数理解析研究所講究録
1093 巻 1999 年 188-193

単純回帰ネットワークの計算能力について

守谷 純之介 (Junnosuke Moriya)　西野 哲朗 (Tetsuro Nishino)

電気通信大学大学院 電気通信学研究科

〒 182-8585 東京都調布市調布ヶ丘 1-5-1
e-mail: jmoriya@sw.cas.uec.ac.jp, nishino@sw.cas.uec.ac.jp

1 Introduction

In the filed of cognitive psychology, *simple recurrent networks* are used for recognizing sequences of symbols and modeling the language processing in the human brain. A simple recurrent network is a circuit which consists of a finite number of gates, each of which computes a linear function whose range is a closed interval $[0.0, 1.0]$ [4, 5]. McClelland et al. showed on an experimental basis that simple recurrent networks can simulate finite automata [8]. Furthermore, Elman showed on an experimental basis that simple recurrent networks can predict the rightmost word in sentential forms of a particular context-free grammar with high probability, after a learning process based on the sample words which are generated by the grammar [4, 5]. Concerning these results, it is natural ask whether the computational capability of simple recurrent networks is sufficient to recognize natural languages or not.

It is known that *processor nets* used by Siegelmann and Sontag in [10] can simulate Turing Machines [10]. Processor nets are recurrent networks which consist of a finite number of processors (or gates), each of which computes a *saturated-linear* function. Siegelmann and Sontag showed the following facts in [10]: if the weights of the connections in a processor net are rational numbers, it can simulate an arbitrary Turing Machine. The proof of this proposition is constructive. Moreover, if the weights of the connections are real numbers, processor nets can recognize arbitrary languages and if the weights of connections are integers, processor nets can only recognize regular languages. Note that when the weights are rational numbers (or real numbers), the number of the sates of processor nets is infinite.

It is trivial that if each gate of simple recurrent networks computes a saturated linear function, simple recurrent networks can simulate processor nets in a straightforward fashion. This means that simple recurrent networks can recognize recursive languages. In these works, however, the range of a function computed at each gate is infinite. In this paper, we assume that the range of a function computed at each gate of a simple recurrent networks is a *finite set*. This is a quite realistic assumption especially when we perform a computer simulation of a simple recurrnet network.

On the other hand, computational complexity classes on processor nets has been studied. In [10], Siegelmann and Sontag showed that a class of languages decided by processor nets in polynomial time equals a class of languages decided by Turing Machines in polynomial time, and 1058 processors are sufficient in this simulation. Indyk improved this result and showed that 25 processors are sufficient [7]. Moreover, Balcázar et al. develop further relationships between languages which are decided by processor nets and other complexity classes [3]. The details of the relationships between time complexity classes on processor nets and circuit complexity on nonuniform circuit families are shown in [11].

A mathematical definition of simple recurrent networks will be given in section 2.1, and we adopt a cirtain assumption on gates of simple recurrent networks. Then, we define an equivalence relation between simple recurrent networks and Mealy machines which are a finite automata with output. Our first result is a construction of a Mealy machine which simulates a simple recurrent network. This result shows that, under our assumption, simple recurrent networks can only learn a regular language and the computational capabilty of simple recurrent networks is not sufficient to recognize natural languages.

Next, we define an equivalence relation between

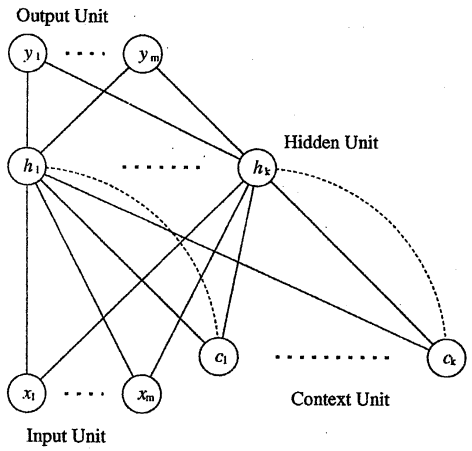

Figure 1: A simple recurrent network

simple recurrent networks and Moore machines which are also finite automata with output. Our second result is a construction of a simple recurrent network which simulates a Moore machine under our assumption. Therefore, these two constructions show that the computational capability of simple recurrent networks is equal to that of finite automata with output under our assumption.

In section 5, we discuss the relationships between the number of states of a finite automaton with output and the size of a simple recurrent networks.

2 Preliminaries

2.1 Simple Recurrent Networks

A *simple recurrent network* (SRN) consists of an *input unit*, an *output unit*, a *hidden unit* and a *context unit* (see Figure 1). The hidden unit, the context unit and the output unit are sets of gates each of which computes a function $f[w_1,\ldots,w_m](x_1,\ldots,x_m) : \mathbf{R}^m \to \mathbf{R}, w_1,\ldots,w_m \in \mathbf{R}, m \in \mathbf{N}$ defined by

$$f[w_1,\ldots,w_m](x_1,\ldots,x_m) = \begin{cases} 0 & \text{if } \sum_{i=1}^m w_i x_i < 0 \\ \sum_{i=1}^m w_i x_i & \text{if } 0 \le \sum_{i=1}^m w_i x_i \le 1 \\ 1 & \text{if } \sum_{i=1}^m w_i x_i > 1. \end{cases}$$

For convenience, we shall use $F[\sum_{i=1}^m w_i x_i]$ instead of $f[w_1,\ldots,w_m](x_1,\ldots,x_m)$ from here on.

The context unit of a SRN holds a copy of the outputs of the hidden unit at a previous time step. The outputs of the hidden unit depend on the outputs of the input unit and the context unit, while the outputs of the output unit only depends on the outputs from the hidden unit. More formally, we define a SRN as follows.

Definition 2.1 *A simple recurrent network is a 7-tuple $\mathcal{E} = (G, I, O, H, C, A, w)$, where*

1. *$G = (V, E)$ is a directed graph, where V is a finite set of nodes which is divided into the following four sets of gates: an input unit $I = \{x_1,\ldots,x_n\} \subseteq V$, an output unit $O = \{y_1,\ldots,y_m\} \subseteq V$, a hidden unit $H = \{h_1,\ldots,h_k\} \subseteq V$ and a context unit $C = \{c_1,\ldots,c_k\} \subseteq V$. Then, $E = \{(v_1,v_2) \mid (v_1,v_2) \in I\times H \cup C\times H \cup H\times C \cup H\times O\}$ is a set of edges of G. Edges from the hidden unit to the context unit must be of the form $(h_i, c_i), h_i \in H, c_i \in C, 1 \le i \le k$.*

2. *$A = \{A_1,\ldots,A_k\} \in \mathbf{R}^k$ is a set of k real numbers, called initial output of the context unit.*

3. *$w : E \to \mathbf{R}$ is a weight assignment to the edges in E. We denote the weight of an edge (v_1,v_2) by $w(v_1,v_2)$. We assume that the weight of the edge $(h_i, c_i), h_i \in H, c_i \in C, 1 \le i \le k$ is 1.*

In this paper, we adopt the following realistic assumption on gates of simple recurrent networks:

Assumption 2.1 *A function computed at each gate of a simple recurrent network is a function whose range is a finite set.*

This means that we can only physically implement a logic gate whose output is a value of finite precision. From Assumption 1, for each gate, there exists a finite set of real numbers which the gate can output. We define a finite ordered set VAL which contains these real numbers as follows:

$$\mathrm{VAL} = \{val_1,\ldots,val_i,\ldots,val_j,\ldots,val_p\},$$

where $val_k \in \mathbf{R}$, $1 \le k \le p$, $0 \le val_i < val_j \le 1$, $i < j$, $i,j,k \in \mathbf{N}$. In the sequel, we assume that a gate compute a function $F[\sum_{i=1}^m w_i x_i] : \mathbf{R}^m \to \mathrm{VAL}$, $m \in \mathbf{N}$.

Now, we define an *input sequence* and an *output sequence* of a SRN. For $v \in V, t \ge 0$, let $S_v(t)$ denote the *output* of the gate v at time t. The *input* for a SRN at time t, denoted $\mathcal{I}(t)$, is the

sequence $S_{x_1}(t)\cdots S_{x_n}(t)$, where $x_i \in I, S_{x_i} \in \{0,1\}$, $1 \leq i \leq n$. The *output* of a SRN at time $t \geq 2$ is the sequence $S_{y_1}(t)\cdots S_{y_m}(t)$, where $y_i \in O$, $1 \leq i \leq m$.

Definition 2.2 *An input sequence $\mathcal{I}$ with length n is $\mathcal{I} = \mathcal{I}(0)\mathcal{I}(2)\mathcal{I}(4)\cdots\mathcal{I}(2(n-1))$. For a input sequence $\mathcal{I}$ with length n and a SRN $\mathcal{E}$, an output sequence of $\mathcal{E}$ is $T_{\mathcal{E}}(\mathcal{I}) = \mathcal{O}(2)\mathcal{O}(4)\mathcal{O}(6)\cdots\mathcal{O}(2n)$.*

2.2 Mealy Machines

A *Mealy machine* is a deterministic finite automaton with output [1, 6].

Definition 2.3 *A Mealy machine M is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is an input alphabet, Δ is an output alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\lambda : Q \times \Sigma \to \Delta$ is an output function.*

Let $q(t) \in Q$ denote the state of M at time t. We define an input sequence and an output sequence of a Mealy Machine as follows:

Definition 2.4 *An input sequence w of a Mealy machine with length $i \in \mathbf{N}$, is a string $w = a_0a_1\cdots a_{i-1}$, $a_j \in \Sigma$, $0 \leq j \leq i-1$.*

Definition 2.5 *For an input sequence $w = a_0a_1\cdots a_{i-1}$, an output sequence of a Mealy machine M, $T_M(w)$ is a string*

$$T_M(w) = \lambda(q(0), a_0)\lambda(q(1), a_1)\cdots\lambda(q(i-1), a_{i-1})$$

such that $q(t+1) = \delta(q(t), a_t)$, $0 \leq t \leq i-1$.

We define an equivalence relation between simple recurrent machines and Mealy machines.

Definition 2.6 *A SRN $\mathcal{E}$ is said to be equivalent to a Mealy machine M if $T_{\mathcal{E}}(w) = T_M(w)$ for any input sequence w.*

2.3 Moore Machines

A *Moore machine* is also a deterministic finite automaton with output [6].

Definition 2.7 *A Moore machine M is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is an input alphabet, Δ is an output alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\lambda : Q \to \Delta$ is an output function.*

We define an *input sequence* and an *output sequence* of a Moore machine in the same fashion of those of a Mealy machine. The following lemma is known [6].

Lemma 2.1 *For a Mealy machine M', there exists a Moore machine M which is equivalent to M'.*

2.4 Vector representations

Suppose that the number of states of a Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is m, the number of elements of the input alphabet is k, and the number of elements of the output alphabet is l. Namely, let $Q = \{q_0, \ldots, q_{m-1}\}$, $\Sigma = \{a_0, \ldots, a_{k-1}\}$ and $\Delta = \{b_0, \ldots, b_{l-1}\}$.

We first define a *state vector* of a Moore machine. Each entry of a state vector corresponds to a tuple of a state and an input symbol of a Moore machine.

Definition 2.8 *A state vector of M at time t is $v(t) \in \{0,1\}^{km}$ and satisfies the following properties:*

1. *If $t = 0$ and $q(t) = q_i$, from the $ik+1$th entry to the $ik+k$th entry of $v(t)$ are all 1's and the others are 0's.*
2. *If $t \neq 0$ and $q(t) = q_i$, only one entry from the $ik+1$th entry to the $ik+k$th entry of $v(t)$ is 1 and the others are 0's.*

Next, we define an *input vector* of a Moore machine. Let $I(t)$ denote a given input symbol of a Moore machine at time t.

Definition 2.9 *An input vector of M at time t is $w(t) \in \{0,1\}^k$, and if $I(t) = a_i$, the $i+1$th entry of $w(t)$ is 1 and the others are 0's.*

We define a *state-input vector* based on the above definitions.

Definition 2.10 *Let $v(t) = (v_1(t), \ldots, v_{km}(t)) \in \{0,1\}^{km}$ a state vector and $w(t) = (w_1(t), \ldots, w_k(t)) \in \{0,1\}^k$ an input vector of M at time t. Then, a state-input vector of M at time t is $V(t) = (v_1(t), \ldots, v_{km}(t), w_1(t), \ldots, w_k(t)) \in \{0,1\}^{km+k}$.*

Next, we define an *output vector* of a Moore machine. Let $O(t)$ denote an output symbol of a Moore machine at time t.

Definition 2.11 *If an output vector of M at time t is $u(t) \in \{0,1\}^l$, and $O(t) = b_i$, then the $i+1$th entry of $u(t)$ is 1 and the others are 0's.*

Finally, we give the definition of equivalence relation between simple recurrent networks and Moore machines.

Definition 2.12 *For a input sequence w, suppose that a Moore machine M yeilds the output sequence $T_M(w)$ and a SRN $\mathcal{E}$ yeilds the output sequence $T_{\mathcal{E}}(w)$. Then, $\mathcal{E}$ is said to be equivalent to M if $T_M(w) = bT_{\mathcal{E}}(w)$ for all w where b is the output alphabet for the initial state of M.*

3 Constructing Mealy Machines

In this section, we will construct a Mealy machine which is equivalent to a given SRN. We describe how to define the states,the transition function and the output function of the SRN.

Suppose that a SRN $\mathcal{E} = (G, I, O, H, C, A, w)$ is given, where $|I| = n, |O| = m, |H| = |C| = k$. We construct a Mealy machine $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ which simulates $\mathcal{E}$ in the following way.

1. Each state $p \in Q$ of M represents the output pattern $a_{c_1} \cdots a_{c_k}$ of the context unit, where $a_{c_i} \in \mathrm{VAL}$ and $c_i \in \mathcal{C}$. Note that every output pattern of the context unit can be represented as a word in VAL^k. Thus, the states of M includes all the representation of the output pattern of the context unit of $\mathcal{E}$.

   The initial output pattern of the context unit of $\mathcal{E}$ corresponds to the initial state q_0 of M.

2. Since the number of the gates in the context unit is equal to that of the hidden unit, each state $p \in Q$ of M also represents the output pattern $a_{h_1} \cdots a_{h_k}$ of the hidden unit, where $a_{h_i} \in \mathrm{VAL}$ and $h_i \in \mathcal{H}$.

3. We construct the transition function δ of M. Suppose that the output pattern of the input unit of $\mathcal{E}$ is $a_{x_1} \cdots a_{x_n}$ and the output pattern of the context unit of $\mathcal{E}$ is $a_{c_1} \cdots a_{c_n}$. The output pattern of the hidden unit depends on $a_{c_1} \cdots a_{c_k}$ and $a_{x_1} \cdots a_{x_n}$. That is, the output of the gate a_{h_i} in the hidden unit is given by

$$a_{h_i} = F\left[\sum_{j=1}^{k} w(c_j, h_i) a_{c_j} + \sum_{j=1}^{n} w(x_j, h_i) a_{x_j}\right],$$

   for each i, $1 \le i \le k$. Then, the transition function δ of M is defined by $\delta(p, \sigma) = p'$ where $p = a_{c_1} \cdots a_{c_k} \in Q, \sigma = a_{x_1} \cdots a_{x_n} \in \Sigma$ and $p' = a_{h_1} \cdots a_{h_k} \in Q$ satisfy the above equations.

4. Finally, we construct the output function λ of M. Suppose that the output pattern of the input unit of $\mathcal{E}$ is $a_{x_1} \cdots a_{x_n}$, the output pattern of the context unit is $a_{c_1} \cdots a_{c_n}$ and the output pattern of the hidden unit is $a_{h_1} \cdots a_{h_k}$. The output of $\mathcal{E}$ depends on the output pattern of the hidden unit. That is, the output of the gates a_{y_i} in the output unit is given by

$$a_{y_i} = F\left[\sum_{j=1}^{k} w(h_j, y_i) a_{h_k}\right].$$

   Then, the output function λ of M is defined by $\lambda(p, \sigma) = \gamma$ where $p = a_{c_1} \cdots a_{c_k} \in Q, \sigma = a_{x_1} \cdots a_{x_n} \in \Sigma$, $\gamma = a_{y_1} \cdots a_{y_m} \in Q$ satisfy the above equations.

We have the following result:

Theorem 3.1 *Under Assumption 2.1, for any simple recurrent network $\mathcal{E}$, there exists a Mealy machine M which is equivalent to $\mathcal{E}$.*

4 Constructing Simple Recurrent Networks

In this section, we will show that there exits a SRN which simulates a Moore Machine. For this purpose, we use a *connection matrix* of a Moore machine [2]. Our first goal is to show that a connection matrix of a Moore machine can be implemented on a SRN. Through this section, consider a Moore machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ with $Q = \{q_0, q_1, \ldots, q_{m-1}\}$ and $\Sigma = \{a_0, a_1, \ldots, a_{k-1}\}$.

4.1 Connection matrices

We first define a *connection matrix* of a transition function of a Moore machine [2].

Definition 4.1 *A connection matrix α of transition function of M is*

$$\alpha_{ij} = \{x \in \Sigma \mid \exists q_i, q_j \in Q, \delta(q_i, x) = q_j\},$$

where α_{ij} is an entry of the ith row and the jth column of the matrix α.

4.2 Weight matrices

From the definition, a connection matrix α of M is an $m \times m$ matrix. We define a *weight matrix* of M.

Definition 4.2 *For a Moore machine M, a weight matrix α' of M is a $km \times (km+k)$ matrix which satisfies the following properties:*

- *For i, j, $1 \leq i, j \leq km$, the entry α'_{ij} is $\frac{1}{k+1}$ if and only if there exists h, $0 \leq h \leq k-1$, such that $\delta(q_{\lfloor (i-1)/k \rfloor}, a_h) = q_{\lfloor (j-1)/k \rfloor}$, and the entry α'_{ij} is 0 if and only if there does not exist h, $0 \leq h \leq k-1$, such that $\delta(q_{\lfloor (i-1)/k \rfloor}, a_h) = q_{\lfloor (j-1)/k \rfloor}$.*
- *For i, j, $1 \leq i \leq km$, $km \leq j \leq km+k$, the entry α'_{ij} is $\frac{2k+1}{2k+2}$ if $(j-km) \bmod k = i \bmod k$ and the entry α'_{ij} is 0 if $(j-km) \bmod k \neq i \bmod k$.*

Now, we construct a weight matrix α' from a connection matrix α as follows:

1. Let α^T be a transposed matrix of α.
2. For an $m \times m$ matrix α of M, let a matrix α' be a $km \times (km+k)$ matrix.
3. Entries in the 1st,...,the kmth rows and the 1st,..., the kmth columns of α' are defined as:
   (a) If the entry α_{ij} contains a symbol $a_t \in \Sigma$, k entries in the $k(j-1)+1$th, ..., the $k(j-1)+k$th columns and the $k(i-1)+t$th row are all $\frac{1}{k+1}$.
   (b) If the entry α_{ij} does not contains a symbol $a_t \in \Sigma$, k entries in the $k(j-1)+1$th, ..., the $k(j-1)+k$th columns and the $k(i-1)+t$th row are all 0.
4. Entries in the 1st, ...,the kmth rows and the $km+1$th, ..., the $km+k$th columns of α' are defined as follows: for the ith row, an entry in the ith row and the $km+(i \bmod k)$th column of α' is $\frac{2k+1}{2k+2}$ and the other entries are all 0.

4.3 Implementing matrices on simple recurrent networks

In this paper, we already have assume that the range of a function computed at each gate of a SRN is a finite set, namely VAL $= \{0, 1\}$. Therefore, a function computed at each gate is

$$F\left[\sum_{i=1}^{m} w_i x_i\right] = \begin{cases} 0 & \text{if } \sum_{i=1}^{m} w_i x_i < 0 \\ 0 & \text{if } 0 \leq \sum_{i=1}^{m} w_i x_i \leq 1 \\ 1 & \text{if } \sum_{i=1}^{m} w_i x_i > 1. \end{cases}$$

We will define a function σ from the function F.

Definition 4.3 *For an $n \times m$ matrix A, a function $\sigma_A : \{0,1\}^m \to \{0,1\}^n$ is*

$$\sigma_A(x) = \begin{pmatrix} F[\sum_{i=1}^{m} A_{1,i} x_i] \\ F[\sum_{i=1}^{m} A_{2,i} x_i] \\ \vdots \\ F[\sum_{i=1}^{m} A_{n,i} x_i] \end{pmatrix},$$

where $x = (x_1, \ldots, x_m)$ and

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,m} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \cdots & A_{n,m} \end{bmatrix}.$$

The outputs of units of SRN's are regarded as vectors. A transition function of M can be implemented by using a function $\sigma_{\alpha'}$.

Lemma 4.1 *Suppose that $\delta(q_i, a_j) = q_h$, a state vector of the state q_i is v, an input vector of the input symbol a_j is w, and a state-input vector of v and w is V. Then, a vector $\sigma_{\alpha'}(V)$ is a state vector of the state q_h.*

4.4 Constructing simple recurrent networks which simulate Moore machines

In this section, we will construct a SRN which is equivalent to a given Moore machine. We describe how to define the input unit, the output unit, the context unit, the hidden unit, the initial output and the weights.

Let $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ be a Moore machine and $m = |Q|$, $k = |\Sigma|$, and $l = |\Delta|$. We will construct a SRN $\mathcal{E} = (G, I, O, H, C, A, w)$ as follows:

1. The input unit is $I = \{x_1, \ldots, x_k\}$.
2. The output unit is $O = \{y_1, \ldots, y_l\}$.
3. The context unit is $C = \{c_1, \ldots, c_{km}\}$, and the hidden unit is $H = \{h_1, \ldots, h_{km}\}$.

4. Let $v(0)$ be a state vector of the initial state q_0 and $v(0) = (v_1(0), \ldots, v_{km}(0))$, where $v_i(0) \in \{0,1\}, 1 \leq i \leq km$. Then, the initial output A is $A = \{A_1, A_2, \ldots A_{km}\}$, where $A_i = v_i(0)$, $1 \leq i \leq km$.

5. Note that the weight matrix α' of M is a $km \times (km+k)$ matrix. We define the weights as follows:

   (a) For an edge (x_i, h_j), $x_i \in I$, $h_j \in H$, the weight $w(x_i, h_j)$ is the entry in the jth row and $km+1$th column of α'.

   (b) For a edge (c_i, h_j), $c_i \in C$, $h_j \in H$, the weight $w(c_i, h_j)$ is the entry in the jth row and ith column f α'.

6. If $\lambda(q_i) = b_j$, the weight $w(h_s, y_j)$ is an arbitrary value that is greater than 1 for h_s, $ik \leq s \leq ik+k$. The weights of the other edges are all 0.

We obtain the following theorem.

Theorem 4.1 *Under Assumption 2.1, for any Moore machine M, there exists a simple recurrent network $\mathcal{E}_M$ which is equivalent to M.*

5 The Size of Simple Recurrent Networks

In this section, we discuss the relationships between the number of states of automata with output and the size of a SRN. Suppose that a SRN $\mathcal{E}$ is equivalnet to automata with output M. Let $K(M)$ be a set of automata which are equivalnet to M and $S(\mathcal{E})$ be a set of SRN's which are equivalnet to $\mathcal{E}$. For M, let $\phi(M)$ denote a number of the states of M. For $\mathcal{E}$, let $size(\mathcal{E})$ denote the size of $\mathcal{E}$, where the size of a SRN is the number of gates of $\mathcal{E}$.

Definition 5.1 *M_{min} is a element of $K(M)$ which satisfies*

$$\phi(M_{min}) = \min\{\phi(M) \mid M \in K(M)\}.$$

Definition 5.2 *$\mathcal{E}_{min}$ is a element of $S(\mathcal{E})$ which satisfies*

$$size(\mathcal{E}_{min}) = \min\{size(\mathcal{E}) \mid \mathcal{E} \in S(\mathcal{E})\}.$$

From the above two construction, we obtain the following theorem.

Theorem 5.1 *For M and $\mathcal{E}$, there exit constants C and C' such that*

$$C \log \phi(M_{min}) \leq size(\mathcal{E}_{min}) \leq C'\phi(M_{min}).$$

6 Conclusion

In this paper, we show that when the range of a function computed at each gate of a SRN is a finite set, there exists a Mealy machine which can simulates the SRN. This result means that, under our assumption, the computational power of simple recurrent networks is not exceed that of finite automata with output, and thus, is not sufficient to recognize natural languages.

On the other hand, a simple recurrent network can simulates a Moore machine as we showed in Section 4.4. From our two results, we have shown that the computational capability of simple recurrent networks is equal to that of finite automata under our assumption.

References

[1] Noga Alon, A. K. Dewdney, and Teunis J. Ott. Efficient simulation of finite automata by neural nets. *Journal of Association for Computing Machinery*, Vol. 38, No. 2, pp. 495–514, April 1991.

[2] M. A. Arbib, A. J. Kfoury, and Robert N. Moll. *A Basis for Theoretical Computer Science.* Springer-Verlag, 1981.

[3] José L. Balcázar, Ricard Gavaldà, Have T. Siegelmann, and Eduardo D. Sontag. Some structural complexity aspects of neural computation. In *Proceedings of the IEEE Structure in Complexity Theory Conference*, pp. 253–265, San Diego, CA, 1993.

[4] Jeffrey L. Elman. Finding structure in time. *Congnitive Science*, Vol. 14, pp. 179–211, 1990.

[5] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, Vol. 7, pp. 195–225, 1991.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language and Computation.* Reading. Addison-Wesley, MA, 1979.

[7] P. Indyk. Optimal simulation of automata by neural nets. In *Proceedings of the 12th Annual Symposium on Theoreical Aspects of Computer Science (STACS'95)*, pp. 337–348, New York, April 1995. Springer-Verlag.

[8] James L. McClelland, Axel Cleeremans, and David Servan-Schreiber. Parallel distributed processing : Bridging the gap between human and machine intelligence. *Journal of Japanese Society of Artificial Intelligence*, Vol. 5, No. 1, pp. 3–14, January 1990.

[9] 守谷純之介, 西野哲朗. 「単純回帰ネットワークを模倣する mealy 機械の構成法」. 電子情報通信学会コンピュテーション研究会資料, COMP98-27, 1998.

[10] Have T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory*, Pittsburgh, PA, July 1992.

[11] Have T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, Vol. 131, pp. 331–360, 1994.