Neural Language Acceptors

Šíma, Jiří
1995

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

NEURAL LANGUAGE ACCEPTORS

Jiří Šíma and Jiří Wiedermann [1]

Technical report No. 625

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+422) 66414244 fax: (+422) 8585789
e-mail: sima@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

NEURAL LANGUAGE ACCEPTORS

Jiří Šíma and Jiří Wiedermann [1]

Technical report No. 625

Abstract

A finite discrete recurrent neural network, working in parallel computation mode, is considered a language acceptor. Such neural acceptors recognize only regular languages. Both, the size of neural acceptors (i.e., the number of neurons) and their descriptional complexity (i.e., the number of bits in the neural acceptor representation) are studied. It is shown that any regular language given by a regular expression of length n is recognized by a neural acceptor with $\Theta(n)$ neurons. Further, it is proved that this network size is, in the worst case, optimal. Then, two specialized constructions of neural acceptors of the optimal descriptional complexity $\Theta(n)$ for a single n–bit string recognition are described. They both require $O(n^{\frac{1}{2}})$ neurons and either $O(n)$ connections with constant weights or $O(n^{\frac{1}{2}})$ edges with weights of the $O(2^{\sqrt{n}})$ size. Finally, the concept of Hopfield languages is introduced by means of so–called Hopfield acceptors (i.e., of neural networks with symmetric weights) and several properties are mentioned.

Keywords

1 Introduction

One less commonly studied task in neurocomputing [3], [7], [8] which we will be dealing with is to compare the computational power of neural networks with the traditional finite models of computation such as recognizers of regular languages. It appears that a finite discrete recurrent neural network can be used for language recognition in parallel mode: at each time step one bit of an input string is presented to the network via an input neuron and an output neuron signals whether the input string that has been read so far belongs to the relevant language. In this way, a language can be recognized by a so-called neural acceptor. It is clear that the neural acceptors recognize only regular languages [5].

A similar definition of neural acceptors appeared in [2], [4] where the problem of language recognition by neural networks has been explored in the context of finite automata. It was shown in [2] that every m-state finite automaton can be realized as a discrete neural net with $O(m^{\frac{3}{4}})$ neurons and that at least $\Omega((m \log m)^{\frac{1}{3}})$ neurons are necessary for such construction. This upper and lower bound was improved in [4] by showing that $O(m^{\frac{1}{2}})$ neurons suffice and that the most of finite automata require $\Omega(m^{\frac{1}{2}})$ neurons when the values of weights in the network are polynomial with respect to the network size. Several practical experiments have been also done with the learning of finite automata by recurrent neural networks using the standard neural learning heuristics back-propagation [6].

In the present paper we relate the size of neural acceptors to regular expressions that on one hand are known to possess the same expressive power as finite automata, but on the other hand they represent a tool whose descriptional efficiency can exceed that of deterministic finite automata.

First, in section 2 we will introduce the basic formalism for dealing with neural acceptors. Then, in section 3 we will prove that any regular language described by a regular expression of a length n can be recognized by a neural acceptor consisting of $O(n)$ neurons. Subsequently, in section 4 we will show that, in general, this result cannot be improved because there is a regular language given by a regular expression of length n requiring neural acceptors of size $\Omega(n)$. Therefore, the respective neural acceptor construction made of regular expressions is size-optimal.

Next, in section 5 we will present two specialized constructions of neural acceptors for single n–bit string recognition that, both, require $O(n^{\frac{1}{2}})$ neurons and either $O(n)$ connections with constant weights or $O(n^{\frac{1}{2}})$ edges with weights of the size $O(2^{\sqrt{n}})$. The number of bits required for the entire string acceptor description in both cases is proportional to the length of the string. This means that these constructions are optimal with respect to the descriptional complexity of neural acceptors. They can be exploited as a part of a more complex efficient architectural neural network design, for example for the construction of a cyclic neural network, with $O(2^{\frac{n}{2}})$ neurons and edges, which computes any boolean function.

In section 6 we will introduce the concept of Hopfield languages as the languages that are recognized by the so–called Hopfield acceptors which are based on symmetric neural networks (Hopfield networks). Hopfield networks have been studied widely outside the framework of formal languages because of their convergence properties.

From the formal language theoretical point of view, it is an interesting fact that the class of Hopfield languages is strictly contained in the class of regular languages. Hence, they represent a natural proper subclass of regular languages. Further, we will formulate the necessary and sufficient, so-called, Hopfield condition stating when a regular language is a Hopfield language. Finally the closure properties of Hopfield languages are mentioned. More information about Hopfield languages, including proofs, can be found in [9].

All previous results jointly point to the fact that neural acceptors present quite an efficient tool not only for the recognition of regular languages and of their subclasses respectively but also for their description.

2 Neural Acceptors

In this section we formalize the concept of a so-called neural acceptor which is a discrete recurrent neural network (further neural network) exploited for a language recognition in the following way: During the network computation, an input string is presented bit after bit to the network by means of one input neuron. All neurons of the network work in parallel. Following this, the output neuron shows whether the input string, that has been already read, is from the relevant language. Similar definition appeared in [2] and [4]. Next, we observe that the neural acceptor can be viewed as a finite automaton [5] and therefore, neural acceptors recognize just regular languages because of Kleene's theorem [1].

Definition 1 *A* neural acceptor *is a 7-tuple $N = (V, inp, out, E, w, h, s_{init})$, where V is the set of n neurons including the input neuron $inp \in V$, and the output neuron $out \in V$, $E \subseteq V \times (V - \{inp\}) - \{\langle inp, out\rangle\}$ is the set of edges, $w : E \longrightarrow \mathcal{Z}$ ($\mathcal{Z}$ is the set of integers) is the* weight *function (we use the abbreviation $w(\langle j, i\rangle) = w_{ij}$), $h : V - \{inp\} \longrightarrow \mathcal{Z}$ is the* threshold *function (the abbreviation $h(i) = h_i$ is used), and $s_{init} : V - \{inp\} \longrightarrow \{0, 1\}$ is the* initial *state of the network. The graph (V, E) is called the* architecture *of the neural network N and $n = |V|$ is the* size *of the neural acceptor. The number of bits that are needed for the whole neural acceptor representation (especially for the weight and threshold functions) is called the descriptional complexity of neural acceptor.*

Definition 2 *Let $x = x_1 x_2 \cdots x_m \in \{0, 1\}^m$, followed formally by some $x_{m+i} \in \{0, 1\}$, $i \geq 1$ (due to the notation consistency), be the* input *for the neural acceptor $N = (V, inp, out, E, w, h, s_{init})$. The* state *of the neural network at the time t is a mapping $s_t : V \longrightarrow \{0, 1\}$. At the beginning of a neural network computation the state s_0 is set to $s_0(i) = s_{init}(i)$ for $i \in V - \{inp\}$ and $s_0(inp) = x_1$. Then at each time step $1 \leq t \leq m + 1$, the network computes its new state s_t from the old state s_{t-1} as follows:*

$$s_t(i) = \begin{cases} x_{t+1} & \text{if } i = inp \\ S\Big(\sum\limits_{\langle j,i\rangle \in E} w_{ij} s_{t-1}(j) - h_i\Big) & \text{if } i \in V - \{inp\} \end{cases}$$

where $S(z) = 1$ for $z \geq 0$ and $S(z) = 0$ otherwise. For the neural acceptor N and its input $x \in \{0, 1\}^m$ we denote the state of the output neuron $out \in V$ in the time step

$m+1$ *by* $N(x) = s_{m+1}(out)$. *Then* $L(N) = \{x \in \{0,1\}^\star \mid N(x) = 1\}$ *is the language* recognized *by the neural acceptor* N.

Theorem 1 *Let* $L = L(N)$ *be a language recognized by some neural acceptor* N. *Then* L *is regular.*

Proof: Let $N = (V, inp, out, E, w, h, s_{init})$ be a neural acceptor. We define a deterministic finite automaton $A = (Q, \Sigma = \{0,1\}, \delta, q_0, F)$ so that $L(N) = L(A)$. Let $Q = \{s : V - \{inp\} \longrightarrow \Sigma\}$ be the set of automaton state and $q_0 = s_{init}$ be an initial automaton state. The transition function is defined for $s \in Q$ and $x \in \Sigma$ as follows:

$$\delta(s,x)(i) = S\Big(\sum_{\langle j,i\rangle \in E} w_{ij} s(j) - h_i\Big) \quad \text{for } i \in V - \{inp\}$$

where $s(inp) = x$. Finally, $F = \{s \in Q \mid \delta(s,x)(out) = 1,\ x \in \Sigma\}$ is the set of final automaton states. Then the proposition follows from Kleene's theorem [1]. □

3 Upper Bound

We show that any regular language, given by a regular expression of a length n, may be recognized by a neural acceptor of size $O(n)$. The constructed architecture of the neural network corresponds to the structure of the regular expression. The neural acceptor passes through all oriented network paths that correspond to all strings generated by this expression and that, at the same time, match the input string.

Theorem 2 *For every regular language* L *denoted by a regular expression* $L = [\alpha]$ *there exists a neural acceptor* N *of the size* $O(|\alpha|)$ *such that* $L = L(N)$ *is recognized by* N.

Proof: Let $L = [\alpha]$ be a regular language denoted by a regular expression α. We construct a neural acceptor $N_\alpha = (V, inp, out, E, w, h, s_{init})$ of size $O(|\alpha|)$ so that $L = L(N_\alpha)$.

We first build an architecture (V, E) of the neural network N_α recursively with respect to the structure of the regular expression α. For that purpose we define the sequence of graphs (V_k, E_k), $k = 0, \ldots, p$ where (V_0, E_0) has one vertex corresponding to the whole expression α which is recursively partitioned into shorter regular subexpressions, so that (V_p, E_p) have vertices corresponding only to the elementary subexpressions 0 or 1 of α (we say, vertices of the type 0 or 1). For the sake of notation simplicity we identify the subexpressions of α with the vertices of these graphs.

1. $V_0 = \{start, \alpha, out\}$
   $E_0 = \{\langle start, \alpha\rangle, \langle \alpha, out\rangle\}$

2. Assume that (V_k, E_k), $0 \le k < p$ have been already constructed and $\beta \in V_k$ is a subexpression of α different from 0 or 1. Hence besides the empty language and the empty string, the regular expression β can denote union, concatenation,

or iteration of subexpressions of β. With respect to the relevant regular operation the vertex β is fractioned and possibly new vertices corresponding to the subexpressions of β arise in the graph (V_{k+1}, E_{k+1}). To be really rigorous we should first remove the vertex β and then add the new vertices. However, due to the notational simplicity we do not insist on such rigorousness and therefore we can identify one of the new vertex with the old β. That is why we write rather non-exactly for example 'β has the form $\beta + \gamma$'. Moreover we substitute $e + \beta^+$ for $\beta^\star$.

- β is $\emptyset$: $V_{k+1} = V_k - \{\beta\}$, $E_{k+1} = E_k - \{\langle x, \beta\rangle, \langle \beta, y\rangle \in E_k\}$.
- β is e: $V_{k+1} = V_k - \{\beta\}$,
  $E_{k+1} = (E_k - \{\langle x, \beta\rangle, \langle \beta, y\rangle \in E_k\}) \cup$
  $\{\langle x, y\rangle \mid \langle x, \beta\rangle, \langle \beta, y\rangle \in E_k - \{\langle \beta, \beta\rangle\}\}$.
- β has the form $\beta + \gamma$: $V_{k+1} = V_k \cup \{\gamma\}$,
  $E_{k+1} = E_k \cup \{\langle x, \gamma\rangle, \langle \gamma, y\rangle \mid \langle x, \beta\rangle, \langle \beta, y\rangle \in E_k\} \cup \{\langle \gamma, \gamma\rangle \mid \langle \beta, \beta\rangle \in E_k\}$.
- β has the form $\beta \cdot \gamma$: $V_{k+1} = V_k \cup \{\gamma\}$,
  $E_{k+1} = (E_k - \{\langle \beta, y\rangle \in E_k\}) \cup \{\langle \beta, \gamma\rangle\} \cup \{\langle \gamma, y\rangle \mid \langle \beta, y\rangle \in E_k\}$.
- β has the form β^+: $V_{k+1} = V_k$, $E_{k+1} = E_k \cup \{\langle \beta, \beta\rangle\}$.

This construction is finished after $p = O(|\alpha|)$ steps when V_p contains only subexpressions 0 or 1. Then we define the network architecture in the following way:

$$V = V_p \cup \{inp\},\ E = E_p \cup \{\langle inp, \beta\rangle \mid \beta \in V_p - \{start, out\}\}.$$

For $i \in V$ denote by $d(i) = |\{j \in V_p \mid \langle j, i\rangle \in E\}|$. Now we can define the weight function w and the threshold function h:

- $i \in V$ is the neuron of the type 1: $w_{ij} = 1$ for $\langle j, i\rangle \in E_p$ and $w_{i,inp} = d(i)$, $h_i = d(i) + 1$.
- $i \in V$ is the neuron of the type 0: $w_{ij} = 1$ for $\langle j, i\rangle \in E_p$ and $w_{i,inp} = -d(i)$, $h_i = 1$.
- $start \in V$: $h_{start} = 1$.
- $out \in V$: $w_{out,j} = 1$ for $\langle j, out\rangle \in E_p$, $h_{out} = 1$.

To define the initial state remains; $s_0(i) = 0$ for $i \in V_p - \{start\}$ and $s_0(start) = 1$.

The set V contains three special neurons $(inp, out, start)$ as well as other neurons of the type 0 or 1 — one for each subexpression 0 or 1 in α, hence $|V| = O(|\alpha|)$. An example of the neural acceptor for the regular language $[((0 + e)1)^\star]$ is in figure 3.1 (the types of neurons are depicted inside the circles representing neurons; thresholds are depicted as weights of edges with constant inputs -1).

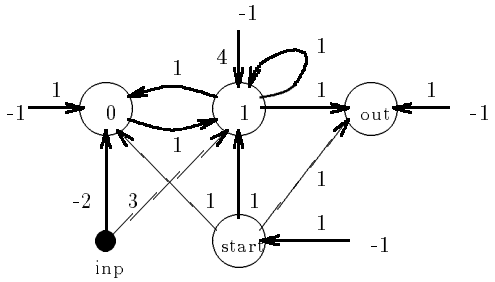

Figure 3.1: Neural acceptor for $[((0+e)1)^\star]$.

We prove that $L = [\alpha] = L(N_\alpha)$. From the construction of (V_p, E_p) given above it is easy to observe that this graph corresponds to the structure of the regular expression α. This means that for every string $x = x_1 \cdots x_m \in L \subseteq \{0,1\}^\star$ there is an oriented path $start \to x_1 \to x_2 \to \cdots \to x_m \to out$ leading from $start \in V_p$ to $out \in V_p$ and containing the vertices of the relevant types (0 or 1). On the other hand for any such path there is a corresponding string in L.

The neural acceptor N_α passes through all possible paths that match the network input. In the beginning the only non-input neuron $start \in V$ is active (its state is 1). It sends a signal to all connected neurons and subsequently becomes passive (its state is 0) due to the dominant threshold. The connected neurons of the type 0 or 1 compare its type with the network input and become active only when it matches, otherwise they remain passive. Due to the weight and threshold values it follows that any neuron of the type 1 becomes active iff $inp \in V$ is active and at least one of $j \neq inp$, $\langle j, i\rangle \in E$, is active, and any neuron of the type 0 becomes active iff $inp \in V$ is passive and at least one of $j \neq inp$, $\langle j, i\rangle \in E$, is active. This way all relevant paths are being traversed, and their traverse ends in the neuron $out \in V$ which realizes the logical disjunction, and is active iff the prefix of the input string, that has been read so far, belongs to L. This completes the proof that $L = L(N_\alpha)$. □

4 Lower Bound

In this section we show the lower bound $\Omega(n)$ for the number of neurons that, in the worst case, are necessary for the recognition of a regular language which is described by a regular expression of the length n. As a consequence, it follows that the construction of the neural acceptor from section 3 is size-optimal.

The standard technique is employed for this purpose. For a given length of regular expression we define a regular language and the corresponding set with exponential number of prefixes for this language. We prove that these prefixes must bring any neural acceptor into the exponential number of different states in order to provide a correct recognition. This will imply the desired lower bound.

Definition 3 *For $n \geq 1$ and $k \geq 0$ define the following regular languages:*

$$L_n = \Big[\Big(10 \;+\; 1(e+0)10 \;+\; (1(e+0))^2\,10 \;+\; \cdots \;+\; \cdots \;+\; (1(e+0))^{n-3}\,10 \;+\; (1(e+0))^{n-1}\,(1+0)\Big)^\star\Big],$$

$$\Pi_k = \left[(1(e+0))^k\right], \qquad P_n = \bigcup_{k=0}^{n-1} \Pi_k.$$

It is clear that P_n, $n \geq 1$ is the set of prefixes for the language L_n. We prove several lemmas concerning properties of these regular languages. The regular expression which defines the language L_n in Definition 3 is in fact of $O(n^2)$ length because the abbreviation for repeated concatenation is not included when determining its length. Therefore, we first show that there is a regular expression α_n, of the linear length only, denoting the same language L_n. The number of prefixes in P_n is shown to be exponential with respect to n.

Lemma 1

(i) $\exists \alpha_n \in RE \quad L_n = [\alpha_n] \quad and \quad |\alpha_n| = O(n)$.

(ii) $|P_n| = 2^n - 1$.

Proof:

(i) In the regular expression which denotes the language L_n from Definition 3, we can subsequently factor out $(n-3)$ times the subexpression $1(e+0)$ to obtain the desired regular expression

$$\alpha_n = \Big(10 + 1(e+0)\Big(10 + 1(e+0)\Big(\cdots 1(e+0)\Big(10 + 1(e+0)^2(1+0)\Big)\cdots\Big)\Big)\Big)^\star$$

of the linear length $|\alpha_n| = O(n)$ which defines the same language $L_n = [\alpha_n]$.

(ii) It follows from Definition 3 that $|\Pi_k| = 2^k$ and therefore, $|P_n| = \sum_{k=0}^{n-1} 2^k = 2^n - 1$. □

The following lemma shows how the prefixes in P_n can be completed to strings from L_n.

Lemma 2

(i) $x = x'1 \in P_n \longrightarrow x0 \in L_n$.

(ii) *Let* $x \in P_n$ *then* $x1 \in L_n \longleftrightarrow x \in \Pi_{n-1}$.

(iii) $x = x'1 \in L_n \longrightarrow \exists x_1 \in L_n, \ \exists x_2 \in \Pi_{n-1} \ \ x = x_1 x_2 1$.

Proof:

(i), (ii) follow from Definition 3.

(iii) Assume $x = x'1 \in L_n$. The language L_n is defined via iteration in Definition 3. Hencefore, we can write $x = x'_1 x'_2 \cdots x'_r$, where $x'_i \in L_n$, for $i = 1, \ldots, r$. Define $x_1 = x'_1 x'_2 \cdots x'_{r-1} \in L_n$ and x_2 so that $x'_r = x_2 1$. Obviously $x_2 \in \Pi_{n-1}$. □

Now we prove that any two different prefixes from P_n can be completed by the same suffix, so that one of the resulting string is in L_n while the other one is not.

Lemma 3 $\forall x_1, x_2 \in P_n \;\; x_1 \neq x_2 \longrightarrow \exists y \in \{0,1\}^\star \;\; x_1 y \in L_n \;\; and \;\; x_2 y \notin L_n.$

Proof: Assume $x_1, x_2 \in P_n$. Then there exist $0 \leq k_1, k_2 < n$ such that $x_1 \in \Pi_{k_1}$, $x_2 \in \Pi_{k_2}$. We will distinguish two cases:

1. $k_1 \neq k_2$:
   Without loss of generality suppose $n > k_1 > k_2$. Define $y = 1^{n-k_1}$.

   - Denote $x_1' = x_1 1^{n-k_1-1} \in \Pi_{n-1}$. From (ii) Lemma 2 we obtain $x_1 y = x_1' 1 \in L_n$.
   - Denote $x_2' = x_2 1^{n-k_1-1} \in \Pi_{k_2+n-k_1-1}$. From $k_1 > k_2$ it follows $k_2 + n - k_1 - 1 < n - 1$. Therefore $x_2 y = x_2' 1 \notin L_n$ due to (ii) Lemma 2.

2. $k_1 = k_2 = k$:
   We can write $x_1 = 1a_1 1a_2 \cdots 1a_k$, $a_j \in \{e, 0\}$ and $x_2 = 1b_1 1b_2 \cdots 1b_k$, $b_j \in \{e, 0\}$, for $j = 1, \ldots, k$. Let $i = \min\{j \mid a_j \neq b_j\}$. Without loss of generality $a_i = 0$, $b_i = e$. Define $y = 1^{n-k+i}$.

   - Denote $z_1 = 1a_1 1a_2 \cdots 1a_{i-1} 1 \in \Pi_i$, $z_2 = 1a_{i+1} 1a_{i+2} \cdots 1a_k 1^{n-k+i-1} \in \Pi_{n-1}$. From (i) Lemma 2 $z_1 0 \in L_n$ and $z_2 1 \in L_n$ from (ii) Lemma 2. This implies that $x_1 y = z_1 0 z_2 1 \in L_n$ because L_n is closed under the concatenation.
   - Denote $x_2' = x_2 1^{n-k+i-1} \in \Pi_{n+i-1}$. To the contrary suppose that $x_2 y = x_2' 1 \in L_n$. From (iii) Lemma 2 $z_1 \in L_n$, $z_2 \in \Pi_{n-1}$ such that $x_2 y = z_1 z_2 1$ exist. Hence $z_2 = 1b_{i+1} 1b_{i+2} \cdots 1b_k 1^{n-k+i-1} \in \Pi_{n-1}$ because $x_2 y = 1b_1 1b_2 \cdots 1b_k 1^{n-k+i}$. Denote $z_1' = 1b_1 1b_2 \cdots 1b_{i-1}$, $i < n$ and from (ii) Lemma 2 it follows $z_1 = z_1' 1 \notin L_n$ which is a contradiction. Thus $x_2 y \notin L_n$. □

Now we are ready to prove the following theorem concerning the lower bound.

Theorem 3 *Any neural acceptor N that recognizes the language $L_n = L(N)$ requires at least $\Omega(n)$ neurons.*

Proof: The neural acceptor N, that recognizes the language $L_n = L(N)$ from Definition 3, must have $\Omega(2^n)$ different states which are reached when taking input prefixes from P_n because any different $x_1, x_2 \in P_n$ can be completed by y from Lemma 3 so that $x_1 y \in L_n$ and $x_2 y \notin L_n$. This implies that N needs $\Omega(n)$ binary neurons. □

5 Neural String Acceptors

The previous results showed that neural acceptors have the same descriptive capabilities as regular expressions. In this section we will study how powerful they are when we confine ourselves on a certain subclass of regular expressions. Here, we will deal with the simplest case considering just fixed binary strings from $\{0,1\}^\star$. We present two constructions of neural acceptors for a single string recognition. For n–bit string they both require $O(n^{\frac{1}{2}})$ neurons and either $O(n)$ connections with constant weights or

$O(n^{\frac{1}{2}})$ edges with weights of the $O(2^{\sqrt{n}})$ size. The number of bits required for the entire string acceptor description is in both cases proportional to the length of the string. This means that these constructions are optimal from the neural acceptors descriptional complexity point of view.

This elementary case can be useful because the single string recognition is very often a part of more complicated tasks. In a personal communication Piotr Indyk pointed out that the latter string acceptor construction can also be exploited for building a cyclic neural network, with $O(2^{\frac{n}{2}})$ neurons and edges, which computes any boolean function.

Theorem 4 *For any string $a = a_1 \dots a_n \in \{0,1\}^n$ there exists a neural acceptor of the size $O(n^{\frac{1}{2}})$ neurons with $O(n)$ connections of constant weights. Thus, the descriptional complexity of the neural acceptor is $\Theta(n)$.*

Proof: For the sake of simplicity suppose first that $n = p^2$ for some positive integer p. The idea of the neural acceptor construction is to split the string $a \in \{0,1\}^n$ into p pieces of the length p and to encode these p substrings using p^2 (for each p) binary weights of edges leading to p comparator neurons $c_1, \dots, c_p$. The input string $x = x_1 \dots x_n \in \{0,1\}^n$ is being gradually stored per p bits into p buffer neurons $b_1, \dots, b_p$. When the buffer is full, the relevant comparator neuron c_i compares the assigned part of the string $a_{(i-1)p+1}, \cdots, a_{ip}$ with the corresponding part of the input string $x_{(i-1)p+1}, \cdots, x_{ip}$, $1 \leq i \leq p$, that is stored in the buffer, and sends the result of this comparison to the next comparator neuron. The synchronization of the comparison is performed by $2p$ clock neurons, where neurons $s_1, \dots, s_p$ tick at each time step of the network computation, and neurons $m_1, \dots, m_p$ tick once in a period of p such steps. The last comparator neuron c_p represents the output neuron and reports at the end whether the input string x matches a.

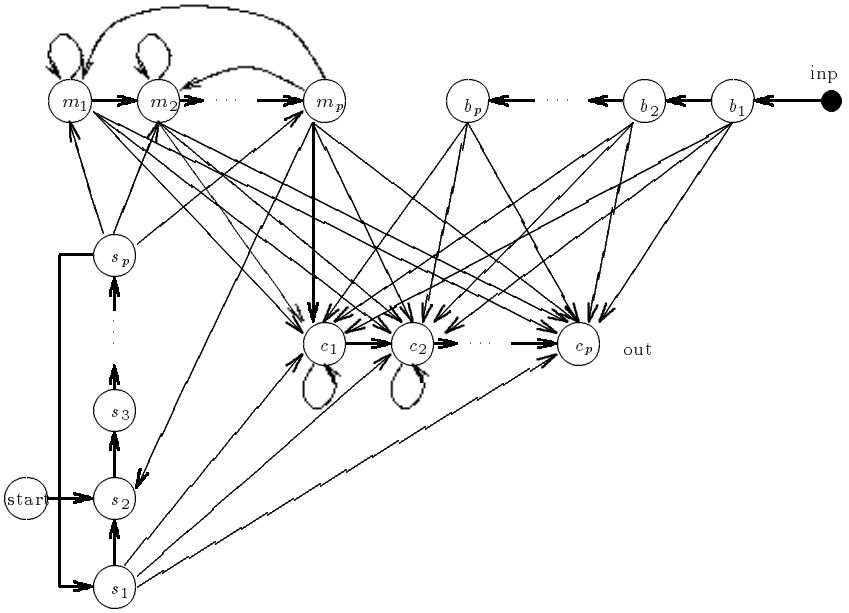

Figure 5.1: Architecture for a neural string acceptor with $O(n)$ edges.

A formal definition of the neural acceptor $N = (V, inp, out, E, w, h, s_{init})$ for the recognition of the string a follows (see also figure 5.1). Define

$$V = \{inp, start, c_1, \ldots, c_p = out, b_1, \ldots, b_p, s_1, \ldots, s_p, m_1, \ldots, m_p\},$$

$$\begin{aligned} E = \; & \{\langle b_i, c_j\rangle, \langle m_i, c_j\rangle \mid 1 \le i, j \le p\} \cup \{\langle s_1, c_i\rangle, \langle s_p, m_i\rangle \mid i = 1, \ldots, p\} \cup \\ & \{\langle c_i, c_{i+1}\rangle, \langle b_i, b_{i+1}\rangle, \langle s_i, s_{i+1}\rangle, \langle m_i, m_{i+1}\rangle \mid i = 1, \ldots, p-1\} \cup \\ & \{\langle c_i, c_i\rangle, \langle m_i, m_i\rangle, \langle m_p, m_i\rangle \mid i = 1, \ldots, p-1\} \cup \\ & \{\langle inp, b_1\rangle, \langle start, s_2\rangle, \langle s_p, s_1\rangle, \langle m_p, s_2\rangle\}. \end{aligned}$$

Denote by $d_i = \left|\{j \mid a_{(i-1)p+j} = 1,\ 1 \le j \le p\}\right|$. Then put

$$w(\langle b_j, c_i\rangle) = 2a_{(i-1)p+j} - 1, \quad w(\langle m_j, c_i\rangle) = \begin{cases} 1 & \text{if } j \le i \\ -1 & \text{if } j > i \end{cases} \qquad 1 \le i, j \le p$$

$$w(\langle c_i, c_i\rangle) = 2p - d_i - i + 1 \quad i = 1, \ldots, p-1$$

$$w(\langle m_p, s_2\rangle) = -1, \quad w(\langle m_p, m_i\rangle) = -1 \quad i = 1, \ldots, p-1$$

$$h(c_i) = \begin{cases} d_i + i + 1 & \text{if } i = 1 \\ d_i + i + 2 & \text{if } 2 \le i \le p \end{cases} \qquad h(m_i) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } 2 \le i \le p. \end{cases}$$

All remaining weights and thresholds are set to 1. Finally the initial state is determined: $s_0(v) = 0$ for $v \in V - \{inp, star\}$ and $s_0(start) = 1$.

Note that the weights $w(\langle c_i, c_i\rangle) = O(p)$, $i = 1, \ldots, p-1$ are not constant as required. It is due to the neuron c_i which should keep its state 1 after it becomes active in order to transfer the possible positive result of the comparison to the next comparator neuron. Therefore the relevant feedback must exceed the sum of all other inputs to achieve the threshold value. This can be avoided by inserting $p-1$ auxiliary neurons that remember the result of preceding comparisons between neighbor comparator neurons. All these neurons have a constant feedback because they have only one input, from the previous comparator neuron, to be exceeded.

The construction of the neural acceptor can also be easily adapted for $n = p^2 + r$, $1 \le r < p$. The technique from the proof of Theorem 2 is employed for the recognition of the last r bits of the string a. The resulting architecture of size $O(r)$ is then connected to the neural string acceptor by identifying the neuron $start$ with the above-mentioned neuron $out = c_p$. □

Theorem 5 *For any string $a \in \{0,1\}^n$ there exists a neural acceptor of the size $O(n^{\frac{1}{2}})$ neurons with $O(n^{\frac{1}{2}})$ edges with weights of the size $O(2^{\sqrt{n}})$. Thus, the descriptional complexity of the neural acceptor is $\Theta(n)$.*

Proof: The design of the desired neural acceptor is very similar to the construction from the proof of Theorem 4 except that the number of comparator neurons is reduced to two neurons $c_{\le}, c_{\ge}$. In this case the substrings $a_{(i-1)p+1}, \cdots, a_{ip}$, $1 \le i \le p$ are encoded by $O(p)$ (each by 2) weights of the size $O(2^p)$ corresponding to the connection leading from the clock neurons m_i to these comparison neurons. The contents of the input buffer, viewed as a binary number, are first converted into an integer. This integer is then compared with the relevant encoded part of the string a by the comparator

neuron $c_{\leq}$ to see whether it is smaller or equal. At the same time it is compared by the comparator neuron $c_{\geq}$ to see whether it is greater or equal. The neuron which realizes the logical conjuction of comparator outputs is added to indicate whether the part of the input string matches the corresponding part of the string a. However, this leads to one more computational step of the neural acceptor which is inconsistent with Definition 2. The correct synchronization can be achieved by exploiting the above-mentioned additional architecture for the recognition of the last r bits $a_{q+1}, a_{q+2}, \ldots, a_{q+r}$ $(q = p^2)$ of the string a. Details of the synchronization are omitted as well as a complete formal definition of the neural string acceptor. We give only the definition of the weights that are relevant for the comparisons:

$$w\left(\langle b_i, c_{\leq}\rangle\right) = -2^{i-1}, \quad w\left(\langle b_i, c_{\geq}\rangle\right) = 2^{i-1} \quad i = 1, \ldots, p$$

$$w\left(\langle m_i, c_{\leq}\rangle\right) = \begin{cases} \sum_{j=1}^{p} 2^{p-j} a_j & \text{for } i = 1 \\ \sum_{j=1}^{p} 2^{p-j} a_{(i-1)p+j} - \sum_{j=1}^{p} 2^{p-j} a_{(i-2)p+j} & \text{for } i = 2, \ldots, p \end{cases}$$

$$w\left(\langle m_i, c_{\geq}\rangle\right) = -w\left(\langle m_i, c_{\leq}\rangle\right), \quad h\left(c_{\leq}\right) = h\left(c_{\geq}\right) = 0.$$

The weights $w\left(\langle m_i, c_{\leq}\rangle\right) = -w\left(\langle m_i, c_{\geq}\rangle\right)$, $i = 1, \ldots, p$ are defined as differences because all clock neurons $m_j = 1$ for $1 \leq j \leq i$ are active only when the part $x_{(i-1)p+1}, \cdots, x_{ip}$ of the input is in the buffer. The architecture of the neural string acceptor is depicted in figure 5.2 (instead of the above-mentioned conjuction the neuron *reset* is added to realize the negation of comparator conjuction to possibly terminate the clock). □

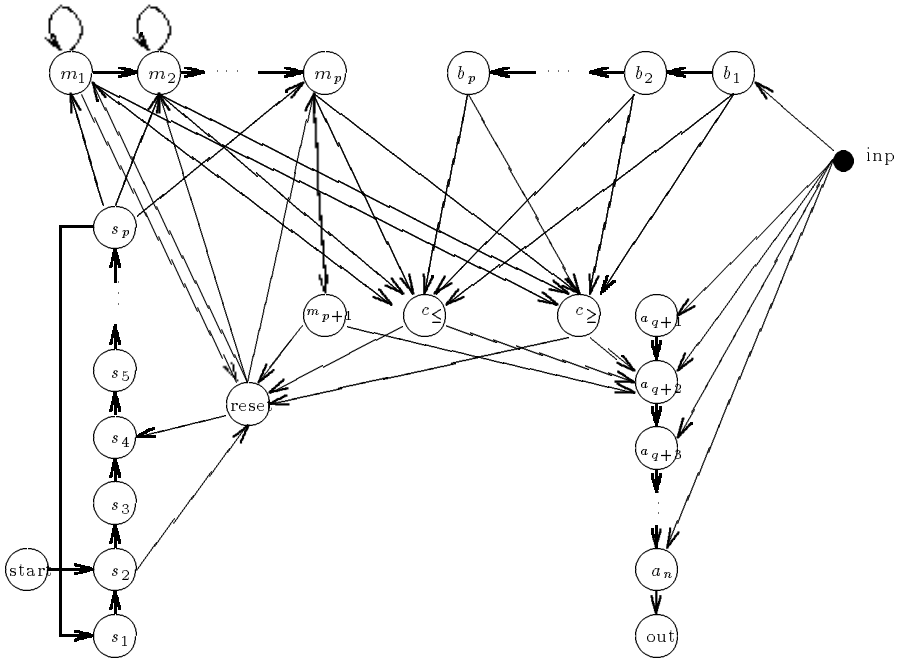

Figure 5.2: Architecture for a neural string acceptor with $O(n^{\frac{1}{2}})$ edges.

6 Hopfield Languages

In section 5 we have restricted ourselves to a special subclass of regular expressions. In this section we will concentrate on a special type of neural acceptors, the so–called Hopfield acceptors which are based on symmetric neural networks (Hopfield networks). In these networks the weights are symmetric and therefore, the architecture of such neural acceptors is given by an undirected graph.

Hopfield networks have been traditionally studied [3], [10] and used due to their convergence properties. These networks are also of particular interest, because their natural physical realizations exist (e.g. Ising spin glasses, 'optical computers').

Using the concept of Hopfield acceptors, we will define a class of Hopfield languages that are recognized by these acceptors. We mention several properties of this class. In this section all proofs are omitted and can be found in [9].

Definition 4 *The neural acceptor* $N = (V, inp, out, E, w, h, s_{init})$ *that is based on symmetric neural network (Hopfield network) where* $\langle i,j \rangle \in E \longleftrightarrow \langle j,i \rangle \in E$, *and* $w_{ij} = w_{ji}$, *for every* $i,j \in V - \{inp\}$ *is called* Hopfield acceptor. *The language* $L = L(N)$ *recognized by Hopfield acceptor* N *is called* Hopfield language.

The class of Hopfield languages is strictly contained within the class of regular languages. We give an evidence of this fact by formulating the necessary and sufficient condition — the so-called Hopfield condition when a regular language is a Hopfield language. Intuitively, Hopfield languages cannot include words with those potentially infinite substrings which allow Hopfield acceptor to converge and to forget relevant information about the previous part of the input string which is recognized.

Definition 5 *A regular language* L *is said to satisfy a* Hopfield condition *iff for every* $v_1, v_2 \in \{0,1\}^\star$ *and* $x \in \{0,1\}^2$ *there exists* $m_0 > 0$ *such that either* $(\forall m \geq m_0 \;\; v_1 x^m v_2 \in L)$ *or* $(\forall m \geq m_0 \; v_1 x^m v_2 \notin L)$.

The following theorem gives a characterization of the Hopfield languages class. The construction of a Hopfield acceptor for a regular languague satisfying the Hopfield condition can be realized with $O(|\alpha|)$ neurons and so it is size-optimal as well.

Theorem 6 *Let* L *be a regular language. Then* L *is a Hopfield language iff* L *satisfies the Hopfield condition.*

For example it follows from Theorem 6 that the regular languages $[(000)^\star]$, $[(1010)^\star]$ are not Hopfield languages because they do not satisfy the Hopfield condition.

We also investigate the closure properties of the Hopfield languages class.

Theorem 7 *The class of Hopfield languages is closed under the union, concatenation, intersection, and complement. It is not closed under the iteration.*

Acknowledgement

We are grateful to Piotr Indyk for the stimulating discussion related to the topics of this paper. Further we thank Tereza Bedaňová who realized the pictures in LaTex environment.

Bibliography

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. (Addison-Wesley, 1974).

[2] N. Alon, A. K. Dewdney and T. J. Ott, *Efficient Simulation of Finite Automata by Neural Nets*. Journal of ACM **38** (1991), 495-514.

[3] P. Floréen, P. Orponen, *Complexity Issues in Discrete Hopfield Networks*. Report A-1994-4 (Dept. of Computer Science, University of Helsinki, 1994).

[4] P. Indyk, *Optimal Simulation of Automata by Neural Nets*. Proc. of the 12th Annual Symposium on Theoretical Aspects of Computer Science STACS'95 LNCS 900 (1995), 337-348.

[5] S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata. In: C. E. Shannon and J. McCarthy (eds.), *Automata Studies*. (Princeton Univ. Press, New Jersey, 1965), 3-43.

[6] P. Manolios and R. Fanelli, *First-Order Recurrent Neural Networks and Deterministic Finite State Automata*. Neural Computation **6** (1994), 1155-1173.

[7] P. Orponen, *Computational Complexity of Neural Networks: A Survey*. Nordic Journal of Computing **1** (1994), 94-110.

[8] I. Parberry, *Circuit Complexity and Neural Networks*. (The MIT Press, Cambridge, Massachusetts, 1994).

[9] J. Šíma, *Hopfield Languages*. To appear at the SOFSEM Winter School, (Milovy, Czech rep., 1995).

[10] J. Wiedermann, *Complexity Issues in Discrete Neurocomputing*. Neural Network World **4** (1994), 99-119.