Theory of Neuromata

Jiří Šíma, Jiří Wiedermann *
Department of Theoretical Informatics
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8
Czech Republic
e-mail: sima@uivt.cas.cz

Abstract

A finite automaton — the so-called neuromaton, realized by a finite discrete recurrent neural network, working in parallel computation mode, is considered. Both the size of neuromata (i.e., the number of neurons) and their descriptional complexity (i.e., the number of bits in the neuromaton representation) are studied. It is proved that a constant time delay of the neuromaton output does not play a role within a polynomial descriptional complexity. It is shown that any regular language given by a regular expression of length n is recognized by a neuromaton with $\Theta(n)$ neurons. Further, it is proved that this network size is, in the worst case, optimal. On the other hand, generally there is not an equivalent polynomial length regular expression for a given neuromaton. 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. Furthermore, the concept of Hopfield languages is introduced by means of so–called Hopfield neuromata (i.e., of neural networks with symmetric weights). It is proved that the class of Hopfield languages is strictly contained in the class of regular languages. The necessary and sufficient so–called Hopfield condition stating when a regular language is a Hopfield language, is formulated. A construction of a Hopfield neuromaton is presented for a regular language satisfying the Hopfield condition. The class of Hopfield languages is shown to be closed under union, intersection, concatenation and complement, and it is not closed under iteration. Finally, the problem whether a regular language given by a neuromaton (or by a Hopfield acceptor) is nonempty, is proved to be PSPACE-complete. As a consequence, the same result for a neuromaton equivalence problem is achieved.

1 Introduction

Neural networks [7] are models of computation motivated by our ideas about brain functioning. Both their computational power and their efficiency have been traditionally investigated [4, 14, 15, 19, 21] within the framework of computer science.

One less commonly studied task which we will be addressing is the comparison of the computational power of neural networks with the traditional finite models of computation, such as recognizers of regular languages. It appears that a finite

*This research was supported by GA ČR Grant No. 201/95/0976.

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, possibly with a constant time delay, whether the input string which has been read, so far, belongs to the relevant language. In this way, a language can be recognized by a neural acceptor (briefly, by a neuromaton). It is clear that the neuromata recognize just the regular languages [12].

A similar definition of a neural acceptor appeared in [2, 8], 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 deterministic 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 [6, 8] by showing that $\Theta(m^{\frac{1}{2}})$ neurons suffice and that, in the worst case, this network size cannot be decreased assuming either at most $O(\log m)$-time simulation delay [6] or polynomial weights [8]. Moreover, several experiments to train the (second-order) recurrent neural networks from examples to behave like deterministic finite automata either for a practical exploitation or for a further rule extraction have also been done [5, 13, 20, 22] using the standard neural learning heuristics back-propagation.

In the present paper we relate the size of neural acceptors to the length of 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 and 3, respectively, we will introduce the basic formalism for dealing with regular languages and neuromata. We will prove that a constant time delay of the neuromaton output does not play a role within a linear neuromata size. Therefore, we will restrict ourselves to neuromata which respond only one computational time step after the input string is read. Then, in section 4 we will prove that any regular language described by a regular expression of a length n can be recognized by a neuromaton consisting of $O(n)$ neurons. Subsequently, in section 5 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 neuromaton construction from a regular expression is size-optimal. This can be used, for example, for constructive neural learning when a learning algorithm for regular expressions from example strings [3] is first employed. On the other hand, in section 6 this construction is proven not to be efficiently reversible because there exists a neuromaton for which every equivalent regular expression is of an exponential length.

Next, in section 7 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 automata constructions are optimal from the descriptional complexity point of view. They can be exploited as a part of a more complex 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 [9].

In section 8 we will introduce the concept of Hopfield languages as the languages that are recognized by the so–called Hopfield acceptors (Hopfield neuromata) which are based on symmetric neural networks (Hopfield networks). Hopfield networks have been studied widely outside of the framework of formal languages, because of their convergence properties. From the formal language theoretical point of view we will prove an interesting fact, namely 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.

Furthermore, we will formulate the necessary and sufficient, so-called, Hopfield condition stating when a regular language is a Hopfield language. In section 9 we will show a construction of a Hopfield neuromaton with $O(n)$ neurons for a regular language satisfying the Hopfield condition. Thus, we will obtain a complete characterization of the class of Hopfield languages. As far as, the closure properties of Hopfield languages are concerned, we will show that the class of Hopfield languages is closed under union, intersection, concatenation and complement and that it is not closed under iteration.

Finally, in section 10 we investigate the complexity of the emptiness problem for regular languages given by neuromata or by Hopfield acceptors. We will prove that both problems are PSPACE-complete. This is a somewhat surprising because the identical problems for regular expressions, deterministic and non-deterministic finite automata are known to only be NL-complete [10, 11]. It confirms the fact from section 6 that neuromata can be stronger than regular expressions from the descriptional complexity point of view. As a next consequence we will obtain that the equivalence problem for neuromata is PSPACE-complete as well.

All previous results jointly point to the fact that neuromata present quite an efficient tool not only for the recognition of regular languages and of their subclasses respectively, but also for their description. In addition, the above-mentioned constructions can be generalized for the analog neural networks [18].

A preliminary version of this paper concerning general neuromata and Hopfield languages, respectively, appeared in [16] and [17].

2 Regular Languages

We recall some basic notions from language theory [1]. We introduce the definition of regular expressions which determine regular languages. The concept of a (deterministic) finite automaton is defined and Kleene's theorem about the correspondence between finite automata and regular languages is mentioned as well.

Definition 1 *An* alphabet *is a finite set of symbols. A* string *over an alphabet* Σ *is a finite-length sequence of symbols from* Σ*. The* empty string*, denoted by* e*, is the string with no symbols. If* x *and* y *are strings, then the* concatenation *of* x *and* y *is the string* xy*. The string* $\underbrace{xx\cdots x}_{n\,times}$ *is abbreviated to* x^n*. The* length *of a string* x*, denoted by* $|x|$*, is the total number of symbols in* x*.*

Definition 2 *A* language *over an alphabet* Σ *is a set of strings over* Σ*. Let* L_1 *and* L_2 *be two languages. The language* $L_1 \cdot L_2$*, called the* concatenation *of* L_1 *and* L_2*, is* $\{xy | x \in L_1, y \in L_2\}$*. Let* L *be a language. Then define* $L^0 = \{e\}$ *and* $L^n = L \cdot L^{n-1}$ *for* $n \geq 1$*. The* iteration *of* L*, denoted* $L^\star$*, is the language* $L^\star = \bigcup_{n=0}^{\infty} L^n$*. Similarly the* positive iteration $L^+ = \bigcup_{n=1}^{\infty} L^n$*.*

Definition 3 *The set* RE *of* regular expressions *over an alphabet* $\Sigma = \{0, 1\}$ *is defined as the minimal language over an alphabet* $\{0, 1, \emptyset, e, +, \cdot, ^\star, (,)\}$ *satisfying the following conditions:*

1. $\emptyset, e, 0, 1 \in RE$
2. *if* $\alpha, \beta \in RE$ *then also* $(\alpha + \beta), (\alpha \cdot \beta), \alpha^\star \in RE$*.*

In writing a regular expression we can omit many parentheses if we assume that * has higher precedence than concatenation and the latter has higher precedence than $+$. For example, $((0(1^\star))+0)$ may be written $01^\star+0$. We also abbreviate the expression $\underbrace{\alpha\alpha\cdots\alpha}_{n\,times}$ to α^n but $|\alpha^n|$ remains $n\cdot|\alpha|$.

Definition 4 *The set $RL=\{[\alpha]\,|\,\alpha\in RE\}$ is the set of* regular languages $[\alpha]$ *which are denoted by regular expressions α as follows:*

1. $[\emptyset]=\emptyset, [e]=\{e\}, [0]=\{0\}, [1]=\{1\}$
2. *if $\alpha,\beta\in RE$ then $[\alpha+\beta]=[\alpha]\cup[\beta], [\alpha\cdot\beta]=[\alpha]\cdot[\beta], [\alpha^\star]=[\alpha]^\star$.*

We also use a regular expression α^+ corresponding to the positive iteration $[\alpha]^+$. Clearly $[\alpha^\star]=[e+\alpha^+]$.

Definition 5 *A* (deterministic) finite automaton *is a 5-tuple $A=(Q,\Sigma,\delta,q_0,F)$ where $Q\neq\emptyset$ is a finite set of* automaton states, *Σ is an* input alphabet *(in our case $\Sigma=\{0,1\}$), $\delta: Q\times\Sigma\longrightarrow Q$ is the* transition function, *$q_0\in Q$ is the* initial *state of the automaton, and $F\subseteq Q$ is a set of* accepting *states.*

Definition 6 *The* generalized *transition function $\delta^\star: Q\times\Sigma^\star\longrightarrow Q$ of the automaton is defined in the following way:*

1. $\delta^\star(q,e)=q$ *for $q\in Q$.*
2. $\delta^\star(q,wa)=\delta(\delta^\star(q,w),a)$ *for $q\in Q,\ w\in\Sigma^\star,\ a\in\Sigma$.*

Then $L(A)=\{w\in\Sigma\,|\,\delta^\star(q_0,w)\in F\}$ is the language recognized by the finite automaton A.

Theorem 1 (Kleene). *A language L is regular iff it is recognized by some finite automaton A (i.e., $L=L(A)$).*

3 Neuromata

In this section we formalize the concept of a neural acceptor — the so-called neuromaton which is a discrete recurrent neural network (or neural network, for short) 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 a single predetermined input neuron. All neurons of the network work in parallel. Following this, with a possible constant time delay the output neuron shows whether the input string, that has been already read, is from the relevant language. A similar definition appeared in [2] and [8].

Definition 7 *A* neural acceptor *(briefly, a* neuromaton*) 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 $out\in V$, $E\subseteq V\times(V-\{inp\})$ 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 neuromaton. The number of bits that are needed for the whole neuromaton representation (especially for the weight and threshold functions) is called the* descriptional complexity *of neuromaton.*

Definition 8 *Let $x = x_1x_2 \cdots x_m \in \{0,1\}^m$, followed formally, due to the notational consistency, by arbitrary $x_{m+l} \in \{0,1\}$, $l \geq 1$, be the* input *for the neuromaton $N = (V, inp, out, E, w, h, s_{init})$. Further, assume that all oriented paths from inp to out in the architecture (V, E) have length at least $k+1$ for some $k \geq 1$. The* state *of the neural network at the discrete 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:*

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

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 $m+k$ by $N_k(x) = s_{m+k}(out)$. Then $L_k(N) = \{x \in \{0,1\}^\star \mid N_k(x) = 1\}$ is the language recognized *by the neuromaton N with the* time delay *k.*

First, we show with respect to language recognition capabilities that a constant time delay of the neuromaton output does not play a role, within a linear neuromata size. More precisely, all languages recognized with the time delay k by a neuromaton of size n can be recognized with the time delay 1 by a neuromaton of size $O(2^k n)$.

Lemma 1 *Let $N = (V, inp, out, E, w, h, s_{init})$ be a neuromaton of size n such that all oriented paths from inp to out in the architecture (V, E) have length at least $k+1$ for some $k \geq 1$. Then there exists a neuromaton $N^\star = (V^\star, inp^\star, out^\star, E^\star, w^\star, h^\star, s^\star_{init})$ of the size $2^k(n-1)+2$ such that $L_k(N) = L_1(N^\star)$.*

Proof: The idea of the proof is to construct $N^\star$ in such a way that it checks ahead all possible computations of N for the next $k-1$ steps and cancels all wrong computations that do not match the actual input being read with the time delay $k-1$.

For this purpose besides the input and output neurons $inp^\star$, $out^\star$, the neuromaton $N^\star$ consists of 2^k blocks N_x ($x \in \{0,1\}^k$), which foresee the computation of N, each of them for one of the possible next k bits x of input. Denote neurons in these blocks in the same way as in N but each of them indexed by the relevant k bits. Then the state of the neuron v_x of $N^\star$ is equal to the state of the neuron $v \in V$ of N after the computation over the next $k-1$ input bits (i.e., over the first $k-1$ bits of $x \in \{0,1\}^k$) is performed. This is achieved as follows.

The block N_{yb} where $y \in \{0,1\}^{k-1}$, $b \in \{0,1\}$ computes its new state from the old state of the block N_{ay} where $a \in \{0,1\}$. Therefore neurons in N_{ay} are connected to neurons of N_{yb} and the corresponding edges are labeled with relevant weights with respect to the original weight function w of N. The block N_{yb} has the constant input b which is taken into account by modifying all thresholds of neurons in this block, especially when $b = 1$.

The input bit $c \in \{0,1\}$ of $N^\star$ indicates that the computations of N_{ay}, where $a \neq c$, are not valid. Therefore the input neuron $inp^\star$ cancels these computations by setting all neuron states in N_{ay} to zero. Thus, all neurons in 2^{k-1} blocks N_{ay} ($y \in \{0,1\}^{k-1}$) have zero states when the previous input bit does not match a. This also enables the block N_{yb} to execute the correct computation because the influence of one invalid block from either N_{0y} or N_{1y} which are connected to N_{yb}, is suppressed, due to zero states.

The neurons in the blocks N_x which are connected to out_x lead to the output neuron $out^\star$ as well. They are labeled with the same weights. We know that half of them do not have any influence on $out^\star$. Moreover, among the remaining ones, the corresponding neurons v_x ($v \in V$) from all blocks N_x ($x \in \{0,1\}^k$) have the same state because the distance between *inp* and *out* in N is at least $k+1$ and the last k input bits cannot influence the output. It is sufficient to multiply the threshold of $out^\star$ by 2^{k-1} in order to preserve the function computed by *out*. In this way, the correct recognition is accomplished with $k-1$ step lookahead.

A formal definition of the neuromaton $N^\star = (V^\star, inp^\star, out^\star, E^\star, w^\star, h^\star, s^\star_{init})$ follows:

$$V^\star = \left\{ v_x \mid v \in V - \{inp\}, x \in \{0,1\}^k \right\} \cup \{inp^\star, out^\star\},$$

$$\begin{aligned} E^\star = \; & \left\{ \langle u_{ay}, v_{yb} \rangle \mid \langle u, v \rangle \in E; y \in \{0,1\}^{k-1}; a, b \in \{0,1\} \right\} \cup \\ & \left\{ \langle inp^\star, v_x \rangle \mid v \in V, x \in \{0,1\}^k \right\} \cup \left\{ \langle v_x, out^\star \rangle \mid \langle v, out \rangle \in E, x \in \{0,1\}^k \right\}, \end{aligned}$$

$$\begin{aligned} w^\star(\langle u_{ay}, v_{yb} \rangle) &= w(\langle u, v \rangle) \quad y \in \{0,1\}^{k-1}; a, b \in \{0,1\}, \\ w^\star(\langle inp^\star, v_{0y} \rangle) &= h(v) - 1 - \sum_{u \in V : w(\langle u, v \rangle) > 0} w(\langle u, v \rangle), \\ w^\star(\langle inp^\star, v_{1y} \rangle) &= -w^\star(\langle inp^\star, v_{0y} \rangle) \quad y \in \{0,1\}^{k-1}, \\ w^\star(\langle v_x, out^\star \rangle) &= w(\langle v, out \rangle) \quad x \in \{0,1\}^k, \end{aligned}$$

$$\begin{aligned} h^\star(v_{ayb}) &= h(v) + aw^\star(\langle inp^\star, v_{ayb} \rangle) - bw(\langle inp, v \rangle) \quad y \in \{0,1\}^{k-2}; a, b \in \{0,1\}, \\ h^\star(out^\star) &= 2^{k-1} h(out), \end{aligned}$$

$$s^\star_{init}(v_{ya}) = a s_{k-1}(v)(y) \quad y \in \{0,1\}^{k-1}, a \in \{0,1\}$$

where $s_{k-1}(v)(y)$ is the state of neuron v in the neuromaton N for the input $y \in \{0,1\}^{k-1}$ at the time step $k-1$.

The term $bw(\langle inp, v \rangle)$ in the threshold definition takes into account the weight of the constant input in the block N_{y1} ($y \in \{0,1\}^{k-1}$) corresponding to the original weight associated with the edge leading from the input *inp* to the relevant neuron v in the neuromaton N. The definition of $w^\star(\langle inp^\star, v_{0y} \rangle)$ ensures setting the states of all neurons in the block N_{0y} ($y \in \{0,1\}^{k-1}$) to zero iff the input $inp^\star$ is 1. Similarly the definition of weights $w^\star(\langle inp^\star, v_{1y} \rangle)$ together with the term $aw^\star(\langle inp^\star, v_{ayb} \rangle)$ in the threshold definition cause zero states of all neurons in the block N_{1y} ($y \in \{0,1\}^{k-1}$) iff the input $inp^\star$ is 0.

Clearly, the size of the neuromaton $N^\star$ is $2^k(n-1)+2$. □

Following Lemma 1, we can restrict ourselves to neuromata which respond only one computational time step after the input string is read because their size is, up to a constant multiplication factor, as large as the size of the equivalent neuromata with a constant time delay. Therefore, in the rest of this paper we will assume that the time delay in the neuromaton recognition is 1 and that any neuromaton architecture does not contain an edge from the input to the output neuron. We will also denote $L_1(N)$ by $L(N)$ for any neuromaton N.

Next, we will prove that the neural acceptor can be viewed as a finite automaton [12] and therefore, neuromata recognize exactly regular languages due to Theorem 1.

Theorem 2 *Let $L = L(N)$ be a language recognized by some neuromaton 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 $L(N) = L(A)$. Put $Q = \{s : V - \{inp\} \longrightarrow \Sigma\}$ and $q_0 = s_{init}$. 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, $F = \{s \in Q \mid \delta(s,x)(out) = 1,\, x \in \Sigma\}$. Then the proposition follows from Theorem 1. $\square$

4 Upper Bound

We show that any regular language, given by a regular expression of length n, may be recognized by a neuromaton of size $O(n)$. The idea of recognition by neuromaton is to compare an input string with all possible strings generated by the regular expression and to report via the output neuron whether some of these strings match the input. Therefore, the constructed architecture of the neural network corresponds to the structure of the regular expression. The neuromaton proceeds through all oriented network paths that correspond to all strings generated by this expression and that, at the same time, match the part of the input string that has been read so far.

Theorem 3 *For every regular language $L \in RL$ denoted by a regular expression $L = [\alpha]$ there exists a neuromaton N of the size $O(|\alpha|)$ such that L is recognized by N (i.e., $L = L(N)$).*

Proof: Let $L = [\alpha]$ be a regular language denoted by a regular expression α. We construct a neuromaton $N_\alpha = (V, inp, out, E, w, h, s_{init})$ 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 notational 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 already been 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 rigor and therefore, we can identify one of the new vertices with the old β. That is why we write rather inexactly, 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: $V_{k+1} = V_k - \{\beta\}$,
     $$E_{k+1} = \begin{array}{l}(E_k - \{\langle x,\beta\rangle, \langle\beta, y\rangle \in E_k\}) \cup \\ \{\langle x, y\rangle \mid \langle x,\beta\rangle, \langle \beta, y\rangle \in E_k - \{\langle\beta,\beta\rangle\}\}.\end{array}$$
   - 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 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 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 $\langle j, out \rangle \in E_p$, $h_{out} = 1$.

The initial state is defined as $s_0(i) = 0$ 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 neuromaton for the regular language $[(1(0 + 0(1 + 0)))^\star]$ is in figure 1 (the types of neurons are depicted inside the circles representing neurons; thresholds are depicted as weights of edges with constant inputs -1).

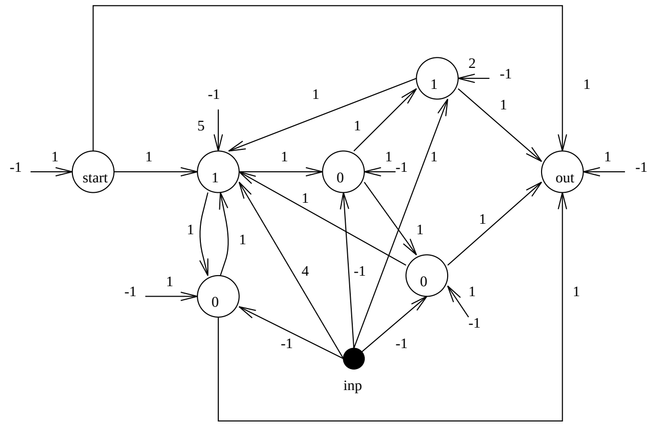

Figure 1: Neuromaton for $[(1(0 + 0(1 + 0)))^\star]$.

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 their types with the network input and become active only when they match, 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 $L = L(N_\alpha)$. □

5 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 regular languages which are described by regular expressions of the length n. As a consequence, it follows that the construction of the neuromaton from section 4 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 an exponential number of prefixes for this language. We prove that these prefixes must bring any neuromaton to an exponential number of different states in order to provide a correct recognition. This will imply the desired lower bound.

Definition 9 *For $n \geq 1$ and $k \geq 0$ we denote by L_n, Π_k, and P_n, respectively, the following regular languages:*

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

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

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 9 is in fact of $O\left(n^2\right)$ length because the abbreviation for a 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 2

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

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

Proof:

(i) In the regular expression which denotes the language L_n from Definition 9, we can subsequently factor out $(n-3)$ times the subexpression $1(e+0)$ to obtain the desired regular expression

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

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

(ii) It follows from Definition 9 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 3

(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 9.

(iii) Assume $x = x'1 \in L_n$. The language L_n is defined via iteration in Definition 9. Henceforth, 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 strings is in L_n while the other one is not.

Lemma 4 $\forall x_1, x_2 \in P_n\ \ x_1 \neq x_2 \longrightarrow \exists y \in \{0,1\}^\star\ \ x_1 y \in L_n$ *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 3 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 3.

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 3 $z_1 0 \in L_n$ and $z_2 1 \in L_n$ from (ii) Lemma 3. This implies that $x_1 y = z_1 0 z_2 1 \in L_n$ because L_n is closed under 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 3 $z_1 \in L_n$, $z_2 \in \Pi_{n-1}$ 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 3 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 4 *Any neuromaton N that recognizes the language $L_n = L(N)$ requires at least $\Omega(n)$ neurons.*

Proof: The neuromaton N, that recognizes the language $L_n = L(N)$ from Definition 9, 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 4, so that $x_1 y \in L_n$ and $x_2 y \notin L_n$. This implies that N needs $\Omega(n)$ binary neurons. □

6 Neuromata Are Stronger

Although from the results of sections 4, 5 it seems that from the descriptional complexity point of view, neuromata and regular expressions are polynomially equivalent, in this section, we will show that there exists a neuromaton for which every equivalent regular expression is of an exponential length. This means that the neuromata construction from regular expressions that has been described in section 4 is not efficiently reversible.

Theorem 5 *For every $n \geq 1$ there exists a regular language L_n recognized by a neuromaton N_n of size $O(n)$ and of descriptional complexity $O(n^2)$ such that any regular expression α_n which defines $L_n = [\alpha_n] = L(N_n)$ is of length $|\alpha_n| = \Omega(2^n)$.*

Proof: Let $m = 2^n$. We define the finite language $L_n = \{0,1\}^m$ to be the set of all binary strings of length m.

The neuromaton $N_n = (V, inp, v_{n-2}, E, w, h, s_{init})$ which recognizes L_n is a binary n-bit counter which accepts the input string iff its length is m. The neuron v_i corresponding to the ith bit of the counter should change its state iff $v_0 = v_1 = \cdots = v_{i-1} = 1$. However, the corresponding Boolean function cannot be computed by only one neuron and, therefore, a small subnetwork of three neurons a_i, b_i, v_i is introduced for this purpose. Then v_i is a disjunction of a_i and b_i where $a_i = 1$ iff $v_i = 0$ and $v_0 = v_1 = \cdots = v_{i-1} = 1$ and similarly $b_i = 1$ iff $v_i = 1$ and at least one of $v_0, v_1, \ldots, v_{i-1}$ is 0. Clearly, v_i changes its state as required but it takes two computational steps. This means that the counter is two times slower and only $n-1$ bits are required to count till m. Moreover the neuron v_0 generates the binary sequence $(0011)^\star$ and a special neuron rst is introduced to suppress the firing of the output neuron v_{n-2} after m bits are read.

A formal definition of the counter follows:

$$V = \{a_i, b_i, v_i \mid i = 1, \ldots, n-2\} \cup \{inp, a_0, v_0, rst\},$$

$$\begin{aligned} E = \; & \{\langle v_j, a_i\rangle, \langle v_j, b_i\rangle \mid 0 \leq j \leq i; i = 1, \ldots, n-2\} \cup \\ & \{\langle a_i, v_i\rangle, \langle b_i, a_i\rangle \mid i = 1, \ldots, n-2\} \cup \\ & \{\langle v_0, a_0\rangle, \langle a_0, v_0\rangle, \langle a_{n-2}, rst\rangle, \langle b_{n-2}, rst\rangle, \langle rst, rst\rangle, \langle rst, v_{n-2}\rangle\}, \end{aligned}$$

$$w(\langle v_j, a_i\rangle) = -w(\langle v_j, b_i\rangle) = 1 \quad 0 \leq j < i; \quad i = 1, \ldots, n-2,$$

$$w(\langle v_i, b_i\rangle) = -w(\langle v_i, a_i\rangle) = i \quad w(\langle a_i, v_i\rangle) = w(\langle b_i, v_i\rangle) = 1 \quad i = 1, \ldots, n-2,$$

$$w(\langle a_0, v_0\rangle) = -w(\langle v_0, a_0\rangle) = 1,$$

$$w(\langle a_{n-2}, rst\rangle) = w(\langle b_{n-2}, rst\rangle) = w(\langle rst, rst\rangle) = -w(\langle rst, v_{n-2}\rangle) = 1,$$

$$h(v_i) = 1, \quad h(a_i) = i \quad i = 0, \ldots, n-2,$$

$$h(b_i) = 1 \quad i = 1, \ldots, n-2, \quad h(rst) = 1,$$

$$s_{init}(v) = 0 \quad v \in V - \{inp\}.$$

Clearly, the size of N_n is of order $O(n)$ and its descriptional complexity is $O(n^2)$.

Let α_n be a minimal length regular expression which defines the language $L_n = [\alpha_n] = L(N_n)$. We prove that $|\alpha_n| = \Omega(m)$. The expression α does not contain iterations because it defines the finite language (the part containing iterations would denote the empty language and, thus, could be omitted). To generate strings from $[\alpha_n]$ the expression α_n is to be read from the left to the right side without any returning since no iteration is allowed. But any string of L_n is of length m and that is why α_n must contain at least m symbols from $\{0,1\}$. Hence $|\alpha_n| = \Omega(m)$. □

Theorem 5 shows that, in some sense, there is a big gap between the descriptive power of regular expressions and that of neuromata. We will discuss this issue later in sections 10, 11.

7 Neural String Acceptors

The previous results showed that neuromata have the descriptive capabilities of regular expressions. In this section we will study how powerful they are when we confine ourselves to a certain subclass of regular expressions. Here, we will deal with the simplest case only considering fixed binary strings from $\{0,1\}^\star$. We present two constructions of neural acceptors for a single string recognition. For n–bit strings 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 their descriptional complexity point of view.

Studying this elementary case is useful because single string recognition is very often a part of more complicated tasks. The techniques developed for the architectural neural network design can, for example, sometimes improve the construction of neuromata from section 4, when a regular expression consists of long binary substrings. The other example of application is a constructive neural learning, where strings are viewed as training patterns, and the resulting network is composed of neural acceptors for these strings that work in parallel.

Theorem 6 *For any string $a = a_1 \ldots a_n \in \{0,1\}^n$ 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 neuromaton 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 $c_1, \ldots, c_p$. The input string $x = x_1 \ldots x_n \in \{0,1\}^n$ is being gradually stored per p bits into p buffer neurons $b_1, \ldots, b_p$. 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 $s_1, \ldots, s_p$ tick at each time step of the network computation, and neurons $m_1, \ldots, m_p$ 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.

A formal definition of the neuromaton $N = (V, inp, out, E, w, h, s_{init})$ for the recognition of the string a follows (see also figure 2). Define

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

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

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

$$\begin{array}{l} w\left(\langle b_j, c_i\rangle\right) = 2a_{(i-1)p+j} - 1, \quad w\left(\langle m_j, c_i\rangle\right) = \left\{\begin{array}{ll} 1 & \text{if } j \leq i \\ -1 & \text{if } j > i \end{array}\right. \qquad 1 \leq i,j \leq p \\ w\left(\langle c_i, c_i\rangle\right) = 2p - d_i - i + 1 \quad i = 1,\ldots,p-1 \\ w\left(\langle m_p, s_2\rangle\right) = -1, \quad w\left(\langle m_p, m_i\rangle\right) = -1 \quad i = 1,\ldots,p-1 \\ h\left(c_i\right) = \left\{\begin{array}{ll} d_i + i + 1 & \text{if } i = 1 \\ d_i + i + 2 & \text{if } 2 \leq i \leq p \end{array}\right. \qquad h\left(m_i\right) = \left\{\begin{array}{ll} 1 & \text{if } i = 1 \\ 2 & \text{if } 2 \leq i \leq p. \end{array}\right. \end{array}$$

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 $s_0(start) = 1$.

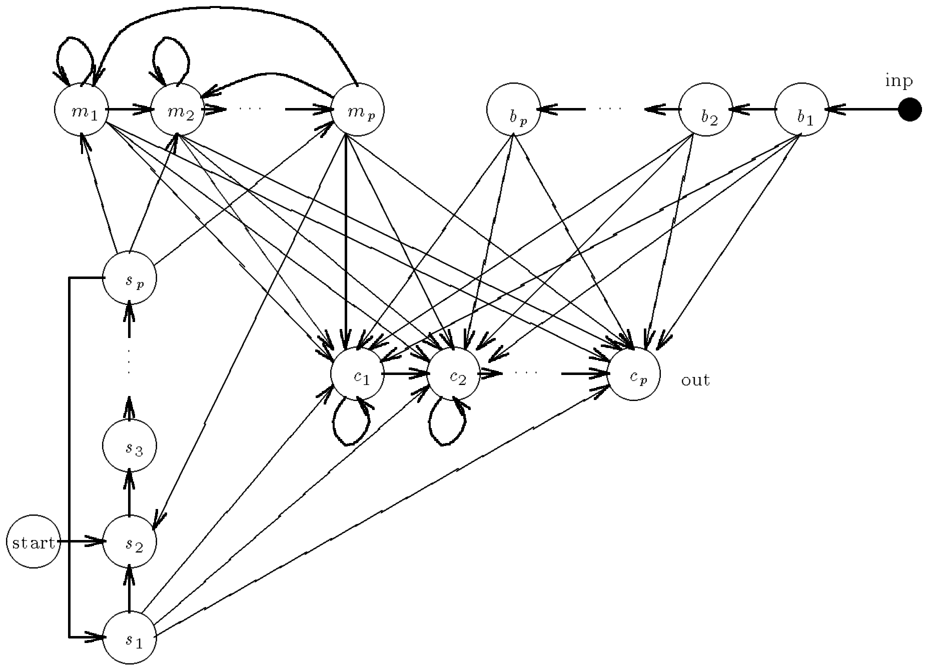

Figure 2: Architecture for a neural string acceptor with $O(n)$ edges.

Note that the weights $w\left(\langle c_i, c_i\rangle\right) = O(p)$, $i = 1,\ldots,p-1$ 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 of these neurons have a constant feedback because they only have one input, from the previous comparator neuron, to be exceeded.

The construction of the neural acceptor can also be easily adapted for $n = p^2 + r$, $1 \leq r < p$. The technique from the proof of theorem 3 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 $out = c_p$. □

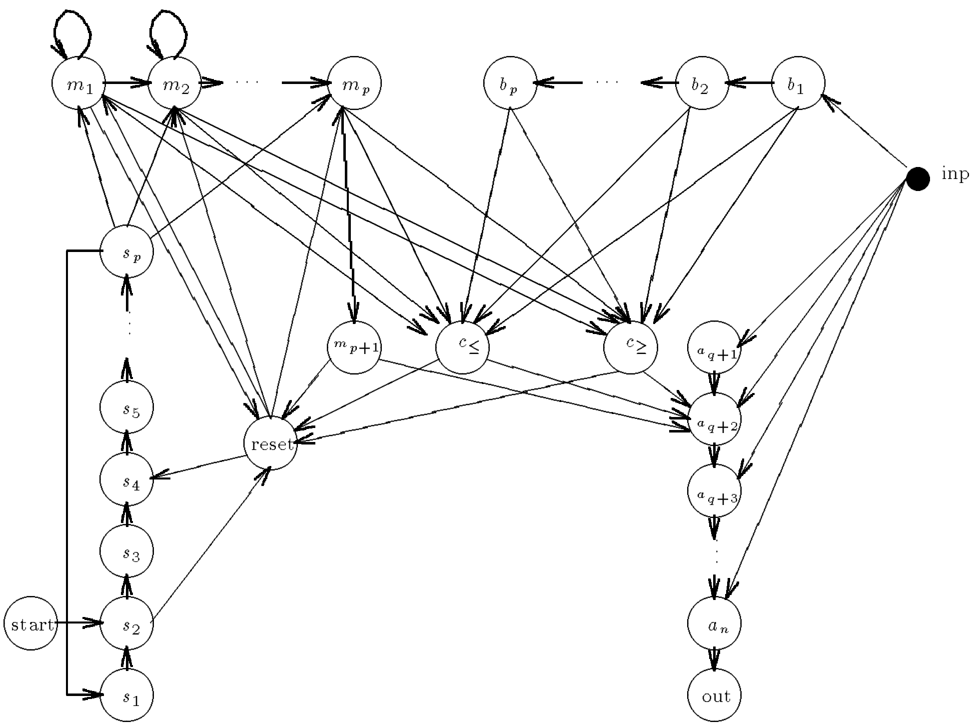

Figure 3: Architecture for a neural string acceptor with $O(n^{\frac{1}{2}})$ edges.

Theorem 7 *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 neuromaton is $\Theta(n)$.*

Proof: The design of the desired neural acceptor is very similar to the construction from the proof of Theorem 6, except that the number of comparator neurons is reduced to two neurons $c_\leq, c_\geq$. In this case the substrings $a_{(i-1)p+1}, \cdots, a_{ip}$, $1 \leq i \leq p$ 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 conjunction 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. 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 only give the definition of the weights that are relevant for the comparisons:

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

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

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

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 just 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 3 (instead of the above-mentioned conjunction, the neuron *reset* is added to realize the negation of comparator conjunction to possibly terminate the clock). □

Piotr Indyk [9] pointed out that the latter string acceptor construction from Theorem 7 can also be exploited for building a cyclic neural network, with $O(2^{\frac{n}{2}})$ neurons and edges, which computes any boolean function. In this case the binary vector of all function values is encoded into the string acceptor. The position of the relevant $2^{\frac{n}{2}}$-bit part of this vector which includes the desired output is given by the first $\frac{n}{2}$ bits of input. All possible corresponding $2^{\frac{n}{2}}$-bit substrings are generated and presented to the acceptor to achieve the relevant part of the function value vector from which the relevant bit is extracted. Its position is determined from the last $\frac{n}{2}$ bits of input.

8 Hopfield Languages

In section 7 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 neuromata which are based on symmetric neural networks (Hopfield networks). In these networks the weights are symmetric and therefore, the architecture of such neuromata can be seen as an undirected graph.

Hopfield networks have been traditionally studied [4, 21] and used, due to their favorite 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 neuromata, we will define a class of Hopfield languages that are recognized by these particular acceptors.

Definition 10 *The neuromaton $N = (V, inp, out, E, w, h, s_{init})$ 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 a* Hopfield acceptor *(*Hopfield neuromaton*). The language $L = L(N)$ recognized by Hopfield neuromaton N is called* Hopfield language*.*

First, we will show that the class of Hopfield languages is strictly contained within the class of regular languages. For this purpose we will formulate the necessary 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 the Hopfield neuromaton to converge and to forget relevant information about the previous part of the input string which is recognized. The idea of the proof is to find a necessary condition to prevent a Hopfield neuromaton from converging.

Definition 11 *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 $(\forall m \geq m_0 \;\; v_1 x^m v_2 \notin L)$.*

Theorem 8 *Every Hopfield language satisfies the Hopfield condition.*

Proof: Let $L = L(N)$ be a Hopfield language recognized by the Hopfield neuromaton $N = (V, inp, out, E, w, h, s_{init})$. Denote $V - \{inp\} = \{1, \ldots, n\}$ and define

$w_{ij} = 0$ for $\langle j, i\rangle \notin E$. Let $\vec{w}_0 = (w_{1,inp}, \ldots, w_{n,inp})$, $\vec{h} = (h_1, \ldots, h_n)$ be the integer vectors of size $n \times 1$ and let $W = (w_{ij})_{i,j \in V-\{inp\}}$ be the integer matrix of size $n \times n$. Note that the matrix W is symmetric, since N is the Hopfield acceptor.

Let us present an input string $v_1 x^{m_0}$, $m_0 \geq 1$, with $v_1 \in \{0,1\}^\star$ and $x \in \{0,1\}^2$ to the acceptor N. For a sufficiently large m_0, the network's computation must start cycling over this input because the network has only 2^{n+1} of possible states. Let $s_1, \ldots, s_p$ be the different states in this cycle and $x_1 = s_1(inp), \ldots, x_p = s_p(inp)$ be the corresponding input bits, so that the state s_i is followed by the state $s_{\pi(i)}$, where $\pi : \{1, \ldots, p\} \longrightarrow \{1, \ldots, p\}$ is the index shifting permutation: $\pi(i) = i+1$ for $1 \leq i < p$ and $\pi(p) = 1$. Let $\sigma = \pi^{-1}$ be the inverse permutation of π, and let π^r be the composed permutations, for any $r \geq 1$. Further, let $\vec{c}_i = (s_i(1), \ldots, s_i(n))$, $i = 1, \ldots, p$ be the binary vectors of size $n \times 1$. Then $(\vec{w}_0 x_i + W\vec{c}_i - \vec{h})_j \geq 0 \longleftrightarrow c_{\pi(i)j} = 1$ for $j = 1, \ldots, n$ and $1 \leq i \leq p$ follows from Definition 8.

For each state of the cycle we define an integer

$$E_i = \vec{c}_i^T \vec{w}_0 x_{\sigma(i)} + \vec{c}_{\pi(i)}^T \vec{w}_0 x_i + \frac{1}{2}\left(\vec{c}_i^T W \vec{c}_{\pi(i)} + \vec{c}_{\pi(i)}^T W c_i\right) - \left(\vec{c}_i^T + \vec{c}_{\pi(i)}^T\right)\vec{h}$$

where $1 \leq i \leq p$ and the symbol ^{T} denotes the transposition of a vector. Obviously, $E_{\pi^{rp}(i)} = E_i$ for $r \geq 1$ and $1 \leq i \leq p$. Using the fact that the matrix W is symmetric we obtain

$$E_{\pi(i)} - E_i = \vec{c}_{\pi^2(i)}^T \vec{w}_0 x_{\pi(i)} - \vec{c}_i^T \vec{w}_0 x_{\sigma(i)} + \left(\vec{c}_{\pi^2(i)}^T - \vec{c}_i^T\right)\left(W\vec{c}_{\pi(i)}^T - h\right) \quad 1 \leq i \leq p.$$

Moreover, $x_{\pi(i)} = x_{\sigma(i)}$ for $i = 1, \ldots, p$ because $|x| = 2$. So we can write

$$E_{\pi(i)} - E_i = \left(\vec{c}_{\pi^2(i)}^T - \vec{c}_i^T\right)\left(\vec{w}_0 x_{\pi(i)} + W\vec{c}_{\pi(i)}^T - h\right) \quad i = 1, \ldots, p.$$

We know that $\left(\vec{w}_0 x_{\pi(i)} + W\vec{c}_{\pi(i)}^T - h\right)_j \geq 0 \longleftrightarrow c_{\pi^2(i)j} = 1$, $1 \leq j \leq n$. Therefore, $\left(\vec{c}_{\pi^2(i)}^T - \vec{c}_i^T\right)_j \left(\vec{w}_0 x_{\pi(i)} + W\vec{c}_{\pi(i)}^T - h\right)_j \geq 0$ for $j = 1, \ldots, n$ which implies $E_{\pi(i)} - E_i \geq 0$ for $i = 1, \ldots, p$. But from $E_{\pi^p(i)} = E_i$ it follows that $E_{\pi(i)} - E_i = 0$ for $i = 1, \ldots, p$ and hence, $\left(\vec{c}_{\pi^2(i)}^T - \vec{c}_i^T\right)_j \left(\vec{w}_0 x_{\pi(i)} + W\vec{c}_{\pi(i)}^T - h\right)_j = 0$.

Then we cannot have both $c_{\pi^2(i)j} = 0$ and $c_{ij} = 1$, $1 \leq j \leq n$, at the same time because in this case the inequality of $\left(\vec{c}_{\pi^2(i)}^T - \vec{c}_i^T\right)_j \left(\vec{w}_0 x_{\pi(i)} + W\vec{c}_{\pi(i)}^T - h\right)_j > 0$ is strict. The complementary case of $c_{\pi^2(i)j} = 1$ and $c_{ij} = 0$ simultaneously is impossible as well. Since then, the number of 1's in $\vec{c}_{\pi^2(i)}$ would be greater than the number of 1's in $\vec{c}_i$ for $i = 1, \ldots, p$, which contradicts $c_{\pi^p(i)} = c_i$.

Therefore, we can conclude that $\vec{c}_{\pi^2(i)} = \vec{c}_i$ for $i = 1, \ldots, p$ and consequently $\pi^2(i) = i$. This implies that the cycle length $p \leq 2$. Hence, for every $v_2 \in \{0,1\}^\star$ either $(\forall m \geq m_0 \;\; v_1 x^m v_2 \in L)$ or $(\forall m \geq m_0 \;\; v_1 x^m v_2 \notin L)$. This completes the proof that L satisfies the Hopfield condition. $\square$

For example, it follows from Theorem 8 that the regular languages $[(000)^\star]$, $[(1010)^\star]$ are not Hopfield languages because they do not satisfy the Hopfield condition.

9 The Hopfield Condition Is Sufficient

In this section we will prove that the necessary Hopfield condition from Definition 11, stating when a regular language is a Hopfield language, is sufficient as well. A construction of a Hopfield neuromaton is shown for a regular language satisfying the Hopfield condition.

Theorem 9 *For every regular language $L = [\alpha]$ satisfying the Hopfield condition there exists a Hopfield neuromaton N of the size $O(|\alpha|)$ such that L is recognized by N. Hence, L is a Hopfield language.*

Proof: The architecture of the Hopfield neuromaton for the regular language $L = [\alpha]$ satisfying the Hopfield condition is given by the general construction from the proof of Theorem 3. As a result we obtain an oriented network N' that corresponds to the structure of the regular expression α where each neuron n_s of N' (besides the special neurons *inp*, *out*, *start*) is associated with one symbol $s \in \{0,1\}$ from α (i.e., is of the type s). The task of n_s is to check whether s agrees with the input bit.

We will transform N' into an equivalent Hopfield network N. Supposing that α contains iterations of binary substrings with at most two bits, the standard technique [15, 21] of the transformation of acyclic neural networks to Hopfield networks can be employed. The idea consists in adjusting weights to prevent propagating a signal backwards while preserving the original function of neurons. The transformation starts in the neuron *out*, it is carried out in the opposite direction to oriented edges and ends up in the neuron *start*. For a neuron whose outgoing weights have been already adjusted, its threshold and incoming weights are multiplied by a sufficiently large integer which exceeds the sum of absolute values of outgoing weights. This is sufficient to suppress the influence of outgoing edges on the neuron. After the transformation is accomplished all oriented paths leading from *start* to *out* are labeled by decreasing sequences in weights.

The problem lies in realizing general iterations using only the symmetric weights. Consider a subnetwork I of N' corresponding to the iteration of some substring of α. Let the subnetwork I have arisen from a subexpression β^+ of α in the proof of theorem 3. After the above-mentioned transformation is performed, any path leading from the incoming edges of I to the outgoing ones is labeled by a decreasing sequence of weights in order to avoid the backward signal from spreading. But the signal should be propagated from any output of the subnetwork I back to each subnetwork input, as the iteration requires. On one hand, an integer weight associated with such a connection should be small enough in order to suppress the backward signal propagation. On the other hand, this weight should be sufficiently large enough to influence the subnetwork input neuron. Clearly, these two requirements are contradictory.

Consider a simple cycle C in the subnetwork I consisting of an oriented path passing through I and of one backward edge leading from the end of this path (i.e., the output of I) to its beginning (i.e., the input of I). Let the types of neurons in the cycle C establish an iteration a^+ where $a \in \{0,1\}^\star$ and $|a| > 2$. Moreover, suppose that $x \in \{0,1\}^2$ and $a = x^k$ for some $k \geq 2$. In the Hopfield condition set v_2 to be a postfix of L associated with a path leading from C to *out*. Similarly set v_1 to be a prefix of L associated with a path leading from *start* to C such that for every $m \geq 0$ it holds $v_1 x^{mk} v_2 \in L$ and $v_1 x^{mk+1} v_2 \notin L$ which contradicts the Hopfield condition. Such prefix v_1 exists because, otherwise, the cycle C could be realized as an iteration of two bits. Therefore $a \neq x^k$. This implies that strings a^i, $i \geq 2$ contain a substring of the form $by\bar{b}$ where $b, y, \bar{b} \in \{0,1\}$ and $b \neq \bar{b}$. Hence, the string a has the form either $a = a_1 by\bar{b} a_2$, $a_1, a_2 \in \{0,1\}^\star$, or $a = y\bar{b}a_2 b$ (e.g., $a = 10101$). For notational simplicity we confine ourselves to the former case while the latter remains similar. Furthermore, we consider $a = a_1 by\bar{b}a_2$ with a minimal $|a_2|$.

We shift the decreasing sequence of weights in C to start and to end in the neuron n_y while relevant weights in N' are modified by the above-mentioned procedure to ensure a consistent linkage of C within N'. For example, this means that all edges leading from the output neuron of I in C to the input neurons of I are evaluated by sufficiently large weights to realize the corresponding iterations. Now the problem lies in signal propagation from the neuron n_b to the neuron n_y. Assume $b = 1$. To support the small weight in the connection between n_b and n_y a new neuron *id* that copies the state of the input neuron *inp* is connected to the neuron n_y via a sufficiently large weight which strengthens the small weight in the connection from n_b. Obviously, both the neuron n_b ($b = 1$) and the new neuron *id* are active at the same time and both enable the required signal propagation from n_b to n_y together. On the other hand, when the neuron $n_{\bar{b}}$ ($\bar{b} = 0$) is active, the neuron *id* is passive due to the fact that it is copying the input. This prevents the neuron n_y from becoming active at that time.

However, for some symbols $b' = b \in \{0, 1\}$ in α there can be neurons $n_{b'}$ outside the cycle C (but within the subnetwork I) to which the edges from n_y lead. This situation corresponds to y being concatenated with a union operation within β. In this case the active neurons $n_{b'}$, *id* would cause the neuron n_y to fire. To avoid this, we add another new neuron $n_{y'}$ that behaves identically as the neuron n_y for the symbol $y' = y \in \{0, 1\}$. Thus, the same neurons that are connected to n_y are linked to $n_{y'}$ and the edges originally outgoing from n_y to $n_{b'}$ for all corresponding b' are reconnected to lead only from $n_{y'}$.

A similar approach is used in the opposite case when $b = 0$. The above-described procedure is applied for each simple cycle C in the subnetwork I corresponding to the iteration β^+. These cycles are not necessarily disjoint, but the decomposition of $a = a_1 b y \bar{b} a_2$ with minimal $|a_2|$ ensures their consistent synthesis. Similarly, the whole transformation process is performed for each iteration in α. In this case some iteration can be a part of another iteration and the magnitude of weights in the inner iteration will need to be accommodated to embody this into the outer iteration. It is also possible that the neuron *id* has to support both iterations in the same point. Finally, the number of simple cycles in α is $O(|\alpha|)$. Hence, the size of the resulting Hopfield neuromaton remains of order $O(|\alpha|)$.

In figure 4 the preceding construction is illustrated by an example of the Hopfield neuromaton for the regular language $[(1(0 + 0(1 + 0)))^\star]$ (compare with figure 1). A simple cycle consisting of neurons $n_b, n_y, n_{\bar{b}}$ is clarified here in details. Notice the decreasing sequence of weights (7,5,1) in this cycle, starting and ending in the neuron n_y, as well as, the neuron *id* which enables the signal propagation from n_b to n_y. The neuron $n_{y'}$, identical with n_y, has also been created because the neuron $n_{b'}$ ($b' = b$) was originally connected to n_y (see figure 1). □

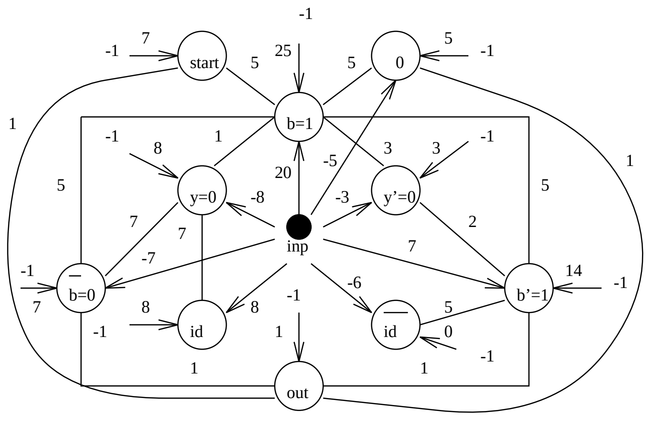

Figure 4: Hopfield neuromaton for $[(1(0 + 0(1 + 0)))^\star]$.

Corollary 1 *Let L be a regular language. Then L is a Hopfield language iff L satisfies the Hopfield condition.*

Finally, we will briefly investigate the closure properties of the class of Hopfield languages.

Theorem 10 *The class of Hopfield languages is closed under union, concatenation, intersection, and complement. It is not closed under iteration.*

Proof: The closeness of Hopfield languages under union and concatenation follows from Corollary 1. To obtain the Hopfield neuromaton for the complement, negate the function of the output neuron *out* by multiplying the associated weights (and the threshold) by -2 and by adding 1 to the threshold. Hence, the Hopfield languages are closed under intersection as well. Finally, due to Theorem 9, [1010] is a Hopfield language, whereas [(1010)*] is not a Hopfield language because it does not satisfy the Hopfield condition from Theorem 8. $\square$

10 Emptiness problem

In order to further illustrate the descriptional power of neuromata we investigate the complexity of the emptiness problem for regular languages given by neuromata or by Hopfield acceptors. We will prove that both problems are PSPACE-complete.

Definition 12 *Given a (Hopfield) neuromaton N, the* (Hopfield) neuromaton emptiness problem*, which is denoted* NEP *(*HNEP*), is the issue of deciding whether the language $L = L(N)$ recognized by the (Hopfield) neuromaton N is nonempty.*

Theorem 11 *NEP, HNEP are PSPACE-complete.*

Proof: To show that both $NEP, HNEP \in PSPACE$, an input string for the (Hopfield) neuromaton N is being guessed bit by bit and its acceptance is checked by simulating the network computation in a polynomial space to witness the nonemptiness of $L = L(N)$.

Next, we show that NEP is PSPACE-hard. Let A be an arbitrary language in PSPACE. For each $x \in \{0,1\}^\star$ we will, in a polynomial time, construct the corresponding neuromaton N such that $x \in A$ iff $N \in NEP$. Further, let M be a polynomial space bounded Turing machine which recognizes A. First, a cyclic neural network N' which simulates M can be constructed in a polynomial time using the standard technique [4]. The idea of this construction is that for each tape cell of M there is a subnetwork which simulates a tape head when it is in this position during the computation (i.e., the local transition rule). The neighbor subnetworks are connected to enable the head moves. The input x for M is encoded in the initial state of N' and at the end of the neural network computation one neuron of N', called *result*, signals whether $x \in A$. The neural network N' is embodied into the neuromaton N as follows. The input neuron *inp* of N is not connected to any other neuron and the output neuron *out* of N is identified with the neuron *result* of N'.

Now, $x \in A$ iff M accepts x iff the neuron *out* is active at the end of the simulation iff $L(N)$ contains all words of the length which is equal to the length of the computation of M on x iff $N \in NEP$. Thus, $x \in A$ iff $N \in NEP$. This completes the proof that NEP is PSPACE-complete.

For the Hopfield neuromata a similar simulation of M can be achieved using the symmetric neural network N'. It is because any convergent computation of an arbitrary asymmetric neural network can be simulated by a symmetric network of a polynomial size [4] and we can assume, without loss of generality, that M stops for every input. Hence, HNEP is PSPACE-complete as well. $\square$

Theorem 11 is a somewhat surprising because the identical problems for regular expressions, deterministic and non-deterministic finite automata are known to only be NL-complete [10, 11]. In some sense this shows that there is a big gap between the descriptive power of regular expressions and that of neuromata and confirms the result from section 6. We will discuss this issue later in section 11.

The difference between the descriptive power of regular expressions and of neuromata can also be illustrated by the complement operation. While the emptiness problem for the complement of the regular expression becomes PSPACE-complete [1] the emptiness problem complexity of the complement of a neuromaton does not change because the output neuron can be easily negated.

As a next consequence we will show that the neuromaton equivalence problem is PSPACE-complete as well.

Definition 13 *Given two (Hopfield) neuromata N_1, N_2 the* (Hopfield) neuromaton equivalence problem *which is denoted* NEQP *(*HNEQP*) is the issue of deciding whether the languages recognized by these (Hopfield) neuromata are the same, i.e. whether $L(N_1) = L(N_2)$.*

Corollary 2 *NEQP, HNEQP are PSPACE-complete.*

Proof: To prove that $NEQP \in PSPACE = NPSPACE$ it is sufficient to show that its complement is in $PSPACE$. For this purpose an input string for the neuromata N_1, N_2 is being guessed and its acceptance for one of these two neuromata and its rejection for the other one are checked by simulating both network computations in a polynomial space to witness the non-equivalence $L(N_1) \neq L(N_2)$.

To show that NEQP is PSPACE-hard, the complement of NEP denoted co-NEP, which is PSPACE-complete as well, is in a polynomial time reduced to NEQP. Let the neuromaton N be an instance of co-NEP. We will in a polynomial time construct the corresponding instance N_1, N_2 for NEQP so that $L(N)$ is empty iff $L(N_1) = L(N_2)$. The neuromaton N_1 is identified with N. Let N_2 be an arbitrary small neuromaton which recognizes the empty language. It is easy to see that this is a correct reduction. Hence, NEQP is PSPACE-complete. The PSPACE-completeness of HNEQP can be achieved similarly. □

11 Conclusions

In this paper we have explored an alternative formalism for the regular language representation, based on neural networks. Comparing the so-called neuromata with the classical regular expressions we have obtained the result that within a polynomial descriptive complexity, the non-determinism, which is captured in the regular expressions, can be simulated in a parallel mode by neuromata which have a deterministic nature.

In our opinion, the descriptive power of the neuromata consists in an efficient encoding of the transition function. While the transition function of the deterministic (nondeterministic) finite automata is usually specified by the list of its values *(old state, input symbol)/new state*, the same function in the neuromata is given by the vector of formulae — each for one neuron which evaluates it. This encoding can be interpreted more like a general program. From this point of view it is easy to observe that the table of transition rules in the deterministic finite automata is a special case of such program. Moreover, the neuromata can encode the nondeterministic transition function efficiently using parallelism. In this way, the behavior of the neuromata in the exponential number of all possible states can be described in a polynomial size. However, this process is not reversible. The neural transition program cannot be generally rewritten as a polynomial size table or a polynomial length regular expression. Moreover, the number of neuromaton states (although exponential) is limited and that is why the matching lower bound on the neuromaton size can be achieved using the standard technique: there exists a regular language which requires the exponential number of neuromaton states to be recognized.

Also, a more complex neural transition rule specification makes the emptiness problem for the neuromata harder than for the classical formalism (finite automata, regular expressions). On the other hand, while the complement operation in the nondeterministic case of this formalism can cause an exponential growth of the descriptional complexity, the complement of the neuromata can be easily achieved.

We have also investigated the Hopfield neuromata which are studied widely due to their convergence properties. We have shown that Hopfield neuromata determine the proper subclass of regular languages — the so-called Hopfield languages. Via the so-called Hopfield condition, we have completely characterized the class of Hopfield languages.

We can conclude that the neuromata present quite an efficient tool not only for the recognition of regular languages and of their subclasses respectively, but also for their description.

Acknowledgement

We are grateful to Markus Holzer, Piotr Indyk, and Petr Savický for the stimulating discussions related to the topics of this paper. Further we thank Tereza Bedaňová who realized the pictures in LaTex environment.

References

[1] Aho, A. V., Hopcroft, J. E., Ullman, J. D. 1974. The Design and Analysis of Computer Algorithms. Addison–Wesley, Reading.

[2] Alon, N., Dewdney, A. K., Ott, T. J. 1991. Efficient Simulation of Finite Automata by Neural Nets. Journal of the ACM, 38, 495–514.

[3] Brāzma, A. 1995. Learning of Regular Expressions by Pattern Matching. Proceedings of the 2^{nd} European Conference Computational Learning Theory Euro–COLT'95, Barcelona, Springer–Verlag, LNAI 904, 392–403.

[4] Floréen, P., Orponen, P. 1994. Complexity Issues in Discrete Hopfield Networks. Research Report A–1994–4, Dept. of Computer Science, University of Helsinki.

[5] Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z., Lee, Y. C. 1992. Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks. Neural Computation, 4, 393–405.

[6] Horne, B. G., Hush, D. R. 1996. Bounds on the Complexity of Recurrent Neural Network Implementations of Finite State Machines. Neural Networks, 9, 243–252.

[7] Hecht-Nielsen, R. 1990. Neurocomputing. Addison–Wesley, California.

[8] Indyk, P. 1995. Optimal Simulation of Automata by Neural Nets. Proceedings of the 12^{th} Annual Symposium on Theoretical Aspects of Computer Science STACS'95, Springer–Verlag, LNCS 900, 337–348.

[9] Indyk, P. 1995. Personal Communication.

[10] Jiang, T., Ravikumar, R. 1991. A Note on the Space Complexity of Some Decision Problems for Finite Automata. Information Processing Letters, 40, 25–31.

[11] Jones, N. 1975. Space-Bounded Reducibility Among Combinatorial Problems. Journal of Computer and System Sciences, 11, 68–85.

[12] Kleene, S. C. 1956. Representation of Events in Nerve Nets and Finite Automata. Automata Studies, volume 34 of Annals of Mathematics Studies, Princeton University Press, New Jersey, 3–41.

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

[14] Orponen, P. 1994. Computational Complexity of Neural Networks: A Survey. Nordic Journal of Computing, 1, 94–110.

[15] Parberry, I. 1994. Circuit Complexity and Neural Networks. MIT Press, Cambridge, Massachusetts.

[16] Šíma, J., Wiedermann, J. 1996. Neural Language Acceptors. Proceedings of the 2^{nd} International Conference Developments in Language Theory DLT'95, Magdeburg, World Scientific Publishing, 430–439.

[17] Šíma, J. 1995. Hopfield Languages. Proceedings of the SOFSEM'95 Seminar on Current Trends in Theory and Practice of Informatics, Milovy, Springer–Verlag, LNCS 1012, 461–468.

[18] Šíma, J. 1997. Analog Stable Simulation of Discrete Neural Networks. Neural Network World, 7.

[19] Siu, K.-Y., Roychowdhury, V., Kailath, T. 1995. Discrete Neural Computation *A Theoretical Foundation*. Prentice–Hall PTR, Englewood Cliffs, New Jersey.

[20] Tiňo, P., Šajda, J. 1995. Learning and Extracting Initial Mealy Automata with a Modular Neural Network Model. Neural Computation, 7, 822–844.

[21] Wiedermann, J. 1994. Complexity Issues in Discrete Neurocomputing. Neural Network World, 4, 99–119.

[22] Zeng, Z., Smyth, P. 1993. Learning Finite State Machines with Self-Clustering Recurrent Networks. Neural Computation, 5, 976–990, 1993.