STUDIES IN LOGIC, GRAMMAR AND RHETORIC 8 (21) 2005

Dariusz Surowik
Białystok University

ON THE COMPUTATIONAL POWER OF SOME MODELS OF COMPUTATION

Introduction

In the 1930's mathematicians began to think about: What it means to be able to compute a function? It was the time, when a computability theory started. We may ask the next question: What is a computation? A simple answer would be as follows: a computation involves the mapping of a set of numbers to another set of numbers. But computation involves more. Namely, computation involves the use of finite procedures or algorithms to generate number mappings.

A computation model is an entity that is capable of carrying out computations. One famous example of a computation model is the Turing Machine.

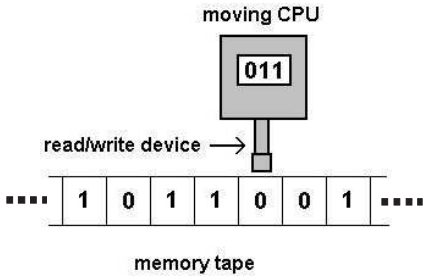
Turing machines

A Turing Machine (TM) is a mathematical model for a computing device. A TM has a potentially infinite tape to hold the input data and to store the results. The tape is divided into cells. Any cell may contain a symbol from a finite alphabet. The TM has also a read/write head which moves along the tape and reads one symbol and replaces it by another. There is some restriction, namely: there can be only finitely many non-blank cells on the tape.

This behavior of the machine is completely determined by three parameters:

1. the state the machine is in,
2. the number on the cell it is scanning, and
3. a table of instructions.

ISBN 83-7431-059-6 ISSN 0860-150X



The table of instructions specifies, for each state and each binary input, what the machine should write, which direction it should move in, and which state it should go into. The table can list only finitely many states, each of which becomes implicitly defined by the role it plays in the table of instructions.

Although a Turing Machine is only a mental construction, but any given Turing Machine can be realized or implemented on a different physical computing devices.

Formal definition of deterministic Turing Machine is as follows:

Definition

A *Turing Machine* $M = (Q, \delta, q_0, F)$, where:

- Q is a finite set of *states*,
- q_0 is a *start state*
- F is a set of *accepting states* and
- δ is a function such that: $\delta : Q \times X \rightarrow Q \times X \times \{L, R\}$, where the *alphabet* X contains a symbol B ("blank").

Definition

Let M be a class of machines. The class of functions computed by M is the set $\{\phi^n_{\mathcal{M}} \in M \text{ and } n \in \mathbb{N}\}$.

The class of functions computed by Turing Machines is the class of partial recursive functions of Kleene[1].

Variants of Turing machine

Nondeterministic Turing Machines

A Nondeterministic Turing Machine (NTM) has a finite number of choices of next move (state, new symbol, and head move) for each state and symbol scanned, i.e., its transition function

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}.$$

[1] Full discussion on this class of functions you may see in Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967, chapter 10.

The following theorem holds:

Theorem

Every non-deterministic TM has an equivalent deterministic TM.

The above theorem says, that deterministic Turing Machines are just as powerful as non-deterministic ones, and so they accept the same languages.

Multi-tape Turing Machine

We can to consider a multi-tape Turing Machine. The transition function of this machine is as follows:

$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$, k: *the number of types*

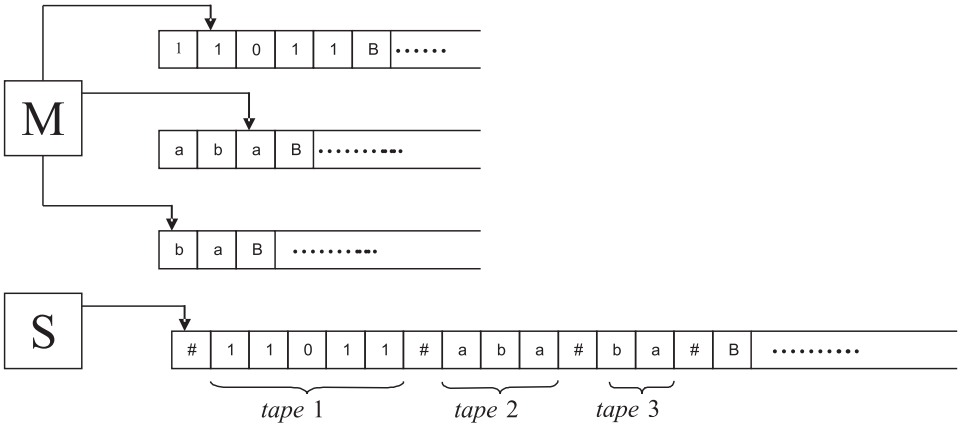
$\delta(q_1, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, L, \ldots, L)$.

More tapes in the machine not increase of computational power of the machine.

Theorem

Every multi-tape Turing Machine has an equivalent single tape Turing Machine[2].

We can emulate a multi-tape Turing Machine with single-tape Turing Machine.



Universal Turing Machine

For the computation of each function, it is necessary to construct a separate Turing Machine. We can build a Turing Machine that can do the

[2] Equivalent only in the sense that the classes of computable functions are the same. From the point of view of complexity classes, these models are not equivalent.

job of any other Turing Machine. This is known as a *Universal Turing Machine*[3] (UTM). The UTM interprets the input symbols on the tape as the program. Every conventional computer is logically (not physically) equivalent to a UTM[4].

Parallel Turing Machines

Parallel Turing machines (PTM) can be viewed as a generalization of cellular automata (CA) where an additional measure called processor complexity can be defined which indicates the "amount of parallelism" used.

The definition of PTM is as follows:

Definition[5]

A *Parallel Turing Machine* consists of a usual one-dimensional Turing tape, on which a number of finite automata are working. It is characterized by an 8-tuple $P = (Q, q_0, F_+, F_-, B, A, \square, \delta)$. Q is the set of states and contains an initial state q_0. The disjoint subsets F_+ and F_- of Q contain the accepting respectively rejecting final states. It is required that $q_0 \notin F_+ \cup F_-$. B is the tape alphabet containing at least the blank symbol $\square$ and the symbols of the input alphabet A.

A configuration of a PTM $P = (Q, q_0, F_+, F_-, B, A, \square, \delta)$ is a pair $c = (p, b)$ of mappings $p : \mathbb{Z} \to 2^{\mathbb{Q}}$ and $b : \mathbb{Z} \to B$ where $p(i)$ is the set of states of the finite automata currently visiting square i and $b(i)$ is the symbol written on it.

Each step of a PTM, i.e. the transition from a configuration c to its successor configuration $c' = (p', b')$ is determined by the transition function $\delta : 2^Q \times B \to 2^{Q \times D} \times B$ where D is the set $\{-1, 0, 1\}$ of possible movements of a finite automaton. In order to compute c', δ is simultaneously applied at all tape positions $i \in \mathbb{Z}$. The arguments used are the set of states of the finite automata currently visiting square i and its tape symbol. Let $(M'_i, b'_i) := \delta(p(i), b(i))$. Then the new symbol on square i in configuration c' is $b'(i) := b'_i$. The set of finite automata on square i is replaced by a new set of finite automata (defined by $M'_i \subseteq Q \times D$) each of which has to change the tape square according to the indicated direction of movement, i.e., $p'(i) := \{q | (q, 1) \in M'_{i-1} \vee (q, 0) \in M'_i \vee (q, -1) \in M'_{i+1}\}$.

---

[3] Wolfram describes a Turing Machine with 2 states and 5 symbols per cell, currently the *smallest known Universal Turing Machine*.

[4] All computer instruction sets, high level languages and computer architectures, including multi-processor parallel computers, can be shown to be UTM-equivalent.

[5] Thomas Worsch, *Parallel Turing Machines With One-Head Control Units And Cellular Automata*.

Thomas Worsch defines for total functions s, t and h from $\mathbb{N}_+$ into $\mathbb{N}_-$, the complexity class $PTM - STP(s,t,h)$ as the family of all languages L for which there is a PTM recognizing L and satisfying, for all $n \in \mathbb{N}_+$, Space$p(n) \leq s(n)$, Time$p(n) \leq t(n)$, and Proc$p(n) \leq h(n)$. He also uses $PTM - ST(s,t)$, and so on. Furthermore he writes $PTM - T(O(t))$ instead of $\bigcup_{t' \in O(t)} PTM - T(t')$ and so on.

There is a close relation between Parallel Turing Machines and Cellular Automata. The following theorem holds:

Theorem[6]

For all functions $s(n) \geq$, $t(n) \geq n$ and $h(n) \geq 1$ where h is fully PTM processor constructible in space s, time t, and with h processors, holds:

- $PTM - STP(O(s), O(t), O(h)) \subseteq CA - ST(O(s), O(t))$
- $CA - ST(O(s), O(t)) \subseteq PTM - STP(O(s), O(st/h), O(h))$
- $PTM - STP(O(s), O(t), O(s)) = CA - ST(O(s), O(t))$

The relations between computational power of Turing Machines and Parallel Turing Machines are explained by the following theorem:

Theorem

For all functions $s(n) \geq n$, $t(n) \geq n$ holds

- $TM - ST(O(s), O(t)) \subseteq PTM - ST(O(s), O(t), O(s))$
- $PTM - ST(O(s), O(t), O(h)) \subseteq TM - ST(O(s), O(t\sqrt{h}))$

It means that computational power of Turing Machines and Parallel Turing Machines is the same, but Parallel Turing Machines are faster than Turing Machines.

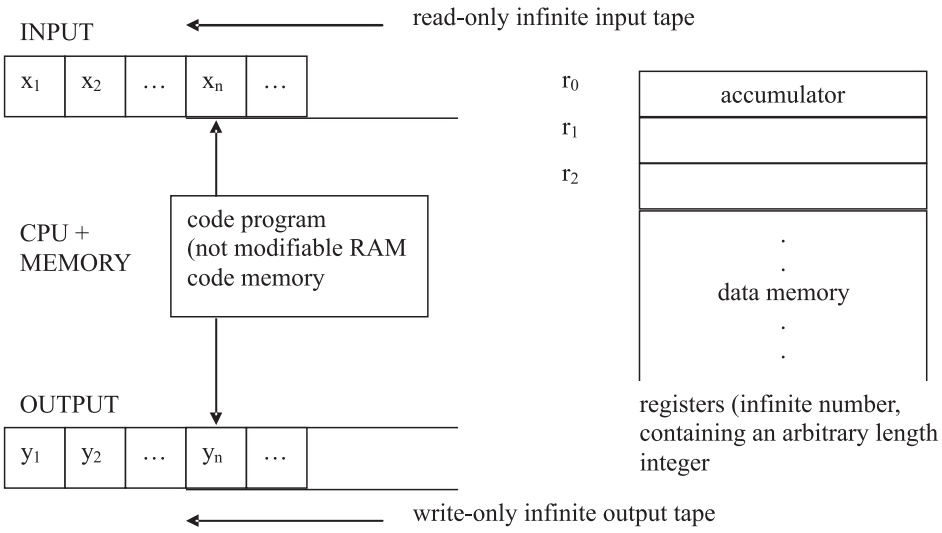
Random access machines[7] (RAMs)

A *Random Access Machine* is an idealized computer with a random access memory consisting of a finite number of idealized registers (i.e., they can hold any sized number) $R_1, R_2, \ldots$ whose contents are strings over some alphabet $\sum_k$ and which has a finite set of machine instructions. The scheme of such machine is as follows:

---

[6] Thomas Worsch, *Parallel Turing Machines...*

[7] AV Aho, JE Hopcropft, JD Ullman, *The Desing and Analysis of Computer Algorithms*, addison-Wesley Publishing Copmpany, London, 1974.



One of a possible (equivalent) set of instructions (optimally labeled):

1. LOAD operand ; loads to accumulator contents of a register
2. STORE operand ; store accumulator to one of the registers
3. ADD operand ; integer addition: AC + op, result → AC
4. SUB operand ; subtraction: AC – op, result in AC
5. MULT operand ; multiplication (note, no overflow)
6. DIV operand ; integer division: AC/op
7. READ operand ; integer from input tape to a register
8. WRITE operand ; contents of a register to an output tape
9. JUMP label ; program counter set to label value
10. JGTZ label ; jump if accumulator greater than zero
11. JZERO (or JLTZ) label ; jump if zero
12. HALT ; stop execution

Other possible computer instructions can be simulated by 12 RAM instructions.

Theorem

Computational power of a RAM machine is the same as a Turing Machine.

This means that a sequential algorithm can be computed by RAM machine. Turing Machines have the same computational power as RAM machines but they have different computational time and different memory complexity.

Theorem (Aho, Hopcropft, Ullman)[8]

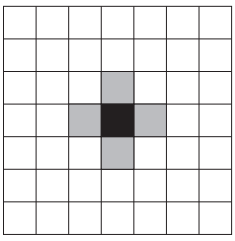
If algorithm is accepted by a RAM machine in time $T(n)$ using logarithmic cost criterion and RAM does not perform multiplications or divisions, then there exists a multi-tape Turing Machine accepting the same algorithm in time $O(T^2(n))$.

This means that although RAM machine has the same computational power as a Turing Machine, a *RAM machine is faster than a Turing Machine.*

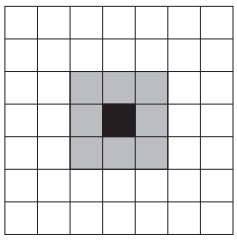
Cellular automata

Cellular Automata were introduced by John von Neumann[9] and Stanislaw Ulam in the late 1940's. From the more practical point of view Cellular Automata was introduced in the late 1960's when John Horton Conway developed the *Game of Life.*

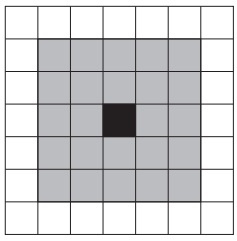
The basic element of a Cellular Automata is the cell. A cell is a kind of a memory element and stores states. In the simplest case, each cell can have the binary states 0 or 1 but in more complex case the cells can have more different states. These cells are arranged in a spatial web – a lattice. The simplest one is the *one dimensional* "lattice", meaning that all cells are arranged in the form of a tape. The most common CA's are built in one or two dimensions. To introduce *dynamic* into the system, there are some rules. These rules define the state of the cells for *the next time step.* In cellular automata a rule defines the state of a cell in dependence of the neighborhood of the cell. Most famous neighborhoods:



von Neumann Neighborhood



Moore Neighborhood



Extended Moore Neighborhood

[8] AV Aho, JE Hopcropft, and JD Ullman, "The Desing and Analysis of Computer Algorithms", addison-Wesley Publishing Copmpany, London, 1974.

[9] Von Neumann proved that the typical feature of living systems and their tendency to reproduce themselves, can be simulated by an automaton with 200,000 cells, if each cell has 29 possible states and the four orthogonal neighboring cells as environment. Although this idea is justified by a mathematically precise proof, it is hard to realize in technical computers.

Definition

Cellular Automata are quadruples (d, Q, N, δ),
where:

d is the *dimension* of the space the cellular automaton works on.

Q is a finite set of *states* of cells.

$N = (x_1, \ldots, x_k)$ is *neighborhood*. It is a k-tuple of distinct vectors of $\mathbb{Z}^d$. The x_i's are the relative positions of the neighbor cells with respect to the cell, whose new state is being computed. The states of these neighbors are used to compute the new state of the center cell by the *local function* of the cellular automaton $\delta : Q^k \to Q$.

Definition

A Cellular Automata simulates a Turing Machine if there is a bijection from the possible instantaneous descriptions of the Turing Machine to the possible instantaneous descriptions of the Cellular Automata, so that if the Cellular Automata is run with initial state M, for a fixed number i, the i^{th} instantaneous description of the Cellular Automata is equal to the i^{th} instantaneous description of the Turing Machine if run with the initial state M.

A Turing Machine can be simulated by Cellular Automata[10].

Theorem[11]

Any $TM[n; m]$[12] can be simulated by a k-states CA where:

1. $k = (n + 1) \cdot m$
2. $k = m + 2n$
3. $k = m + n + 4$
4. $k = m + n + 2$
5. $k = \max[m; n] + 4$

Wolfram shows how Turing Machines can be built to emulate cellular automata and vice versa. Therefore, these two different machines are effectively equivalent in their computational power and he argues, many systems in nature are equivalent as well. *Because Turing machines have*

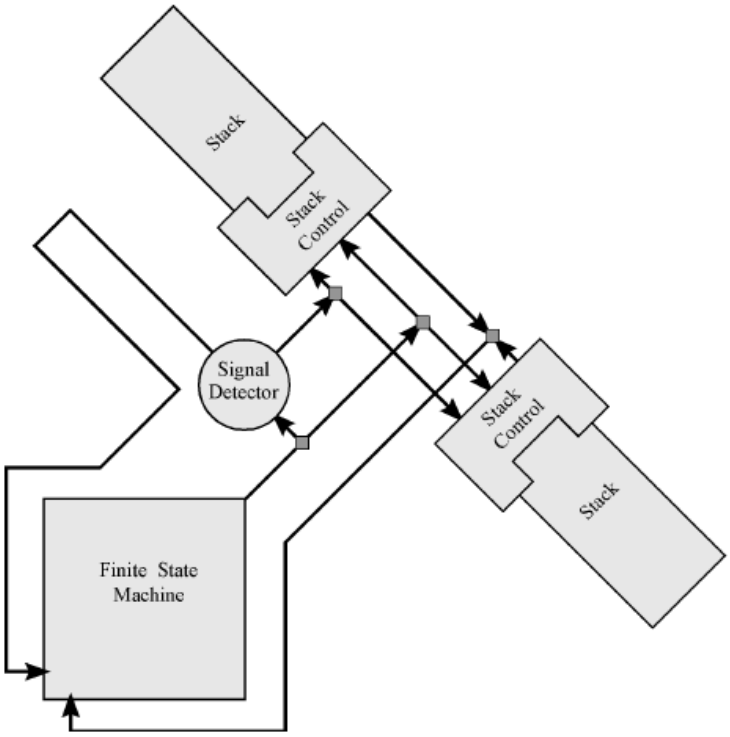
[10] Any 1-tape Turing Machine can be simulated without loss of time by an invertible partitionned cellular automaton. Dubacq Jean-Christophe, *How to simulate Turing machines by invertible one-dimensional cellular automata*, Departement de Mathematiques et d'Informatique, Ecole Normale Sup'erieure de Lyon, 1997.

[11] Claudio Baiocchi, *Some results on cellular automata*. Rend. Mat. Acc. Linceis. 9, v. 9: 307–316 (1998).

[12] A $TM[n; m]$ is a Turing Machine with n internal-states and m tape-symbols.

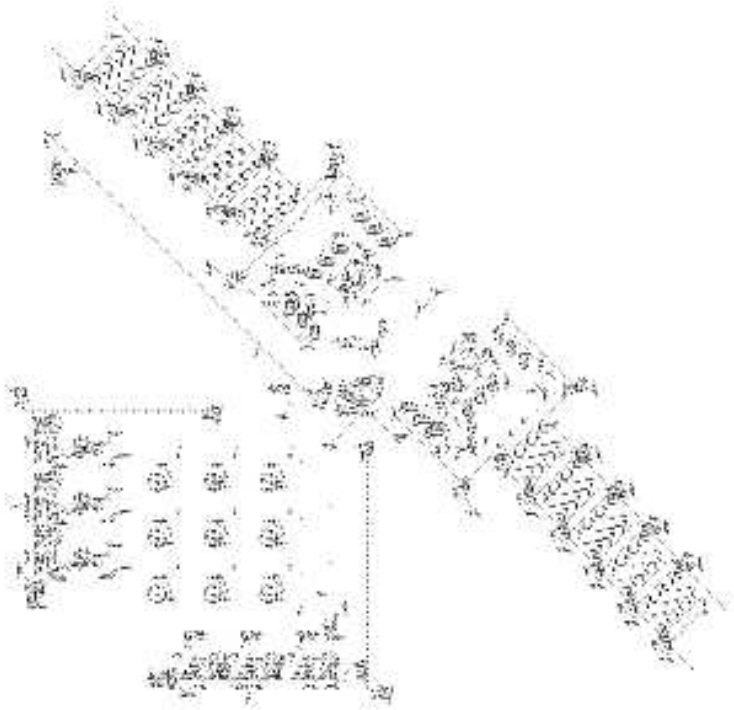
been shown to be universal computers capable of arbitrarily complex computations, so are universal cellular automata which emulate Turing machines[13].

It was proved a long time ago that a Turing Machine could be simulated in The Game of Life. This proof is based on the fact, that simple logic can be performed and therefore simulation can be built.



The above figure shows a diagram of the Turing Machine. The finite state machine contains the memory unit built up of the memory cells. In each cycle of the Turing Machine the finite state machine sends its output to the signal detector and the stacks[14]. The simulation of this machine in the Game of Life is as follows:

[13] Sam Reid Virtual Machines: *Turing Machines, Cellular Automata and Java*, 2003.
[14] AA Book P. Rendell Chapter Draft 3.



Many alternative schemes for simulating Turing Machines in cellular automata have been formulated over the years. From the other hand, the following theorem has been proved:

Theorem

A cellular automaton can be simulated by a 2-tape Turing Machine.

Neural network

Artificial neural networks were proposed as a tool for machine learning. Many results have been obtained by their application to practical problems. Usually, a neural network is trained during a *supervised* training session to recognize associations between inputs and outputs. These associations are incorporated into the weights of the network, which encode a representation of the information contained in the input. Once trained, the network will compute an input/output mapping which, if the training data was representative enough.

Definition[15]

A (discrete) neural network is defined as a 6-tuple $N = (V, I, O, \Lambda, w, h)$, where:

V is a finite set of units, which we assume are indexed as $V = \{1, \ldots, p\}$,

$I \subseteq V$ and $O \subseteq V$ are the sets of input and output units, respectively,

$\Lambda \subseteq V$ is a set of initially active units, of which we require that $\Lambda \cap I = \emptyset$,

$w : V \times V \to Z$ is the edge weight matrix, and

$h : V \to Z$ is the threshold vector.

The size of a network is its number of units, $|V| = p$, and the weight of a network is defined as its sum total of edge weights, $\sum_{i,j \in V} |w_{ij}|$[16].

Given a neural network N, let us denote $|I| = n$, $|O| = m$. Moreover, let us assume that the units are indexed so that the input units appear at indices 1 to n. The network computes a partial mapping $f_N : \{0,1\}^n \to \{0,1\}^m$ as follows. Given an input x, $|x| = n$ the states s_i of the input units are initialized as $s_i = x_i$. The states of the units in set Λ are initialized to 1, and the states of the remaining units are initialized to 0. Then new states s'_i, $i = 1, \ldots, p$ are computed simultaneously for all the units according to the rule $s'_i = \mathrm{sgn}(\sum_{i=1}^{p} w_{ij} s_j - h_i)$ where $\mathrm{sgn}(t) = 1$ for $t \geq 0$, and $\mathrm{sgn}(t) = 0$ for $t < 0$.

Simulation of Turing Machines by neural networks was done first by McCulloch and Pitts in 1943[17]. The result that neural networks can simulate Turing Machines[18] is well-known. The computational power increases considerably for rational weights[19] and thresholds. For instance, a "rational" recurrent net is, up to a polynomial time computation, equivalent to a Turing Machine. In particular, a network that simulates a universal Turing Machine does exist and one could refer to such a network as "universal" in the Turing sense. It is important to note that the number of nodes

---

[15] Pekka Orponen, *The Computational Power of Discrete Hopfield Nets with Hidden Units*.

[16] A Hopfield net, for example, (with hidden units) is a neural network N whose weight matrix is *symmetric*.

[17] McCulloch, W. and Pitts, W., *A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics*, 7: 115–133, 1943.

[18] Any computatable function can be computed on a neural network.

[19] *Irrational weights provide a further boost in computation power. If the net is allowed exponential computation time, then arbitrary Boolean functions (including non-computable functions) are recognizable. However, if only polynomial computation time isallowed, then nets have less power and recognize exactly the languagescomputable by polynomial-size Boolean circuits*. See: Bhaskar DasGupta, Georg Schnitger, *On the Computational Power of Analog NeuralNetworks*, p. 11.

in the simulating recurrent net is fixed (i.e., *does not grow* with increasing input length)[20].

The computational power of recurrent neural networks was investigated by Siegelmann and Sontag. They proved the following theorem:

Theorem (Siegelmann and Sontag 1991, 1995)[21]

A finite recurrent neural network with rational weights can compute, in real time, any function computable by a Turing Machine[22].

However, in the presence of noise, the behavior of recurrent neural network with analytic function of activation of neuron is not good. The computational power of this kind of network falls to a level below the computational power of finite automata[23].

There are some neural networks, which have very interesting properties, for example probabilistic recurrernt networks. The following theorem holds:

Theorem

Probabilistic Recurrent Networks (PRN) and Probabilistic Turing Machines (PTM) are polynomially equivalent. More specifically a PRN can be simulated by a PTM with a polynomial increase in running time and conversely a PTM of time complexity $T(n)$ can be simulated by a PRN of size at most polynomial in $T(n)$.

This is an important result. The significance of the above result lies in the fact that a PRN with polynomial number of processors can therefore learn NP language problems. whereas for example a Hopfield network with polynomial number of processors is already proved not to have such capabilities even if allowed to run for exponentially many steps and hence PRNs are more powerful.

And finally we can to write the following theorem:

---

[20] Bhaskar DasGupta, Georg Schnitger, *On the Computational Power of Analog Neural Networks*.

[21] Siegelmann and Sontag shows that if one moves from binary state to analog-state neurons, then arbitrary machines may be simulated by single, finite recurrent networks. The original construction required 1058 saturated-linear eurons to simulate a universal Turing Machine, but this has later been improved to even 25 neurons.

[22] The universal network possesses at most 884 nodes. The computation time is essentially enchanged. The potential infinity of rational values plays the role of the infinite tape in the Turning Machine.

[23] Wolfgang Maass, Eduardo D. Sontag. *Analog neural nets with gaussian or other common noise distribution cannot recognize arbitrary regular languages*. Neural Computation, 1998 or Siegelmann H. T., Roitershtein A. *Noisy analog neural networks and definite languages: stochastic kernels approach*. Technical Report, Technion, Haifa, Israel, 1998.

Theorem

Every self-map $T : \mathbb{Z} \to \mathbb{Z}$ realizable on a cellular automaton can be implemented by some neural network, and every neural network can be implemented by some random neural network, i.e.

$$TM \subset CA \subseteq NN \subseteq RN$$

Sketch of proof

TM can be simulated by one-dimensional cellular automaton thus $TM \subseteq CA$. The inclusion is proper, because there exist problems solvable by Celullar Automata, but not by Turing Machine. For example, one-dimensional Celullar Automata taking as an input a real number (as an infinite binary expansion) and stabilizing iff it is an integer. A Turing Machine cannot solve the problem since the integer 1 could be given as 0.999... and hence it will not even finish reading its input in finite time, i.e. $TM \subset CA$.

Every Celullar Automata can be simulated by Neural Network, i.e. $CA \subseteq NN$ because if $\delta : Q \times Q^d \to Q$, $|Q| = m$, $\delta(q_0, q_1, \ldots, q_d) = q_0'$ then we construct Neural Network using the same digraph as for CA.

Every NN can be simulated by Random Network, i.e. $NN \subseteq RN$ because NN cell (neuron) is a FSM with local transition: $\delta(x_i, x_{i_1}, \ldots, x_{i_d}) := f_i(\sum w_{i_j} x_{i_j})$.

We have to agree with a version of the Church-Turing thesis:

No one has ever invented a more powerful computing model than a Turing Machine[24].

Bibliography

[1] AV Aho, JE Hopcropft, and JD Ullman, *The Desing and Analysis of Computer Algorithms*, addison-Wesley Publishing Copmpany, London, 1974

[2] Baiocchi Claudio, *Some results on cellular automata*. Rend. Mat. Acc. Linceis. 9, v. :307–316, 1998

[24] It is now known that quantum computing gives us unprecedented possibilities in solving problems beyond the abilities of classical computers. For example Shor's algorithm gives a polynomial solution (on a quantum computer) for the problem of prime factorization, which is believed to be classically intractable. There are some proponents for the idea that *Quantum Neural Networks* may be developed that have abilities beyond the restrictions imposed by the Church-Turing thesis.

[3] Bhaskar DasGupta, Georg Schnitger, *On the Computational Power of Analog NeuralNetworks*

[4] Dubacq Jean-Christophe, *How to simulate Turing machines by invertible one-dimensional cellular automata*, Departement de Mathematiques et d'Informatique, Ecole Normale Sup'erieure de Lyon, 1997

[5] Kroll Jason, *Introduction to Neural Networks and Computational Complexity*

[6] Minsky Marvin L., *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967

[7] Mikel L. Forcada, *Neural Networks: Automata and Formal Models of Computation*, 2002

[8] McCulloch, W. and Pitts, W., *A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics*, 7:115–133, 1943

[9] Melanie Mitchell, *Computation in Cellular Automata: A Selected Review*

[10] Heikki Hyötyniemi, *Turing Machines are Recurrent Neural Networks*

[11] Orponen Pekka, *The Computational Power of Discrete Hopfield Nets with Hidden Units*

[12] Reid Sam, *Virtual Machines: Turing Machines, Cellular Automata and Java*, 2003

[13] Ricardo Joel Marques dos Santos Silva, *On the Computational Power of Sigmoidal Neural Networks*, Diploma Thesis, Universidade Tecnica de Lisboa, 2002

[14] Saharon Shelah, John T. Baldwin, *On the classifiability of Cellular Automata*

[15] Siegelmann H. T., *Neural Networks and Analog Computation: Beyond the Turing Limit*, Birkhauser publishers, 1998

[16] Siegelmann H. T. and Sontag E. D., *Analog computation, neural networks, and circuits*, Theoretical Computer Science, 131, 331–360, 1994

[17] Siegelmann H. T. and Sontag E. D., *On the Computational Power of Neural Nets*, Journal of Computer and System Sciences, 50, 132–150, 1995

[18] Siegelmann H. T., Roitershtein A., *Noisy analog neural networks and definite languages: stochastic kernels approach.* Technical Report, Technion, Haifa, Israel, 1998

[19] Wolfram, S., *Computation theory of cellular automata.* Communications in Mathematical Physics, 96, 1984

[20] Wolfram, S., *Universality and complexity in cellular automata.* Physica D, 10, 1984

[21] Wolfgang Maass, Eduardo D. Sontag. *Analog neural nets with gaussian or other common noise distribution cannot recognize arbitrary regular languages.* Neural Computation, 1998

[22] Worsch T., *Parallel Turing Machines With One-Head Control Units And Cellular Automata*

Dariusz Surowik
Department of Logic, Informatics and Philosophy of Science
Białystok University
e-mail: surowik@hum.uwb.edu.pl