Energy Complexity of Recurrent Neural Networks

Jiří Šíma

Institute of Computer Science,

Academy of Sciences of the Czech Republic,

P. O. Box 5, 18207 Prague 8, Czech Republic, *sima@cs.cas.cz*

Keywords: Neural network, finite automaton, energy complexity, optimal size

Abstract

Recently, a new so-called energy complexity measure has been introduced and studied for feedforward perceptron networks. This measure is inspired by the fact that biological neurons require more energy to transmit a spike than not to fire, and the activity of neurons in the brain is quite sparse, with only about 1% of neurons firing. In this paper, we investigate the energy complexity of recurrent networks which counts the number of active neurons at any time instant of a computation. We prove that any deterministic finite automaton with m states can be simulated by a neural network of optimal size $s = \Theta(\sqrt{m})$ with the time overhead of $\tau = O(s/e)$ per one input bit, using the energy $O(e)$, for any e such that $e = \Omega(\log s)$ and $e = O(s)$, which shows the time-energy tradeoff in recurrent networks. In addition, for the time overhead τ satisfying $\tau^\tau = o(s)$, we obtain the lower bound of $s^{c/\tau}$ on the energy of such a simulation, for some constant $c > 0$ and for infinitely many s.

1 Introduction

In biological neural networks the energy cost of a firing neuron is relatively high[1], while energy supplied to the brain is limited and hence the activity of neurons in the brain is quite sparse, with only about 1% of neurons firing (Lennie, 2003). This is in contrast to artificial neural networks in which, on average, every second unit fires during a computation. This fact has recently motivated the definition of a new complexity measure for *feedforward* perceptron networks (threshold circuits), the so-called energy complexity (Uchizawa et al., 2006) which is the maximum number of units in the network which output 1, taken over all the inputs to the circuit. Energy complexity has been shown to be closely related by tradeoff results to other complexity measures such as the network size (i.e., the number of neurons) (Uchizawa et al., 2008, 2011b), the circuit depth (i.e., parallel computational time) (Uchizawa et al., 2010, 2008), and the fan-in (i.e., the maximum number of inputs to a single unit) (Suzuki et al., 2011) etc. In addition, energy complexity has found its use in circuit complexity, e.g. as a tool for proving the lower bounds (Uchizawa et al., 2011a) etc.

[1]The relatively small difference in the oxygen consumption of a used vs. not used functional part of brain (e.g. visual cortex), which is documented by the fMRI studies, is caused by the fact that the corresponding neurons fire although the respective region is not employed for its purpose.

In this paper, we investigate, for the first time, the energy complexity of *recurrent* neural networks which we define to be the maximum number of neurons outputting 1 at any time instant, taken over all possible computations. It has been known for a long time that the computational power of binary-state recurrent networks corresponds to that of finite automata since the network of size s units can reach only a finite number (at most 2^s) different states (Šíma et al., 2003). A simple way of simulating a given deterministic finite automaton A with m states by a neural network N of size $O(m)$ is to implement each of the $2m$ transitions of A (having 0 and 1 transitions for each state) by a single unit in N which checks whether the input bit agrees with the respective type of transition (Minsky, 1967). Clearly, this simple linear-size implementation of finite automata requires only a constant energy.

Much effort had been given to reducing the size of neural automata (e.g., Alon et al., 1991; Horne et al., 1996; Indyk, 1995; Šíma et al., 1998) and, indeed, neural networks of size $\Theta(\sqrt{m})$ implementing a given deterministic finite automaton with m states were proposed and proven to be size-optimal (Horne et al., 1996; Indyk, 1995). A natural question arises: what is the energy consumption when simulating finite automata by optimal-size neural networks? We answer this question by proving the tradeoff between the energy and the time overhead of the simulation. In particular, we prove that an optimal-size neural network of $s = \Theta(\sqrt{m})$ units can be constructed to simulate a deterministic finite automaton with m states using the energy $O(e)$ for any function e such that $e = \Omega(\log s)$ and $e = O(s)$, while the time overhead for processing one input bit is $\tau = O(s/e)$. For this purpose, we adapt the asymptotically optimal method of threshold circuit synthesis due to Lupanov (1973).

In addition, we derive lower bounds on the energy consumption e of a neural network of size s simulating a finite automaton within the time overhead τ per one input bit, by using the technique due to Uchizawa et al. (2008) which is based on communication complexity (Kushilevitz et al., 1997). In particular, for less than sublogarithmic time overhead τ satisfying $\tau \log \tau = o(\log s)$, we obtain the lower bound $\log e = \Omega_\infty \left(\frac{1}{\tau} \log s \right)$ which implies $e \geq s^{c/\tau}$ for some constant $c > 0$ and for infinitely many s. For example, this means that for time overhead $\tau = O(1)$, the energy of any simulation must fulfill $e \geq s^\delta$ for some constant δ such that $0 < \delta < 1$, and for infinitely many s, which can be compared to the energy $e = O(s)$ consumed by our simulation. For $\tau = O(\log^\alpha s)$ where $0 < \alpha < 1$, any simulation requires $e = \Omega_\infty \left(s^{\frac{\log \log s}{\log^\delta s}} \right)$ for any $\delta > \alpha$, while $e = O\left(s / \log^\alpha s\right)$ is sufficient for our implementation.

This paper is organized as follows. After a brief review of the basic definitions in section 2, the main result concerning a low-energy simulation of finite automata by neural nets is formulated in section 3 including the basic ideas of the proof. The subsequent two sections are devoted to the technical details of the proof: section 4 deals with a decomposition of the transition function and section 5 describes the construction of low-energy neural automata. The lower bounds on the energy consumption of such neural automata are derived and compared to the respective upper bounds in section 6. A concluding summary is given in section 7. A preliminary version appeared as an extended abstract (Šíma, 2013).

2 Neural Networks as Finite Automata

We will first specify the model of an (artificial) *neural network* N. The network consists of s *units (neurons, threshold gates)*, indexed as $V = \{1, \ldots, s\}$, where s is called the network *size*. The units are connected into a directed graph representing the *architecture* of N, in which each edge (i, j) leading from unit i to j is labeled with an integer *weight* $w(i, j)$. The absence of a connection within the architecture corresponds to a zero weight between the respective neurons, and vice versa.

In contrast to general *recurrent* networks, which have cyclic architectures, the architecture of a *feedforward* network (or a so-called *threshold circuit*) is an acyclic graph. Hence, units in a feedforward network can be grouped in a unique minimal way into a sequence of $d + 1$ pairwise disjoint *layers* $\alpha_0, \ldots, \alpha_d \subseteq V$ so that neurons in any layer α_t are connected only to neurons in subsequent layers α_u, $u > t$. Usually the zeroth, or *input layer* α_0 consists of external inputs and is not counted in the number of layers and in the network size. The last, or *output layer* α_d is composed of output neurons. The number of layers d excluding the input one is called the *depth* of threshold circuit.

The *computational dynamics* of (not necessarily feedforward) network N determines for each unit $j \in V$ its binary *state (output)* $y_j^{(t)} \in \{0, 1\}$ at discrete time instants $t = 0, 1, 2, \ldots$. We say that neuron j is *active (fires)* at time t if $y_j^{(t)} = 1$, while j is *passive* for $y_j^{(t)} = 0$. This establishes the *network state* $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_s^{(t)}) \in \{0, 1\}^s$ at each discrete time instant $t \geq 0$. At the beginning of a computation, the neural network N is placed in an *initial state* $\mathbf{y}^{(0)}$ which may also include an external input. At discrete time instant $t \geq 0$, an *excitation* of any neuron $j \in V$ is defined as

$$\xi_j^{(t)} = \sum_{i=1}^{s} w(i,j) y_i^{(t)} - h(j) \tag{1}$$

including an integer *threshold* $h(j)$ local to unit j. At the next instant $t+1$, the neurons $j \in \alpha_{t+1}$ from a selected subset $\alpha_{t+1} \subseteq V$ update their states $y_j^{(t+1)} = H(\xi_j^{(t)})$ in parallel by applying the *Heaviside function* $H : \mathbf{R} \longrightarrow \{0,1\}$ which is defined as

$$H(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0 \, . \end{cases} \tag{2}$$

The remaining units $j \in V \setminus \alpha_{t+1}$ do not change their outputs, that is $y_j^{(t+1)} = y_j^{(t)}$ for $j \notin \alpha_{t+1}$. In this way, the new network state $\mathbf{y}^{(t+1)}$ at time $t+1$ is determined.

Without loss of efficiency (Orponen, 1997), we implicitly assume *synchronous* computations. Thus, the sets α_t which define the computational dynamics of N are predestined deterministically for each time instant t (e.g. $\alpha_t = V$ for any $t \geq 1$ means fully parallel synchronous updates). Note that computations in feedforward networks proceed layer by layer from the input layer up to the output one (i.e. sets α_t naturally coincide with layers), which implement Boolean functions. We define the *energy complexity* of N to be the maximum number of active units $\sum_{j=1}^{s} y_j^{(t)}$ at any time instant $t \geq 0$, taken over all the computations of N.

The computational power of recurrent neural networks has been studied analogously to the traditional models of computations so that the networks are exploited as acceptors of formal languages $L \subseteq \{0,1\}^*$ over the binary alphabet. For the finite networks that are to recognize regular languages, the following input/output protocol has been used (Alon et al., 1991; Horne et al., 1996; Indyk, 1995; Siegelmann et al., 1995; Šíma et al., 1998, 2003). A binary input word (string) $\mathbf{x} = x_1 \ldots x_n \in \{0,1\}^n$ of arbitrary length $n \geq 0$ is sequentially presented to the network bit by bit via an *input neuron* in $\in V$. The state of this unit is externally set (and clamped) to the respective input bits at prescribed time instants, regardless of any influence from the remaining neurons in the network, that is,

$$y_{\mathrm{in}}^{(\tau(i-1))} = x_i \tag{3}$$

for $i = 1, \ldots, n$ where an integer parameter $\tau \geq 1$ is the *period* or *time overhead* for processing a single input bit. Then, an *output neuron* out $\in V$ signals at time τn whether the input word belongs to underlying language L, that is,

$$y_{\mathrm{out}}^{(\tau n)} = \begin{cases} 1 & \text{for } \mathbf{x} \in L \\ 0 & \text{for } \mathbf{x} \notin L\,. \end{cases} \tag{4}$$

As usual, we will describe the limiting behavior (rate of growth) of functions when the argument tends towards infinity in terms of simpler functions by using Landau or big O notation. Recall that for functions $f \geq 1$ and $g \geq 1$ defined for all natural numbers, notations $g = O(f)$ and $g = \Omega(f)$ mean that for some real constant $c > 0$ and for all but finitely many natural numbers n, $g(n) \leq c \cdot f(n)$ and $g(n) \geq c \cdot f(n)$, respectively. In addition, $g = \Theta(f)$ if $g = O(f)$ and $g = \Omega(f)$ simultaneously. Similarly, $g = o(f)$ denotes that for *every* real constant $c > 0$ and for all but finitely many natural numbers n, $g(n) \leq c \cdot f(n)$, while $g = \Omega_\infty(f)$ means that for some real constant $c > 0$ and for infinitely many natural numbers n, $g(n) \geq c \cdot f(n)$. Clearly, $g = o(f)$ iff $\lim_{n\to\infty} g(n)/f(n) = 0$ iff $g \neq \Omega_\infty(f)$.

3 Low-Energy Optimal-Size Neural Finite Automata

Now, we can formulate our main result concerning a low-energy implementation of finite automata by optimal-size neural nets:

Theorem 1 *A given deterministic finite automaton A with m states can be simulated by a neural network N of optimal size $s = \Theta(\sqrt{m})$ neurons with time overhead $\tau = O(s/e)$ per one input bit, using the energy $O(e)$, where e is any function satisfying $e = \Omega(\log s)$ and $e = O(s)$.*

Proof: We will first outline the main ideas of the proof while the following two sections provide the detailed argument. As we are interested in asymptotic analysis we will hereafter assume m to be sufficiently large. A set Q of m states of a given deterministic finite automaton A can be arbitrarily enumerated so that each $q \in Q$ is binary encoded using $p = \lceil \log m \rceil + 1$ bits including one additional (e.g. the pth) bit which indicates the final states (i.e. its value is 1 just for the final states of A). Then, the respective transition function $\delta : Q \times \{0,1\} \longrightarrow Q$ of automaton A, producing its new state $q_{\text{new}} = \delta(q_{\text{old}}, x) \in Q$ from the old state $q_{\text{old}} \in Q$ and current input bit $x \in \{0,1\}$, can be viewed as a vector Boolean function $\mathbf{f} : \{0,1\}^{p+1} \longrightarrow \{0,1\}^p$ in terms of binary encoding of automaton's states.

Furthermore, "transition" function $\mathbf{f}$ is implemented by a four-layer neural network C of asymptotically optimal size $s = \Theta(\sqrt{2^p}) = \Theta(\sqrt{m})$ using the method of threshold circuit synthesis due to Lupanov (1973). Feedforward network C implementing the transition function δ of A can then simply be transformed to a recurrent neural network N simulating A by adding the recurrent connections from the fourth layer to the first one (in fact, the fourth output layer of C is identified with the zeroth input one in N) which replace the code of the old state of A by the new one. Using this approach, a finite automaton can be implemented by an optimal-size neural net (Horne et al., 1996).

Unfortunately, the second layer of C in Lupanov's construction (Lupanov, 1973) contains $\Theta(s)$ neurons, half of which fire for any input to C, which results in an unacceptably high energy consumption $\Omega(s)$. In order to achieve a low-energy implementation of A, this layer of $\Theta(s)$ neurons is properly partitioned into $O(s/e)$ blocks of $O(e)$ units each. Then, so-called control units are introduced which ensure that these blocks are updated successively one by one so that the energy consumption (i.e. the maximum number of simultaneously active neurons) of $O(e)$ is guaranteed while the time overhead for processing a single input bit increases to $O(s/e)$. In addition, the results of computation of particular blocks must somehow be preserved using the available energy.

In particular, we will decompose transition function $\mathbf{f}$ in section 4 using the method of threshold circuit synthesis (Lupanov, 1973) so that $\mathbf{f}$ can be implemented by a four-layer threshold circuit C of asymptotically optimal size. This decomposition is then used in section 5 for constructing a low-energy recurrent neural network N which incorporates circuit C and thus simulates finite automaton A.

4 The Transition Function Decomposition

In this section, we will employ the asymptotically optimal method of threshold circuit synthesis due to Lupanov (1973) for a decomposition of function $\mathbf{f} : \{0,1\}^{p+1} \longrightarrow$

$\{0,1\}^p$ which implements the transition function δ of finite automaton A in terms of binary encoding of its states. This decomposition allows $\mathbf{f}$ to be evaluated by a four-layer threshold circuit C of asymptotically optimal size $\Theta(\sqrt{2^p})$ which will later (in section 5) be incorporated into a low-energy recurrent neural network N simulating A. In particular, the resulting formula (18) for function $\mathbf{f}$ is derived below which will be exploited for computing $\mathbf{f}$ using four layers of perceptron units as it is summarized in the beginning of section 5. Unlike the result by Horne et al. (1996), the decomposition technicalities are needed for minimizing the energy demands when A is being implemented by N. Therefore, in the rest of this section, we will completely reformulate the method due to Lupanov (1973) in a detailed and compact form which is simplified and adopted for our future use.[2]

The $p+1$ arguments of vector function $\mathbf{f}(\mathbf{u},\mathbf{v},\mathbf{z})$ are split into three groups $\mathbf{u} = (u_1,\ldots,u_{p_1})$, $\mathbf{v} = (v_1,\ldots,v_{p_2})$, and $\mathbf{z} = (z_1,\ldots,z_{p_3})$, respectively, where

$$\begin{aligned} p_3 &= \lfloor \log(p+1-\log p) - 2 \rfloor && (5)\\ p_1 &= \left\lfloor \frac{p+1-\log p - \log(p+1-\log p)}{2} \right\rfloor && (6)\\ p_2 &= p+1-p_3-p_1\,. && (7) \end{aligned}$$

Parameters p_1, p_2, p_3 are chosen so that the resulting circuit for $\mathbf{f}$ will have the asymptotically optimal size $\Theta(\sqrt{2^p})$ (see section 5.6). Then, each function element f_k :

[2] To the best of our knowledge the classical result due to Lupanov (1973) is described in full details only in his original Russian paper.

$\{0,1\}^{p+1} \longrightarrow \{0,1\}$ ($1 \leq k \leq p$) of vector function $\mathbf{f} = (f_1, \ldots, f_p)$ is decomposed to

$$\begin{aligned} f_k(\mathbf{u},\mathbf{v},\mathbf{z}) &= \bigvee_{\mathbf{c}\in\{0,1\}^{p_3}} (f_k(\mathbf{u},\mathbf{v},\mathbf{c}) \wedge (\mathbf{z} = \mathbf{c})) \\ &= \bigvee_{\mathbf{c}\in\{0,1\}^{p_3}} \left(f_k(\mathbf{u},\mathbf{v},\mathbf{c}) \wedge \bigwedge_{j=1}^{p_3} \ell_{c_j}(z_j) \right) , \end{aligned} \tag{8}$$

where the respective literals are defined as

$$\ell_c(z) = \begin{cases} z & \text{for } c = 1 \\ \neg z & \text{for } c = 0\,. \end{cases} \tag{9}$$

Furthermore, we define vector functions $\mathbf{g}_k : \{0,1\}^{p_1+p_2} \longrightarrow \{0,1\}^{p_1}$ for $k = 1, \ldots, p$ as

$$\mathbf{g}_k(\mathbf{u},\mathbf{v}) = (f_k(\mathbf{u},\mathbf{v},[0]^{p_3}), f_k(\mathbf{u},\mathbf{v},[1]^{p_3}), \ldots, f_k(\mathbf{u},\mathbf{v},[2^{p_3}-1]^{p_3}), 0, \ldots, 0) \tag{10}$$

where $[j]^n = \mathbf{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$ denotes an n-bit binary representation of integer $j \geq 0$, that is, $j = \langle \mathbf{c} \rangle = \sum_{i=1}^{n} 2^{i-1} c_i$. Note that we use $\langle \mathbf{c} \rangle$ for denoting the inverse value to $[j]^n$, e.g. $\langle [j]^n \rangle = j$. The vector produced by $\mathbf{g}_k$ in equation (10) has p_1 elements out of which the first 2^{p_3} items are defined using f_k for all possible values of argument $\mathbf{z} \in \{0,1\}^{p_3}$, while the remaining ones are 0s, which is a correct definition since $2^{p_3} < p_1$ for sufficiently large p according to formulas (5) and (6).

In the following lemma we will further decompose functions $\mathbf{g}_k$ ($1 \leq k \leq p$). For this purpose, we split the first part $\mathbf{u} \in \{0,1\}^{p_1}$ of $\mathbf{g}_k$'s argument into its first bit $a \in \{0,1\}$ and the remaining bits $\mathbf{u}' \in \{0,1\}^r$ where $r = p_1 - 1$. Thus, we will use notation $\mathbf{g}_k(a,\mathbf{u}',\mathbf{v})$ as an alternative to $\mathbf{g}_k(\mathbf{u},\mathbf{v})$ introduced in equation (10) when the first bit of $\mathbf{g}_k$'s argument needs to be specified explicitly (similarly for functions $\mathbf{f}$, f_k etc.).

Lemma 1 *For each* $\mathbf{g}_k$ *(*$1 \leq k \leq p$*), one can construct four vector functions* $\mathbf{g}_k^a : \{0,1\}^{r+p_2} \longrightarrow \{0,1\}^{p_1}$ *and* $\mathbf{h}_k^a : \{0,1\}^{r+p_2} \longrightarrow \{0,1\}^{p_1}$ *for* $a \in \{0,1\}$ *satisfying the following two conditions:*

1. *For any* $a \in \{0,1\}$, $\mathbf{u}' \in \{0,1\}^r$, *and* $\mathbf{v} \in \{0,1\}^{p_2}$,
$$\mathbf{g}_k(a, \mathbf{u}', \mathbf{v}) = \mathbf{g}_k^a(\mathbf{u}', \mathbf{v}) \oplus \mathbf{h}_k^a(\mathbf{u}', \mathbf{v}) \tag{11}$$
where $\oplus$ *denotes a bitwise parity.*

2. *Functions* $\mathbf{g}_k^a, \mathbf{h}_k^a$ *are injective in the first vector argument* $\mathbf{u}'$*, that is, for any* $a \in \{0,1\}$, $\mathbf{v} \in \{0,1\}^{p_2}$, *and* $\mathbf{u}'_1, \mathbf{u}'_2 \in \{0,1\}^r$,
$$\textit{if } \mathbf{u}'_1 \neq \mathbf{u}'_2\,, \textit{ then } \mathbf{g}_k^a(\mathbf{u}'_1, \mathbf{v}) \neq \mathbf{g}_k^a(\mathbf{u}'_2, \mathbf{v}) \textit{ and } \mathbf{h}_k^a(\mathbf{u}'_1, \mathbf{v}) \neq \mathbf{h}_k^a(\mathbf{u}'_2, \mathbf{v})\,. \tag{12}$$

Recall that the parity $\mathbf{z} = \mathbf{x} \oplus \mathbf{y} \in \{0,1\}^n$ used in equation (11) is defined for vectors $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, $\mathbf{y} = (y_1, \ldots, y_n) \in \{0,1\}^n$, and $\mathbf{z} = (z_1, \ldots, z_n) \in \{0,1\}^n$ as
$$z_i = (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i) \tag{13}$$
(i.e. $z_i = 1$ iff $x_i \neq y_i$) for every $i = 1, \ldots, n$, which is an associative operation.

Proof: For any $\mathbf{v} \in \{0,1\}^{p_2}$, the function values $\mathbf{g}_k^a([i]^r, \mathbf{v})$ are defined inductively for $i = 0, \ldots, 2^r - 1$ as $\mathbf{g}_k^a([i]^r, \mathbf{v})$ is chosen arbitrarily from $\{0,1\}^{p_1} \setminus G_k^a(i, \mathbf{v})$ where
$$\begin{aligned} G_k^a(i, \mathbf{v}) \;=\; & \{\, \mathbf{g}_k^a([j]^r, \mathbf{v}) \mid j = 0, \ldots, i-1 \} \\ & \cup \{\, \mathbf{g}_k(a, [i]^r, \mathbf{v}) \oplus \mathbf{g}_k(a, [j]^r, \mathbf{v}) \oplus \mathbf{g}_k^a([j]^r, \mathbf{v}) \mid j = 0, \ldots, i-1 \}\;, \end{aligned} \tag{14}$$
and functions $\mathbf{h}_k^a$ are defined so that equation (11) is met:
$$\mathbf{h}_k^a(\mathbf{u}', \mathbf{v}) = \mathbf{g}_k(a, \mathbf{u}', \mathbf{v}) \oplus \mathbf{g}_k^a(\mathbf{u}', \mathbf{v})\,. \tag{15}$$

Note that $\emptyset = G_k^a(0,\mathbf{v}) \subseteq G_k^a(1,\mathbf{v}) \subseteq \cdots \subseteq G_k^a(2^r-1,\mathbf{v})$ and $|G_k^a(i,\mathbf{v})| \leq 2i$ according to definition (14), which implies $|G_k^a(i,\mathbf{v})| \leq |G_k^a(2^r-1,\mathbf{v})| \leq 2(2^r-1)$. Hence, $|\{0,1\}^{p_1} \setminus G_k^a(i,\mathbf{v})| \geq 2^{p_1} - 2(2^r-1) = 2$ for any $i = 0,\ldots,2^r-1$, which ensures that $\mathbf{g}_k^a(\mathbf{u}',\mathbf{v})$ is correctly defined for all the values of argument $\mathbf{u}' \in \{0,1\}^r$. Moreover, condition (12) is satisfied because for any $i,j \in \{0,\ldots,2^r-1\}$ such that $i > j$, definition (14) secures $\mathbf{g}_k^a([i]^r,\mathbf{v}) \neq \mathbf{g}_k^a([j]^r,\mathbf{v})$ and $\mathbf{h}_k^a([i]^r,\mathbf{v}) = \mathbf{g}_k(a,[i]^r,\mathbf{v}) \oplus \mathbf{g}_k^a([i]^r,\mathbf{v}) \neq \mathbf{g}_k(a,[i]^r,\mathbf{v}) \oplus \mathbf{g}_k(a,[i]^r,\mathbf{v}) \oplus \mathbf{g}_k(a,[j]^r,\mathbf{v}) \oplus \mathbf{g}_k^a([j]^r,\mathbf{v}) = \mathbf{h}_k^a([j]^r,\mathbf{v})$ by using condition (15) and the fact that $\mathbf{x} \oplus \mathbf{x} \oplus \mathbf{y} = \mathbf{y}$. This completes the proof of lemma 1. $\square$

We further rewrite $\mathbf{g}_k^a$ and $\mathbf{h}_k^a$ from lemma 1 by using the functions $\varphi_k^a : \{0,1\}^{r+p_2} \longrightarrow \{0,\ldots,2^{p_1}-1\}$ and $\psi_k^a : \{0,1\}^{r+p_2} \longrightarrow \{0,\ldots,2^{p_1}-1\}$ so that

$$\mathbf{g}_k^a(\mathbf{u}',\mathbf{v}) = [\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1} \text{ and } \mathbf{h}_k^a(\mathbf{u}',\mathbf{v}) = [\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1}\ , \tag{16}$$

respectively, which for any $a \in \{0,1\}$, $\mathbf{v} \in \{0,1\}^{p_2}$, and $\mathbf{u}'_1, \mathbf{u}'_2 \in \{0,1\}^r$, satisfy

$$\text{if } \mathbf{u}'_1 \neq \mathbf{u}'_2\,,\ \text{then } \varphi_k^a(\mathbf{u}'_1,\mathbf{v}) \neq \varphi_k^a(\mathbf{u}'_2,\mathbf{v}) \text{ and } \psi_k^a(\mathbf{u}'_1,\mathbf{v}) \neq \psi_k^a(\mathbf{u}'_2,\mathbf{v}) \tag{17}$$

according to condition (12). Now, we can plug formulas (10), (11), (13), and (16) into equation (8), which results in the final $\mathbf{f}$'s decomposition

$$\begin{aligned} f_k(a,\mathbf{u}',\mathbf{v},\mathbf{z}) &= \bigvee_{\mathbf{c}\in\{0,1\}^{p_3}} \left((\mathbf{g}_k(a,\mathbf{u}',\mathbf{v}))_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i) \right) \\ &= \bigvee_{\mathbf{c}\in\{0,1\}^{p_3}} \left((\mathbf{g}_k^a(\mathbf{u}',\mathbf{v}) \oplus \mathbf{h}_k^a(\mathbf{u}',\mathbf{v}))_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i) \right) \end{aligned}$$

$$= \bigvee_{\mathbf{c}\in\{0,1\}^{p_3}} \left(\left(\left([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1}\right)_{\langle\mathbf{c}\rangle} \wedge \neg \left([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1}\right)_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i) \right) \right.$$
$$\left. \vee \left(\neg \left([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1}\right)_{\langle\mathbf{c}\rangle} \wedge \left([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1}\right)_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i) \right) \right) \tag{18}$$

where $(\mathbf{x})_i$ denotes the ith element of vector $\mathbf{x}$.

Formula (18) can be used for implementing a four-layer circuit C which computes the transition function $\mathbf{f}$ of finite automaton A using the asymptotically optimal number $\Theta(\sqrt{2^p})$ of threshold gates. The details of such an implementation are presented in section 5 where this circuit C will be incorporated into a low-energy recurrent neural network N simulating A.

5 The Finite Automaton Implementation

In this section, we will introduce the construction of a low-energy recurrent neural network N simulating a given finite automaton A. In particular, a set of neurons V is composed of four disjoint layers $V = \nu_0 \cup \nu_1 \cup \nu_2 \cup \nu_3$ which mainly evaluate the transition function $\mathbf{f}$ according to its decomposition (18) derived in section 4 as follows.

The zeroth layer ν_0 is composed of $p + 1$ units storing an input bit and a current state of A, which are split according to $\mathbf{f}$'s arguments $a, \mathbf{u}', \mathbf{v}, \mathbf{z}$ (section 5.1). The gates in the first layer ν_1 computes all possible monomials $\bigwedge_{i=1}^{p_2} \ell_{b_i}(v_i)$ for $\mathbf{b} \in \{0,1\}^{p_2}$ over variables $\mathbf{v}$ (section 5.2), which are used for evaluating functions $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$ in the second layer ν_2 (section 5.3). The third layer ν_3 computes conjunctions $([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \neg([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$ and $\neg([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge ([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$ from formula (18) while their disjunction for $\mathbf{c} \in \{0,1\}^{p_3}$

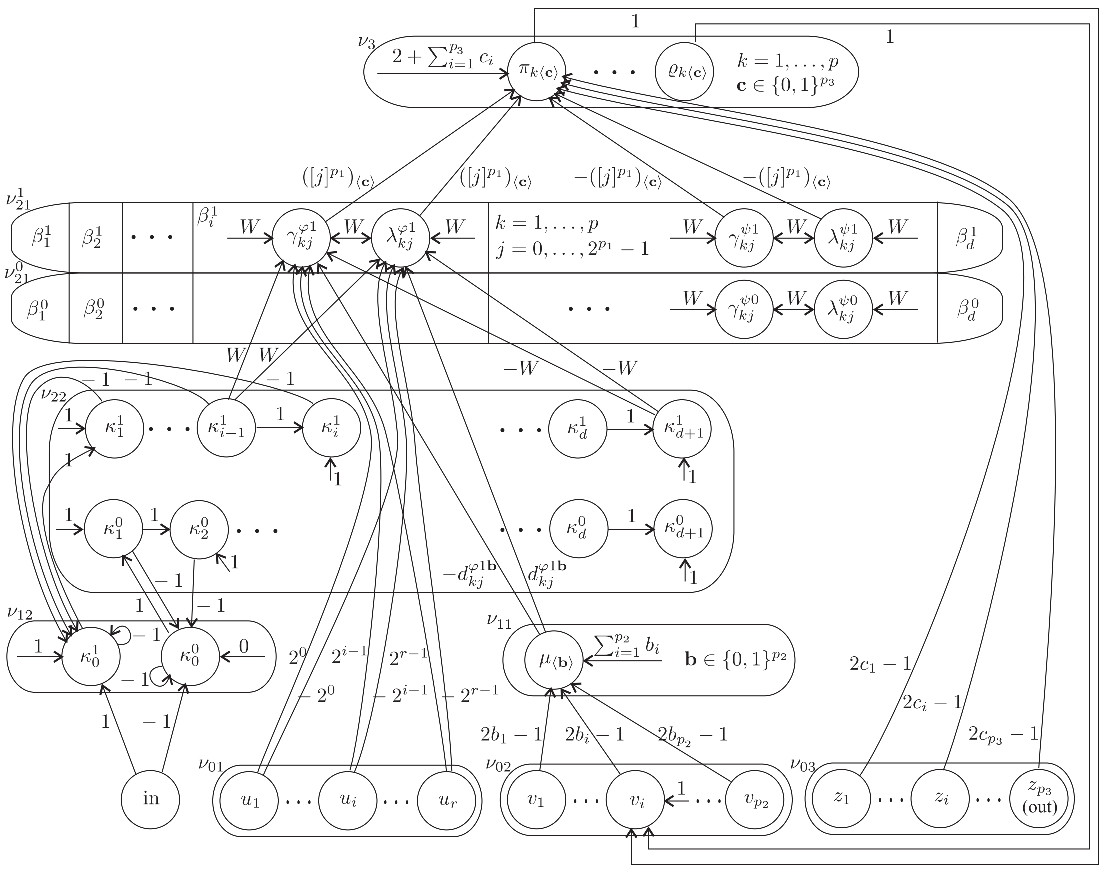

Figure 1: A schema of neural network N implementing finite automaton A

is evaluated for each f_k in the fourth layer which coincides with zeroth layer ν_0 as the old state of A is replaced by the new one (section 5.5) .

In addition, so-called control units are introduced in layers ν_1 and ν_2 for minimizing the energy demands of N (section 5.4). The global computational dynamics of N together with its energy complexity is specified in section 5.6. The network is schematically depicted in Figure 1 where the layers or their parts are indicated and only a few representative units and connections are drawn. The directed edges connecting units are labeled with the corresponding weights whereas the edges drawn without an originating unit correspond to the threshold parameters.

5.1 Layer ν_0 Stores an Input Bit and Automaton's State

An input bit and a current state of A are stored using $p+1$ neurons which constitute layer ν_0. Thus, set ν_0 includes the input neuron $\mathrm{in} \in \nu_0$ and the output neuron $\mathrm{out} \in \nu_0$ which stores the bit (in the state encoding) that indicates the final states. We will implement formula (18) in N for evaluating the transition function $\mathbf{f}$ in terms of binary encoding of states in order to compute the new state of A. For this purpose, layer $\nu_0 = \{\mathrm{in}\} \cup \nu_{01} \cup \nu_{02} \cup \nu_{03}$ is disjointly split into four parts corresponding to the partition of arguments of $\mathbf{f}(a, \mathbf{u}', \mathbf{v}, \mathbf{z})$, respectively, that is, the input neuron in represents the first variable a as $y_{\mathrm{in}} = a$, $\nu_{01} = \{u_1, \ldots, u_r\}$, $\nu_{02} = \{v_1, \ldots, v_{p_2}\}$, and $\nu_{03} = \{z_1, \ldots, z_{p_3}\}$ where neuron z_{p_3} corresponds to the output unit out as the last bit of state encoding has been chosen to indicate the final states of A. Note that we identify the names of neurons in $\nu_0 \setminus \{\mathrm{in}\}$ with the arguments of $\mathbf{f}$ encoding automaton's state so that these variables describe the states of corresponding units, e.g. $y_{v_i} = v_i$. For notational simplicity, we often omit the time index in the states of neurons (e.g. we write $y_{\mathrm{in}} = a$ instead of $y_{\mathrm{in}}^{(t)} = a$) and we implicitly assume step-by-step timing which follows the directed connections among neurons in the natural order as they are introduced when describing the implementation of $\mathbf{f}$ in N. The precise timing will be formalized within the global computational dynamics of N simulating A in section 5.6.

5.2 Layer ν_1 Computes Monomials $\bigwedge_{i=1}^{p_2} \ell_{b_i}(v_i)$

The next layer $\nu_1 = \nu_{11} \cup \nu_{12}$ consists of 2^{p_2} neurons in $\nu_{11} = \{\mu_{\langle \mathbf{b} \rangle} \mid \mathbf{b} \in \{0,1\}^{p_2}\}$ for computing all possible monomials $\bigwedge_{i=1}^{p_2} \ell_{b_i}(v_i)$ for $\mathbf{b} \in \{0,1\}^{p_2}$ over variables $\mathbf{v}$, and two other, so-called *control units* in $\nu_{12} = \{\kappa_0^0, \kappa_0^1\}$ which indicate the input bit value. These monomials will be used in section 5.3 for evaluating functions $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$ in formula (18). Thus, we introduce weights $w(v_i,\mu_{\langle\mathbf{b}\rangle}) = 2b_i - 1$ (i.e. $w(v_i,\mu_{\langle\mathbf{b}\rangle}) = 1$ for $b_i = 1$ whereas $w(v_i,\mu_{\langle\mathbf{b}\rangle}) = -1$ for $b_i = 0$) for $i = 1,\ldots,p_2$, and threshold $h(\mu_{\langle\mathbf{b}\rangle}) = \sum_{i=1}^{p_2} b_i = |\{i \in \{1,\ldots,p_2\}\,|\,b_i = 1\}|$, for any $\mathbf{b} = (b_1,\ldots,b_{p_2}) \in \{0,1\}^{p_2}$ so that the following lemma trivially holds:

Lemma 2 *Unit $\mu_{\langle\mathbf{b}\rangle}$ fires for input $\mathbf{v} \in \{0,1\}^{p_2}$ iff $\mathbf{b} = \mathbf{v}$.*

In addition, we define $w(\mathrm{in},\kappa_0^1) = 1$, $w(\mathrm{in},\kappa_0^0) = -1$ and $h(\kappa_0^1) = 1$, $h(\kappa_0^0) = 0$, which ensures that κ_0^1 fires just one computational step after the current input bit is presented to in iff $y_{\mathrm{in}} = 1$, and similarly, κ_0^0 is active iff $y_{\mathrm{in}} = 0$ (cf. Lemma 6).

5.3 Layer ν_2 Computes $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$

Furthermore, layer $\nu_2 = \nu_{21} \cup \nu_{22}$ where $\nu_{21} = \{\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}, \gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a}\,|\,1 \leq k \leq p\,,\ a \in \{0,1\}\,, j = 0,\ldots,2^{p_1}-1\}$, and $\nu_{22} = \{\kappa_i^a\,|\,a \in \{0,1\}\,,\ i = 1,\ldots,d+1\}$ with $d = \lceil|\nu_{21}|/(4e)\rceil = \lceil 2p2^{p_1}/e\rceil$, serves for a low-energy computation of functions $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$ for any $1 \leq k \leq p$ and $a \in \{0,1\}$.

In this section, we will first show how to implement functions $\varphi_k^a(\mathbf{u}',\mathbf{v})$ for any $1 \leq k \leq p$ and $a \in \{0,1\}$ with *no constraints on energy* by using the outputs of neurons from ν_{01} and ν_{11} which provide the values of argument $\mathbf{u}'$ (see section 5.1) and monomials $\bigwedge_{i=1}^{p_2} \ell_{b_i}(v_i)$ for all $\mathbf{b} \in \{0,1\}^{p_2}$ over variables $\mathbf{v}$ (see section 5.2), respectively. In particular, 2^{p_1} pairs of neurons $\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a} \in \nu_{21}$ for $j = 0,\ldots,2^{p_1}-1$ are employed for this purpose having zero thresholds for now (their thresholds will be defined in section 5.4 for the low-energy implementation) and weights $w(u_i,\gamma_{kj}^{\varphi a}) = -w(u_i,\lambda_{kj}^{\varphi a}) = 2^{i-1}$ for $i = 1,\ldots,r$ and $w(\mu_{\langle\mathbf{b}\rangle},\gamma_{kj}^{\varphi a}) = -w(\mu_{\langle\mathbf{b}\rangle},\lambda_{kj}^{\varphi a}) = -d_{kj}^{\varphi a\mathbf{b}} \in$

$\{0,\ldots,2^r-1\}$ such that $j=\varphi_k^a([d_{kj}^{\varphi a\mathbf{b}}]^r,\mathbf{b})\in\{0,\ldots,2^{p_1}-1\}$ for $\mathbf{b}\in\{0,1\}^{p_2}$. Note that $d_{kj}^{\varphi a\mathbf{b}}$ is uniquely defined according to condition (17). For this definition of weights, the following lemma shows how the outputs of $\gamma_{kj}^{\varphi a},\lambda_{kj}^{\varphi a}\in\nu_{21}$ for $j=0,\ldots,2^{p_1}-1$ represent the function value $\varphi_k^a(\mathbf{u}',\mathbf{v})\in\{0,\ldots,2^{p_1}-1\}$.

Lemma 3 *For any input $\mathbf{u}'\in\{0,1\}^r$ and $\mathbf{v}\in\{0,1\}^{p_2}$, at least one unit from the pair $\gamma_{kj}^{\varphi a},\lambda_{kj}^{\varphi a}$ fires for each $j\in\{0,\ldots,2^{p_1}-1\}$. In addition, both units $\gamma_{kj}^{\varphi a}$ and $\lambda_{kj}^{\varphi a}$ fire simultaneously iff $j=\varphi_k^a(\mathbf{u}',\mathbf{v})$.*

Proof: It follows that for given $\mathbf{u}'\in\{0,1\}^r$ and $\mathbf{v}\in\{0,1\}^{p_2}$, neuron $\gamma_{kj}^{\varphi a}$ fires iff

$$\sum_{i=1}^{r} w(u_i,\gamma_{kj}^{\varphi a})y_{u_i}+\sum_{\mathbf{b}\in\{0,1\}^{p_2}} w(\mu_{\langle\mathbf{b}\rangle},\gamma_{kj}^{\varphi a})y_{\mu_{\langle\mathbf{b}\rangle}}\geq 0 \quad\text{iff}$$
$$\sum_{i=1}^{r}2^{i-1}u_i'-d_{kj}^{\varphi a\mathbf{v}}\geq 0\quad\text{iff}\quad\langle\mathbf{u}'\rangle\geq d_{kj}^{\varphi a\mathbf{v}}\,, \tag{19}$$

since $y_{\mu_{\langle\mathbf{b}\rangle}}=1$ iff $\mathbf{b}=\mathbf{v}$ according to lemma 2. Similarly, neuron $\lambda_{kj}^{\varphi a}$ is active iff $\langle\mathbf{u}'\rangle\leq d_{kj}^{\varphi a\mathbf{v}}$. Clearly, either $\langle\mathbf{u}'\rangle\geq d_{kj}^{\varphi a\mathbf{v}}$ or $\langle\mathbf{u}'\rangle\leq d_{kj}^{\varphi a\mathbf{v}}$, and hence, at least one unit from the pair $\gamma_{kj}^{\varphi a},\lambda_{kj}^{\varphi a}$ is always active while both these neurons fire at the same time iff $\langle\mathbf{u}'\rangle=d_{kj}^{\varphi a\mathbf{v}}$ iff $j=\varphi_k^a(\mathbf{u}',\mathbf{v})$, which implements function $\varphi_k^a(\mathbf{u}',\mathbf{v})$. □

Functions $\psi_k^a(\mathbf{u}',\mathbf{v})$ for any $1\leq k\leq p$ and $a\in\{0,1\}$ are implemented analogously (replace φ by ψ above) using 2^{p_1} pairs of neurons $\gamma_{kj}^{\psi a},\lambda_{kj}^{\psi a}\in\nu_{21}$ for $j=0,\ldots,2^{p_1}-1$ so that the following lemma holds (cf. lemma 3):

Lemma 4 *For any input $\mathbf{u}'\in\{0,1\}^r$ and $\mathbf{v}\in\{0,1\}^{p_2}$, at least one unit from the pair $\gamma_{kj}^{\psi a},\lambda_{kj}^{\psi a}$ fires for each $j\in\{0,\ldots,2^{p_1}-1\}$. In addition, both units $\gamma_{kj}^{\psi a}$ and $\lambda_{kj}^{\psi a}$ fire simultaneously iff $j=\psi_k^a(\mathbf{u}',\mathbf{v})$.*

5.4 Low-Energy Implementation of $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$

We employ so-called *control units* $\kappa_i^a \in \nu_{12} \cup \nu_{22}$ for $a \in \{0,1\}$ and $i = 0,\ldots,d+1$, for synchronizing the computation of functions $\varphi_k^a(\mathbf{u}',\mathbf{v})$, $\psi_k^a(\mathbf{u}',\mathbf{v})$ by neurons from ν_{21} as described in section 5.3 so that their energy consumption is bounded by $e+2$.

For this purpose, we split set $\nu_{21} = \nu_{21}^0 \cup \nu_{21}^1$ into two parts $\nu_{21}^a = \{\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}, \gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a} \mid 1 \le k \le p\,,\, j = 0,\ldots,2^{p_1}-1\}$ of size $4p2^{p_1}$ according to $a \in \{0,1\}$, and both these parts are further partitioned into d blocks, each of size at most $2e$, that is, $\nu_{21}^a = \bigcup_{i=1}^d \beta_i^a$ where $|\beta_i^a| \le 2e$. In addition, we require that any pair $\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}$ respectively $\gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a}$ is included into one block, which means for each $i = 1,\ldots,d$, if $\gamma_{kj}^{\varphi a} \in \beta_i^a$, then $\lambda_{kj}^{\varphi a} \in \beta_i^a$, and if $\gamma_{kj}^{\psi a} \in \beta_i^a$, then $\lambda_{kj}^{\psi a} \in \beta_i^a$. Furthermore, all neurons $j \in \nu_{21}$ whose thresholds were originally assumed to be zero (see section 5.3) are now blocked by large thresholds $h(j) = W$ where $W = 2^r$, which make them passive. Then, for any $1 \le i \le d$ and $a \in \{0,1\}$, the neurons in block β_i^a are released to fire by control unit κ_{i-1}^a using the weights $w(\kappa_{i-1}^a, j) = W$ for all $j \in \beta_i^a$. Thus, if control unit κ_{i-1}^a fires, then threshold $h(j) = W$ of unit $j \in \beta_i^a \subseteq \nu_{21}$ is canceled by weight $w(\kappa_{i-1}^a, j) = W$ and neuron j takes part in the computation of $\varphi_k^a(\mathbf{u}',\mathbf{v})$ and $\psi_k^a(\mathbf{u}',\mathbf{v})$ according to lemma 3 and 4, respectively, as described in section 5.3. This is summarized in the following lemma:

Lemma 5 *For any $1 \le i \le d$ and $a \in \{0,1\}$, neurons in $\beta_i^a \subseteq \nu_{21}$ can fire (according to lemma 3 and 4) iff control unit κ_{i-1}^a is active.*

For current input bit $y_{\text{in}} = a \in \{0,1\}$, the control units $\kappa_0^a,\ldots,\kappa_{d+1}^a$ fire successively one by one, which is achieved by weights $w(\kappa_i^a, \kappa_{i+1}^a) = 1$ for $i = 0,\ldots,d$, $w(\kappa_i^a, \kappa_0^a) = -1$ for $i = 0,\ldots,d+1$, and thresholds $h(\kappa_i^a) = 1$ for $i = 1,\ldots,d+1$, as the following lemma formally proves:

Lemma 6 *For every $i = 0, \ldots, d+1$ and for current input bit $y_{\mathrm{in}}^{(t_0+i+1)} = a \in \{0,1\}$ which is first presented to network N at time instant $t_0 = \tau(\iota - 1)$ (i.e. $x_\iota = a$), $y_{\kappa_i^a}^{(t_0+i+1)} = 1$ whereas $y_{\kappa_i^a}^{(t_0+j+1)} = 0$ for any $j \in \{0, \ldots, d+1\}$ such that $j \neq i$.*

Proof: The argument for $y_{\kappa_i^a}^{(t_0+i+1)} = 1$ proceeds by induction on $i = 0, \ldots, d+1$. For $i = 0$, we know from section 5.2 that $y_{\kappa_0^a}^{(t_0+1)} = 1$ for $y_{\mathrm{in}}^{(t_0)} = a$ (all the control units are assumed to be passive at time instant t_0). For $i > 0$, the excitation of unit κ_i^a at time instant $t_0 + i$ can be evaluated as $\xi_{\kappa_i^a}^{(t_0+i)} = w(\kappa_{i-1}^a, \kappa_i^a) y_{\kappa_{i-1}^a}^{(t_0+i)} - h(\kappa_i^a) = 0$ because $y_{\kappa_{i-1}^a}^{(t_0+i)} = 1$ by induction hypothesis. Hence, $y_{\kappa_i^a}^{(t_0+i+1)} = H(\xi_{\kappa_i^a}^{(t_0+i)}) = 1$. It follows that $y_{\kappa_0^a}^{(t_0+i+1)} = 0$ for $i = 1, \ldots, d+1$ since $w(\kappa_{i-1}^a, \kappa_0^a) = -1$ and $y_{\kappa_{i-1}^a}^{(t_0+i)} = 1$, which makes and keeps neuron κ_i^a ($1 \leq i \leq d+1$) passive after κ_i^a fires at time instant $t_0 + i + 1$, that is, $y_{\kappa_i^a}^{(t_0+j+1)} = 0$ for any $j \in \{0, \ldots, d+1\}$ such that $j \neq i$. $\Box$

Lemmas 5 and 6 ensure that only the neurons from one block β_i^a of size at most $2e$ can fire simultaneously. Note that the respective units from β_i^a, which were released by active control unit κ_{i-1}^a, fire at the same time as the next control unit κ_i^a becomes active. In fact, we know from lemma 3 and 4 that just one unit of each pair $\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a} \in \beta_i^a$ or $\gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a} \in \beta_i^a$ is active except for the special pairs of both firing units $\gamma_{kj_\varphi}^{\varphi a}, \lambda_{kj_\varphi}^{\varphi a}$ and $\gamma_{kj_\psi}^{\psi a}, \lambda_{kj_\psi}^{\psi a}$ such that $\varphi_k^a(\mathbf{u}', \mathbf{v}) = j_\varphi$ and $\psi_k^a(\mathbf{u}', \mathbf{v}) = j_\psi$, respectively. Hence, the energy consumption of ν_{21} is bounded by $e + 2$.

In addition, we must also guarantee that the resulting function values $\varphi_k^a(\mathbf{u}', \mathbf{v}) = j_\varphi$, $\psi_k^a(\mathbf{u}', \mathbf{v}) = j_\psi$ are preserved, that is, neurons $\gamma_{kj_\varphi}^{\varphi a}, \lambda_{kj_\varphi}^{\varphi a}, \gamma_{kj_\psi}^{\psi a}, \lambda_{kj_\psi}^{\psi a}$ remain active without any support from corresponding control units until all the blocks perform computation and are blocked, which is indicated by control unit κ_{d+1}^a. This is implemented by symmetric weights $w(\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}) = w(\lambda_{kj}^{\varphi a}, \gamma_{kj}^{\varphi a}) = w(\gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a}) = w(\lambda_{kj}^{\psi a}, \gamma_{kj}^{\psi a}) = W$ for $a \in \{0,1\}$, $k = 1, \ldots, p$, $j = 0, \ldots, 2^{p_1} - 1$. Note that in the case when only one unit from the pair $\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}$ (similarly for $\gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a}$) is active, say $\gamma_{kj}^{\varphi a}$ fires and $\lambda_{kj}^{\varphi a}$ is passive, the introduced weight $w(\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}) = W$ does not cause the other unit $\lambda_{kj}^{\varphi a}$ to fire although this weight cancels its threshold $h(\lambda_{kj}^{\varphi a}) = W$ since $\lambda_{kj}^{\varphi a}$ is passive for zero threshold anyway. Moreover, neuron κ_{d+1}^{a} eventually resets all neurons in ν_{21} before becoming itself passive which is accomplished by weights $w(\kappa_{d+1}^{a}, j) = -W$ for all $j \in \nu_{21}^{a}$ and $a \in \{0,1\}$.

5.5 Layers ν_3 and ν_0 Evaluate f

Finally, layer $\nu_3 = \{\pi_{k\langle\mathbf{c}\rangle}, \varrho_{k\langle\mathbf{c}\rangle} \,|\, 1 \leq k \leq p\,,\, \mathbf{c} \in \{0,1\}^{p_3}\}$ is composed of 2^{p_3} pairs of neurons $\pi_{k\langle\mathbf{c}\rangle}, \varrho_{k\langle\mathbf{c}\rangle}$ for each $k = 1, \ldots, p$ which evaluate conjunctions $([\varphi_k^a(\mathbf{u}', \mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle}$ $\wedge \neg([\psi_k^a(\mathbf{u}', \mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$ and $\neg([\varphi_k^a(\mathbf{u}', \mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge ([\psi_k^a(\mathbf{u}', \mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$ from formula (18), respectively, for current input $y_{\text{in}} = a \in \{0,1\}$. For this purpose, the states of neurons from ν_{03} which store the values of argument $\mathbf{z}$, and the outputs of units $\gamma_{kj}^{\varphi a}, \lambda_{kj}^{\varphi a}, \gamma_{kj}^{\psi a}, \lambda_{kj}^{\psi a} \in \nu_{21}$ for $j = 0, \ldots, 2^{p_1} - 1$, which represent the function values $\varphi_k^a(\mathbf{u}', \mathbf{v}), \psi_k^a(\mathbf{u}', \mathbf{v})$ according to lemma 3 and 4 after κ_{d+1}^{a} fires, are used. For $\mathbf{c} \in \{0,1\}^{p_3}$, we define weights $w(\gamma_{kj}^{\varphi a}, \pi_{k\langle\mathbf{c}\rangle}) = w(\lambda_{kj}^{\varphi a}, \pi_{k\langle\mathbf{c}\rangle}) = -w(\gamma_{kj}^{\psi a}, \pi_{k\langle\mathbf{c}\rangle}) = -w(\lambda_{kj}^{\psi a}, \pi_{k\langle\mathbf{c}\rangle}) = -w(\gamma_{kj}^{\varphi a}, \varrho_{k\langle\mathbf{c}\rangle}) = -w(\lambda_{kj}^{\varphi a}, \varrho_{k\langle\mathbf{c}\rangle}) = w(\gamma_{kj}^{\psi a}, \varrho_{k\langle\mathbf{c}\rangle})$ $= w(\lambda_{kj}^{\psi a}, \varrho_{k\langle\mathbf{c}\rangle}) = ([j]^{p_1})_{\langle\mathbf{c}\rangle} \in \{0,1\}$ for $a \in \{0,1\}$, $j = 0, \ldots, 2^{p_1} - 1$, and $w(z_i, \pi_{k\langle\mathbf{c}\rangle})$ $= w(z_i, \varrho_{k\langle\mathbf{c}\rangle}) = 2c_i - 1$ for $i = 1, \ldots, p_3$, and threshold $h(\pi_{k\langle\mathbf{c}\rangle}) = h(\varrho_{k\langle\mathbf{c}\rangle}) = 2 + \sum_{i=1}^{p_3} c_i$. The correctness of this definition is shown in the following lemma:

Lemma 7 *For any values $a, \mathbf{u}', \mathbf{v}, \mathbf{z}$ of* $\mathbf{f}$*'s arguments, for every $k = 1, \ldots, p$, and*

$\mathbf{c} \in \{0,1\}^{p_3}$, *unit* $\pi_{k\langle\mathbf{c}\rangle}$ *fires iff* $([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \neg([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$ *and neuron* $\varrho_{k\langle\mathbf{c}\rangle}$ *is active iff* $\neg([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge ([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} \wedge \bigwedge_{i=1}^{p_3} \ell_{c_i}(z_i)$.

Proof: We know that for $y_{\text{in}} = a$ and each $k = 1,\ldots,p$, only one pair of neurons $\gamma_{kj_\varphi}^{\varphi a}, \lambda_{kj_\varphi}^{\varphi a}$ for $0 \leq j_\varphi \leq 2^{p_1}-1$ fires such that $j_\varphi = \varphi_k^a(\mathbf{u}',\mathbf{v})$ by lemma 3, and only one pair of units $\gamma_{kj_\psi}^{\psi a}, \lambda_{kj_\psi}^{\psi a}$ for $0 \leq j_\psi \leq 2^{p_1}-1$ is active such that $j_\psi = \psi_k^a(\mathbf{u}',\mathbf{v})$ according to lemma 4, while the remaining units in v_{21} are passive (blocked) after κ_{d+1}^a fires, which follows from lemma 5 and 6. Hence, neuron $\pi_{k\langle\mathbf{c}\rangle}$ is active iff $([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} = ([j_\varphi]^{p_1})_{\langle\mathbf{c}\rangle} = 1$ and $([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} = ([j_\psi]^{p_1})_{\langle\mathbf{c}\rangle} = 0$, and $y_{z_i} = c_i$ for every $i = 1,\ldots,p_3$, when active neurons $\gamma_{kj_\varphi}^{\varphi a}, \lambda_{kj_\varphi}^{\varphi a}$ contribute to excitation $\xi_{\pi_{k\langle\mathbf{c}\rangle}}$ by 2, while the contribution of units $z_1,\ldots,z_{p_3}$ to $\xi_{\pi_{k\langle\mathbf{c}\rangle}}$ reaches $\sum_{i=1}^{p_3} c_i$, which altogether equals the threshold $h(\pi_{k\langle\mathbf{c}\rangle})$ (cf. lemma 2). Analogously, neuron $\varrho_{k\langle\mathbf{c}\rangle}$ fires iff $([\varphi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} = 0$ and $([\psi_k^a(\mathbf{u}',\mathbf{v})]^{p_1})_{\langle\mathbf{c}\rangle} = 1$, and $y_{z_i} = c_i$ for every $i = 1,\ldots,p_3$. $\Box$

It follows from lemma 7 that for any $1 \leq k \leq p$, at most one unit among $\pi_{k\langle\mathbf{c}\rangle}, \varrho_{k\langle\mathbf{c}\rangle} \in \nu_3$ over $\mathbf{c} \in \{0,1\}^{p_3}$ is active, which determines the value of $f_k(a,\mathbf{u}',\mathbf{v},\mathbf{z})$ according to formula (18), as described in the following lemma:

Lemma 8 *For any values* $a,\mathbf{u}',\mathbf{v},\mathbf{z}$ *of* $\mathbf{f}$*'s arguments and for every* $k = 1,\ldots,p$, $f_k(a,\mathbf{u}',\mathbf{v},\mathbf{z}) = 1$ *iff either* $\pi_{k\langle\mathbf{z}\rangle}$ *or* $\varrho_{k\langle\mathbf{z}\rangle}$ *fires. In addition, the remaining units* $\pi_{k\langle\mathbf{c}\rangle}$, $\varrho_{k\langle\mathbf{c}\rangle} \in \nu_3$ *for* $\mathbf{c} \neq \mathbf{z}$ *are passive.*

Thus, a binary encoding $\mathbf{f}(a,\mathbf{u}',\mathbf{v},\mathbf{z})$ of the new state of automaton A is computed as disjunctions (18) over $\mathbf{c} \in \{0,1\}^{p_3}$ for $k = 1,\ldots,p$ by units from $\nu_0 \setminus \{\text{in}\}$ (which rewrite the code of the old state of A) using the recurrent connections leading from neurons of ν_3. After re-indexing the units in layer $\nu_0 \setminus \{\text{in}\} = \{1,\ldots,p\}$ properly, for each $k = 1, \dots, p$, the kth disjunction is implemented by weights $w(\pi_{k\langle\mathbf{c}\rangle}, k) = w(\varrho_{k\langle\mathbf{c}\rangle}, k) = 1$ for every $\mathbf{c} \in \{0,1\}^{p_3}$, and threshold $h(k) = 1$, according to lemma 8.

5.6 Computational Dynamics and Complexity of N

Now we specify the computational dynamics of neural network N simulating the finite automaton A. At the beginning, the states of neurons from $\nu_0 \setminus \{\mathrm{in}\}$ are placed in an initial state of A. Each bit x_i ($1 \leq i \leq n$) of input word $\mathbf{x} = x_1, \dots, x_n$, which is read by input neuron $\mathrm{in} \in \nu_0$ at time instant $\tau(i-1)$ (i.e. $y_{\mathrm{in}}^{(\tau(i-1))} = x_i$), is being processed by N within the desired period of $\tau = d+4 = O(p2^{p_1}/e) = O(\sqrt{2^p}/e) = O(\sqrt{m}/e)$ time steps. The states of neurons in N are successively updated in the order following the architecture of layers. Thus, we define sets α_t of units updated at time instants $t \geq 1$ as $\alpha_{\tau(i-1)+1} = \nu_1$, $\alpha_{\tau(i-1)+j+1} = \nu_{12} \cup \nu_2$ for $j = 1, \dots, d+1$, $\alpha_{\tau(i-1)+d+3} = \nu_{12} \cup \nu_2 \cup \nu_3$, and $\alpha_{\tau i} = \nu_0 \setminus \{\mathrm{in}\}$, for $i = 1, \dots, n$. Eventually, the output neuron $\mathrm{out} \in \nu_0$ signals at time instant τn whether input word $\mathbf{x}$ belongs to underlying language L, that is, $y_{\mathrm{out}}^{(\tau n)} = 1$ iff $\mathbf{x} \in L$.

The size of N simulating the finite automaton A with m states can be expressed as $s = |\nu_0|+|\nu_1|+|\nu_2|+|\nu_3| = p+1+2^{p_2}+2+8p2^{p_1}+2(d+1)+p2^{p_3} = O(\sqrt{2^p}) = O(\sqrt{m})$ in terms of m according to formulas (5)–(7), which matches the known lower bound (Horne et al., 1996; Indyk, 1995). Finally, energy consumption can be bounded for particular layers as follows. Layer ν_0 can possibly require all $p+1$ units to fire for storing the binary encoding of a current automaton's state (see section 5.1). Moreover, there is only one active unit among neurons in ν_{11} which serve for evaluating all possible monomials over variables $\mathbf{v}$ according to lemma 2, and also only one control unit from

$\nu_{12} \cup \nu_{22}$ fires at one time instant by lemma 6. In addition, we know that the energy consumption by ν_{21} is at most $e + 2$ (see section 5.4), and at most p neurons among $\pi_{k\langle \mathbf{c} \rangle}, \varrho_{k\langle \mathbf{c} \rangle}$ from ν_3 fire (one for each $k = 1, \ldots, p$) according to lemma 8. Altogether, the global energy consumption of N is bounded by $e + 2p + 5 = O(e + \log s) = O(e)$ as $e = \Omega(\log s)$ is assumed. This completes the proof of theorem 1. $\square$

6 The Lower Bound

In this section, we will show lower bounds on the energy complexity of neural networks implementing finite automata. For this purpose, we will employ the technique due to Uchizawa et al. (2008) which is based on communication complexity (Kushilevitz et al., 1997). Assume that $f : \{0,1\}^n \times \{0,1\}^n \longrightarrow \{0,1\}$ is a Boolean function whose value $f(\mathbf{x}, \mathbf{y})$ has to be computed by two players with unlimited computational power, each receiving only his/her part of the input $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$, respectively, while they wish to exchange with each other the least possible number of bits. In particular, they communicate according to a randomized protocol additionally making use of the same public random bit string. For any error probability ε satisfying $0 \leq \varepsilon < 1/2$, the *communication complexity* $R_\varepsilon(f)$ of function f is defined to be the maximum number of bits needed to be exchanged for the best randomized protocol to make the two players compute the correct value of $f(\mathbf{x}, \mathbf{y})$ with probability at least $1 - \varepsilon$, for every input assignment $\mathbf{x}$ and $\mathbf{y}$.

It is well known (Kushilevitz et al., 1997) that almost all Boolean functions f of $2n$

variables have large communication complexity

$$R_\varepsilon(f) = \Omega\left(n + \log\left(\frac{1}{2} - \varepsilon\right)\right) \tag{20}$$

for any error probability ε such that $0 \leq \varepsilon < 1/2$. An example of a particular function that meets condition (20) is the Boolean inner product $IP_n : \{0,1\}^{2n} \longrightarrow \{0,1\}$, defined as

$$IP_n(x_1, \ldots, x_n, y_1, \ldots, y_n,) = \bigoplus_{i=1}^{n} (x_i \wedge y_i)\,. \tag{21}$$

On the other hand, Uchizawa et al. (2008) proved the upper bound on the communication complexity of Boolean function f in terms of the size, depth, and energy complexity of a feedforward network computing f:

Theorem 2 (Uchizawa et al. (2008)) *If a Boolean function $f : \{0,1\}^{2n} \longrightarrow \{0,1\}$ can be computed by a threshold circuit of size S, depth d, and energy complexity E, then*

$$R_\varepsilon(f) = O\left((E + d)(\log n + (E+1)^d \log S\right) \tag{22}$$

for error probability

$$\varepsilon = \frac{1}{2} - \frac{1}{4S^{3(E+1)^d}}\,. \tag{23}$$

The lower and upper bounds on the communication complexity (20) and (22), respectively, are put together in the following lemma:

Lemma 9 *Let $f : \{0,1\}^{2n} \longrightarrow \{0,1\}$ be a Boolean function of $2n$ variables whose communication complexity satisfies condition (20), which can be computed by a threshold circuit of size S, depth d, and energy complexity E such that $n = O(S)$ and $d = O(E)$. Then $n = O(E^{d+1} \log S)$.*

Proof: It follows from condition (20) applied to formula (23) that there is a constant $c_\ell > 0$ such that

$$R_\varepsilon(f) \geq c_\ell \left(n - 3(E+1)^d \log S - 2\right) \tag{24}$$

for sufficiently large n. On the other hand, formula (22) together with $n = O(S)$ and $d = O(E)$ gives

$$R_\varepsilon(f) \leq c_u E^{d+1} \log S \tag{25}$$

for some constant $c_u > 0$, as the term $(1 + 1/E)^{E+1}$ is bounded. Putting inequalities (24) and (25) together, we get

$$n \leq \frac{c_u}{c_\ell} E^{d+1} \log S + c_\ell \left(3(E+1)^d \log S + 2\right) \tag{26}$$

which implies $n = O(E^{d+1} \log S)$. $\square$

Now we will formulate the result providing the lower bound $e \geq s^{c/\tau}$ (for some constant $c > 0$ and for infinitely many s) on the energy complexity e of a recurrent neural network of size s neurons implementing a given finite automaton with time overhead τ such that $\tau^\tau = o(s)$. This means the lower bound is valid for less than sublogarithmic time overheads.

Theorem 3 *Let* $\tau \log \tau = o(\log s)$. *There exists a neural network of size s neurons simulating a finite automaton with time overhead τ per one input bit which needs energy e such that* $\log e = \Omega_\infty \left(\frac{1}{\tau} \log s\right)$.

Proof: Let N be a neural network of size s neurons simulating a finite automaton A with time overhead τ per one input bit. The states of A are represented by the 2^{s-1} states of N (excluding the input neuron in) and the transition function of A is computed by N

within τ time steps. Clearly, network N can be "unwound" into a threshold circuit C of depth $d = \tau$ and size $S = \tau s$ which implements the transition function of A so that each layer is a copy of N (Savage, 1972). Thus, the states of neurons in the ith layer of C coincide with the network state $\mathbf{y}^{(k\tau+i)}$ for $0 \leq i \leq \tau$, when the new state $\mathbf{y}^{((k+1)\tau)}$ of A is produced from the old one $\mathbf{y}^{(k\tau)}$ including the current input bit. Hence, the energy complexity of C is a τ multiple of the energy consumed by N, that is, $E = \tau e$.

As component $f_k : \{0,1\}^s \longrightarrow \{0,1\}$ (for $1 \leq k \leq s$) of the transition function defining A can be arbitrary, there is a neural network N simulating A such that f_k implemented by C has large communication complexity satisfying condition (20). Moreover, $n = \lceil s/2 \rceil = O(S)$ and $d = \tau = O(E)$, which meets the remaining assumptions of lemma 9. It follows that

$$\lceil s/2 \rceil = n = O\left(E^{d+1} \log S\right) = O\left((\tau e)^{t+1} \log \tau s\right) \tag{27}$$

according to lemma 9. On the contrary, suppose that

$$\log e = o\left(\frac{\log s}{\tau}\right) . \tag{28}$$

We will prove that $(\tau e)^{\tau+1} \log \tau s = o(s)$ which contradicts equation (27). For this purpose, it suffices to show that $\log((\tau e)^{\tau+1} \log \tau s) = o(\log s)$. This can be rewritten as $(\tau + 1) \log \tau + (\tau + 1) \log e + \log\log \tau + \log\log s = o(\log s)$ which follows from the assumption of the theorem and equation (28), completing the argument. $\Box$

In the following corollary we will present the lower bounds on energy complexity in terms of the network size for selected cases of sublogarithmic time overhead.

Corollary 1

1. If $\tau = O(1)$, then $e \geq s^\delta$ for some δ such that $0 < \delta < 1$ and for infinitely many s.

2. If $\tau = O(\log\log s)$, then $e = \Omega_\infty\left(s^{\frac{1}{\log^\delta s}}\right) = \Omega_\infty\left(2^{\log^{1-\delta} s}\right)$ for any δ such that $0 < \delta < 1$.

3. If $\tau = O(\log^\alpha s)$ for some $0 < \alpha < 1$, then $e = \Omega_\infty\left(s^{\frac{\log\log s}{\log^\delta s}}\right) = \Omega_\infty\left((\log s)^{\log^{1-\delta} s}\right)$ for any δ such that $\delta > \alpha$.

Proof:

1. For $\tau = O(1)$, the assumption $\tau\log\tau = o(\log s)$ trivially holds and the proposition follows straightforwardly from theorem 3.

2. For $\tau = O(\log\log s)$, there is $c_u > 0$ such that for all but finitely many s we have $\tau\log\tau \le c_u(\log\log s)\log\log\log s + (c_u\log c_u)\log\log s = o(\log s)$. According to theorem 3, there is $c_\ell > 0$ such that $\log e \ge \frac{c_\ell \log s}{c_u \log\log s}$ for infinitely many s. On the contrary, suppose that $e = o\left(s^{\frac{1}{\log^\delta s}}\right)$ for some δ satisfying $0 < \delta < 1$, which implies $\log^{1-\delta} s \ge \frac{c_\ell \log s}{c_u \log\log s}$ leading to a contradiction $\frac{\log\log s}{\log^\delta s} \ge \frac{c_\ell}{c_u} > 0$.

3. If $\tau = O(\log^\alpha s)$ for some $0 < \alpha < 1$, then there is $c_u > 0$ such that for all but finitely many s we have $\tau\log\tau \le c_u\left(\log^\alpha s\right)\log\log^\alpha s + (c_u\log c_u)\log^\alpha s = o(\log s)$. According to theorem 3, there is $c_\ell > 0$ such that $\log e \ge \frac{c_u}{c_\ell}\log^{1-\alpha} s$ for infinitely many s. On the contrary, suppose that $e = o\left(s^{\frac{\log\log s}{\log^\delta s}}\right)$ for some δ satisfying $\delta > \alpha$, which implies $(\log\log s)\log^{1-\delta} s \ge \frac{c_u}{c_\ell}\log^{1-\alpha} s$ leading to a contradiction $\frac{\log\log s}{\log^{\delta-\alpha} s} \ge \frac{c_u}{c_\ell} > 0$. $\square$

We can compare the lower bounds on energy complexity of simulating the finite automata by neural nets presented in corollary 1 to the respective upper bounds provided by theorem 1. For the constant time overhead $\tau = O(1)$, the construction from theorem 1 achieves the energy consumption of $e = O(s)$, while any simulation requires energy $e \ge s^\delta$ for some constant δ such that $0 < \delta < 1$ and for infinitely many s, according to corollary 1. Similarly, for the time overhead of $\tau = O(\log^{\alpha} s)$ where $0 < \alpha < 1$, we have the upper bound of $e = O\left(s/\log^{\alpha} s\right)$ which compares to the lower bound of $e = \Omega_{\infty}\left(s^{\frac{\log\log s}{\log^{\delta} s}}\right)$. Clearly, there are still gaps between these lower and upper bounds, respectively, which need to be eliminated.

7 Conclusions

We have, for the first time, applied the energy complexity measure to recurrent neural nets. This measure has recently been introduced and studied for feedforward perceptron networks. The binary-state recurrent neural networks recognize exactly the regular languages so we have investigated their energy consumption of simulating the finite automata with the asymptotically optimal number of neurons. We have presented a low-energy implementation of finite automata by optimal-size neural nets with the tradeoff between the time overhead for processing one input bit and the energy varying from the logarithm to the full network size. We have also achieved lower bounds for the energy consumption of neural finite automata which are valid for less than sublogarithmic time overheads and are still not tight. An open problem remains for further research whether these bounds can be improved. In addition, we have so far assumed the worst case energy consumption while the average case analysis would be another challenge.

Acknowledgments

The presentation of this paper benefited significantly from valuable comments of anonymous reviewers. Author's research was done with institutional support RVO: 67985807 and partially supported by the grant of the Czech Science Foundation No. P202/10/1333.

References

Alon, N., Dewdney, A.K., & Ott, T.J. (1991). Efficient simulation of finite automata by neural nets. *Journal of the ACM*, *14*(2), 495–514.

Horne, B.G., & Hush, D.R. (1996). Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, *9*(2), 243–252.

Indyk, P. (1995). Optimal simulation of automata by neural nets. In E.W. Mayr, & C. Puech (Eds.), *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1995), LNCS 900* (pp. 337–348). Berlin: Springer-Verlag.

Kushilevitz, E., & Nisan, N. (1997). *Communication Complexity*. Cambridge: Cambridge University Press.

Lennie, P. (2003). The cost of cortical computation. *Current Biology*, *13*(6), 493–497.

Lupanov, O. (1973). On the synthesis of threshold circuits. *Problemy Kibernetiki*, *26*, 109–140.

Minsky, M. (1967). *Computations: Finite and infinite machines.* Englewood Cliffs, NJ: Prentice-Hall.

Orponen, P. (1997). Computing with truly asynchronous threshold logic networks. *Theoretical Computer Science*, *174*(1-2), 123–136.

Savage, J.E. (1972). Computational work and time on finite machines. *Journal of the ACM*, *19*(4), 660–674.

Siegelmann, H.T., & Sontag, E.D. (1995). Computational power of neural networks. *Journal of Computer System Science*, *50*(1), 132–150.

Šíma, J. (2013). A low-energy implementation of finite automata by optimal-size neural nets. In V. Mladenov, P.D. Koprinkova-Hristova, G. Palm, A.E.P. Villa, B. Appollini, & N. Kasabov (Eds.), *Proceedings of the 23rd International Conference on Artificial Neural Networks (ICANN 2013), LNCS 8131* (pp. 114–121). Berlin: Springer-Verlag.

Šíma, J., & Orponen, P. (2003). General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, *15*(12), 2727–2778.

Šíma, J., & Wiedermann, J. (1998). Theory of neuromata. *Journal of the ACM*, *45*(1), 155–178.

Suzuki, A., Uchizawa, K., & Zhou, X. (2011). Energy and fan-in of threshold circuits computing mod functions. In M. Ogihara, & J. Tarui (Eds.), *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation (TAMC 2011), LNCS 6648* (pp. 154–163). Berlin: Springer-Verlag.

Uchizawa, K., Douglas, R., & Maass, W. (2006). On the computational power of threshold circuits with sparse activity. *Neural Computation*, *18*(12), 2994–3008.

Uchizawa, K., Nishizeki, T., & Takimoto E. (2010). Energy and depth of threshold circuits. *Theoretical Computer Science*, *411*(44-46), 3938–3946.

Uchizawa, K., & Takimoto, E. (2008). Exponential lower bounds on the size of constant-depth threshold circuits with small energy complexity. *Theoretical Computer Science*, *407*(1-3), 474–487.

Uchizawa, K., & Takimoto, E. (2011a). Lower bounds for linear decision trees via an energy complexity argument. In F. Murlak, & P. Sankowski (Eds.), *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS 2011), LNCS 6907* (pp. 568–579). Berlin: Springer-Verlag.

Uchizawa, K., Takimoto, E., & Nishizeki, T. (2011b). Size-energy tradeoffs for unate circuits computing symmetric Boolean functions. *Theoretical Computer Science*, *412*(8-10), 773–782.