Forcing Simple Recurrent Neural Networks to Encode Context[1]

Arun Maskara
New Jersey Institute of Technology
Department of Computer and Information Sciences
University Heights, Newark, NJ 07102
arun@hertz.njit.edu

Andrew Noetzel
The William Paterson College
Department of Computer Science
Wayne, NJ 07470

Abstract

The Simple Recurrent Network (SRN) is a neural network model that has been designed for the recognition of symbol sequences. It is a back-propagation network with a single hidden layer of units. The symbols of a sequence are presented one at a time at the input layer. But the activation pattern in the hidden units during the previous input symbol is also presented as an auxiliary input. In previous research, it has been shown that the SRN can be trained to behave as a finite state automaton (FSA) which accepts the valid strings corresponding to a particular grammar and rejects the invalid strings. It does this by predicting each successive symbol in the input string.

However, the SRN architecture sometime fails to encode the context necessary to predict the next input symbol. This happens when two different states in the FSA generating the strings have the same output, and the SRN develops similar hidden layer encodings for these states. The failure happens more often when number of units in the hidden layer is limited. We have developed a new architecture, called the Forced Simple Recurrent Network (FSRN), that solves this problem. This architecture contains additional output units, which are trained to show the current input and the previous context. Simulation results show that for certain classes of FSA with u states, the SRN with $\lceil \log_2 u \rceil$ units in the hidden layers fails, where as the FSRN with the same number of hidden layer units succeeds.

[1]To appear *In the Proceedings of the 1992 Long Island Conference on Artificial Intelligence and Computer Graphics*

1 Introduction

Time-varying patterns are inherent to the processing of information. Applications such as speech recognition, speech production, image recognition, robotics, and natural language processing require the processing of time-varying patterns. Patterns having a temporal dimension can often be represented as sequences of known patterns, or symbols. The processing of temporal patterns can then be characterized as either sequence recognition or sequence generation.

Neural networks are effective in associating input patterns with output patterns [1]. A recurrent neural network can take each successive symbol of a sequence as input, and produce an output that depends on the input symbol and its context in the sequence.

Since in a sequence generation application the output symbol at each time interval is known, (recurrent) neural networks easily can be trained to generate symbol sequences [2, 3, 4]. And by training a simple recurrent neural network to predict the next symbol in any sequence, the temporal pattern recognition problem is converted to a problem of temporal pattern generation [5].

The simple recurrent network (SRN) introduced by Elman [5] has demonstrated some potential for natural language processing [6, 7, 8, 9]. It has been shown to be capable of being trained to predict each successive symbol of each of the strings generated by a finite state automaton (FSA), after having been trained on a finite subset of those strings [10].

In order to produce an output corresponding to the next symbol of any string, the SRN must create an internal representation, or encoding, of the previous symbols of the string. The symbols preceding any input symbol will be called the *context* for that input.

The SRN sometimes fails to encode the context necessary to predict the next input. We describe a variant version of the SRN, called the Forced Simple Recurrent Network (FSRN), that solves this problem. A few selected simulation experiments are used to demonstrate the source of the problem and the reason for the FSRN capability for solving it.

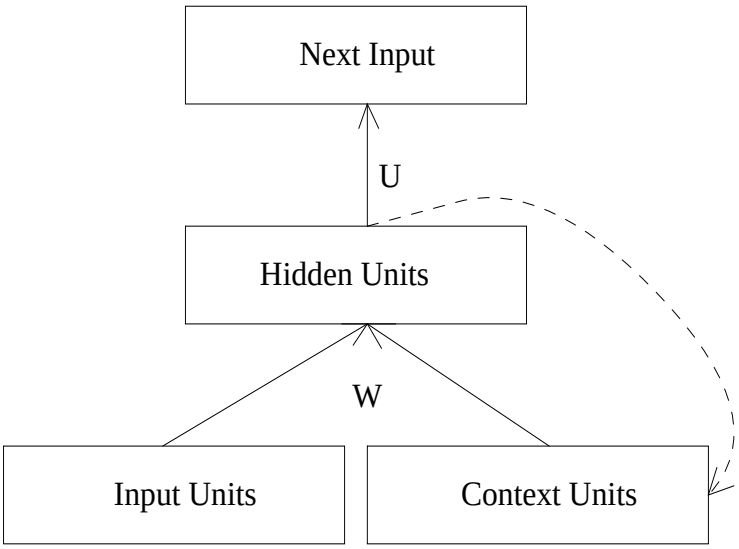


Figure 1: Simple Recurrent Network

2 The Simple Recurrent Network

A diagram of the Simple Recurrent Network (SRN) introduced by Elman [5] is shown in Figure 1. In this model, the input units are made to represent, at successive time intervals, the successive symbols of the input sequence. The input units contribute (through weight matrix W) to the state of the hidden units. Then at the end of each time interval, the state of the hidden units is copied into the context units.

Assuming unit time intervals, at time t the context units hold the state of the hidden units at $t-1$. The context units also contribute (through matrix W) to the state of the hidden units. The network output symbol at t should be prediction of the input symbol at $t+1$.

2.1 Characteristics of the SRN

After training, the SRN behaves like a Moore machine [11]. The Moore machine is a finite state automaton (FSA) denoted by the 5-tuple $(Q, \Sigma, Z, \delta, \omega)$. Q is a set of states, Σ is a finite set of input symbols, Z is a finite set of output symbols, $\delta : Q \times \Sigma \Rightarrow Q$ is the state transition function, and $\omega : Q \Rightarrow Z$

is the output function.

In the SRN the weight matrix W plays the role of the state transition function δ, and the weight matrix U acts as the output function ω. Elman has used the SRN for the special application in which the network is trained to predict the next input symbol. Therefore $Z \equiv \Sigma$. Instead of specifying one particular output, the SRN produces a measure, similar to a probability, indicating the likelihood that each symbol in Σ will be the next symbol.

The SRN also has a special state $q_0 \in Q$ know as the initial state. At the beginning of a sequence, the context units are always set to q_0. A training sequence $(x_1, x_2, \ldots, x_n)$ of size n always has $x_1 = S$, and $x_n = E$, where $S, E \in \Sigma$. The string (S, E) is the smallest string used for training the SRN.

The input and output layers of the SRN use a local representation for the symbols of the sequence: each symbol is represented by the activation of a single unit. The back-propagation algorithm [12] is used to train the network from positive examples of sequences.

After completion of training, when a symbol sequence is presented at the input, the network generates output unit values that represent approximate probabilities for the next input symbol. When a particular subsequence, or *context*, is always followed by one particular $x \in \Sigma$, the network learns to generate a high probability for x and low probabilities for all other symbols. When different instances of a particular context are followed by different symbols, the network will not show a high probability for any of the symbols following the context, but their probabilities will still be higher than those of the symbols that do not follow the context.

The network is said to *accept* a sequence if each symbol in the sequence is predicted; and a symbol is said to be predicted if its associated output unit shows a value of greater than the threshold (nominally 0.3) on the preceding symbol. The sequence is said to be *rejected* if the network is unable to predict any one of the symbols of the sequence.

2.2 Extracting Automata from the Network

To understand how the context is encoded in the hidden layer, a technique of extracting finite state automata from the neural network was developed. Other methods of extracting automata have been developed by Lee Giles [13] and Cleeremans [14]. The activation pattern in the hidden layer generated for each symbol in each context is recorded. Initially, each activation pattern is

considered to be a state. Then each state is compared with the other states. When two states are found to be similar then they are combined into one, represented by the average of the two activations. Two states are considered similar if the activations of each pair of their corresponding units differ at most by 0.05. If a state is found to be similar to more than one state, it is first combined with the state that is closer in terms of Euclidean distance. The process is repeated until no more states can be combined.

2.3 Deficiencies of the SRN

The process of encoding context in the hidden layer can be conceptualized in the following way. Early in training, each symbol forms a distinct code in the hidden layer. Since the output of the hidden layer at $t-1$ is used as the context input at t, the encoded context gradually begins to show a representation of the previous symbol. Then the hidden layer can begin to encode combinations of the previous and current symbols. With this combination as the context, the hidden layer begins to encode the relevant aspects of three consecutive symbols. Eventually, it encodes the relevant aspects of the entire context.

Sometimes the network is unable to encode the context required to predict the next input. During training, at time t the network encodes in the hidden layer the information necessary to predict the next input. The context at t is the output of hidden layer at $t-1$. If a particular aspect of any context is not encoded at t, it will not be propagated to times greater than t. The network fails to predict properly if at time t a particular aspect of the context is not encoded (since it is not relevant to predict the input at $t+1$), but that aspect of context is required for prediction at a later time. After examination of the formation of hidden layer code for many cases in which the SRN fails, we make the following observation.

Observation: *Suppose that the FSA generating a language has two different states q_i and q_j that have the same output. Then the different contexts required to reach each of q_i and q_j might generate similar hidden layer encodings in the SRN. If these different contexts have similar hidden layer encodings, and the SRN has learned to predict correctly each successive input throughout the contexts of q_i and q_j, then the SRN will never be able to generate different hidden layer encodings for those different contexts.*

Analysis: When the SRN has learned to predict correctly the next input, it implies that the network has reached or is approaching a local minimum [1]. If the different contexts required to reach states q_i and q_j generate similar hidden layer encodings, then if the encoded context correctly predicts the next input, the network in this context has reached a local minimum. If the SRN is to correctly generate the language of the FSA, then the contexts required to reach states q_i and q_j should have different hidden layer encodings. In a specific context, the desired output of the network and the current input remain constant. The only parameter that can change in the future training cycles is the previous context. Since there will be little change in the previous context (as each of the previous states is in a local minimum), and small gradual change will be canceled out by these contexts (as they are also in local minimum). Therefore in the future training cycles these contexts will continue to have similar hidden layer encoding.

A possible exception to this rule might occur if a momentum term is used in back-propagation algorithm. The use of momentum term could amplify small differences in context and take the network out of local minima. However, the use of momentum is not generally effective in taking the network out of a local minima [1]. In most of the cases, the two contexts will continue to have similar hidden layer encodings.

We have simulated the training of the SRN on strings of FSA in which two different states have the same output symbol. In the SRN, this leads to the requirement that two different contexts must be encoded, and yet must make the same prediction. Let u be the number of states in the FSA generating the strings. The SRN with $\lceil \log_2 u \rceil$ units in the hidden layer was never able to encode correctly the desired FSA. In each of these training examples we noticed that once the premise of the above observation became true, it always remained true throughout the subsequent training cycles.

When the number of units in the hidden layer was increased to u, the SRN hidden layer encodings converged in such a way that the premise of above observation never became true.

3 The Forced Simple Recurrent Network

The Forced Simple Recurrent Network (FSRN) is a variation of the SRN that has additional hidden and output units. The hidden units are trained

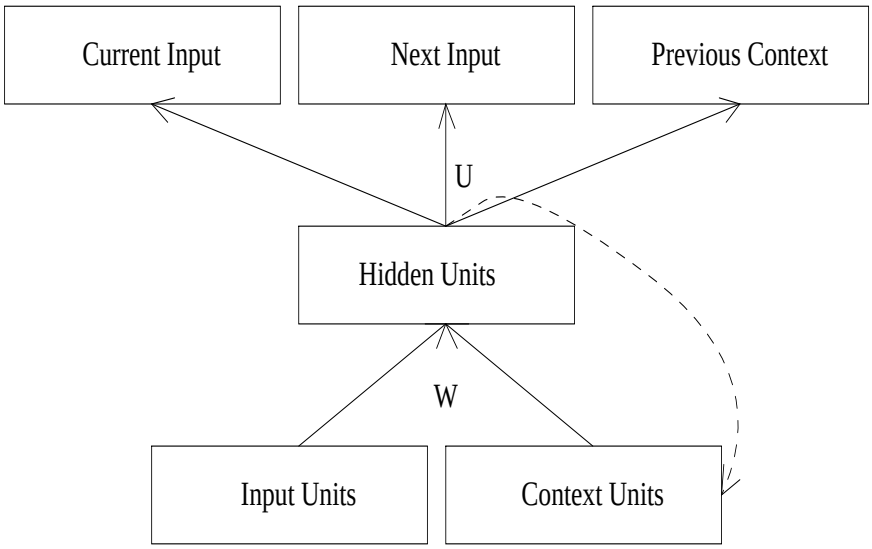


Figure 2: Forced Simple Recurrent Network

to encode a more specific representation of the context, by requiring the output units to show the previous context and current input.

A diagram of the FSRN is shown in Figure 2. At the beginning of interval t, the previous activation pattern in the hidden units (the encoded context) is copied into the context unit. The network is trained through back-propagation to show the current input and the context that is active in the context unit, as well as to predict the next input.

If the FSRN in training ever reaches the condition specified in the premise of the above observation, it will continue to evolve, and will gradually develop different encodings for the two states. The formation of the FSRN internal code begins the same as in the SRN. Initially, each symbol becomes associated with a single internal representation. In the SRN, it is possible that two symbols that result in a common prediction might form the same internal representation. The FSRN architecture guarantees that each symbol will have a distinct representation, since the hidden layer must learn to represent the current symbol. In the SRN, only the context necessary to predict the next input is gradually encoded. But since the FSRN forces the hidden layer to show the previous context, the entire sequence is always gradually encoded.

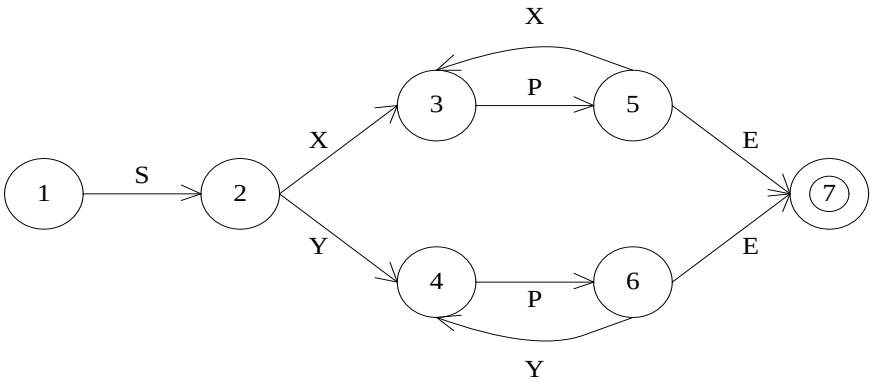


Figure 3: The FSA for Experiment 1.

4 Simulation Results

We have performed a number of experiments that demonstrate the problems of the SRN and the advantage of forcing the network to encode the current input and previous context. Here we discuss the results of two such experiments. In both experiments, the common weights of both SRN and FSRN were initialized with the same random values for each trial. During training, the input strings were presented in the same random order in both cases.

4.1 Experiment 1

The first experiment consisted of training the both SRN and FSRN to recognize the strings generated by the FSA shown in Figure 3. In this FSA, state 7 is the final state. The special symbols S and E are used to indicate the beginning and the end of each string. Training was limited to the following six positive examples of the strings of the language of the FSA.

$$SXPE,\ SXPXPE,\ SXPXPXPE$$

$$SYPE,\ SYPYPE,\ SYPYPYPE$$

The FSA generating these strings had 7 states. The contexts to be encoded in the hidden layers can be identified with the symbol sequences required to reach each of these states. Therefore if $n = \lceil \log_2 7 \rceil = 3$ units are

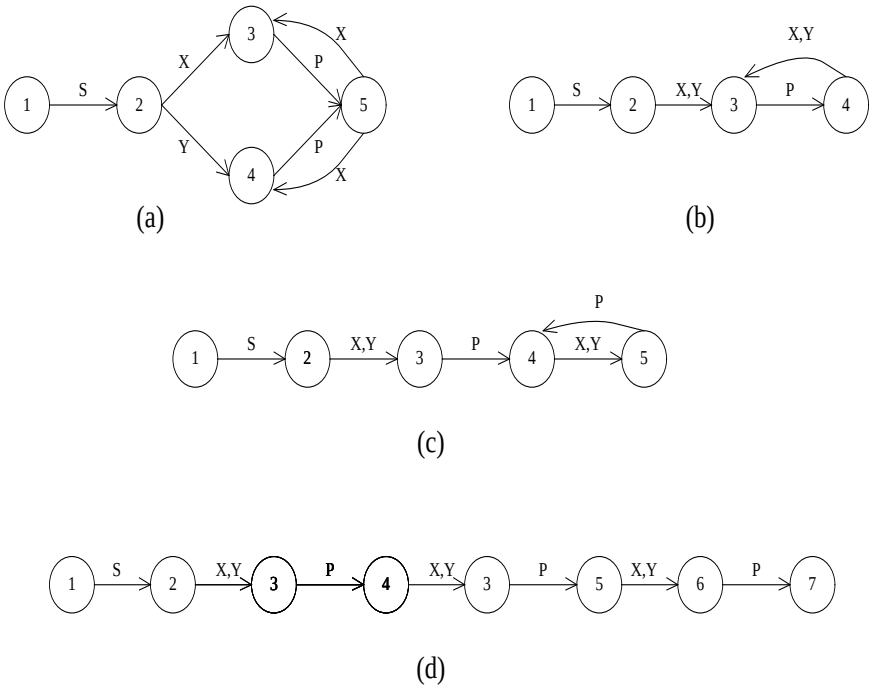


Figure 4: The evolution of states is SRN for Experiment 1 (a) before training (b) after 300 trials (c) after 700 trials (d) after 6300 trials

in the hidden layer, it should be possible for the SRN or FSRN to encode each state of the FSA as a binary number. This is the smallest number of hidden units that can encode the states of the FSA.

The SRN and the FSRN each had three units in the hidden layer. These were initialized with the same random weights. After training in each case, the FSRN was successful in learning the desired finite state recognizer, while the SRN was not. Examination of the hidden layer code of the SRN revealed that the symbols X and Y had developed the same internal representation.

In one of the examples the automaton encoded in the hidden unit was extracted from the SRN. The evolution of states in the extracted automaton is shown in Figure 4. The extracted automaton before training is shown in Figure 4a. After 300 training trials the contexts SX and SY had generated similar hidden layer code (Figure 4b). For the duration of the training the encoding of these contexts remained similar. This was consequence of the

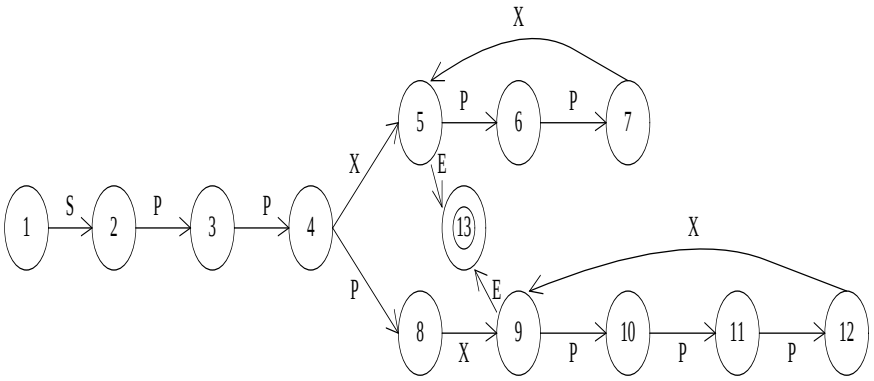


Figure 5: The FSA for Experiment 2.

common prediction P that results from either X or Y.

Figure 4c shows that after 700 trials, S followed by X or Y and P followed by X or Y developed different codes and hence were assigned different states. This happened because the SRN is gradually encoding the entire path to each state. This encoding continued until after 6300 trials it formed the linear automaton shown in Figure 4c. This experiment shows the need for forcing the network to retain a representation of the current input, so that different symbols will always have distinct internal representations.

4.2 Experiment 2

In the second experiment, strings that required longer contexts for correct prediction were used. Both the SRN and the FSRN were trained to recognize strings generated generated by the 13-state FSA shown in Figure 5. Only the following six examples were used in training:

$$SPPXE,\ SPPXPPXE,\ SPPXPPXPPXE$$

$$SPPPXE,\ SPPPXPPPXE,\ SPPPXPPPXPPPXE$$

In these examples, if P appears after X, the number of P's appearing after X is the same as the number of P's appearing before the first occurrence of X.

Both the SRN and the FSRN had four units in the hidden layer. Since $\lceil \log_2 13 \rceil = 4$, this is the smallest possible hidden layer that can succeed in this case. At the end of training, the FSRN was always able to predict number of P's after X, while the SRN was unable to do this.

In the SRN, it was found that each occurrence of X, whether preceded by two P's or three P's, had the same internal code. For one of the training examples the automaton was extracted from the SRN. The evolution of states in the extracted automaton is shown in Figure 6. The extracted automaton before training is shown in Figure 6a. After 300 training trials the contexts $SPPX$ and $SPPPX$ were encoded as the same state, but the SRN had not yet been trained to make correct predictions (Figure 6b). After 700 trials $SPPX$ and $SPPPX$ are encoded as separate states (Figure 6c), and this remained true even after 1500 trials (Figure 6d), while the the SRNs predictions improved. After 3200 trials, the contexts $SPPX$ and $SPPPX$ were again encoded as one state (Figure 6e). Since the SRN is predicting correctly throughout these contexts, their encodings remained similar for the duration of the training. Since the number of P's preceding X is not required to predict the next symbol, this aspect of the context is not encoded. This shows necessity of forcing the hidden layer to remember the previous context.

5 Conclusions

The FRSN is a new network architecture that can be used in any application where the SRN has been used. For every test case that required encoding the entire context, the performance of the FSRN was superior to that of the SRN. All the test cases verified our observation that when different states in the FSA generating the language have the same output symbols, the SRN will often fail to distinguish those states, and will therefore fail to recognize the sequences of that FSA's language. In fact, when the number of units in the hidden layer is restricted to $\lceil \log_2 u \rceil$, where u is the total number of states in the FSA, the SRN always fails. However, the FSRN with same number of nodes in the hidden layer succeeds.

References

[1] David E. Rumelhart and James L. McClelland, editors. *Parallel Distributed Processing.* MIT press, 1986.

[2] Michael I. Jordan. Attractors dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, 1986.

[3] Barak A. Pearlmutter. Learning state space trajectory in recurrent neural networks. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 113–117. Morgan Kaufmann, 1988.

[4] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1:270–280, 1989.

[5] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[6] Jeffrey L. Elman. Distributed representation, simple recurrent network, and grammatical structure. *Machine Learning*, 7(2/3), 1991.

[7] Andreas Stolcke. Learning feature-based semantics with simple recurrent networks. Technical Report TR-90-015, International Computer Science Institute, 1947 Center Street, Berkeley, California 94704, 1990.

[8] Ronan Reilly. A connectionist technique for on-line parsing. *Network*, To appear.

[9] Robert B. Allen. Connectionist language users. *Connection Science*, 2(4):279–311, 1990.

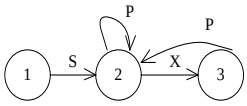
[10] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. Learning sequential structure in simple recurrent networks. In *Advances in Neural Information Processing Systems 1*, pages 643–652. Morgan Kaufmann, 1989.

[11] Taylor L. Booth. *Sequential Machines and Automata Theory.* John Wiley and Sons, 1967.

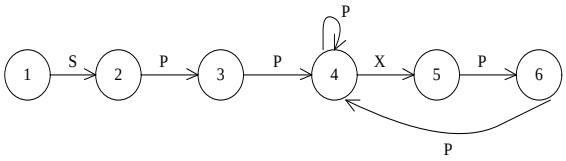
[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323, 1986.

[13] C. Lee Giles et al. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, To appear.

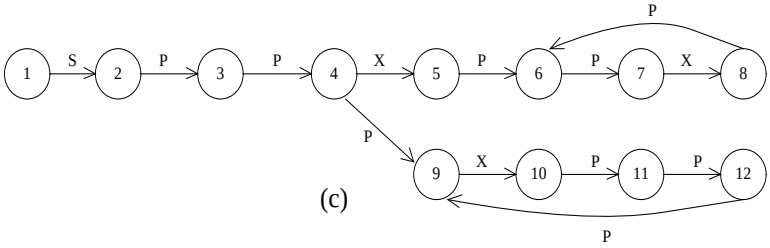
[14] Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.



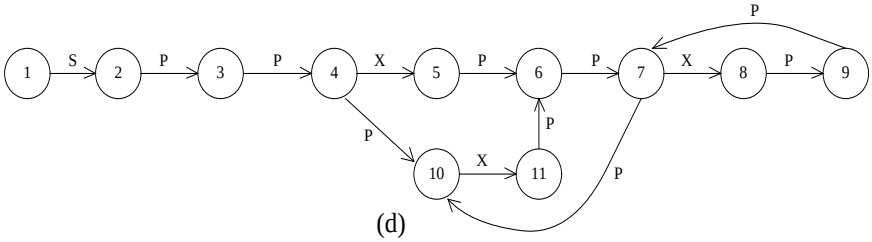
(a)



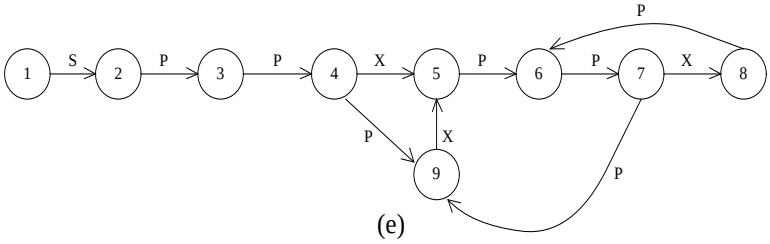
(b)



(c)



(d)



(e)

Figure 6: The evolution of states is SRN for Experiment 2 (a) before training (b) after 300 trials (c) after 700 trials (d) after 1500 trials (e) after 3200 trials