70th anniversary of publication: Warren McCulloch & Walter Pitts - A logical calculus of the ideas immanent to nervous activity

Vladimír KVASNIČKA a Jiří POSPÍCHAL[1]

Abstract. In 1943 was published a paper of Warren McCulloch & Walter Pitts entitled „*A logical calculus of the ideas immanent to nervous activity*“, which is now considered as one of the seminal papers that initiated the formation of artificial intelligence and cognitive science. In this paper, concepts of logical (threshold) neurons and neural networks were introduced. There was proved that an arbitrary Boolean function may be represented by a feedforward (acyclic) neural network composed of threshold neurons, i. e. this type of neural network is a universal approximator in the domain of Boolean functions. Later, S. Kleene and N. Minsky extended this theory by a study of relationships between neural networks and finite state machines (Mealy automata). They proved two important theorems. The first one claims that for an arbitrary neural network (composed of logical neurons) there exists an equivalent finite state machine. In a similar way, the second theorem claims that for an arbitrary finite state machine there exists an equivalent recurrent neural network. From these important properties it immediately follows that symbolic and subsymbolic approaches to the study of cognitive properties of human mind are mutually equivalent.

1 Introduction and basic concepts

Logical neurons and neural networks were initially studied in 1943 by Warren McCulloch and Walter Pitts´s paper [6] „*A logical calculus of the ideas immanent to nervous activity*", which is considered as a milestone of connectionist metaphor in artificial intelligence and cognitive science. This paper demonstrated that neural networks are universal approximators for a domain of Boolean functions, i. e. an arbitrary Boolean function can be represented by a feedforward neural network composed of threshold neurons. But, we have to mention from the very beginning that this work is very difficult to read, its mathematical-logical part was probably written by Walter Pitts, who

[1] Faculty of informatics and information technologies, Slovak Technical University in Bratislava, Ilkovičova 3, 812 19 Bratislava, E-mail: kvasnicka@fiit.stuba.sk, pospichal@fiit.stuba.sk.

was in both sciences total autodidact. Thanks to logician S. Kleene [3] and computer scientist M. Minsky [7,8] this work has been "translated" at the end of fifties into a form using standard language of contemporary logic and mathematics and its important ideas became generally available and accepted.

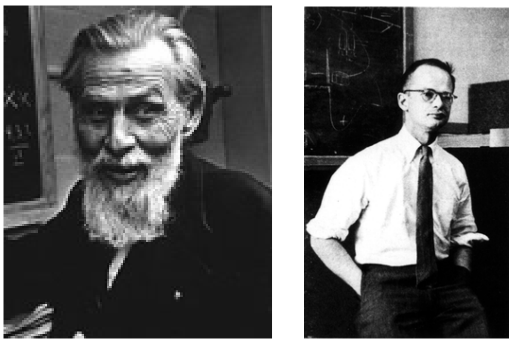
Figure 1. Warren McCulloch (1889 - 1969) and Walter Pitts (1923 - 1969)

An elementary unit of neural networks is ***threshold (logical) neuron*** of McCulloch and Pitts. It has two binary values (i. e. either state 1 or state 0). It may be interpreted as a simple electrical device - relay. Let us postulate that a dendritic system of threshold neuron is composed of excitation inputs (described by binary variables x_1, x_2, ..., x_n, which amplify an output response) and inhibition inputs (described by binary variables x_{n+1}, x_{n+2}, ..., x_m, which are weakening an output response), see fig. 2.

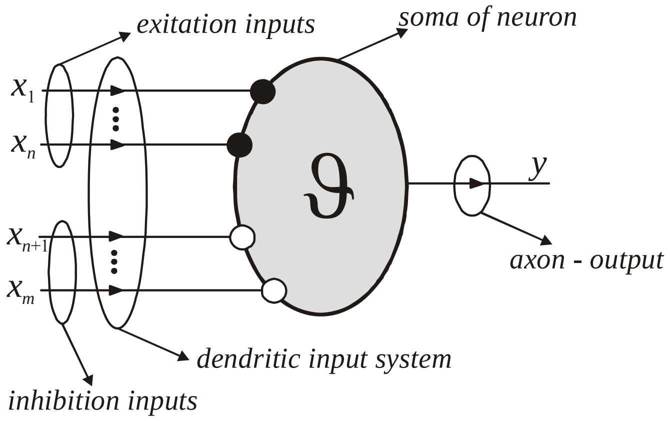

Figure 2. Diagrammatic visualization of McCulloch and Pitts neuron, which is composed of dendritic system for information input (excitation or inhibition) activities, and axon for information output. A body of neuron is called the soma, it is specified by a threshold coefficient ϑ.

An activity of threshold neuron is set to one, if the difference between a sum of excitation input activities and a sum of inhibition activities is greater than or equal to the threshold coefficient ϑ, otherwise it is set to zero

$$y = \begin{cases} 1 & \left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m \geq \vartheta\right) \\ 0 & \left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m < \vartheta\right) \end{cases} \tag{1}$$

If we introduce a simple step function

$$s(\xi) = \begin{cases} 1 & (\xi \geq 0) \\ 0 & (\xi < 0) \end{cases} \tag{2a}$$

then an output activity may be expressed as follows:

$$y = s\left(\underbrace{x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m}_{\xi} - \vartheta\right) \tag{2b}$$

An entity ξ is called the internal potential. This relation (2) may be alternatively interpreted such that excitation activities are incoming to the neuron through connections evaluated by positive unit weight coefficients (w = 1), whereas inhibition activities are incoming through connections evaluated by negative unit weight coefficients (w = -1). Then an activity of neuron may be expressed by a simple formula

$$y = s\left(\underbrace{w_1 x_1 + \ldots + w_m x_m}_{\xi} - \vartheta\right) = s\left(\sum_{i=1}^{m} w_i x_i - \vartheta\right) \tag{3}$$

where weight coefficients are specified by

$$w_{ij} = \begin{cases} 1 & \left(connection\ j \to i\ is\ of\ excitation\ character\right) \\ -1 & \left(connection\ j \to i\ is\ of\ inhibition\ character\right) \\ 0 & \left(connection\ j \to i\ is\ nonexisting\right) \end{cases} \tag{4}$$

In a neural network, weight coefficients are fixed and they are determined by a topology of syntactic tree, which specifies a given Boolean function.

Let us note that the above mentioned simple principles (1-4) “all or none” for neurons have originated in late twenties and early thirties of former century by English physician and electro-physiologist Sir E. Adrian, when he studied output neural activities by making use, in that time, of very modern electronic equipment based on electron-tube amplifiers and cathode-ray tubes for a visualization of measurements.

In the original paper McCulloch and Pitts [6] have discussed a possibility that inhibition is absolute, i. e. any active inhibitory connection forces the neuron into the inactive state (with zero output state). The paper itself shows that this form of inhibition is not necessary, and that „subtractive inhibition“ based on formulae (1-4) gives the same results.

Simple implementations of elementary Boolean functions of disjunctions, conjunctions, implication, and negation are presented in fig. 3. Let us study a function of disjunction for $n = 2$, if we use formulae (1-2) we get

$$y_{OR}\left(x_1,x_2\right)=s\left(x_1+x_2-1\right) \tag{5}$$

Functional values of this Boolean function are specified in tab. 1. It immediately follows from this table that a function y_{OR} simulates Boolean function of disjunction

Table 1. Disjunctive Boolean function

#	x_1	x_2	$y_{OR}(x_1,x_2)$	$x_1 \vee x_2$
1	0	0	$s(-1)$	0
2	0	1	$s(0)$	1
3	1	0	$s(0)$	1
4	1	1	$s(1)$	1

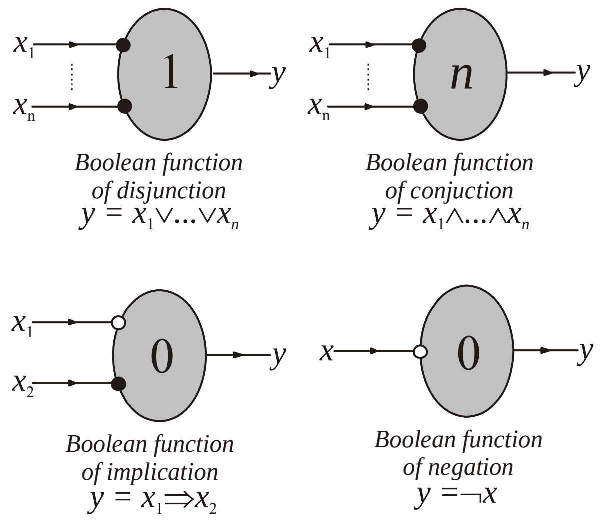
x_1
x_n
1
y
Boolean function of disjunction
$y = x_1 \vee \ldots \vee x_n$
x_1
x_n
n
y
Boolean function of conjuction
$y = x_1 \wedge \ldots \wedge x_n$
x_1
x_2
0
y
Boolean function of implication
$y = x_1 \Rightarrow x_2$
x
0
y
Boolean function of negation
$y = \neg x$

Figure 3. Three different implementations of threshold neurons, which specify Boolean functions of disjunction, conjunction, implication, and negation, respectively. Excitatory connections are terminated by black dot whereas inhibition connections by open dots.

2 Boolean functions

Each Boolean function [5,8] is represented by a syntactic tree (derivation tree), which represents a way of its recurrent building, going bottom up, initiated by

Boolean variables and then terminated (at a root of tree) by a composed Boolean function (formula of propositional logic), see fig. 4, diagram A. Syntactic tree is a very important notion for a construction of its subformulae, each vertex of tree specifies a subformula of the given formula: lowest placed vertices are assigned to trivial subformulae *p* and *q*, forthcoming two vertices are assigned subformulae $p \Rightarrow q$ and $p \wedge q$, highest placed vertex – root of the tree – is represented by the given formula $(p \Rightarrow q) \Rightarrow (p \wedge q)$.

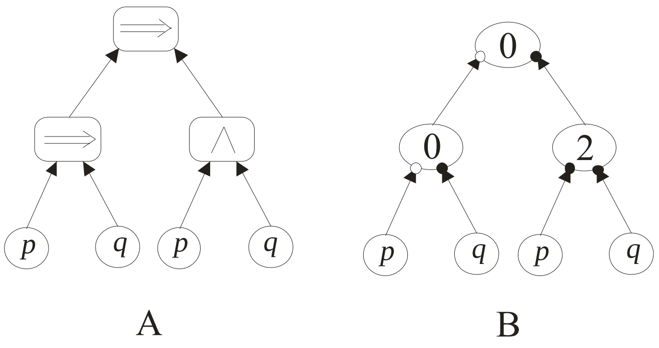

Figure 4. (A) Syntactic tree of a Boolean function (propositional formula) $(p \Rightarrow q) \Rightarrow (p \wedge q)$. Bottom vertices correspond to Boolean variables (propositional variable) *p* and *q*, vertices from the next levels are assigned to connectives implication and conjunction, respectively. An evaluation of the syntactic tree runs bottom up. (B) Neural network composed of logical neurons of connectives, which appear in a given vertex of the syntactic tree of diagram A. We see that between syntactic tree and neural network these exists very closed one-to-one correspondence, their topologies are identical, they are different only in vertices. Pictorially speaking, we may say that a neural network representing a Boolean function φ can be constructed from its syntactic tree by direct substitution of its vertices by proper logical neurons.

We see that for an arbitrary Boolean function we may simply construct a neural network, which simulates functional value of the Boolean function, see fig. 4, where this process is outlined for formula $(p \Rightarrow q) \Rightarrow (p \wedge q)$. It means that these results may be summarized in a form of a theorem.

Theorem 1. Each Boolean function, represented by a syntactic tree, can be alternatively expressed in a form of neural network composed of logical neurons that correspond to connectives from the given formula.

This theorem belongs to basic results of the seminal paper of McCulloch and Pitts [xx]. It claims that an arbitrary Boolean function represented by a syntactic tree, may be expressed in a form of neural network composed of simple logical neurons that are assigned to logical connectives from the tree. It means that neural networks with logical neurons are endowed by an interesting property that these networks have a property of universal approximator in a domain of Boolean functions. The above outlined constructive approach based on an existence of syntactic tree for each Boolean function is capable of accurate simulation of any given Boolean function.

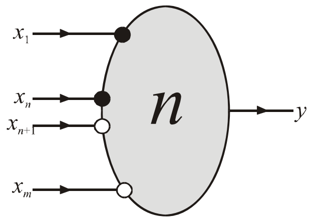

Figure 5. A logic neuron for simulation of an arbitrary conjunctive clause, which is composed of propositional variables or their negations that are mutually connected by conjunctions, $y = x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$.

Architecture of neural network based on the syntactic tree, which is assigned to an arbitrary Boolean function, may be substantially simplified to the so-called 3-layer neural network composed of

(1) a layer of input neurons (which copy input activities, they are not computational units),

(2) a layer of hidden neurons, and

(3) a layer of output neurons;

where neurons from two juxtaposed layers are connected by all possible ways by connections. This architecture is a minimalistic and could not be further simplified. We demonstrate a constructive way how to construct such a neural network for an arbitrary Boolean function.

Applying simple generalization of the concept of logical neuron, we may immediately show that a single logical neuron is capable of simulating a conjunctive clause $x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$, see fig. 5. This Boolean function is true only for variables satisfying $x_1 = ... = x_n = 1$ and $x_{n+1} = ... = x_m = 0$, for all other cases of variables its truth value is 0 (false)

$$val_\tau \left(x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m \right) = \begin{cases} 1 & \left(pre\ \tau = \tau_0 \right) \\ 0 & \left(pre\ \tau \neq \tau_0 \right) \end{cases} \tag{6}$$

where $\tau_0 = \left(x_1/1, ..., x_n/1, x_{n+1}/0, ..., x_m/0 \right)$ is a specification of truth values of variables. It can be easily verified that this conjunctive clause is simulated by logical neuron illustrated in fig. 5, its output activity is determined by simple formula

$$y = s(x_1 + \ldots + x_n - x_{n+1} - \ldots - x_m - n) \tag{7}$$

Its functional value is equal to 1 if and only if

$$x_1 + \ldots + x_n - x_{n+1} - \ldots - x_m \geq n \tag{8}$$

This simple condition is achieved if the first *n* input (excitation) variables are equal to 1 and further (*m*-*n*) input (inhibition) variables are equal to 0.

Table 2. Functional values of a Boolean function.

#	x_1	x_2	x_3	$y = f(x_1, x_2, x_3)$	*clause*
1	0	0	0	0	-
2	0	0	1	0	-
3	0	1	0	1	$\neg x_1 \wedge x_2 \wedge \neg x_3$
4	0	1	1	1	$\neg x_1 \wedge x_2 \wedge x_3$
5	1	0	0	0	-
6	1	0	1	1	$x_1 \wedge \neg x_2 \wedge x_3$
7	1	1	0	0	-
8	1	1	1	0	-

In the theory of Boolean functions is proved very important theorem that each Boolean function may be equivalently written in a form of disjunctive normal form [5,8]

$$\varphi = \bigvee_{\substack{\tau \\ (val_\tau(\varphi)=1)}} x_1^{(\tau)} \wedge x_2^{(\tau)} \wedge \ldots \wedge x_n^{(\tau)} \tag{9}$$

where

$$x_i^{(\tau)} = \begin{cases} x_i & (\text{if } val_\tau(x_i) = 1) \\ \neg x_i & (\text{if } val_\tau(x_i) = 0) \end{cases} \tag{10}$$

In order to illustrate this theorem let us study a Boolean function with functional values specified in tab. 2, where in its rows 3, 4 and 6 are "one" (true) values and in all other rows the function is false. Applying formula (9) we get an „analytic“ form of the given Boolean function specified initially by tab. 2

$$y = f(x_1, x_2, x_3) = (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3) \tag{11}$$

This Boolean function may be further simplified in such a way that the first and second clauses are simplified

$$(\bar{x}_1 \wedge x_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge x_2 \wedge x_3) = \bar{x}_1 \wedge x_2 \wedge \underbrace{(\bar{x}_3 \vee x_3)}_{1} = \bar{x}_1 \wedge x_2 \tag{12}$$

Then a final “analytic” form of the studied Boolean function is

$$y = f\left(x_1, x_2, x_3\right) = \left(\overline{x}_1 \wedge x_2\right) \vee \left(x_1 \wedge \overline{x}_2 \wedge x_3\right) \tag{13}$$

Summarizing our considerations, a clause $x_1^{(\tau)} \wedge x_2^{(\tau)} \wedge \ldots \wedge x_n^{(\tau)}$ may be expressed by single logical neuron, see fig. 5. Outputs from these neurons are mutually connected by a neuron, which represents a disjunction (see fig. 3). A final form of the Boolean function (11) is outlined in fig. 6. Results of this illustrative example may be summarized in a form of the following theorem.

Theorem 2. An arbitrary Boolean function *f can be simulated by a 3-layer neural network.*

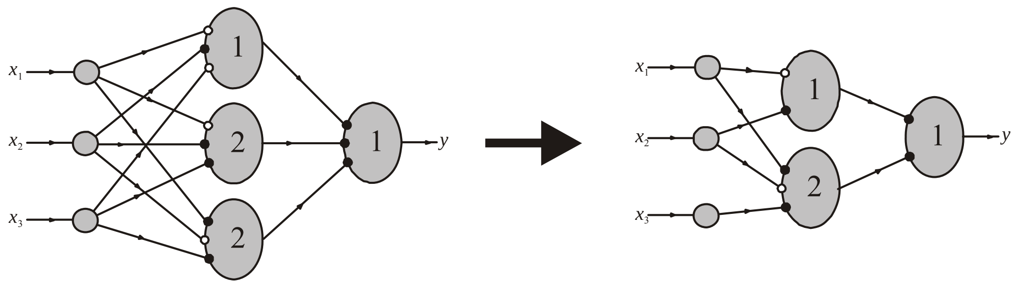

Figure 6. The 3-layer neural network, which simulates Boolean function specified by tab. 2, hidden neurons represent single conjunctive clauses specified in tab. 2, their disjunction is realized by single output "disjunctive" neuron. This neural network may be further simplified in such a way that the first two clauses are combined into a simpler conjunctive clause, see (12-13).

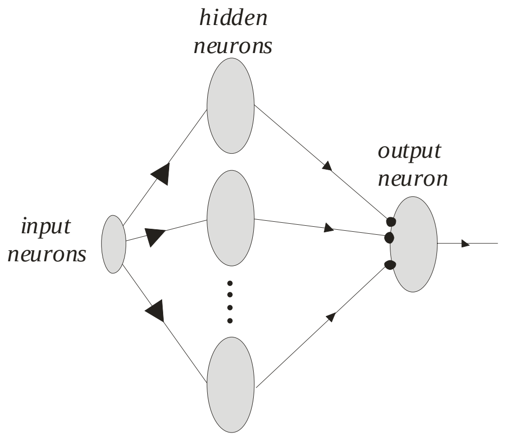

Figure 7. A schematic outline of 3-layer neural network. Going from the left to right, first comes an input layer, which is not a calculating device. The second layer is composed of hidden neurons, which represent single conjunctive clauses of the given Boolean function. The third (last) layer is composed of single output neuron, which performs an addition (disjunction) of activities produced by hidden neurons.

A general form of the 3-layer neural network is illustrated by fig. 7.

We have to note, that according to the theorem 2, the 3-layer neural networks composed of logical neurons are a universal computational device for a domain of Boolean function; each Boolean function may be represented by this “neural device” called the neural network. This fundamental result of McCulloch and Pitts’ paper [6] preceded modern result from the turn of the eighties of last century, after which 3-layer feed-forward neural networks with a continuous activation function are a universal approximator of continuous functions specified by a table of functional values [3,12,13]. Moreover, since the proof of theorem 2 was realized in a constructive manner, we know a simple systematic approach how to construct this neural network for an arbitrary Boolean function. Unfortunately, an optimal form of the constructed neural network is not solved by the theorem 2. In general, there may exist a neural network composed of smaller number of hidden neurons than the one constructed in the systematic manner from the proof of theorem 2. In the theory of Boolean function, many optimization methods have been elaborated to achieve a “minimal” form of the given Boolean function (e. g. Quin and McCluskey’s method [5]). If such an optimization technique is applied in our considerations how to construct a neural network for an arbitrary Boolean function, we arrive at an interesting constructive method that produces neural network composed of minimal number of logical neurons.

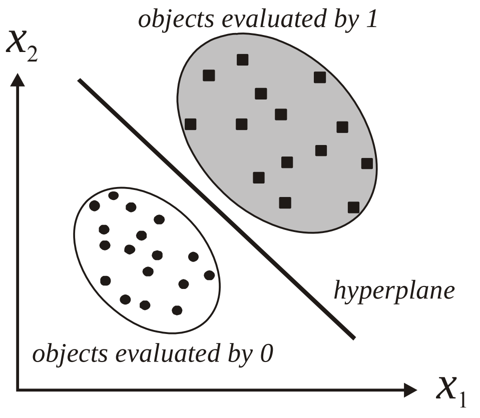

Figure 8. An illustrative outline of the concept "linear separability", where round (square) objects are separated by a hyperplane $w_1x_1+...+w_nx_n-\vartheta = 0$ such that in the first half-space there are situated objects of one kind, whereas in the second half-space there are situated objects of another kind.

We may put a question what kind of Boolean functions a single logical neuron is capable to classify correctly [7,3]? This question may be relatively quickly solved by geometric interpretation of computations running in logical neuron. In fact, logical neuron divides an input spaces onto two halfspaces by a hyperplane $w_1x_1 + w_2x_2 + ... + w_nx_n = \vartheta$, for weight coefficients w_i=0,±1. Then we say that a Boolean function $f(x_1, x_2,..., x_n)$ is *linearly separable*, if and only if there exists such a hyperplane $w_1x_1 + w_2x_2 + ... + w_nx_n = \vartheta$, which separates a space of input activities in such a way that in the first part of space are situated objects evaluated by 0, whereas in the second part of space are situated objects evaluated by 1 (see fig. 8).

Theorem 3. Logical neurons are capable to *simulate* correctly *only those Boolean functions that are linearly separable.*

A classical example of a Boolean function, which is not linearly separable is a logical connective "exclusive disjunction", which may be formally specified as a negation of a connective of equivalence, $(x \oplus y) \Leftrightarrow \neg(x \equiv y)$, in computer-science literature this connective is usually called the XOR Boolean function, $\varphi_{XOR}(x,y) = x \oplus y$, its functional values are specified in tab. 3.

Table 3. XOR Boolean function

#	x	y	$\varphi_{XOR}(x,y)$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

If we introduce its functional values into a state space x - y we get a diagram displayed in fig. 9, which is evidently linearly inseparable.

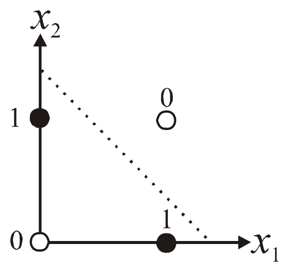

Figure 9. A diagrammatic outline of XOR Boolean function in a state space of its arguments, where objects represented by open (filled) circles are evaluated by 0 (1) . We see from the figure that there could not exist a straight-line (a hyperplane), which divides the whole plane into two sub-planes such that each sub-plane contains two object of the same kind.

Applying a technique from the first part of this chapter, we may construct a neural network, which simulates this inseparable Boolean function. From its functional values presented in tab. 3 we may directly construct its an equivalent form composed of two clauses

$$\varphi_{XOR}\left(x_1,x_2\right)=\left(\neg x_1\wedge x_2\right)\vee\left(x_1\wedge\neg x_2\right) \tag{14}$$

Then this Boolean function is simulated by the following neural network displayed in fig. 10.

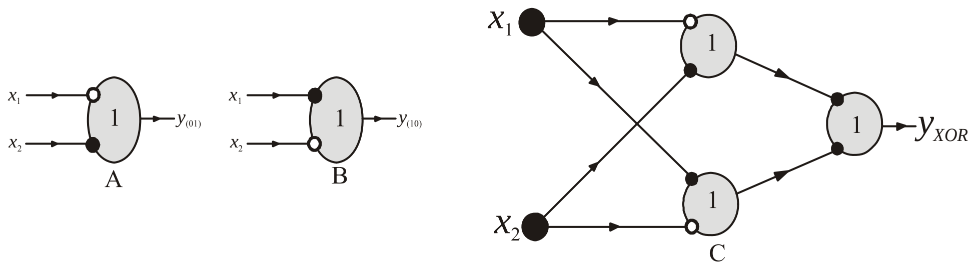

Figure 10. Diagrams A and B simulate single conjunctive clauses from (14). Diagram C represents 3-layer neural network, which hidden neurons are taken from diagrams A and B, respectively. An output neuron corresponds to a disjunctive connective.

Example 1. Construct a neural network, which simulates an addition of two binary numbers:

$$\begin{array}{r}\alpha_1\\ \alpha_2\\ \hline \beta_1\beta_2\end{array}$$

Single output binary variables are specified by $\beta_2=\alpha_1\oplus\alpha_2$ and $\beta_1=\alpha_1\wedge\alpha_2$. If we use (14), then the second output variable may be written in a form $\beta_2=\left(\neg\alpha_1\wedge\alpha_2\right)\vee\left(\alpha_1\wedge\neg\alpha_2\right)$, the corresponding network is displayed in fig. 11.

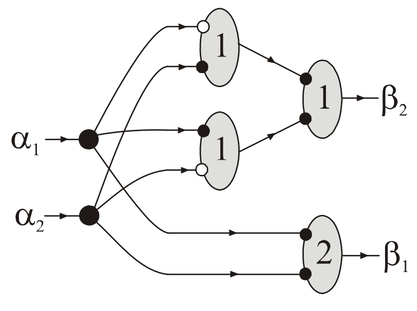

Figure 11. A neural network, which performs an addition of two one-bit variables.

In the previous part of this Chapter there was demonstrated that a single logical neuron is capable to emulate only those Boolean functions that are linearly separable. This severe restriction may be removed if we introduce the higher-order logical neurons [7], which output activity is specified by a generalization of (3) using terms of higher orders

$$y = s\left(\underbrace{\sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1 \\ (i<j)}}^{n} w_{ij} x_i x_j + \ldots + \vartheta}_{\xi}\right) \tag{15}$$

If an internal potential ξ is determined only as a linear combination of input activities (i. e. only by the first summation term), then the logical neuron is a standard one and it is called "the first order logical neuron". After Minsky and Papert [7], this property of the higher-order neurons may be summarized as a theorem.

Theorem 4. An arbitrary Boolean function *f* is simulated by a logical neuron of properly high order.

This theorem claims that each Boolean function may be simulated by a single logical neuron of sufficiently high order; there exist such weight coefficients and a threshold that for each specification of input variables $x_1, x_2, \ldots, x_n$, the calculated output activity is equal to a required value.

Example 2. Let us study, as an illustrative example, the Boolean function XOR, which is not linearly separable. Its functional values are presented in tab. 3. Let activity of a logical neuron be determined by a quadratic potential (i. e. we study a logical neuron of the second order)

$$y = s\left(\underbrace{w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2}_{\xi} - \vartheta\right) \tag{16}$$

For XOR we obtain from single rows in tab. 3 these inequalities

$$\begin{aligned} -\vartheta &< 0 \\ w_2 \quad -\vartheta &\geq 0 \\ w_1 \quad -\vartheta &\geq 0 \\ w_1 + w_2 + w_{12} - \vartheta &< 0 \end{aligned} \tag{17}$$

If we solve successive this system of inequalities, we arrive at a solution

$$\vartheta = 1, w_1 = w_2 = 1, w_{12} = -2 \tag{18}$$

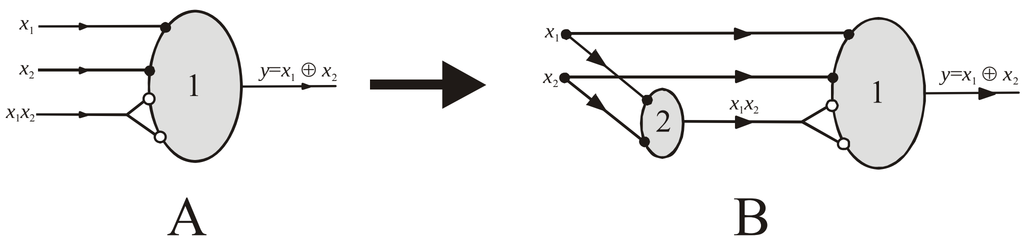

Figure 12. (A) A diagrammatic outline of the second-order logical neuron, which simulates Boolean function *XOR*, where excitation input variables are specified by variables x_1 and x_2, an inhibition activity is assigned to a product x_1x_2. An output activity z is specified by a step function $z = s(x_1 + x_2 - 2x_1x_2 - 1)$. By direct verification for different values of input activities we will see that this single second-order logical neuron simulates the *XOR* function. A fork of inhibitive input means that this input activity is taken into account twice. (B) A transformation of logical neuron of the second order, which simulates the connective *XOR* (diagram A), onto a neural network composed entirely of neurons of the first order. This transformation is based on a construction of product x_1x_2 by making use of single logical neuron (simulating a connection of conjunction), an output from this neuron is used as doubled inhibition input for the output neuron. Thus derived architecture is probably the simplest possible which may be constructed from simple (first order) logical neurons (cf. fig. 9).

In the example 2 we have shown that linearly inseparable function *XOR* may be implemented by making use of a logical neurons with three inputs x_1, x_2, and x_1x_2. In this connection we have to solve an additional problem of calculation of the product x_1x_2, which may be simply performed by a logical connective of conjunction, $x_1x_2 = x_1 \wedge x_2$. If these operation will be performed by a logical neuron of conjunction (see fig. 12, diagram B), then we may create the simplest neural network, which is composed of two neurons, where there are used only two input activities x_1 and x_2. It means that a logical neuron of the second order is capable to simulate correctly Boolean function *XOR*, which is linearly inseparable in 2-dimensional phase space x_1-x_2, but it is linearly separable in 3-dimensional phase space x_1-x_2- x_1x_2 , see fig. 13.

A concept of linearly separable Boolean function can be easily generalized to a quadratic (cubic) separability by making use a concept of quadratic (cubic) hypersurface.

Definition 1. A Boolean function *f* is called quadratic separable if and only if there exist such weight coefficients w_i, w_{ij}, and threshold coefficient ϑ that for each specification of variables $x_1, x_2, ..., x_n$ the following inequalities are satisfied

$$\begin{aligned} y_{req}\left(x_1,x_2,\ldots,x_n\right)=1 &\Rightarrow \sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1\\(i<j)}}^{n} w_{ij} x_i x_j \geq \vartheta \\ y_{req}\left(x_1,x_2,\ldots,x_n\right)=0 &\Rightarrow \sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1\\(i<j)}}^{n} w_{ij} x_i x_j < \vartheta \end{aligned} \tag{19}$$

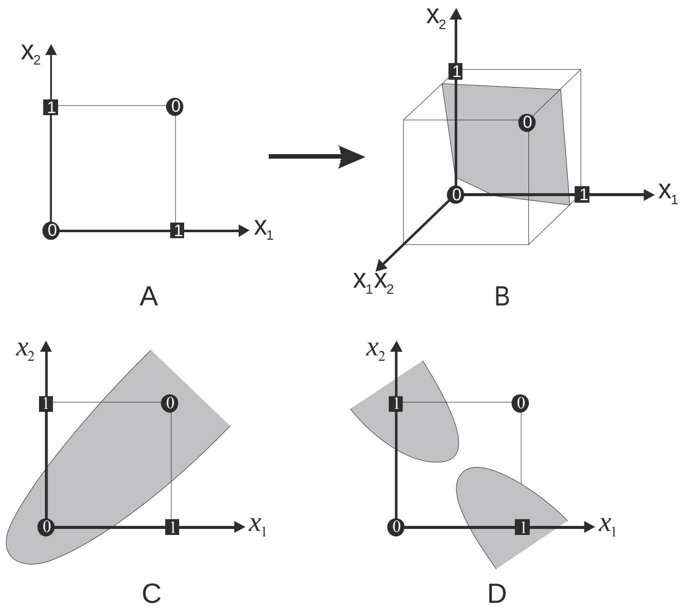

Figure 13. A diagrammatic representation of *XOR* Boolean function. (A) If *XOR* function is represented in 2-dimensional state space x_1-x_2, then objects with unit classification are not linearly separable from objects with zero classification. (B) If *XOR* Boolean function is represented in 3-dimensional phase space x_1-x_2-x_1x_2 , then there exists a hyperplane, which mutually separates objects with different classification. A projection of this hyperplane into a plane x_1-x_2 gives a quadratic curve, which separates objects with different classification, see diagram C and D.

The above outlined approach to a study of separability of Boolean functions can be generalized in a form of a theorem.

Theorem 5. An arbitrary Boolean function *f* can be correctly simulated by a higher-order logical neuron.

This theorem means that for each specification of variables $x_1,x_2,\ldots,x_n$ there exist a higher-order logical neuron (i. e. its weight coefficients and threshold factor), which correctly specifies the given Boolean function for all possible values of its arguments.

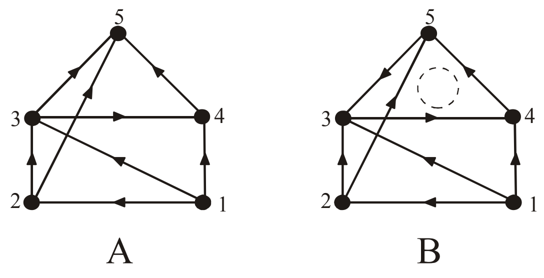

Figure 14. Oriented connected graphs that represent a topology of neural network. The vertex indexed by 1 represents an input neuron, vertices indexed by 2, 3, 4 represent hidden neurons, and finally, the vertex indexed by 5 represents an output neuron. Diagram A is an acyclic graph, whereas diagram B is a cyclic graph (it was created from the l.h.s. graph by reversing orientation of an edge 3-4) .

3 Formal specification of neural networks

From our previous discussion it follows that a concept of neural network belongs to fundamental notions of general theory of neural networks (not only those networks that are composed of logical neurons). Neural network is defined as an ordered triple

$$\mathcal{N} = \left(G, \boldsymbol{w}, \boldsymbol{\vartheta}\right) \tag{20}$$

where G is a connected oriented graph, $\boldsymbol{w}$ is a matrix of weight coefficients, and $\boldsymbol{\vartheta}$ is a vector of threshold coefficients.

Up to now we did not use time information in an explicit form. We postulate that time t is a discrete entity and is represented by natural integers. Activities of neurons in time t are represented by a vector $\boldsymbol{x}^{(t)}$, in the time $t = 0$ a vector $\boldsymbol{x}^{(0)}$ specifies initial activities of a given neural network. Relation (4) for an activity of the ith neuron in time t is specified by

$$x_i^{(t)} = s\left(\sum_j w_{ij} x_j^{(t-1)} - \vartheta_i\right) \tag{21}$$

where summation runs over all neurons that are predecessors of the ith neuron, activities of these neurons are taken in the time t-1. As an example, let us study a neural network displayed in fig. 14, where the neural network is specified by an acyclic graph, activities of single neurons are determined by (21) as follows:

$$
\begin{aligned}
&x_1^{(t)} = \textit{external input}\\
&x_2^{(t)} = s\left(-x_1^{(t-1)} - 0\right)\\
&x_3^{(t)} = s\left(x_1^{(t-1)} + x_2^{(t-1)} - 2\right) \qquad \left(t = 1,2,\ldots,t_{max}\right)\\
&x_4^{(t)} = s\left(-x_1^{(t-1)} + x_3^{(t-1)} - 1\right)\\
&x_5^{(t)} = s\left(x_2^{(t-1)} + x_3^{(t-1)} + x_4^{(t-1)} - 1\right)
\end{aligned} \tag{22}
$$

As a side notice, in a consequence of the fact that the neural network is acyclic, in the course of calculation of an activity $x_i^{(t)}$ we need to know activities of the predecessor neurons in the previous time t-1. Neural network $\mathcal{N}$ may be understood as a function, which maps an activity vector $\boldsymbol{x}^{(t-1)}$ in the time t-1 onto an activity vector $\boldsymbol{x}^{(t)}$ in the time t

$$\boldsymbol{x}^{(t)} = F\left(\boldsymbol{x}^{(t-1)};\mathcal{N}\right) \tag{23}$$

where the function F contains as a parameter the specification $\mathcal{N}$ of the given network.

According to a topology of graphs G from (20), neural networks are divided into two big classes: if graph G is acyclic, then the neural network is called *feedforward*, in the opposite case, if graph G is cyclic, then the network is called recurrent (see fig. 15).

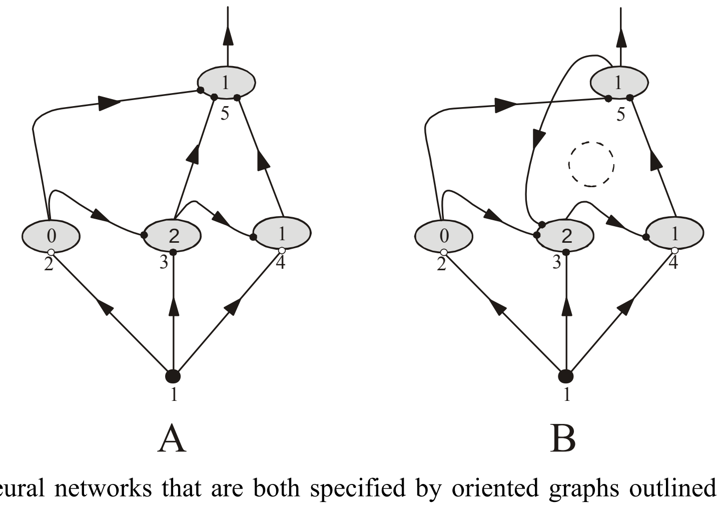

Figure 15. Neural networks that are both specified by oriented graphs outlined in fig. 14. (A) Feedforward neural network specified by the acyclic graph G displayed in fig. 14, diagram A. (B) Recurrent neural network specified by the cyclic graph G displayed in fig. 14, diagram B.

If initial values of activities of neurons indexed by 2-5 for t=1 are zero and input activities are specified by a binary vector of length t_{max}=10 are $\boldsymbol{x}_1$=(1101101010), then activities of hidden and output neurons from networks specified in fig. 14, diagram A, are presented in the following table for times $1 \le t \le 10$.

t	x_1	x_2	x_3	x_4	x_5
1	1	0	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	1	1	0	0	1
5	1	0	1	0	1
6	0	0	0	0	1
7	1	1	0	0	0
8	0	0	1	0	1
9	1	1	0	1	1
10	0	0	1	0	1

In general, we may say, that neural network forms a mapping (with parameters specified by graph topology G, weight coefficients $\boldsymbol{w}$, and threshold coefficients ϑ) of a sequence of input activities onto a sequence of output activities

$$(0001110111) = \tilde{F}\left(1101101010, \textit{parameters of network}\right) \tag{24}$$

Recurrent neural networks [3,12,13] are specified by a cyclic oriented graph, see diagram B, fig. 14. In this case we may say that this type of recurrent network has a ***memory.*** As a consequence of an existence of closed oriented cycles in recurrent networks, a repeating character of dependency of some activities from other neurons may appear. For instance, in the course of calculation of the activity x_2 in time t, as a consequence of oriented cycles we have to know activities of neurons 1, 2, and 5 in time t-1. Moreover, if we calculate an activity x_5 in a time t-1, then we must know activities neurons indexed by 2 and 4 in time t-2. From this simple discussion it follows that an activity of neuron indexed by 5 in time t is determined by previous activities in times t-1 and t-2. In forthcoming steps the “window to history” may be extended, this fact specific for recurrent networks is called the *„the memory of recurrent networks “*.

For a similar sequence of input activities as was used in the previous illustrative example, $\boldsymbol{x}_1$=(1101101010) and for similar initial activities of other neurons for t=1 (activities of neurons 2-5 in t=1 are zero), by using relations (21) we get activities of the neural network for different increasing time, which are outlined in the following table.

t	x_1	x_2	x_3	x_4	x_5
1	1	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	1	1	0	0	0
5	1	0	1	0	1
6	0	0	1	0	0
7	1	1	0	1	0
8	0	0	1	0	1
9	1	1	0	1	0
10	0	0	1	0	1

Similarly as in previous example of feedforward neural network (see fig. 15, diagram A and eq. (24)), also a recurrent neural network (see fig. 15, diagram B) can be interpreted as a mapping of input sequence $\boldsymbol{x}_1$=(1101101010) onto an output sequence $\boldsymbol{x}_5$=(0000100101).

4 Finite state machine (automaton) [3,7,9]

A finite state machine is schematically outlined in fig. 16, this machine works in discrete time events 1, 2,..., t, t+1,... . It contains two tapes of input symbols and output symbols, respectively, where output symbols are s determined by input symbols and internal states s of the machine (see fig. 16)

$$state_{t+1} = f\left(state_t, input\ symbol_t\right) \tag{25a}$$

$$output\ symbol_{t+1} = g\left(state_t, input\ symbol_t\right) \tag{25b}$$

where functions f and g specify the given machine and are considered as its basic specification:

(1) Transition function f determines the next state, this is fully specified by an actual state and an input symbol,

(2) Output function g determines an output symbol, this is fully specified by an actual state and an input symbol.

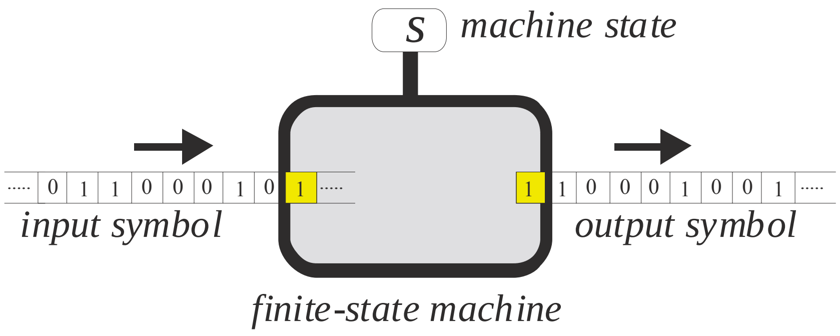

Figure 16. A finite state machine works in discrete time steps 1, 2,...,*t*, *t*+1, ,... . It contains two heads, one for reading of an input symbol and another one for printing of output symbol. In each time step *t* the machine is in specific internal state *s*, in the forthcoming time step *t*+1 an internal state *s* is determined by internal state for a time step *t* and an input symbol also in time step *t* (see relations (26a-b)).

Definition 2. A finite state machine (with an output, called alternatively the Mealy automaton) is defined by an ordered 6-tuple $M=\left(S,I,O,f,g,s_{ini}\right)$, where $S=\left\{s_1,...,s_m\right\}$ is a finite set of internal states, $I=\left\{i_1,i_2,...,i_n\right\}$ is a finite state of input symbols, $O=\left\{o_1,o_2,...,o_p\right\}$ is a finite set of output symbols, $f:S\times I\rightarrow S$ is a transition function, $g:S\times I\rightarrow O$ is an output function, and $s_{ini}\in S$ is an initial state.

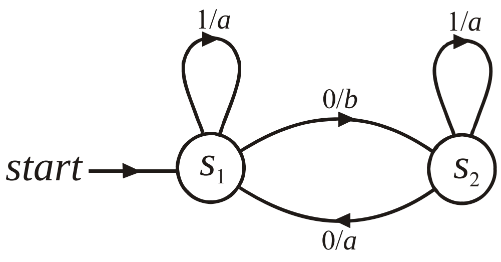

Figure 17. An example of finite state machine, which is composed of two states, $S=\left\{s_1,s_2\right\}$, two input symbols, $I=\left\{0,1\right\}$, two output symbols, $O=\left\{a,b\right\}$, and an initial state s_1. Transition and output functions are specified by tab. 4.

Table 4. Transition and output functions of a finite state machine displayed in fig. 17.

	f		*g*	
state	*transition function*		*output function*	
	0	1	0	1
s_1	s_2	s_1	*b*	*a*
s_2	s_1	s_2	*a*	*a*

Transition and output functions may be used for a construction of a model of a finite state machine, see fig. 17.

Sequences of internal states and output symbols for a finite state machine displayed in fig. 16 are determined by tab. 5 for an input sequence of symbols (100111010...). This device may be interpreted as a mapping of input string of symbols onto output string of symbols

$$G\left(\underbrace{100111010...}_{\text{input string } x}; f, g\right) = \underbrace{\square abaaaabaa...}_{\text{output string } y}$$

where a symbol □ in an output string means an “empty token”, symbols of output string are shifted by one time step with respect to the input string. A mapping *G* is composed of functions *f* and *g*, which specify a „topology“ of the finite state machine. For a construction of relationship between neural network and finite state machine we specify this approach as follows: Let $\boldsymbol{i} = i^{(1)}i^{(2)}i^{(3)}...i^{(t)}...$, $\boldsymbol{o} = o^{(2)}o^{(3)}o^{(4)}...o^{(t+1)}...$, and $\boldsymbol{s} = s^{(1)}s^{(2)}s^{(3)}...s^{(t)}...$ be strings of input symbols, output symbols, and internal states, respectively (see tab. 5). Single symbols from these strings are in two mutual relationships (see fig. 18)

$$s^{(t+1)} = f\left(s^{(t)}, i^{(t)}\right) \tag{26a}$$

$$o^{(t+1)} = g\left(s^{(t)}, i^{(t)}\right) \tag{26b}$$

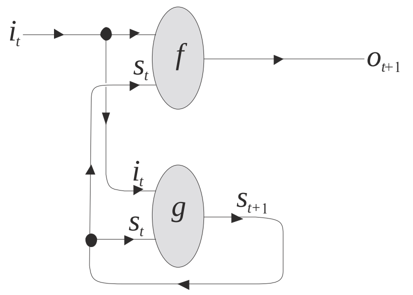

Figure 18. A diagrammatic outline of finite state machine represented by transition and output functions *f* and *g*, respectively (see eq. (26a-b)).

The first equation (26a) specifies the next internal state $s^{(t+1)}$ by a transition function *f*, input symbol $i^{(t)}$, and internal state $s^{(t)}$. In a similar way, the second equation (26b) specifies the new output symbol $o^{(t+1)}$ by an output function *g*, a previous internal state $s^{(t)}$, and an output symbol $i^{(t)}$. We say that a neural network is *equivalent* to a finite state machine if and only if responses of both devices are identical for the same input. For this equivalence it is not important a way of mapping of input symbols onto output symbols, i. e. a type of calculation accompanying this transformation, a substantial feature here is an equality of output strings for the same input strings for both devices (neural network and finite state machine).

Table 5. Sequences of input symbols, internal states, and output symbols for a finite state machine displayed in fig. 15.

input symbol	1	0	0	1	1	1	0	1	0	..
internal state	s_1	s_1	s_2	s_1	s_1	s_1	s_1	s_2	s_2	..
output symbol	□	*a*	*b*	*a*	*a*	*a*	*a*	*b*	*a*	*a*

Theorem 5 [xx]**.** Each neural network can be represented by an equivalent finite state machine with output.

Proof of this theorem is simple and constructive, we show how we can construct for a given neural network single elements from the definition 2, $M=\left(S,I,O,f,g,s_{ini}\right)$. First of all we divide a binary vector of neural-network activities $\boldsymbol{x}$ onto a direct sum $\boldsymbol{x}=\boldsymbol{x}_I\oplus\boldsymbol{x}_H\oplus\boldsymbol{x}_O$, where its components are binary vector of input activities $\boldsymbol{x}_I$, hidden activities $\boldsymbol{x}_H$, and output activities $\boldsymbol{x}_O$, respectively.

(1) The internal-state set S is composed of all possible binary vectors $\boldsymbol{x}_H$, $S=\left\{\boldsymbol{x}_H\right\}$. Let the neural network be composed of n_H hidden neurons, then a cardinality of S is 2^{n_H} .

(2) The set of output symbols is composed of all possible binary vectors $\boldsymbol{x}_I$, $I=\left\{\boldsymbol{x}_I\right\}$, a cardinality of this set is 2^{n_I}, where n_I is number of input neurons.

(3) The set of output symbols is composed of all possible binary vectors $\boldsymbol{x}_O$, $O=\left\{\boldsymbol{x}_O\right\}$, a cardinality of this set is 2^{n_O}, where n_O is number of output neurons.

(4) A function $f : S \times I \to S$ assigns to each couple of internal state and input symbol a new internal state. This function is specified by a mapping (23) produced by the given neural network

$$\boldsymbol{x}_H^{(t+1)} = F\left(\boldsymbol{x}_I^{(t)} \oplus \boldsymbol{x}_H^{(t)};\mathcal{N}\right) \qquad (27)$$

This mapping assigns a new internal state in a time *t* to a couple composed of internal state and input symbol in time *t*-1.

(5) Function $g : S \times I \to O$ assigns a new output symbol to each couple of internal state and input symbol. This function is specified by a mapping

$$\boldsymbol{x}_O^{(t+1)} = \tilde{F}\left(\boldsymbol{x}_I^{(t)} \oplus \boldsymbol{x}_H^{(t)};\mathcal{N}\right) \qquad (28)$$

(6) An initial internal state s_{ini} is usually selected such that all activities of hidden neurons are vanishing (zero).

Summarizing, for a given neural network we unambiguously specify a finite state machine, which is equivalent to the given neural network. This means that any neural network may be represented by an equivalent finite state machine, Q.E.D.

A proof of inverse theorem with respect to theorem 5 (i. e. each finite state machine may be represented by an equivalent neural network) is not a trivial one, the first who proved this inverse form was Minsky in 1967 in his famous book "*Computation: Finite and Infinite Machines*" [7] by making use of very sophisticated constructive approach. Our goal is to construct for a given finite state machine an equivalent neural network.

Theorem 6 [xx]**.** Each finite state machine with output (i. e. the Mealy automaton) can be represented by an equivalent recurrent neural network.

Example 2. In this example we present a simple illustrative proof of the above theorem 6. The constructed neural network will correspond to an example of finite state machine with state diagram displayed in fig. 17. This machine is determined for transition and output functions (see tab. 4), which may be expressed as two Boolean function:

(1) Transition function $state_{t+1} = f\left(state_t, input\ symbol_t\right)$:

state,input symbol	*transition function f*
$(s_1,0) \to (0,0)$	$(b) \to (1)$
$(s_1,1) \to (0,1)$	$(a) \to (0)$
$(s_2,0) \to (1,0)$	$(a) \to (0)$
$(s_2,1) \to (1,1)$	$(a) \to (0)$

(2) Output function $output\ symbol_{t+1} = g\left(state_t, output\ symbol_t\right)$:

state, output symbol	*output function g*
$(s_1,0) \to (0,0)$	$(s_2) \to (1)$
$(s_1,1) \to (0,1)$	$(s_1) \to (0)$
$(s_2,0) \to (1,0)$	$(s_1) \to (0)$
$(s_2,1) \to (1,1)$	$(s_2) \to (1)$

This means that both functions *f* and *g* are specified as Boolean functions

$$f\left(x_1, x_2\right) = \neg x_1 \wedge \neg x_2$$

$$g\left(x_1, x_2\right) = \left(\neg x_1 \wedge \neg x_2\right) \vee \left(x_1 \wedge x_2\right)$$

A representation of both these functions in a form of neural network composed of logical neurons is displayed in fig. 19.

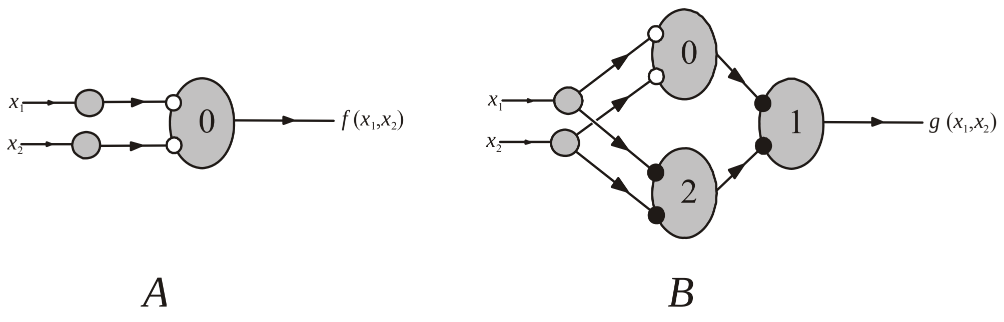

Figure 19. Boolean functions *f* and *g* from example 2.

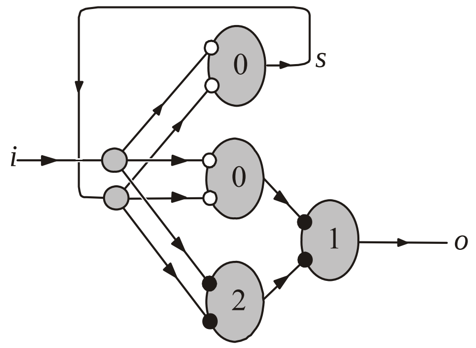

Figure 20. A recurrent neural network, which represents a finite state machine displayed in fig. 17. This network was created by a substitution of Boolean functions *f* and *g* from fig. 19 to diagram displayed in fig. 18.

Let us note that this simple example may serve as a sufficient illustrative specification of a way how to produce a constructive proof of theorem 6, i. e. for any finite state machine (specified by functions *f* and *g*) we know a way of construction of an equivalent recurrent neural network. In the first step we construct a neural representation of functions *f* and *g* by making use the method outlined in section 2 for construction of Boolean function. In the second step the functions *f* and *g* are substituted by their neural representations in general diagram displayed in fig. 18, which specifies finite state machine. This second step may be understood as a finalization of proof of theorem 6, we have demonstrated a constructive method for a construction of neural network equivalent to the given finite state machine.

To summarize our results, we have demonstrated that neural networks composed of logical neurons are powerful calculation device: (1) feedforward neural networks represented by acyclic graph are a universal approximator of Boolean functions and (2) between finite state machine and neural network there exists a property of mutual equivalency. An arbitrary finite state machine may be simulated by a recurrent neural network, and conversely, an arbitrary neural network (feedforward of recurrent) may be simulated by a finite state machine. Both these properties have been proved in constructive way, i. e. we have an algorithm how to construct another device if we know its counterpart. There exists a substantial limitation based on the fact that connection between neurons and their specification as excitatory or inhibitory and also values of threshold coefficients are specified by an architecture of network. In other words, neural networks composed of logical neurons are incapable of learning; a Boolean function (or Boolean functions, if neural network has more than one output neuron) is fully fixed in the course of its counterpart finding process.

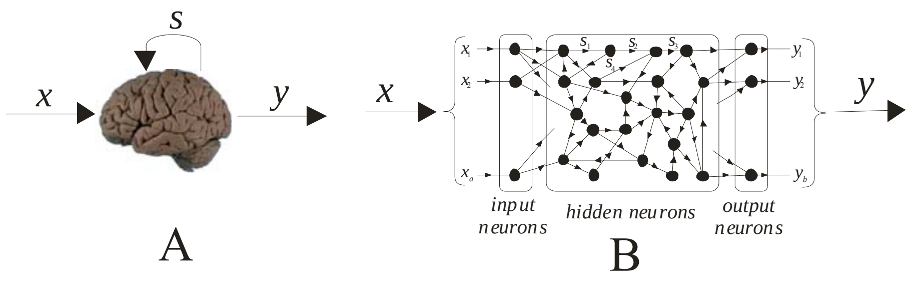

Figure 21. (A) A cybernetic interpretation of brain as a device, which transforms input ***x*** onto output ***y***, where this transformation is affected by internal state ***s*** . It means that we may get two different responses $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ on the same input ***x***. (B) Connectionist (neural) model of the brain implemented by a neural network, which is composed of (1) input neurons (e. g. perception neurons of eye retina), (2) hidden neurons, which are performing a transformation process of input onto output, and (3) output neurons (e. g. neurons controlling motor activities). Activities of hidden neurons form internal states of neural network, different initial values of their activities cause different responses to the same input activity ***x***.

5 A view of artificial intelligence and cognitive science on the problem of relationship between mind and brain

In the first part of this section we give a general view of artificial intelligence and cognitive science on the complex mind – brain as a device, which transforms input data ***x*** (produced by sight, hearing, smell and so on) onto motor impulses ***y*** (whereas this transformation is depending on an internal state ***s*** (see fig. 21, diagram A)). The brain may be considered as a huge parallel computer realized by a neural network, which transforms input information ***x*** onto output information ***y***, where this transformation is affected by internal state (see fig. 21, diagram B). This "neuroscience interpretation" of brain on a microscopic (neural) level does not allow a direct study of higher cognitive activities (solution of problems, understanding of human speech, etc.). We don't say that it is fundamentally impossible, but it is very clumsy and complicated. For instance, a complexity of this problem is similar to a study of macroscopic problem "surface tension" of water by applying methods of quantum mechanics. Of course, in principle this way of study is possible, but it is very numerically as well as theoretically demanding problem. If we apply here a "phenomenological" approach based on macroscopic thermodynamics, then it is substantially simpler than a pure microscopic approach based on quantum mechanic. In the macroscopic approach we may formulate the problem of "surface tension" very quickly in terms of experimentally measured entities; we get a formula, which is immediately experimentally verified. There exists analogical situation for a study of mind – brain relationship. Neural (connectionist) view is effective only for studies of elementary cognitive activities (e. g. initial transformation of visual information from eye retina). Higher cognitive activities are studied entirely by symbolic or cognitivistic approaches based on an idea that human brain is a computer, which activities are based on the following principles. These principles form a basis of the so-called symbolic paradigm:

(1) It transforms symbols by simple syntactic rules onto other symbols, whereas

(2) sought are symbolic representations implemented by applying a language of thinking, and

(3) mental processes are causal sequences of symbols generated by syntactic rules.

An application of term „computer" usually evokes an idea of sequential computer with von-Neumann architecture (e. g. personal computers are endowed by this architecture), where a strict demarcation line between hardware and software is possible; on the same computer may be performed huge number different programs – software. For these computers, a strict dichotomy exists between hardware and software. Unfortunately, a paradigm of a mind as a computer evokes for many people an idea that there is possible to separate brain from the mind, as two "independent" phenomena, where a brain plays a role of a hardware and the mind a role of software (performed on the hardware - brain).

Let us turn our attention to a modern "neuroscience" approach for an understanding of a relationship between brain and mind [1,10], which is based on the connectionist conception of brain and mind. A basal model of brain (based on experimental neuroscience knowledge) consists in facts that it is formed of neurons that are mutually interconnected by directed (one way) synaptic connections (see fig. 21, diagram B). Thereafter we say that *a capability of brain performing not only cognitive activities but also being a memory should be coded in its architecture*. It means that a computational paradigm of human brain must be formulated in such a way that the brain is a parallel and distributed computer composed of a few milliards (GB) neurons, which are mutually interconnected by one-way connections into extremely complex network. A program in this parallel computer is a built-in function of its architecture, i. e. human brain is a single-purpose parallel computer represented by its neural network, which could not be reprogrammed without changes of its architecture. This "neuroscience" contemplations may be summarized in a general conclusion that the brain and mind form one integral unit, where the mind should be understood as a "program" performed by the brain. The brain and mind are nothing but two different views on the same object brain-mind:

> (1) If we speak about a brain, we thought its "hardware" structure biologically realized by neurons and their synaptic connections (formally represented by a neural network), and conversely,
>
> (2) If we speak about a mind, we thought its cognitive and other activities performed by a neural network (which formally represents the brain).

We say a few remarks on relationship between a distributed representation (called the connectionism or subsymbolism) and a localistic representation (called the symbolism or between cognitivism) in theory of mind. Recently, there is used a compromise solution that higher level activities are considered on symbolic level (though there exist good connectionist models), whereas low level cognitive activities are considered on connectionist level. For completeness, we mention that D. Gabbay has published a seminal book [xx] in which he and his coworkers demonstrated connectionist approaches based on neural networks for a study of logical reasoning.

A realistic interpretation of both these approaches is that they offers different views at the same problem. While the symbolism is appropriate for interpretations of higher-order cognitive activities of human brain, its counterpart is appropriate for low-level cognitive activities (e. g. perception). An alternative interpretation of this view is that symbolism could be understood as an approach bottom-up, which interprets higher cognitive activities by making use of different approaches that are known from artificial intelligence. We have to remember that a suggested model must have connectionistic plausibility; i. e. a substrate of human thinking is brain with entirely connectionist architecture. On the other hand, connectionist approaches to a study and interpretation of cognitive activities of the human brain, are fully based on neural networks and represent up-bottom approach. In the course of application of connectionist methods there is necessary to introduce hypothetical blocks (modules) that perform special activities, which are closely related with block structure of symbolic approaches. In an ideal case, we shall expect that both these approaches are met at halfway denoted by dashed line in fig. 22. For instance, connectionist approaches offer an interpretation of modules used in symbolic approaches. In other words, the connectionism offers for symbolic approaches a "microscopic theory" for its phenomenological notions, which is in accordance with recent concepts of a structure and physiology of human brain.

6 Discussion and final notes

McCulloch and Pitts's paper is very ostensibly ''neural'' in the sense that he used an approach for specification of neuron activities based on simple rule all-or-none. However, McCulloch–Pitts neural networks are heavily simplified and idealized when compared to the then known properties of neurons and neural networks. The theory did not offer testable predictions or explanations for observable neural phenomena. It was quite removed from what neurophysiologists could do in their labs. This may be why neuroscientists largely ignored McCulloch and Pitts's theory. For this scientific community, its main power is not consisting in a capability to produce verifiable hypothesis, but it consists in a fact that such extremely simple neural theory offers arguments of basal character for a discussion of “philosophical” problems about a brain and mind relationship. There can not be expected that a further “sophistication” of this theory (e. g. the rule “all-or-none” is substituted by another more realistic rule or “spiking” neurons are used, etc.) will negatively influence general results deduced from the model.

The 1943 paper by McCulloch and Pitts was influential in a large number of domains, some of them unexpected. In the realm of mathematics itself this paper is often given credit for founding of the important field known as finite state automata theory. However, its influence went even further. The paper was published at the height of the Second World War. At that time there were a number of projects in progress to build practical computing machines for various military purposes. The teams involved became aware of the McCulloch–Pitts paper very early on.

One of those influenced was John von Neumann [xx], who is known as a creator of the so-called „von Neumann computer architecture“, which was outlined in his famous 1945 technical report. He mentioned that in existing digital computing devices, various mechanical or electrical devices have been used as elements. It is worth mentioning that the neurons are definitely elements in the above sense. It is easily seen that these simplified neuron functions can be imitated by telegraph relays or by vacuum tubes. The proposed similarity between the computer and the architecture of the brain was taken very seriously by computer scientists at the time. When early computer scientists referred to computers as ‘giant brains’, they were not just using a metaphor, but were referring to what they thought were two computing systems based on the same principles but using different hardware. From the early 1940s the McCulloch–Pitts neuron was considered by many non-neuroscientists to be the most appropriate way to approach neural computation, largely because the work of McCulloch and Pitts was so well known.

Finally, M. Minsky in the early 1970s commented [8] the paper of McCulloch and Pitts as follows: *The McCulloch and Pitts paper is not a correct for many biological neuroscientists in its initial domain of application – in this case brain theory, since the used rule “all-or-none” is very rough and simplifying from the standpoint of modern neurophysiology. But it is immensely valuable in many other places and at many different levels, and secondly, that a tight coupling between brain science and computer science has existed from the earliest beginnings of both fields, and has enriched both.*

Acknowledgment. The authors acknowledge financial support from Slovak grant agency, grant VEGA 1/0553/12.

References

[1] Anderson, J. A., Rosenfeld, E.: *Talking Nets: An Oral History of Neural Networks*. MIT Press, Cambridge, MA, 1998.

[2] D´Avila Garcez, A. S., Lamb, L. C., Gabbay, D. M.: *Neural-Symbolic Cognitive Reasoning*. Springer, Berlin, 2009.

[3] Kleene, S. C.: Representation of events in nerve nets and finite automata. In Shannon, C. E., McCarthy, J. (eds.): *Automata Studies. Annals of Mathematics Studies*, Vol 34. Princeton University Press, Princeton, 1956, pp. 3-41.

[4] Kvasnička, V., Beňušková, Ľ., Pospíchal, J., Farkaš, I., Tiňo, P., Kráľ, A.: An Introduction of Theory of Neural Networks, IRIS, Bratislava, 1997 (in Slovak).

[5] Kvasnička, V., Pospíchal, J.: *Mathematical Logic,* STU Press, Bratislava, 2006 (in Slovak).

[6] McCulloch, W. S., Pitts, W. H.: A Logical Calculus of the Ideas Immanent in nervous Activity. *Bulletin of Mathematical Biophysics* **5**(1943), 115 – 133.

[7] Minsky, M. and Papert, S.: *Perceptrons. An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[8] Minsky, M. L.: *Computation. Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[9] Molnár Ľ., Češka, M., Melichar, B.: *Grammars and Languages:* Publishing House ALFA, Bratislava, 1987 (in Slovak).

[10] Piccinini, G,: The First Computational Theory of Mind and Brain . A Close Look at McCulloch and Pitt's ''Logical Calculus of Ideas Immanent in Nervous Aactivity''. *Synthese, Vol.* 141(2004), 175–215.

[11] Randell, B. (ed.): *The Origins of Digital Computers*. Springer, Berlin, 1973.

[12] Rojas, R.: *Neural Networks. A Systematic Introduction*. Springer, Berlin, 1996.

[13] Šíma, J., Neruda, R.: *Theoretical Questions of Neural Networks*. Matfyzpress, Praha, 1999 (in Czech).

[14] von Neumann, John: *First Draft of a Report on the EDVAC*, 1945. Retrieved October 1, 2012 from http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf,