generalize. The number 01" non-convergent runs is the number of times in the 10 runs that the training had not converged after 1000 epochs and was halted. Note that the number of unsuccessful runs are significantly fewer [or the case with hints than without hints — hence, hints generally improve the reliability of the learning procedure. It is still an open question as to how to avoid overfitting in general by controlling the size of the derived automaton during learning. It should be noted however that the tile networks did not overfit the data when hints were provided.

The hidden unit sizes and training set sizes shown in Table 2(a) and (b) are tile minimum sizes for which generalization could be obtained for each problem - experiments using either less training data or fewer hidden units invariably resulted in less than perfect generalization.

As an example, Fig. 8(a) and (b) show the derived pushdown automata from the networks after being trained on the parenthesis matching grammar and the $a^n b^n$ grammar respectively. As before, each state corresponds to one single point in the network's hidden unit activation space and the transition rules arc derived similarly: set the S_i^{t-1} units to each of the points(states) in the activation space, give the network different combinations 01' input and top-of-stack controls, and thus calculate the next state given such input and stack conditions.

Note that for the parenthesis matching grammar, the network finds a pushdown automaton that has one single state. Starting from an empty stack, when the input is a "(", it pushes this input onto the stack. When the input is a ")", it either POPS a "(" from the stack if the top-of-stack is a "(", or pushes the ")" onto the stack otherwise. Thus, whenever there are more ")"'s than "(" 's, the machine executes a "push stack" operation no matter what the input symbol is, making the stack nonempty (indicating an illegal string) from this point on.

Using a discrete network as well as a discrete stack results in tile advantages of a stable network, and a clear understanding of the operation of the stack. in [3], where a continuous stack was used, the results show that the trained networks do not always generalize perfectly.

From the results in Table 2, it can be seen that providing the network with hints can indeed speed up learning, or even enable the learning of the grammars in cases where the grammar could not be learned without hints . However, unlike [3], we did not find incremental presentation of the training data helped in improving the learning, Incremental presentation means that the network is initially given a small data set consisting of only short strings. After it has learned the current data set, more strings longer in length are added to the training set until all training strings are learned. We found in our experiments that once the network finds a configuration to fit the small data set with snort strings, it is sometimes very hard to drag it away from that configuration to a desired configuration that will fit the later (longer) strings as well. The training times with and without incremental presentation of strings are comparable in our experiments. The numbers listed in '1'able 2(a) and (b) are of runs with the training data set presented to the network all al once.

We postulate that the reason why incremental learning worked for analog networks but not for discrete networks is due to the nature of analog and discrete networks. The analog network always finds a "soft" solution to a data set, which only has clear decisions for snort strings, but is vague on long strings. Thus it is easy for it to "harden" such a solution when more restrictions about longer strings are enforced. The result is a solution whose "hardness" or decisiveness depends on the maximum length of the training strings. On the other hand. the discrete network always finds a "hard" solution to a data set which has clear decisions for strings of any length. once it settles in such a solution it is hard to enforce restrictions about longer strings which contradict the current solution. So one may as well provide all the restrictions to the network at once. As long as there exists sufficient information in the data set, the resulting solution does not depend on the maximum length 01' training strings.

7 Discussion

The primary advantages of introducing discretization into recurrent networks can be summarized as follows:

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

2. No manual clustering (as in [8]) is required to extract the state machine explicitly, since instead of using "cluster clouds" as its state representation, the network forms distinct, isolated points as states. Each point in activation space is a *distinct* state and, hence, the trained network behaves *exactly* like a state machine.

3. In terms of implementation the discretized recurrent network is easier to implement in hardware particularly when an external stack is used.

A reasonable question is, given that the training method for the discretized networks does not use a proper gradient descent algorithm, dots it take longer to train ? It should be noted that convergence on the training data set has a different meaning in the case of discretized networks as opposed to the case of analog networks. In the analog case, learning is considered to have converged when the error for each sample is below a certain *error tol erance level.* In the case of disc, etized networks, however, learning is only stopped and considered to nave 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 however, as pointed out already, this meant perfect training set memorization and test set genera] ization for the discretized network, whereas the analog network was often unstable for long strings.

8 Conclusion

In this paper we introduced discrete recurrent networks as a solution to stability problems with analog networks in learning regular and context-free grammars.

As opposed to analog networks, which form "cluster clouds" in activation space to represent discrete states, the state representation of our discrete networks consists 01' distinct. isolated points in activation space, and is stable regardless 01 string length.

To train such discrete networks, we present a pseudo-gradient learning method. The method is heuristically plausible and experimental results show that the network has similar

capabilities in learning regular and context-free grammars as the original analog 2nd-order net work.

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 ONR and DARPA under grant number N00014-92-J- 1860. In addition this work was carried out in part by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

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

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

[3] S. Das, C. L. Giles, G. Z. Sun, "Using prior knowledge in an NNPDA to learn context-free languages," *Advances in Neural Information Processing Systems,* 1993, in press.

[4] J. L. Elman, "Finding structure in time," *Cognitive Science,* Vol.] 4, pp.1 79-211, 1990.

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

[6] S. E. Fahlman, "l'he recurrent cascade-correlation architecture," *Advances in Neural Information Processing Systems,* pp.] 90-196, 1990.

[7] K. S. Fu, *Syntactic Pattern Recognition and Applications,* Prent ice-f lall, Englewood Cliffs, New Jersey, 1982.

[8] C. L. Giles, C. B. Miller, D. Chen, 1-1. H. Chen, G. Z. Sun, Y. C. Lee, "Second-order recurrent neural network s," *Neural Computation,* VOI.4, No.3, pp.393-405, 1992.

[9] J. E. Hopfcroft, J. D. Ullman, *Introduction to A utomata Th tory, Languages and Computation,* Addison-Wesley, Reading Mass., 1979.

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

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

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

[13] 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.*

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

[15] 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.

[16] Z. Zeng, R. Goodman, P. Smyth, "Learning finite state machines with self-clustering recurrent networks, " *Neural Computation,* in press.

grammar	training set		# of hidden units	mean # of epochs	σ of epochs	mean # of total characters
	# of strings	L_{max}				
Tomita #1	50	5	4	36.4	33.4	5205
Tomita #4	100	8	4	76.6	27.0	31712
Tomita #5	100	8	4	64.4	20.7	26662
Tomita #7	100	10	4	138.5	31.1	70774
Vending machine	365	6	5	231.8	22.4	383165
10-state machine	317	12	8	5798	- -	14315262

Table 1: Experimental results from training the discrete recurrent network on regular grammars. L_{max} is the maximum length of training strings. The numbers for epochs and total characters processed during learning are the average numbers over 5 runs with different random weight initializations, except for the 10-state machine, for which only one run was obtained. σ is the standard deviation of the epochs over the 5 runs. All runs have perfect generalization performance, i.e., 100'70 correct on strings of any length.

grammar	training set		# of hidden units	N_n	No	N_s	mean # of epochs	σ of epochs	mean # of total characters
	# of strings	L_{max}							
Parenthesis	46	6	3	0	0	10	28.8	16.3	5205
Postfix	63	7	4	1	0	9	62.3	17.1	21131
$a^n b^n$	32	6	4	2	0	8	127.3	4.9	16797
$a^{m+n} b^m c^n$	120	8	5	2	0	8	63	36.0	7560
$an\, b^n c b^m a^m$	150	7	7			1	698		516520

(a)

grammar	training set		# of hidden units	N_n	N_o	N_s	mean # of epochs	σ of epochs	mean # of total characters
	# of strings	L_{max}							
Parenthesis	180	6	3	0	0	10	12.0	10.5	11208
Postfix	371	7	4	4	2	4	185.8	149.0	408464
anbn	760	s	5		—	1	63	—	332136

(b)

Table 2: Experimental results from training the discrete recurrent network on contest-free grammars (a) with hints; (b) without hints. Tile training set and hidden unit columns indicate the fixed learning parameters for each grammar. 10 runs with different random initial weights were carried out for each grammar except for the $a^n b^n c b^m a^m$ grammar in (a) and $a^n b^n$ in (b), for which only one run each was obtained. N_s, the number of successful runs is the number of runs (of' the 10 possible) for which the trained network generalized perfectly for strings of any length. The means for the epochs and total characters processed (and the standard deviation for the epochs) were estimated only from the successful runs, N_o, the number of overfitting runs is the number where the network overfitted the data and did not generalize perfectly. N_n, the number of nol~-convergent runs is the number of runs where the network did not converge on the training data after 1000 epochs.

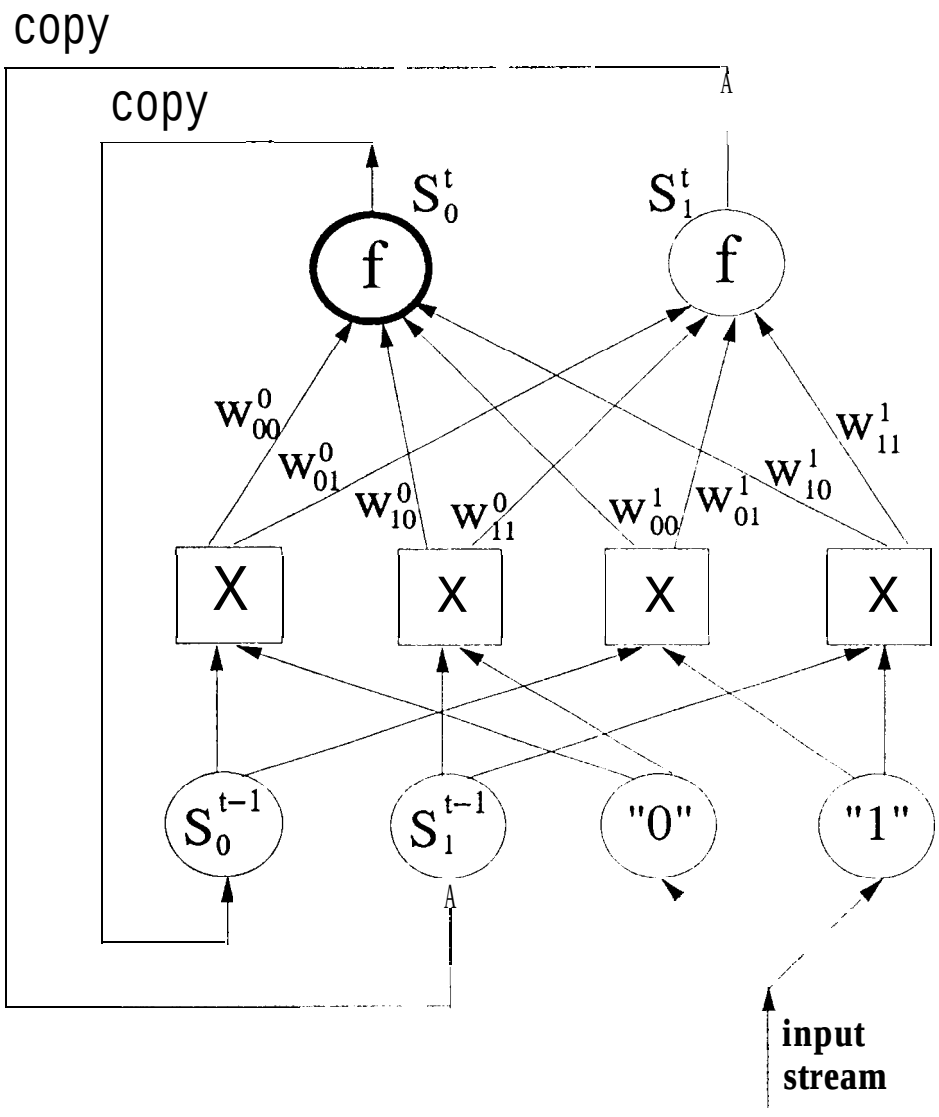


Figure 1: An analog second-order network. Each square unit takes the product of its two inputs as its output. When the current input is 1, input unit "0" has value O, and input unit "l" has value 1, and vice versa, The thick circled unit is the indicator unit, whose desired value is close to 1 when the input is legal and close to O when the input is illegal.

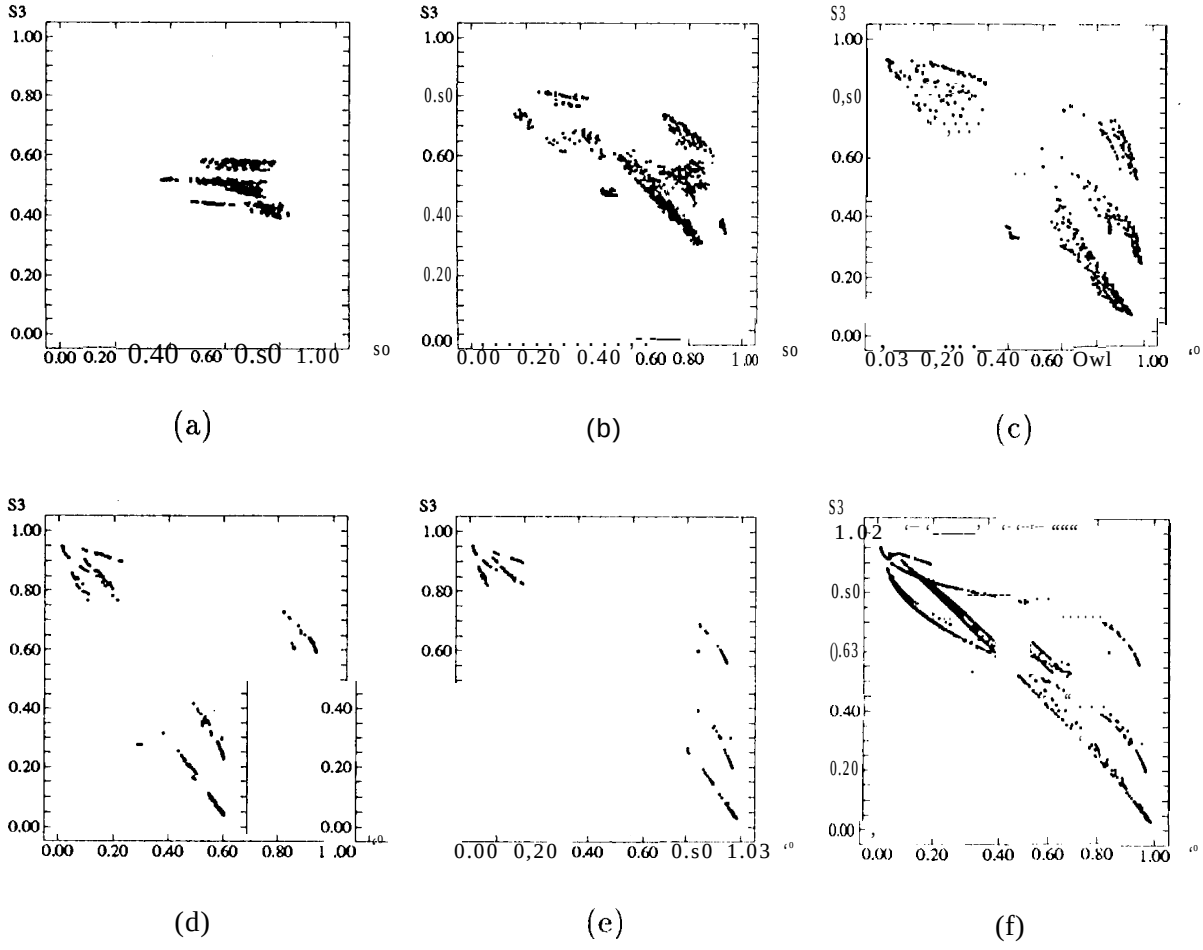


Figure 2: Hidden unit activation plot SO – S_3 in learning I'omits grammar #4 (S_o is the x axis). (a)-(e) are plots of all activations on the training data set: (a)during 1st epoch of training, (b)during 16th epoch of training, (c)during 21st epoch of training, (d)during 31st epoch of training, (e)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

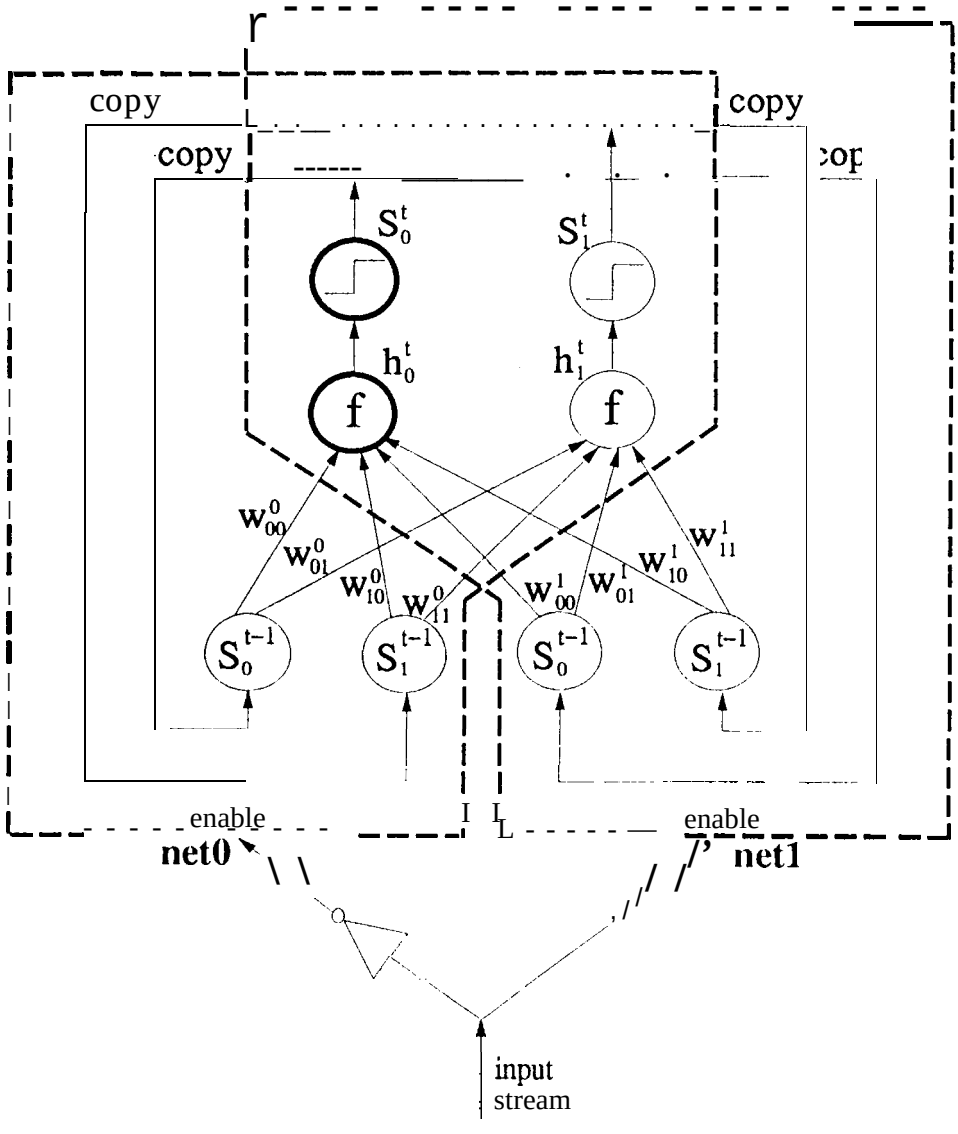


Figure 3: A discretized second-order network. The thick circled unit h_0^t is the indicator unit: h_0^t >0,5 for legal strings and S_0^t <0.5 for illegal strings.

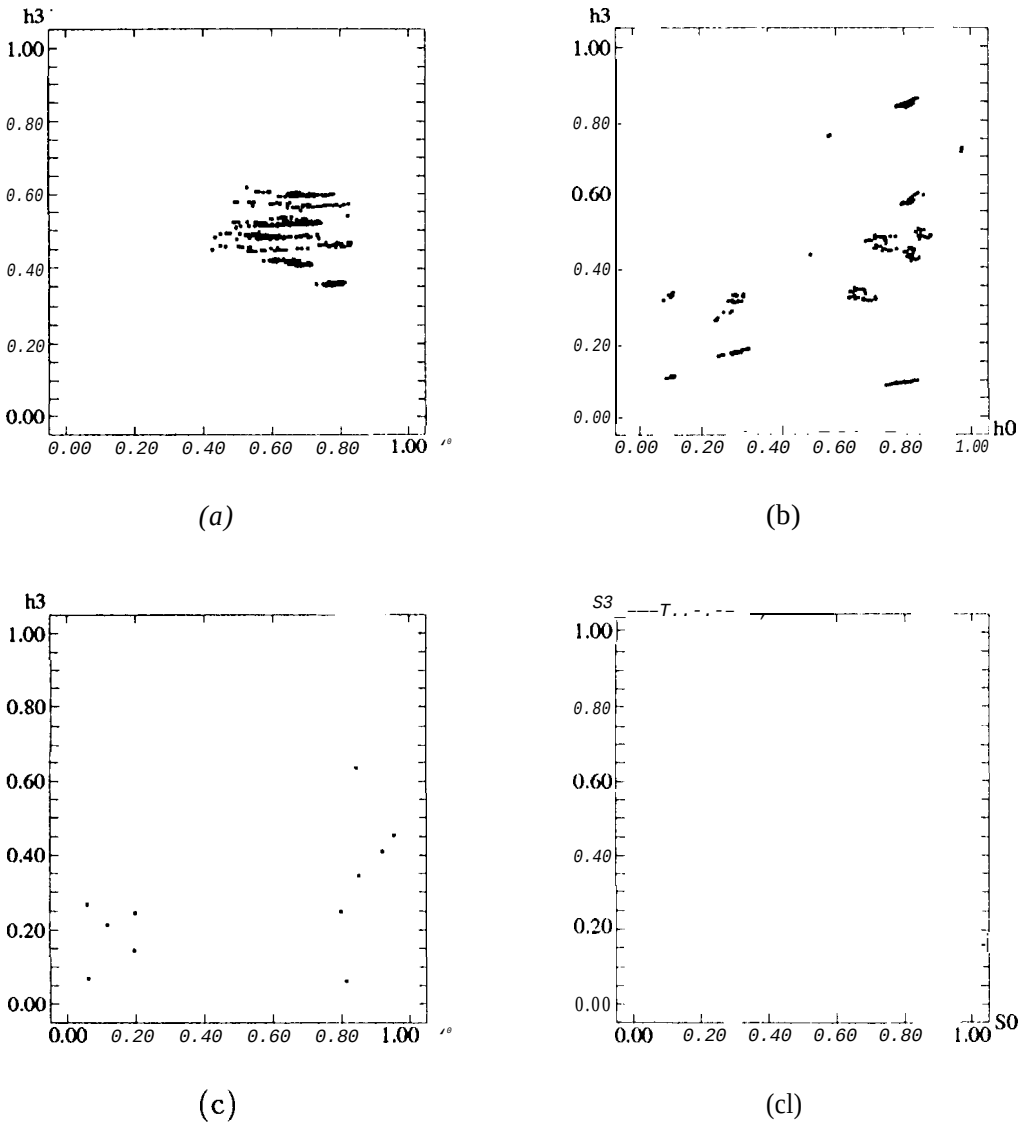
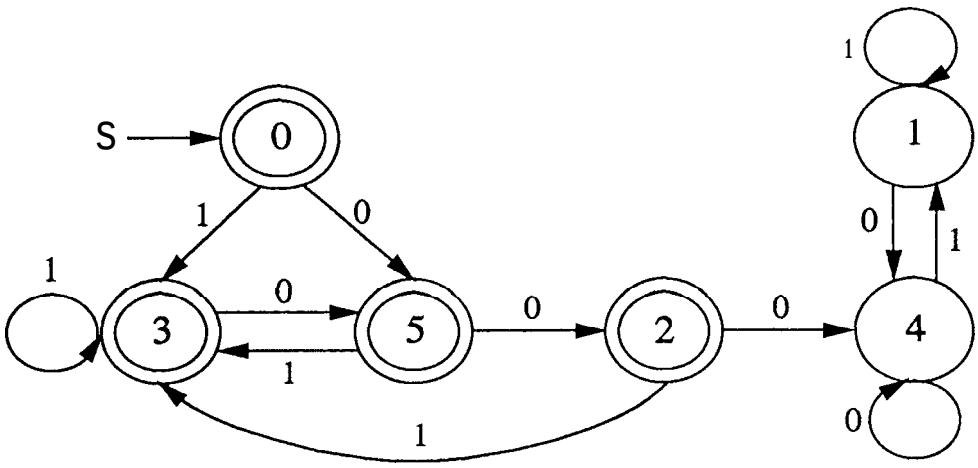
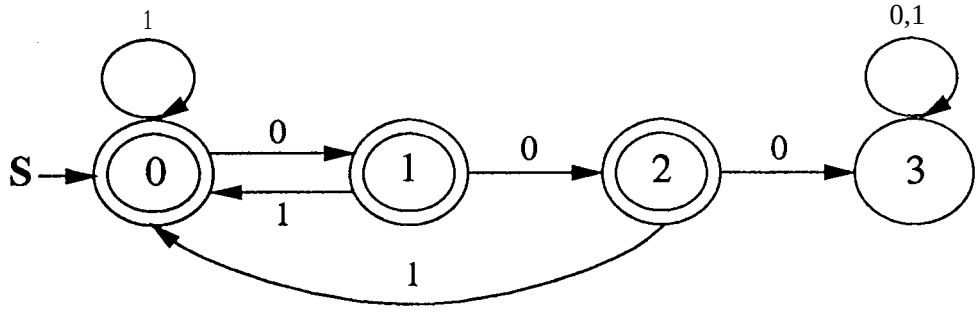


Figure 4: Discretized network learning Tomita grammar #4: (a) $h_0 - h_3$ during 1st epoch of training, (b) $h_0 - h_3$ during 15th epoch of training, (c) ho – }1.₃ after 27 epochs when training succeeds, weights are fixed, (d) So — S_3, the discretized copy of h_0 — h_3 in (c).



(a)



(b)

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

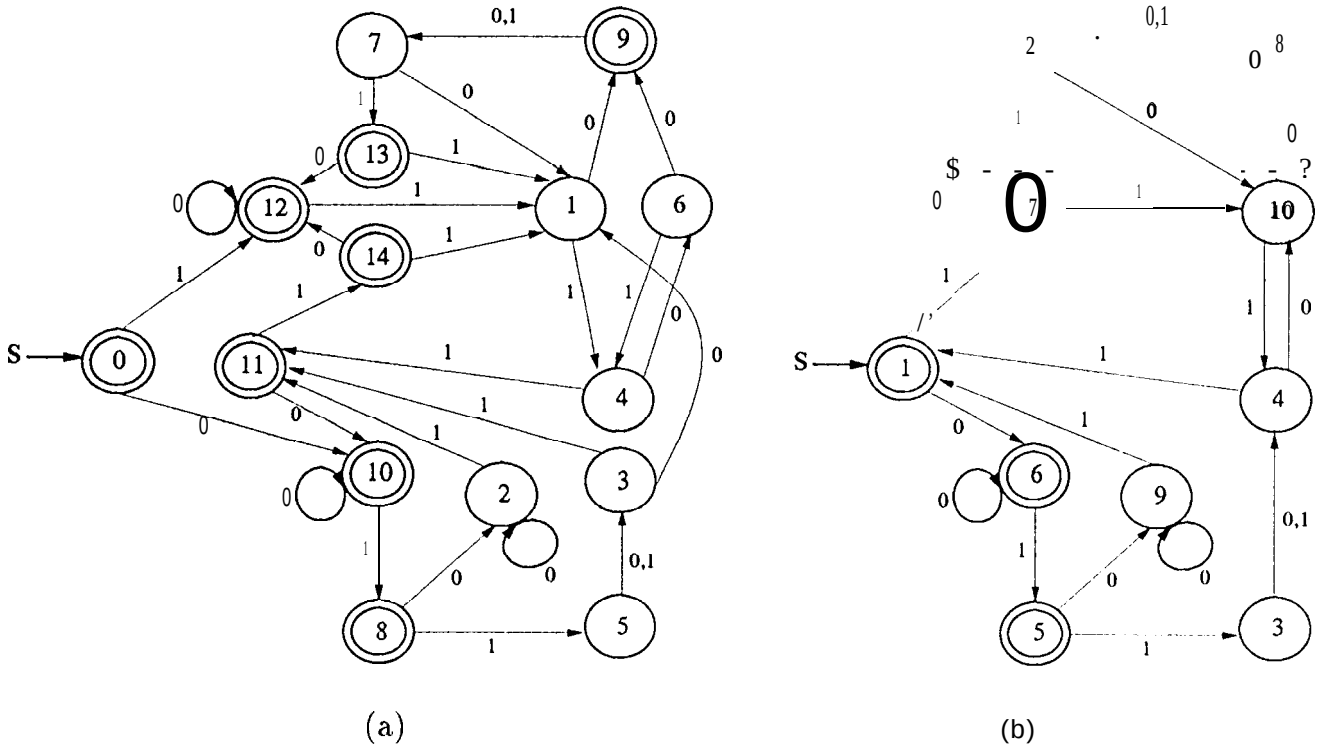


Figure 6: Extracted state machine from the discretized network after learning the 10-state machine: (a) 14-state machine extracted directly from the discrete activation space, (b) equivalent minimal 10-state machine of (a). Note that the state structure in (a) and (b) arc quite similar, for example, states 1 and 6 in (a) are equivalent to 10 in (b), and states 12, 13, and 14 in (a) play a similar role to state 7 in (b).

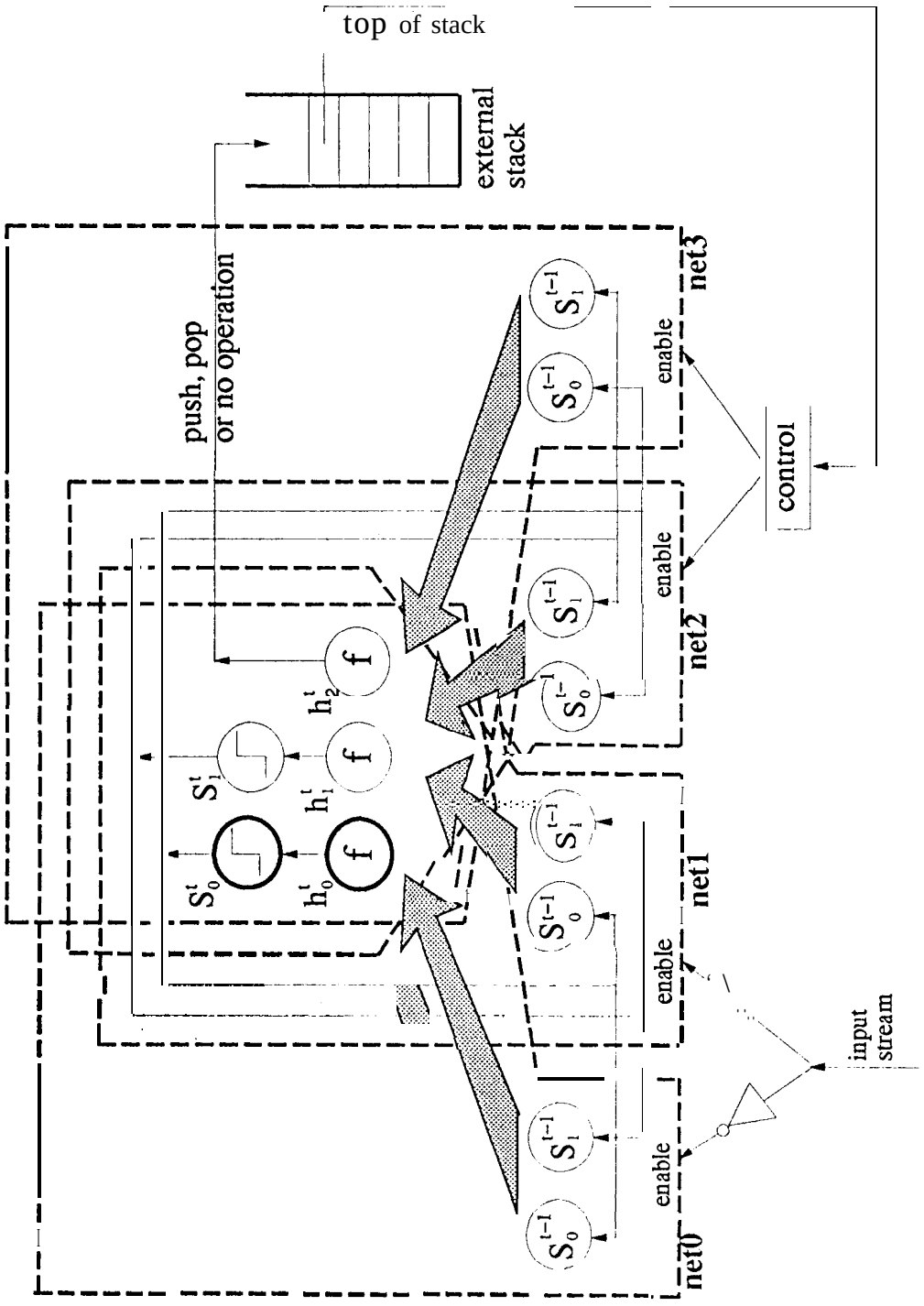


Figure 7: A discretized second-order network with an external stack. The thick circled unit h_0^t is the indicator unit: h_0^t >0.5 for legal strings and h_0^t < 0.5 for illegal strings.

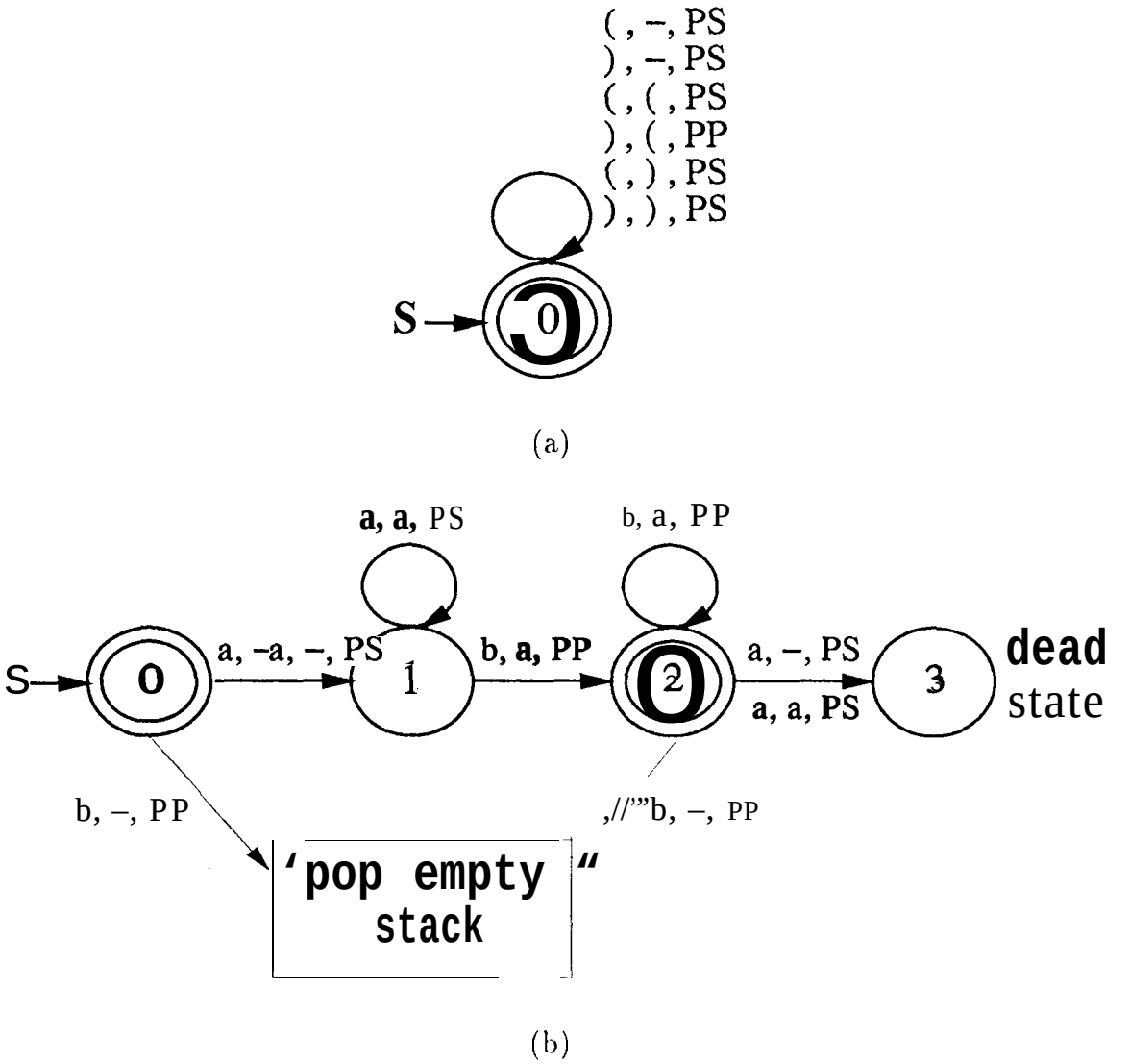


Figure 8: Extracted pushdown automata from the discretized network with an external stack after learning (a) the parenthesis grammar without hints; (b) the grammar $a^n b^n$ with hints. Double circled means the state has an indicator unit on, S_0 = 0.8: thus a processed string is legal if the automaton arrives at such a state *and* if the stack is empty. A dead state means the state has its dead unit on, S_1 = 0.8: a processed string is illegal as soon as the automaton arrives at such a state. A transition rule is labeled by '(x, y,z", where x stands for the current input symbol, y stands for the top-of-stack symbol ("-" means an empty stack), and z stands for the operation taken on the stack: "PS" means push, "PP" means pop.