First-Order vs. Second-Order Single Layer Recurrent Neural Networks *

Mark W. Goudreau † C. Lee Giles ‡ Srimat T. Chakradhar § D. Chen ¶

Abstract

We examine the representational capabilities of first-order and second-order Single Layer Recurrent Neural Networks (SLRNNs) with hard-limiting neurons. We show that a second-order SLRNN is strictly more powerful than a first-order SLRNN. However, if the first-order SLRNN is augmented with output layers of feedforward neurons, it can implement any finite-state recognizer, but only if state-splitting is employed. When a state is split, it is divided into two equivalent states. The judicious use of state-splitting allows for efficient implementation of finite-state recognizers using augmented first-order SLRNNs.

1 Introduction

Recurrent Neural Networks (RNNs) have been used for a variety of problems, including grammatical inference [4, 5], and the implementation of finite automata [1, 7]. There has also been interest in the representational abilities of neural networks [9, 10]. Ideally, we would like to find an RNN architecture that has good representational capabilities and can be efficiently constructed as a VLSI circuit [2]. We show that certain simple RNNs have limited representational capabilities. This is a first step in the design of more sophisticated architectures.

Several RNN architectures have been proposed in the literature [3, 8, 11]. In this work we first consider the Single Layer Recurrent Neural Network (SLRNN) shown in Figure 1. The SLRNN has M inputs, that are labelled $x_1, x_2, \ldots, x_M$. The value of input x_i $(1 \leq i \leq M)$ at time t is denoted by x_i^t. The SLRNN has a *single layer* of N neurons that are labelled $y_1, y_2, \ldots, y_N$. The output value of neuron y_i $(1 \leq i \leq N)$ at time t is denoted by y_i^t. The output values of these neurons are stored in a bank of latches. These values become the "state" of the SLRNN. Each neuron computes its output value based on the current state and the input of the SLRNN. An input vector at time t is represented by $\mathbf{I}^t = [x_1^t, x_2^t, \ldots, x_M^t]^T$. A state vector is represented by $\mathbf{S}^t = [y_1^{t-1}, y_2^{t-1}, \ldots, y_N^{t-1}]^T$.

*Appeared in *IEEE Trans. on Neural Networks*, vol. 5, no. 3, p. 511, 1994.
†Princeton University and NEC Research Institute, Inc.
‡NEC Research Institute, Inc. and University of Maryland
§C&CRL, NEC USA, Inc.
¶University of Maryland

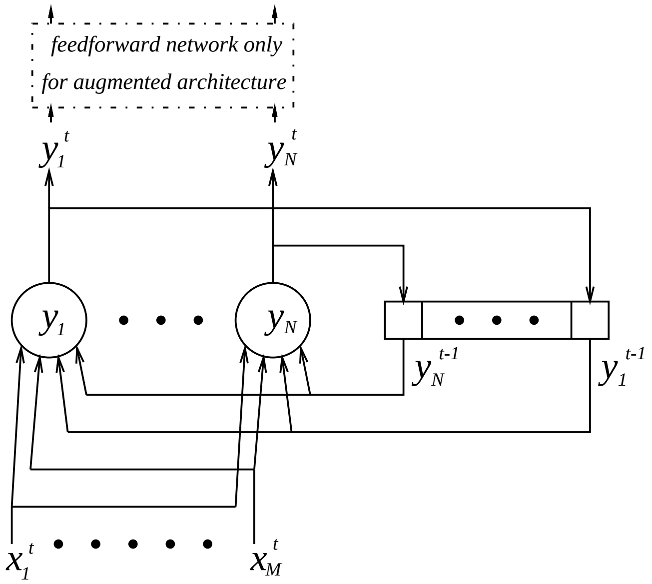


Figure 1: A Single Layer Recurrent Neural Network (SLRNN). There are M input bits, N state bits, and (up to) N output bits. The bank of N latches is shown on the right. Within the dotted lines is a feedforward network that is part of the *augmented* architecture as discussed in Section 3.

In general, one may only be concerned with output values from K of the neurons where $1 \leq K \leq N$. We represent the output vector by $\mathbf{O}^t = [y_1^t, y_2^t, \ldots, y_K^t]^T$.

SLRNN architectures are similar to *Mealy machines*, which can be used to implement arbitrary finite-state machines [6]. A Mealy machine is a quintuple $(I, O, S, \delta, \lambda)$ where I is the set of inputs, O is the set of outputs, S is the set of states, $\delta : I \times S \rightarrow S$ is the state transition function, and $\lambda : I \times S \rightarrow O$ is the output function. A Mealy machine can implement any state transition or output function. However, the structure of the SLRNN (first-order or second-order) may preclude realization of certain state transition functions or output functions.

For a first-order SLRNN the state values are, in a sense, treated as regular input values. To simplify the notation, we will define (for $1 \leq j \leq M + N$):

$$z_j^t = \begin{cases} x_j^t & \text{if } 1 \leq j \leq M \\ y_{j-M}^{t-1} & \text{if } M+1 \leq j \leq M+N \end{cases} \tag{1}$$

A first-order SLRNN contains a set of parameters known as *weights*. Weight w_{ij} causes input (or neuron) j to have an effect on neuron i. A first-order SLRNN has $N(M+N)$ weights. The dynamics of a first-order SLRNN are completely specified by the following equation that computes the output value of neuron i using the output values of the neurons and the inputs to the SLRNN:

$$y_i^t = g(\sum_{j=1}^{M+N} w_{ij} z_j^t) \tag{2}$$

We define the function g to be a hard limiter:

$$g(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{3}$$

In a second-order SLRNN, weight w_{ijk} causes neuron j and input k to have a combined effect on neuron i. A second-order network has N^2M weights. Neuron i computes its output value y_i as follows:

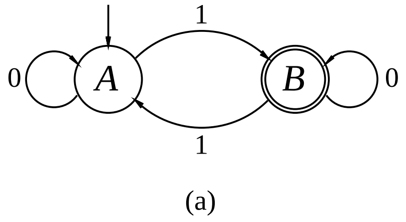
$$y_i^t = g(\sum_{j=1}^{N} \sum_{k=1}^{M} w_{ijk} y_j^{t-1} x_k^t) \tag{4}$$

Here, the function g is as defined by Equation 3.

Any first-order SLRNN that has M inputs bits and N state bits can be simulated with a second-order SLRNN that has $M+1$ input bits and $N+1$ state bits. This can be done by setting one input and one neuron of the second-order SLRNN to "1" at all times.

In this note, we show that there exist second-order SLRNNs that can not be simulated by first-order SLRNNs. Therefore, second-order SLRNNs are strictly more powerful than first-order SLRNNs. Furthermore, we show that second-order SLRNNs can simulate any finite-state recognizer and hence, they have the same representational abilities as Melay machines. If the first-order SLRNN is augmented by a feedforward network of neurons (see Figure 1), Minsky [7] shows that the augmented first-order SLRNN can simulate any finite-state recognizer. Therefore, the augmented first-order SLRNN, the second-order SLRNN, and the Mealy machine have identical representational abilities.

We also investigate the use of augmented first-order SLRNNs to implement finite-state recognizers. Minsky's approach leads to an augmented first-order SLRNN with a large number of neurons. Ideally, one would like to implement an n-state recognizer using only $\log_2 n$ neurons. We show that there exist n-state recognizers that can not be implemented by an augmented first-order SLRNN using only $\log_2 n$ neurons. However, it is possible to realize augmented first-order SLRNNs that have a significantly smaller number of neurons than Minsky's approach by using state-splitting techniques traditionally used in the synthesis of Mealy machines.



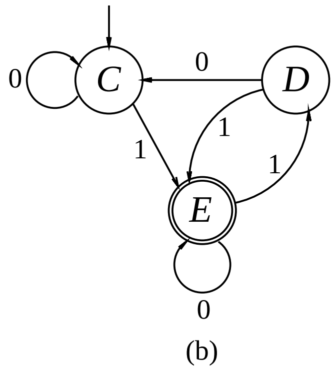


Figure 2: (a) A minimal finite-state recognizer for odd parity. A is the starting state.
(b) A non-minimal finite-state recognizer for odd parity. C is the starting state.

2 Implementation of Finite-State Recognizers

We show that a first-order SLRNN can not implement all finite-state recognizers, while a second-order SLRNN can implement any finite-state recognizer.

2.1 First-Order SLRNNs

For our examination of first-order SLRNNs, we will utilize the odd parity problem. The minimal finite-state recognizer for odd parity is shown in Figure 2(a). Since we are concerned with a recognizer, the set of outputs is $O = \{0, 1\}$. While this implies that only a single neuron's output may need to be observed, the case when $K > 1$ will also be allowed (that is, the "0" and "1" outputs may be encoded in some way). It is required, though, that a single binary vector of length K be used to represent a "0" output and a different binary vector of length K be used to represent a "1" output. These vectors will be called $\mathbf{O}_0$ and $\mathbf{O}_1$, respectively. Since we are allowing for arbitrary output representation of "0" and "1", there is no need to distinguish between odd parity and even parity recognizers. Similarly, the set of inputs to be encoded is $I = \{0, 1\}$. A binary vector of length M will represent a "0" input ($\mathbf{I}_0$), and a different binary vector of length M will represent a "1" input ($\mathbf{I}_1$).

The following theorem presents a shortcoming of first-order SLRNNs.

Theorem 1 *A first-order SLRNN can not implement all finite-state recognizers.*

Proof. It suffices to show that a first-order SLRNN can not implement the finite-state recognizer for odd parity.

Assume that a first-order SLRNN can recognize all strings that have odd parity. The machine must have at least two states that are represented by two different length N binary vectors, $\mathbf{S}_0$ and $\mathbf{S}_1$. The machine has the following properties:

$$\lambda(\mathbf{S}_0, \mathbf{I}_0) = \mathbf{O}_0, \quad \lambda(\mathbf{S}_0, \mathbf{I}_1) = \mathbf{O}_1, \quad \lambda(\mathbf{S}_1, \mathbf{I}_0) = \mathbf{O}_1, \quad \lambda(\mathbf{S}_1, \mathbf{I}_1) = \mathbf{O}_0 \tag{5}$$

Since $\mathbf{O}_0 \neq \mathbf{O}_1$, there must be at least one neuron whose output value is different for the two output vectors. We shall choose one such neuron and call it neuron z. Without loss of generality, assume that neuron z has a value of 0 for $\mathbf{O}_0$ and a value of 1 for $\mathbf{O}_1$. The case where neuron z assumes the value 1 and 0 for output vectors $\mathbf{O}_0$ and $\mathbf{O}_1$, respectively, can be analyzed in a similar way. The neuron z must now be able to implement the following functions:

$$\lambda_z(\mathbf{S}_0, \mathbf{I}_0) = 0, \quad \lambda_z(\mathbf{S}_0, \mathbf{I}_1) = 1, \quad \lambda_z(\mathbf{S}_1, \mathbf{I}_0) = 1, \quad \lambda_z(\mathbf{S}_1, \mathbf{I}_1) = 0 \tag{6}$$

Now let $\mathbf{W}_S$ be a vector of the N weights that are matched with the state values and $\mathbf{W}_I$ be a vector of the M weights that are matched with the input values. Using Equations 1, 2, 3, and the first two parts of Equation 6, we see that the weighted sum of Equation 2 for input $(\mathbf{S}_0, \mathbf{I}_0)$ must be less than or equal to zero, while the weighted sum for input $(\mathbf{S}_0, \mathbf{I}_1)$ must be greater than zero.

$$\mathbf{S}_0\mathbf{W}_S^T + \mathbf{I}_0\mathbf{W}_I^T < \mathbf{S}_0\mathbf{W}_S^T + \mathbf{I}_1\mathbf{W}_I^T$$

$$\mathbf{I}_0\mathbf{W}_I^T < \mathbf{I}_1\mathbf{W}_I^T \tag{7}$$

However, Equations 1, 2, 3, and the last two parts of Equation 6 imply:

$$\mathbf{S}_1\mathbf{W}_S^T + \mathbf{I}_0\mathbf{W}_I^T > \mathbf{S}_1\mathbf{W}_S^T + \mathbf{I}_1\mathbf{W}_I^T$$

$$\mathbf{I}_0\mathbf{W}_I^T > \mathbf{I}_1\mathbf{W}_I^T \tag{8}$$

Since there does not exist a $\mathbf{W}_I$ that satisfies both Equation 7 and Equation 8, no first-order SLRNN can recognize strings with odd (or even) parity. ∎

2.2 Second-Order SLRNNs

Theorem 2 *A second-order SLRNN can implement any finite-state recognizer.*

Proof. We present a constructive method to prove Theorem 2.

Input i will be labelled $\mathbf{I}_i$, output i will be labelled $\mathbf{O}_i$, and state i will be labelled $\mathbf{S}_i$. We will use the obvious one-hot encoding for each input symbol, each output symbol, and each state symbol. There will be one neuron for each state symbol and one neuron for each output symbol. Only the values of the output neurons are examined by the user. Therefore, $M = |I|$, $K = |O|$, and $N = |O| + |S|$. Now the weights can be defined:

$$\text{for } 1 \leq i \leq K, \qquad w_{ijk} = \begin{cases} 1 & \text{if } \lambda(\mathbf{S}_{j-K}, \mathbf{I}_k) = \mathbf{O}_i \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\text{for } K+1 \leq i \leq N, \qquad w_{ijk} = \begin{cases} 1 & \text{if } \delta(\mathbf{S}_{j-K}, \mathbf{I}_k) = \mathbf{S}_{i-K} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

∎

This implementation does not necessarily give the minimal second-order SLRNN for the finite-state recognizer.

3 Augmented First-Order SLRNNs and State-Splitting

It is reasonable to ask whether one can augment the first-order SLRNN architecture to solve the parity problem. Suppose we augment the first-order SLRNN by adding one or more *output layers* of feedforward neurons. In the *augmented* architecture, the neurons $y_1, y_2, \ldots, y_N$, shown in Figure 1, also feed into the additional layers of feedforward neurons, so any mapping of the SLRNN state to the output is possible. Therefore, we will ignore the output mapping and concern ourselves solely with the state transition problem. Minsky has already shown that an augmented first-order SLRNN can implement any finite-state recognizer [7]. However, his approach is likely to lead to implementations that have many more neurons than are necessary. If there are n states and m inputs to a finite-state recognizer, Minsky's approach requires nm neurons in the bottom (feedback) layer.

Note that all of the feedback in the augmented architecture takes place in a single layer of neurons. If this restriction is removed so that feedback may be provided after *several* feedforward layers, then clearly any finite-state recognizer can be implemented.

We show that while the augmented architecture can be used to recognize strings with parity, at least *three* different state vectors are needed to handle this problem for which the minimal machine has only *two* states, as shown in Figure 2(a). This demonstrates that while minimal representations may not be possible, non-minimal representations may exist.

Theorem 3 *An augmented first-order SLRNN can not recognize all strings that have parity if the network is only allowed to utilize two state vectors (i.e., if the network must implement the minimal finite-state recognizer).*

Proof. The proof of Theorem 3 is essentially identical to that of Theorem 1, but in this case there are two different state vectors as opposed to two different output vectors. ∎

But we can show by example that the augmented architecture can solve the parity problem if three different state vectors are allowed. The example actually implements the non-minimal finite-state recognizer shown in Figure 2(b). States C and D of Figure 2(b) are equivalent to state A in Figure 2(a), while state E in Figure 2(b) is equivalent to state B in Figure 2(a). This phenomenon is called state-splitting since state A "splits" into states C and D.

Let $M = 2$ and $N = 2$. Now let $\mathbf{S}_0 = [0,1]^T$ (for state C), $\mathbf{S}_1 = [1,0]^T$ (for state D), and $\mathbf{S}_2 = [1,1]^T$ (for state E). $\mathbf{S}_0$ is the initial state vector. The input vectors are $\mathbf{I}_0 = [1,0]^T$ and $\mathbf{I}_1 = [1,1]^T$. Under these conditions, it can easily be verified that the augmented first-order SLRNN with the following weights recognizes odd parity strings:

$$\begin{array}{llll} w_{11} = 1, & w_{12} = 1, & w_{13} = -\frac{3}{2}, & w_{14} = 1, \\ w_{21} = -1, & w_{22} = -1, & w_{23} = \frac{5}{2}, & w_{24} = -1 \end{array} \tag{11}$$

This implementation of odd parity requires only two state neurons, while Minsky's implementation would require four [7].

4 Conclusion

We have shown that a second-order SLRNN can implement any finite-state recognizer, while a first-order SLRNN can not. This is an example of the improved representational ability achieved by utilizing a second-order network. On the other hand, a second-order SLRNN has N^2M weights while a first-order SLRNN has only $N(M+N)$.

We have also shown that an augmented first-order SLRNN architecture can handle the parity problem. In this case we allowed feedforward output layers of neurons. Intriguingly, the augmented first-order SLRNN was proven to be unable to implement the minimal parity recognizer, but it could implement a non-minimal parity recognizer. It thus becomes obvious that state-splitting is an important method that can be used to expand the representational abilities of augmented first-order SLRNNs.

References

[1] N. Alon, A. Dewdney, and T. Ott, "Efficient simulation of finite automata by neural nets,"

Journal of the Association for Computing Machinery, vol. 38, no. 2, pp. 495–514, April 1991.

[2] L. Atlas and Y. Suzuki, "Digital systems for artificial neural networks," *IEEE Circuits and Devices Magazine*, vol. 5, pp. 20–24, November 1989.

[3] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[4] S. Fahlman, "The recurrent cascade-correlation architecture," in *Advances in Neural Information Processing Systems 3* (R. Lippmann, J. Moody, and D. Touretzky, eds.), (San Mateo, CA), pp. 190–196, Morgan Kaufmann Publishers, 1991.

[5] C. Giles, C. Miller, D. Chen, H. Chen, G. Sun, and Y. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, pp. 393–405, 1992.

[6] Z. Kohavi, *Switching and Finite Automata Theory.* New York, NY: McGraw-Hill, Inc., second ed., 1978.

[7] M. Minsky, *Computation: Finite and Infinite Machines*, ch. 3, pp. 32–66. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1967.

[8] J. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, p. 227, 1991.

[9] D. Seidl and R. Lorenz, "A structure by which a recurrent neural network can approximate a nonlinear dynamic system," in *Proceedings of the International Joint Conference on Neural Networks 1991*, vol. II, pp. 709–714, July 1991.

[10] H. Siegelmann and E. Sontag, "On the computational power of neural nets," in *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, (Pittsburgh, PA), July 1992.

[11] R. Watrous and G. Kuhn, "Induction of finite state languages using second-order recurrent networks," in *Advances in Neural Information Processing Systems 4* (J. Moody, S. Hanson, and R. Lippmann, eds.), (San Mateo, CA), pp. 309–316, Morgan Kaufmann Publishers, 1992.