PERGAMON

Neural Networks 12 (1999) 593–600

Neural Networks

Contributed article

Nine switch-affine neurons suffice for Turing universality

H.T. Siegelmann[a,*], M. Margenstern[b]

[a]Faculty of Industrial Engineering and Management, Technion-Israel Institute of Technology, Haifa 32000, Israel
[b]Institut Universitaire de Technologie de Metz, Université de Metz, L.R.I.M. (Laboratoire de Recherches en Informatique de Metz), 57045 Metz Cedex 01, France

Received 25 November 1996; received in revised form 9 February 1999; accepted 9 February 1999

Abstract

In a previous work Pollack showed that a particular type of heterogeneous processor network is Turing universal. Siegelmann and Sontag (1991) showed the universality of homogeneous networks of first-order neurons having piecewise-linear activation functions. Their result was generalized by Kilian and Siegelmann (1996) to include various sigmoidal activation functions. Here we focus on a type of high-order neurons called switch-affine neurons, with piecewise-linear activation functions, and prove that nine such neurons suffice for simulating universal Turing machines. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Recurrent neural networks; Turing machine; Universal computation; Tag systems

1. Introduction

The current notion of universal computation is based on the model of Turing and the thesis by Church. Analog neural networks are frequently viewed as abstract functional devices able to perform computations. It is thus natural to consider analog networks in terms of their computational power in accordance with the classical discrete Turing model.

In this work our model is an analog recurrent (asymmetric) network of neurons. Each neuron x_i computes a polynomial P_i of its inputs with rational coefficients; the resulting scalar passes through a simple piecewise nonlinearity. The update of a neuron can be written as

$$x_i := \sigma(P_i(x)), \tag{1}$$

where P_i is a polynomial and σ is the saturated-linear function described by

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } x > 1 \end{cases}. \tag{2}$$

Although the description of each neuron is simple and finite, it may compute with unbounded precision. This assumption of a continuous range of values in the neurons does not have to describe reality; the network should rather be considered a mathematical formulation of neurons having many possible activation values. We note that, except for the certain type of "weak robustness" described by Siegelmann and Sontag (1994) the network is not immune against noise, and typically, in a noisy environment, it will not be computationally stronger than finite automata (Casey, 1996; Maass & Orponen, 1998), or even definite language recognizers (Maass & Sontag, 1999).

Our high-order neurons are constrained into *switch-affine* ones. In such a neuron, the function P_i (from Eq. (1)) is a sum of terms; each is a monomial of neuron and input variables adhering to the following constraint: a variable appears with degree 1 in the term, and at most one variable in each term takes a value in (0,1), whereas all the others are in {0,1}. The latter neurons can be considered Boolean or switch variables, and hence the name *switch-affine*.

As a result of the finiteness in the description of the model and its constrained set of operations, these networks are computationally bounded from above by the Turing model. This bound is reached: it is known that such analog networks are Turing universal even when the polynomial is a linear function (Siegelmann & Sontag, 1991; Sun, Chen, Lee, & Giles, 1991). The number of the neurons required for universality with first-order neurons was estimated at 886 (Siegelmann and Sontag, 1995), and in later papers was reduced to 96 (Koiran, Cosnard, & Garzon, 1994) and down to 25 (Indyk, 1995). (The work of Pollack (1987) on the universality of "neuring machines" should be recalled

* Corresponding author. Tel: + 972-4-8294425.
E-mail address: iehava@ie.technion.ac.il (H.T. Siegelmann)

for its originality, although it consists of two different types of neurons and uses discontinuous activation functions. Thus Pollack's model is heterogeneous and not an analog. An intermediate model between Pollack's and the current one appears in Gavaldà and Siegelmann (1999). Also worth mentioning is the pioneering work by Moore (1990) which demonstrates another, though related, finite dimensional construction of universal analog systems.)

Our main theorem states that:

Theorem 1. *There is a universal neural network with nine switch-affine neurons which is Turing universal.*

The proof is constructive and an example network appears in Fig. 2.

The article is organized as follows. We first recall the *tag-systems* model—a universal machine which is the basis of our proof (Section 2), and then we construct a universal neural network with nine switch-affine neurons which simulates a tag system (Section 3).

2. Tag systems

In the late fifties and early sixties much work was conducted to find small Turing machines that are still universal. Shannon (1956) proved the existence of universal Turing machines with only two states or with only two letters. Minsky suggested a machine with 4 symbols and 7 states (Minsky, 1967), which was the world record until 1982. Then, Yuri Rogozhin (Rogozhin, 1982, 1996) established small universal Turing machines with a number of symbols and states of (2,18), (3,10), (4,6), (5,5), (7,4), (10,3), (18,2).

All the above results by Minsky and Rogozhin were obtained by simulating tag-systems, which are a computational model due to Post (1965). As proved by Coke–Minsky (Minsky, 1967), tag-systems can simulate a universal Turing machine, although with a significant computational slow-down, and thus have universal computational power. We take advantage of this same model to find a very small universal neural network.

We take here the definitions and illustrations of (Margenstern, 1995). A p-tag-system, with p a fixed positive integer, is a calculus which is associated with a mapping from alphabet A into A^*, the set of finite-length words on A. The image of a $\in A$ is its *production*. One step in the computation consists of performing the following three operations, illustrated below for $p = 2$.

- a_i, the first letter of the *tagged* word submitted to the tag-system, is memorized;
- the first p letters of the word are erased;
- P_i, the production associated with a_i is appended at the end of what remains from the tagged word.

This process is repeated on the word thus obtained until either the tagged word has fewer than p letters or the first letter of the tagged word is a *halting* letter. Halting letters are, by definition, distinguished letters of A for which the computation halts. A single halting letter is enough, which will be assumed in the sequel.

The Coke–Minsky theorem (see, for instance, Minsky (1967) states that any deterministic Turing machine can be simulated by a 2-tag-system. On the contrary, Wang proved that the halting problem is decidable for any 1-tag-system (Wang, 1963). It is worth noticing that tag-systems yielded by Coke–Minsky theorem happen to halt only by meeting the halting letter on data encoding a Turing configuration. In particular, this means that the tagged word has always at least three letters. This will be assumed in our proof. From now on, say simply *tag-system* for 2-tag-system.

Here is an example of a tag-system with a tag-system computation:

$$\boxed{\begin{array}{l} a \to b \\ b \to bc \\ c \to ! \end{array}} \qquad \begin{array}{l} \text{bbb} \\ \quad\text{bbc} \\ \qquad\text{cbc} \\ \qquad\quad\text{c} \end{array} \tag{3}$$

Universal tag-systems can be obtained either by directly simulating the computation of a universal Turing machine on the alphabet {0,1} as in Coke–Minsky (Minsky, 1967), or by directly simulating a two-register machine computation. As they do not directly concern our purpose, details are omitted here. We only note that tag systems suffer a significant slow-down relative to the standard Turing machines, and thus our result proves only Turing universality and should not be interpreted complexity-wise as a Turing equivalent (Fig. 1).

3. Turing machine simulation with nine neurons

Denote by ω the `tag` word of the input, by **tag** its encoding (to be specified in Section 3.1), by l the number of bits of the encoding, and by `table` the encoding of the internal productions of the tag system. Our universal network consists of six Boolean neurons (with value always in {0,1}) and three continuous neurons. The continuous valued

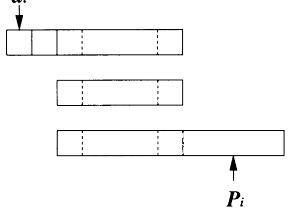

Fig. 1. Tag system.

$$\begin{aligned}
W \quad := \quad & \sigma[(2I_W + 1 + W)\tfrac{I_v}{4} \\
& + C_1(1 - C_2)(1 - E_r)\,W \\
& + C_1(1 - C_2)E_r[(4W - 1)(1 - E_W) + (16W - 7)E_W] \\
& + (1 - C_1)C_2[W + (1 - E_K)\tfrac{1}{4}\,.\,L + E_K\tfrac{3}{4}\,L] \\
& + C_1C_2[(4W - 1)(1 - E_W) + (4W - 3)E_W]]
\end{aligned}$$

$$\begin{aligned}
L \quad := \quad & \sigma[\tfrac{I_{E_W}}{4} + \tfrac{I_v}{4}\,L \\
& + C_1(1 - C_2)(1 - E_r)\,L \\
& + C_1(1 - C_2)E_r[(1 - E_W)\,4\,L + E_W\,16\,L] \\
& + (1 - C_1)C_2\,\tfrac{1}{4}\,L \\
& + C_1C_2\,4\,L]
\end{aligned}$$

$$\begin{aligned}
K \quad := \quad & \sigma[(I_e + C_1C_2E_W)\mathrm{table} \\
& + (C_1 + C_2 - 2C_1C_2) \\
& \quad [(6K - 1(1 - E_K) - 3E_K)(1 - E_r) + (36K - 23)E_r] \\
& - C_1C_2(1 - E_W)]
\end{aligned}$$

$$\begin{aligned}
C_1 &:= \sigma[I_e + C_1(1 - C_2)(1 - E_WE_r) + (1 - C_1)C_2E_r + C_1C_2] \\
C_2 &:= \sigma[C_1(1 - C_2)E_WE_r + C_2(1 - C_1E_W)] \\
E_W &:= \sigma[16W - 6 - 16E_WC_1C_2 - 16E_WE_r(1 - C_2) - 16E_rC_2(1 - C_1)] \\
E_K &:= \sigma[36K - 8 - 36(E_K + E_r)] \\
E_r &:= \sigma[216K - 57 - 216(E_K + E_r)] \\
H &:= \sigma[(1 - C_1)C_2(6K - 4)(1 - I_v)]
\end{aligned}$$

Fig. 2. The universal neural network.

neurons are W, L and K: W encodes the tagged word, L corresponds to the length l of the encoding tag in W (it actually stores the value of 4^{-l}), and K holds **`table`**, an encoding of the production rules. We next describe the role of the Boolean neurons. C_1 and C_2 are two *control neurons*; their value indicates which of the four stages in the simulation (to be discussed in Section 3.3) is being processed. During the network operation strings of digits are read and popped out of W and K. E_W, E_K and E_r assist in recognizing the state of the popped sequence. E_W signifies that the last stream of consecutive digits read out from W is soon to form a code of a letter (from A). E_K and E_r relate to K in a similar manner; E_K signifies that the network is to finish reading an encoded letter from a production rule, while E_r signifies that the stream will soon form an encoding of a whole production rule. H is the Halting neuron.

The network uses four binary input channels: $I\omega$, I_v, I_b, I_e. Channel $I\omega$ transfers the binary tagged word ω as a sequence of bits and it defaults to 0 when it does not transmit. Channel I_v ('v' is for validation) is set to 1 when $I\omega$ starts sending its input and remains set to 1 until the last input digit of the tagged word has reached the network; afterwards, I_v turns to 0 and remains forever with that value. Channels I_b and I_e are in the form of impulse-response: I_b ('b' for beginning of input) is 1 only with the first 1 bit of I_v, and I_e ('e' for end of input) is 1 only at the time when I_v turns from 1 to 0. Both I_b and I_e are in 0 in all other times.

For the sake of readability we first describe the encoding we use for W, L and K (Section 3.1) and only then go to the details of the simulation.

3.1. Encodings

Let us encode the letters of the tag-system alphabet, let it be $A = \{a_1, \ldots, a_N\}$, where N is the number of letters in A, in the following way: a_i is encoded as $\mathrm{cod}(a_i) = 1^i3$, where the symbol "3" has the role of a *right delimiter* in order to distinguish the letters of A. Recall that we have only one halting letter, this will be a_N, the last one in the alphabet.

More generally, the words in the alphabet A are encoded in the following way. Let $m = b_1 \ldots b_k$ be a word on A with

```
$W$ := if $I_v = 1$
      then $\sigma[\frac{(2I_W+1+W)}{4}]$
      else if $C_1 = 1$ and $C_2 = 0$
          then if $E = 0$
              then $W$
              else  % now, $E = 1$
                  if $E_W = 0$
                      then $4W - 1$
                      else $16W - 7$
                  end if
              end if
          else if $C_1 = 0$ and $C_2 = 1$
              then if $D = 0$
                  then $W + \frac{1}{4}.L$
                  else $W + \frac{3}{4}.L$
                  end if
              else  % now $C_1 = 1$ and $C_2 = 1$
                  if $B = 0$
                      then $4W - 1$
                      else $4W - 3$
                      end if
              end if
          end if
      end if
```

Fig. 3. Update of neuron.

$b_j = a_{i_j}$, then m is encoded as $cod(m) = cod(a_{i_1})\ldots cod(a_{i_k}) = 1^{i_1}3\ldots1^{i_k}3$. The encoding of the productions is similar to that of the tagged word. Let $P_1,\ldots,P_{N-1}$ be the productions of the considered tag-system in which P_N is the single halting production that can be

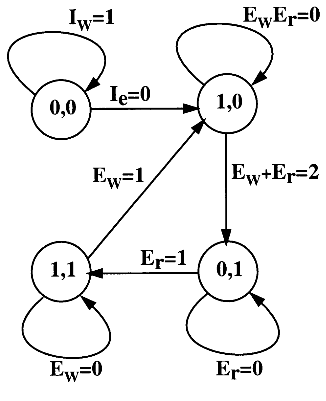

Fig. 4. The control neurons (C_1, C_2).

considered as empty, then this set is encoded by $135cod(P_1)5\ldots cod(P_{N-1})55$.

However, as the neurons are not able to store sequences but rather numbers, the neuron W will keep a base-4 representation of the encoding of the tagged word, and K will constitute a base-6 representation of the encoded productions. W stores the encoding of a tagged word m as the real number $w(m)$ defined by

$$\sum_{j=1}^{k} \frac{1}{4^{(\Sigma_{u<j}|i_u|+1)}} \left(\left(\sum_{u=1}^{|i_j|} \frac{1}{4^u} \right) + \frac{3}{4^{|i_j|+1}} \right),$$

where $|i_j|$ denotes the number of bits in the encoding of a_{ij}. The constant **table** is defined similarly but in base 6. Notice that similar Cantor-like encoding was introduced for example in Siegelmann and Sontag (1994) and its advantage will become clear later in this proof.

3.2. The read-in stage

The initial configuration of our neural network is 0 in all neurons. The network is then activated by the input signals. The process of reading in the input and initiating the network is called the *read-in* stage. This stage occurs when the control neurons have the values (C_1, C_2) = (0,0). It updates by the following equations:

$$W := (2I_W + 1 + W)\frac{1_v}{4},$$

$$L := \frac{I_b}{4} + \frac{I_v}{4}L,$$

$$K := I_e \cdot \texttt{table},$$

$$C_1 := I_e.$$

The first update equation describes the on-line accumulation of cod(ω) in neuron W. At the same time the length l, stored in L, is incremented by 1 each step for as long as $I_v = 1$. When this process is completed, I_v becomes 0, and simultaneously I_e is set to 1. This yields the loading of **table** to K and neuron C_1 is set to 1 as a preparation for the next stage, i.e. the first stage of a computation cycle. The content of the neurons at this moment is as follows:

W	L	K	C_1	C_2	E_W	E_K	E_r	H
tag	$4^{-\ell}$	**table**	1	0	0	0	0	0

We are now ready to describe our nine-neuron network and prove that it simulates a tag-system and thus is universal. The update equations of the neurons are presented in Fig. 2. For clarity, Fig. 3 demonstrates the switch-affine update equation of neuron W in terms of an 'if-then-else' statement with an affine update. The neural update equations

```
Global rules:

$E_K$  :=  if $E_K \bigvee E_r$ then 0 else $[k_1k_2 \geq 13_6]$

$E_r$  :=  if $E_K \bigvee E_r$ then 0 else $[k_1k_2k_3 \geq 135_6]$

Stage (1,0):

$W$  :=  if $\neg E_r$ then $W$
         else if $\neg E_W$ then pop($1_4$) from $W$        % $4W - 1$
         else pop($13_4$) from $W$                         % $16W - 13_4$

$L$  :=  if $\neg E_r$ then $L$
         else if $\neg E_W$ then $4L$                       % $l \leftarrow l - 1$
         else $16L$                                         % $l \leftarrow l - 2$

$K$  :=  if $\neg E_r$ then pop($1_6$) from $K$
           if $\neg E_W$ then pop($1_6$) from $K$ else pop($3_6$) from $K$   % $6K - 1; 6K - 3$
         else pop($35_6$) from $K$                          % $36K - 35_6$

$C_1$  :=  $\neg E_W \bigvee \neg E_r$

$C_2$  :=  $E_W \bigwedge E_r$

$E_W$  :=  if $\neg E_W \bigvee \neg E_r$ then $[W_1W_2 \geq 13_4]$ else 0

$H$  :=  $[k_1 \geq 5_6]$
```

Fig. 5. Update rules during stage (1,0).

are discussed in more detail throughout the rest of this section.

3.3. Computational cycles

Each step of the tag-system computation will be simulated by a cycle. Each cycle is further split into three *stages* corresponding to the three operations in one step of the tag-system. Cycles are characterized by *starting configurations*, to be described later, and our proof boils down to establishing that starting from any such configuration, a sequence of consecutive applications of our neural network leads to the correct next starting configuration.

The Boolean neurons C_1 and C_2 are devoted to marking these stages. $(C_1,C_2) = (0,0)$ was used only once to indicate the initialization read-in stage leading to the first starting configuration. During a computation cycle, stages occur in the order: (1,0), (0,1), (1,1). As can be seen from Fig. 4 the network turns from stage (1,0) to stage (0,1) by simultaneous changes in E_W and E_r from 0 to 1. It then turns from stage (0,1) to (1,1) as E_r turns from 0 to 1. The network turns back to stage (1,0) for the next cycle as E_W turns from 0 to 1.

In what follows we first shortly describe the operations of the different neurons during the different stages and then continue with detailed demonstrations. Let us start with W. In stage (1,0), the network erases the first encoded letter from W and prepares K to start with the prediction rule associated with the letter removed. In stage (0,1), the letters of the rule associated with the letter just removed are appended to W. In stage (1,1), the network removes the second letter of W.

We next consider the operation of the remaining Boolean neurons: E_W, E_K, E_r and H. The Boolean neuron E_W depends on the first two letters in $W : W_1W_2$. Let us call the test $\sigma(16W - 6)$ the *potential value* of E_W. It is equivalent to the symbolic check of whether $W_1W_2 \geq 13_4$. E_W takes the value 1 when its potential value is 1 except for when stage (1,1) turns to stage (1,0) or when stage (1,0) turns to stage (0,1). E_k and E_r depend on the first letters of K. Define the *potential values* of E_K and E_r to be, respectively, $\sigma(36\mathrm{K} - 8)$ and $(\sigma 216\mathrm{K} - 57)$. These correspond to the symbolic checks $K_1K_2 \geq 13_6$ and $K_1K_2K_3 \geq 135_6$. The Boolean neurons E_K and E_r take the value 1 only if their potential value is 1 and when this potential value is 1, the signal is actually 1 only if both previous values of E_K and E_r were 0. The neuron H takes the value 1 when the first digit in K is 5 and when this happens in stage (0,1). H serves as the halting neuron, as will be described towards the end of the proof. We now turn to describe the individual stages with demonstrations.

During state (1,0) the network erases the first letter in the encoding string of W, and locates in the encoding string of **`table`** the production corresponding to that letter of W. The logical structure of the update operations in stage (1,0) is indicated in Fig. 5. Note the correspondence to the update equations in Fig. 2. There, in each sum, the Boolean-valued control neurons C_1 and C_2 select the update values appropriate to the given stage.

Let us indicate several intermediary configurations which will allow us to prove that stage (1,0) leads from configuration (I), where $\bar{P}_i$ denotes $cod(P_i)$:

(I)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	1^n3x	$4^{-\ell}$	$135\bar{P}_1\ldots5\bar{P}_nz$	1	0	0	0	0	0

to the configuration:

(II)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell+n}$	$\bar{P}_nz$	0	1	0	0	0	0

Indeed, a straightforward application of the network equations allows one to prove the following configuration transitions in accordance with the different cases:

(1)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$11x$	$4^{-\ell}$	$135y$	1	0	0	0	0	0
	$11x$	4^{-1}	$35y$	1	0	0	1	1	0
	$1x$	4^{-1+1}	y	1	0	0	0	0	0

(2a)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$13x$	$4^{-\ell}$	$111y$	1	0	0	0	0	0
	$13x$	$4^{-\ell}$	$11y$	1	0	1	0	0	0
	$13x$	$4^{-\ell}$	$1y$	1	0	1	0	0	0

(2b)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$13x$	$4^{-\ell}$	$131y$	1	0	0	0	0	0
	$13x$	$4^{-\ell}$	$31y$	1	0	1	1	0	0
	$13x$	$4^{-\ell}$	$1y$	1	0	1	0	0	0

(2c)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$13x$	$4^{-\ell}$	$135y$	1	0	$\in$	0	0	0
	$13x$	$4^{-\ell}$	$35y$	1	0	1	1	1	0
	x	$4^{-\ell+2}$	y	0	1	0	0	0	0

with, here, $\in\ = 0$ or 1.

(3a)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$11x$	$4^{-\ell}$	$113y$	1	0	0	0	0	0
	$11x$	$4^{-\ell}$	$13y$	1	0	0	0	0	0

(3b)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$11x$	$4^{-\ell}$	$113y$	1	0	0	0	0	0
	$11x$	$4^{-\ell}$	$31y$	1	0	0	1	0	0
	$11x$	$4^{-\ell}$	$1y$	1	0	0	0	0	0

It is now possible to prove that applying the network equations starting from configuration (I), the network arrives at configuration (II) which is the starting configuration of stage (0,1). The proof goes by induction on n. Case $n = 1$ is proved by subtable 2c. Case $n = 2$ is proved by applying first subtable (1) to which subtables 2a to 2c successively apply, subtables 2a and 2b show that the letters of $\bar{P}_1$ are all erased during stage (1,0). The proof of the induction step immediately ensues from subtables 3a, 3b and 2c.

3.3.1. Stage (0,1)

During stage (0,1) the network appends to W the first production occurring in **`table`**, and progressively erases the production each time a letter is transferred to W. The length L of W is updated accordingly.

The following transition tables indicate the evolution of the neuron states during this stage.

(1)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell}$	$11y$	0	1	$\in$	0	0	0
	$x1$	$4^{-\ell-1}$	$1y$	0	1	$\in$	0	0	0

(2a)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell}$	$131y$	0	1	$\in$	0	0	0
	$x1$	$4^{-\ell-1}$	$31y$	0	1	$\in$	1	0	0
	$x13$	$4^{-\ell-2}$	$1y$	0	1	$\in$	0	0	0

W	L	K	C_1	C_2	E_W	E_K	E_r	H
113113113	4^{-9}	13511135113111355	1	0	0	0	0	0
113113113	4^{-9}	351135113111355	1	0	0	1	1	0
13113113	4^{-8}	1135113111355	1	0	0	0	0	0
13113113	4^{-8}	135113111355	1	0	1	0	0	0
13113113	4^{-8}	35113111355	1	0	1	1	1	0
113113	4^{-6}	113111355	0	1	0	0	0	0
1131131	4^{-7}	13111355	0	1	0	0	0	0
11311311	4^{-8}	3111355	0	1	0	1	0	0
113113113	4^{-9}	111355	0	1	0	0	0	0
1131131131	4^{-10}	11355	0	1	0	0	0	0
11311311311	4^{-11}	1355	0	1	0	0	0	0
113113113111	4^{-12}	355	0	1	0	1	1	0
1131131131113	4^{-13}	0	1	1	0	0	0	0
131131131113	4^{-12}	0	1	1	0	0	0	0
31131131113	4^{-11}	0	1	1	1	0	0	0
1131131113	4^{-10}	13511135113111355	1	0	0	0	0	0

Fig. 6. Neural values of system (3).

(2b)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell}$	$135y$	0	1	$\in$	0	0	0
	$x1$	$4^{-\ell-1}$	$35y$	0	1	$\in$	1	1	0
	$x13$	$4^{-\ell-2}$	y	1	1	0	0	0	0

Notice that the equality $E_W = 0$ in the last configuration occurring in (2b) is given by the term $-16E_rC_1(1\text{-}C_2)$ of the network equation defining E_W.

Hence, by induction on the number of letters in the production to be replicated, and for each letter on the number of 1's contained in its encoding, it is easy to prove that during stage (0,1), the network evolves from configuration

(II)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell}$	$\bar{P}_n z$	0	1	$\in$	0	0	0

to configuration:

(III)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$x\bar{P}_n$	4^{-l-l_n}	z	1	1	$\in$	0	0	0

where ℓ_n is the length of $\bar{P}_n$.

3.3.2. Stage (1,1)

During stage (1,1), the network erases the second letter of W and at each step updates the value of L. Thus a new cycle may start again as soon as the appropriate value for C_1 and C_2 are reached.

The following transition tables describe the evolution during this stage:

(1a)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$11x$	$4^{-\ell}$	y	1	1	0	0	0	0
	$1x$	$4^{-\ell+1}$	0	1	1	0	$\in_1$	$\in_2$	0

Value 1 for E_K is possible: this happens if y begins with 13; if y begins with 135 the value 1 is also reached by E_r, but this has no consequence in further transitions as seen in the next subtables.

(1b)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$11x$	$4^{-\ell}$	0	1	1	0	$\in_1$	$\in_2$	0
	$1x$	$4^{-\ell+1}$	0	1	1	0	0	0	0

(2)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	$13x$	$4^{-\ell}$	y	1	1	0	0	0	0
	$3x$	$4^{-\ell+1}$	0	1	1	1	$\in_1$	$\in_2$	0
	x	$4^{-\ell+2}$	`table`	1	0	0	0	0	0

Hence, by induction on the number of 1s contained in the first letter of W (to be erased), it is easy to prove that during stage (1,1), the network evolves from configuration

(III)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	1^k3x	$4^{-\ell}$	y	1	1	0	0	0	0

to the configuration:

(IV)	W	L	K	C_1	C_2	E_W	E_K	E_r	H
	x	$4^{-\ell+k}$	`table`	1	0	0	0	0	0

which is the starting configuration of a new simulating cycle, i.e. the first configuration of stage (1,0) in the next cycle.

To complete the proof, we have to indicate how the network halts. Recall that stage (1,0) leads to the configuration of **`table`** which starts with the production corresponding to the letter removed from the tag word in W. When the first letter of W is the halting letter (we assumed a single one), by construction of our encoding, the located production is empty (by assumption on the halting production) which is indicated by the fact that **`table`** is then simply .5 in base 6. The update equation of neuron H indicates that this situation is detected at the starting step of stage (0,1). *Q.E.D.*

We close this section by demonstrating the simulation of the tag-system labeled (3) in Section 2. According to the above considerations, the set of productions of this system is encoded as 13511351131111355 and, correspondingly, the value of the constant **`table`** is 0.13511351131111355 in basis 6. If as in Section 2 we take as initial tagged word *bbb*, the value of W after the read-in step will be 0.113113113 in basis 4. The simulating computation of the first cycle appears in Fig. 6.

Acknowledgements

This work was funded by Metz University, Institute of Technology, and partially supported by The Binational US/Israel Foundation, The Fund for Promotion of Research at the Technion, and The VPR Fund at the Technion.

References

Casey, M. (1996). The dynamics of discrete-time computation with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, *8* (6), 1135–1178.

Gavaldà, R., & Siegelmann, H. T. (1999). Discontinuities in recurrent neural networks. *Neural Computation*, *11* (3), 715–745.

Indyk, P. (1995). Optimal simulation of automata by neural nets. In E. W. Mayr & C. Puech, *Lecture Notes in Computer Science: Proceedings of the Twelfth Annual Symposium on Theoretical Aspects of Computer Science* (pp. 337). Munich: Springer.

Kilian, J., & Siegelmann, H. T. (1996). The dynamic universality of sigmoidal neural networks. *Information and Computation*, *128* (1), 48–56.

Koiran, P., Cosnard, M., & Garzon, M. (1994). Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, *132*, 113–128.

Maass, W., & Orponen, P. (1998). On the effect of analog noise in discrete-time analog computations. *Neural Computation*, *10*, 1071–1095.

Maass, W., & Sontag, E. D. (1999). Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. *Neural Computation*, *11* (3), 771–782.

Margenstern, M. (1995). Non-erasing Turing machines: a new frontier between a decidable halting problem and universality. *Lecture Notes in Computer Science*, *911*, 386–397.

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ: Prentice Hall.

Moore, C. (1990). Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, *64*, 2354–2357.

Pollack, J.B. (1987). On connectionist models of natural language processing. PhD thesis, Computer Science Dept., University of Illinois, Urbana.

Post, E. L. (1965). Absolutely unsolvable problems and relatively undecidable propositions—account of an anticipation. In M. Davis, *The undecidable* (pp. 317). New York: Raven Press.

Rogozhin, Y. V. (1982). Sem' universal'nykh mashin Tjuringa. *Matematicheskie Issledovanija*, *69*, 76–90 in Russian.

Rogozhin, Yu. V. (1996). Small universal Turing machines. *Theoretical Computer Science*, *168* (2), 215–240.

Shannon, C. E. (1956). A universal Turing machine with two internal states. In J. McCarthy & C. Shannon (Eds.), *Automata Studies*, (pp. 156). Princeton University Press, NJ: Princeton University.

Siegelmann, H. T., & Sontag, E. D. (1991). Turing computability with neural nets. *Applied Mathematics Letters*, *4* (6), 77–80.

Siegelmann, H. T., & Sontag, E. D. (1994). Analog computation via neural networks. *Theoretical Computer Science*, *131*, 331–360.

Siegelmann, H. T., & Sontag, E. D. (1995). On the computational power of neural nets. *Journal of Computers and System Science*, *50* (1), 132–150.

Sun, G. Z., Chen, H. H., Lee, Y. C., Giles, C. L. (1991). Turing equivalence of neural networks with second order connection weights, Proceedings of the IEEE INNS International Joint Conference on Neural Networks—Seattle, IEEE Press, Piscataway, NJ, 2 1991357-362.

Wang, H. (1963). Tag systems and lag systems. *Mat. Annalen*, *152*, 65–74.