70th Anniversary of Publication: Warren McCulloch & Walter Pitts - A Logical Calculus of the Ideas Immanent in Nervous Activity

Jiří Pospíchal and Vladimír Kvasnička

Institute of Applied Informatics, Faculty of Informatics and Information Technologies,
Slovak Technical University, Bratislava, Slovakia
{pospichal,kvasnicka}@fiit.stuba.sk

Abstract. In 1943, a paper by Warren McCulloch & Walter Pitts [6] entitled "*A logical calculus of the ideas immanent to nervous activity*" was published, 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. It 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. The present paper recalls the core achievements of this paper and puts it into perspective from the point of view of further achievements based on their approach. Particularly, S. Kleene [5] and M. Minsky [7] extended this theory by their study of relationships between neural networks and finite state machines. The present paper is not a standard research article where new ideas or approaches would be presented. However, the 70th anniversary of publication of the McCulloch and Pitts paper should be sufficiently important to recall this core event in computer science and artificial intelligence. In particular, the main concept of their paper opened unexpected ways to study processes in the human brain. Their approach offers a way to treat a core philosophical mind/body problem in such a way that the brain is considered as a neural network and the mind is interpreted as a product of its functional properties.

1 Introduction and Basic Concepts

Logical neurons and neural networks were initially introduced in Warren McCulloch´s and Walter Pitts´s paper [6]. 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. 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 was in both sciences a total autodidact. Thanks to logician S. Kleene [5] and computer scientist M. Minsky [7], this work has been "translated" at the end of the fifties into a form using standard language of contemporary logic and mathematics and its important ideas became generally available and accepted.

P. Sinčák et al. (eds.), *Emergent Trends in Robotics and Intelligent Systems*,
Advances in Intelligent Systems and Computing 316, DOI: 10.1007/978-3-319-10783-7_1

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. 1.

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.

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

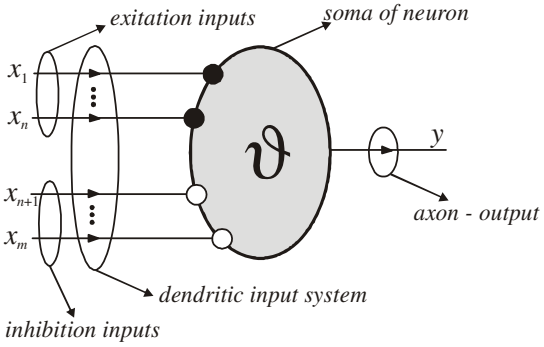

Fig. 1. Diagrammatic visualization of McCulloch and Pitts neuron which is composed of the dendritic system for information input (excitation or inhibition) activities and axon for information output. A body of neuron is called the soma, it is specified by a threshold coefficient ϑ.

If we introduce a simple step function

$$s\left(\xi\right) = \begin{cases} 1\left(\text{if } \xi \geq 0\right) \\ 0\left(\text{otherwise}\right) \end{cases} \tag{2a}$$

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. Simple implementations of elementary Boolean functions of disjunctions, conjunctions, implication and negation are presented in Fig. 2.

Let us note that the above mentioned simple principles (1-2) "all or none" for neurons have been introduced in the late twenties and early thirties of the former century by English physician and electro-physiologist Sir Edgar Adrian [1] when he studied output neural activities by making use of very modern (for that time) electronic equipment based on electron-tube amplifiers and cathode-ray tubes for a visualization of measurements.

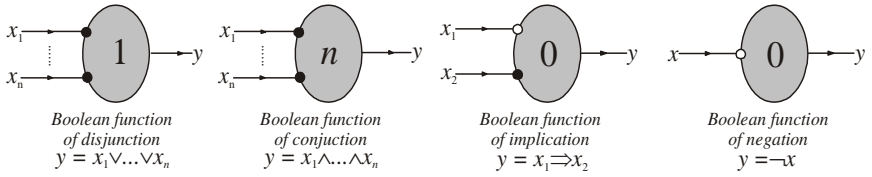

Fig. 2. Four different implementations of threshold neurons which specify Boolean functions of disjunction, conjunction, implication and negation, respectively. Excitatory connections are terminated by a black dot whereas inhibition connections by open circles.

In the original paper [6] McCulloch and Pitts 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 (1-2) gives the same results.

2 Boolean Functions

Each Boolean function [12, 13] 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 the root of tree) by a composed Boolean function (formula of propositional logic), see Fig. 3, diagram A. Syntactic tree is a very important notion for a construction of its subformulae, each vertex of tree specifies sub formulae 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 $(p \Rightarrow q) \Rightarrow (p \wedge q)$.

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. 3, 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.

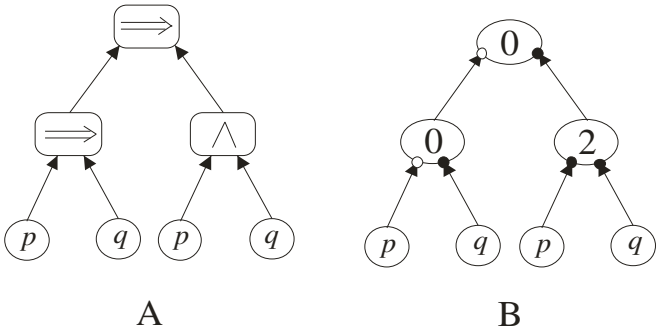

Fig. 3. (A) Syntactic tree of a Boolean function (propositional formula) $(p \Rightarrow q) \Rightarrow (p \wedge q)$. Bottom vertices correspond to Boolean variables (propositional variable) p and q, vertices from the next levels are assigned to connectives implication and conjunction, respectively. 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 there exists a very close one-to-one correspondence, their topologies are identical, they differ only in vertices. Figuratively 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.

This theorem belongs to basic results of the seminal paper by McCulloch and Pitts [6]. 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 with 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 existence of syntactic tree for each Boolean function is capable of accurate simulation of any given Boolean function.

The 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 minimalistic and could not be further simplified. We demonstrate a constructive way of how to construct such a neural network for an arbitrary Boolean function.

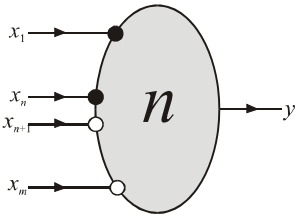

Fig. 4. A logical neuron for simulation of an arbitrary conjunctive clause which is composed of propositional variables or their negations that are mutually connected by conjunctions, $y = x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$

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 $x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$.

In the theory of Boolean functions [12, 13], a very important theorem is proved that each Boolean function may be equivalently written in a form of disjunctive normal form

$$\varphi = \bigvee_{\substack{\tau \\ (val_\tau(\varphi)=1)}} x_1^{(\tau)} \wedge x_2^{(\tau)} \wedge ... \wedge x_n^{(\tau)} \tag{3}$$

where

$$x_i^{(\tau)} = \begin{cases} x_i & \left(\text{if } val_\tau(x_i) = 1\right) \\ \neg x_i & \left(\text{if } val_\tau(x_i) = 0\right) \end{cases} \tag{4}$$

A final form of the Boolean function (3) is outlined in Fig. 4. 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.

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 functions; each Boolean function may be represented by this "neural device" called the neural network. This fundamental result of McCulloch`s and Pitts`s paper [6] preceded modern result, after which 3-layer feed-forward neural network with a continuous activation function is a universal approximator of continuous functions specified by a table of functional values [13].

We may question what kind of Boolean functions is a single logical neuron capable to classify correctly? According to Minsky and Papert, this question may be solved relatively quickly by geometric interpretation of computations running in logical neuron [8]. In fact, logical neuron divides input spaces into two half spaces 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 objects evaluated by 0 are situated in the first part of the space, whereas objects evaluated by 1 are situated in the second part of the space.

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, $\varphi_{XOR}(x, y) = x \oplus y$. Applying the technique from the first part of this chapter, we may construct a neural network which simulates this inseparable Boolean function. From its functional values we may directly construct its equivalent form composed of two clauses

$$\varphi_{XOR}(x_1, x_2) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \tag{5}$$

Then this Boolean function is simulated by the neural network displayed in Fig. 5.

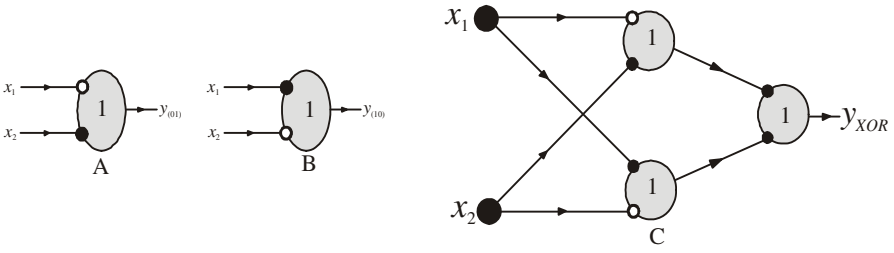

Fig. 5. Diagrams A and B simulate single conjunctive clauses from (5). Diagram C represents 3-layer neural network which hidden neurons are taken from diagrams A and B, respectively. An output neuron corresponds to a disjunctive connective.

3 Formal Specification of Neural Networks

From our previous discussion it follows that a concept of neural network [13] belongs to fundamental notions of artificial intelligence (not only those networks that are composed of logical neurons). Neural network is defined as an ordered triple

$$\mathcal{N} = (G, \boldsymbol{w}, \vartheta) \tag{6}$$

where G is a connected oriented graph, w is a matrix of weight coefficients and ϑ is a vector of threshold coefficients.

Until 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 $x(t)$, in the time $t = 0$ a vector $x(0)$ specifies initial activities of a given neural network. Relation for an activity of the ith neuron in time t is specified by

$$x_i^{(t)} = s\left(\sum_j w_{ij} x_j^{(t-1)} - \vartheta_i \right) \tag{7}$$

where summation runs over all neurons that are predecessors of the ith neuron, activities of these neurons are taken in the 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 , $\boldsymbol{x}^{(t)} = F\left(\boldsymbol{x}^{(t-1)}; \mathcal{N}\right)$, where the function F contains the specification N of the given network as a parameter.

4 Finite State Machine (Automaton)

A finite state machine [4, 5, 7] 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 determined by input symbols and internal states s of the machine

$$state_{t+1} = f\left(state_t, input\ symbol_t\right) \tag{8a}$$

$$output\ symbol_{t+1} = g\left(state_t, input\ symbol_t\right) \tag{8b}$$

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 the output symbol, this is fully specified by an actual state and an input symbol.

Definition 1. 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, $f : S \times I \to S$ is a transition function, $g : S \times I \to O$ is an output function, and $s_{ini} \in S$ is an initial state.

Transition and output functions may be used for a construction of a model of a finite state machine, see Fig. 6.

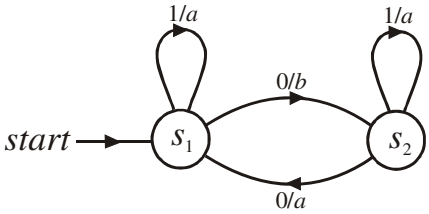

Fig. 6. An example of finite state machine composed of two states, $S=\{s_1,s_2\}$, two input symbols, $I=\{0,1\}$, two output symbols, $O=\{a,b\}$ and an initial state s_1

Finite state machines are determined as a mapping of input string of symbols onto output string of symbols

$$G\left(\underbrace{1\,0\,0\,1\,1\,1\,0\,1\,0}_{input\ string\ x}\ldots;f,g\right)=\underbrace{\square\,a\,b\,a\,a\,a\,a\,b\,a\,a}_{output\ string\ y}\ldots$$

where the 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.

Theorem 4 [5, 8]. Each neural network can be represented by an equivalent finite state machine with output.

Existing proof of this theorem is simple and constructive, we can construct for a given neural network single elements from the definition 1, $M=(S,I,O,f,g,s_{ini})$.

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.

A proof of inverse theorem with respect to theorem 4 (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 a very sophisticated constructive approach. For a given finite state machine, an equivalent neural network can be constructed.

Theorem 5 [7]. Each finite state machine with output (i.e. the Mealy automaton) can be represented by an equivalent recurrent neural network.

To summarize our results, we have demonstrated that neural networks composed of logical neurons are powerful computational devices: (1) feedforward neural networks represented by the acyclic graph are universal approximators of Boolean functions and (2) there is a property of mutual equivalency between finite state machines and neural networks. 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. Further relation between finite automata and neural networks was further studied by many authors, e.g. Noga et al. [10] and Hsien a Honavar [3].

5 Conclusions

McCulloch and Pitts's paper is very ostensibly ''neural'' in the sense that it used an approach for specification of neuron activities based on a 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. Their theory did not offer testable predictions or explanations for observable neural phenomena. It was removed from what neurophysiologists could do in their labs. This may be why neuroscientists largely ignored McCulloch's and Pitts's theory. For this scientific community, its main power is not in a capability to produce verifiable hypothesis but in a fact that such extremely simple neural theory offers arguments of basal character for a discussion about "philosophical" problems concerning a brain and mind relationship. It cannot be expected that a further "sophistication" of this theory (e.g. the rule "all-or-none" [1, 2] is substituted by another more realistic rule or "spiking" neurons are used, etc.) will negatively influence general results deduced from the model.

One example of seminal papers influenced by results of McCulloch and Pitts was the work of well-known John von Neumann [9] 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 various mechanical or electrical devices have been used as elements in existing digital computing devices. It is worth mentioning that the neurons are definitely elements in the above sense. 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.

McCulloch's and Pitts's views – that neural nets perform computations (in the sense of computability theory) and that neural computations explain mental phenomena – permanently belong to the mainstream theory of brain and mind. It may be time to rethink the extent to which those views are justified in light of current knowledge of neural mechanisms. The philosophical impact of the paper of McCulloch and Pitts is broadly discussed in many works oriented to the famous problem of connections between mind and brain [2, 11, 12].

Acknowledgments. This chapter was supported by Grant Agency VEGA SR No. 1/0553/12 and 1/0458/13.

References

1. Adrian, E.D.: The Basis of Sensation. The Action of the Sense Organs. Norton & Company, New York (1928)
2. Boden, M.: Mind As Machine: A History of Cognitive Science, vol. I, II. Oxford University Press, Oxford (2006)

3. Chun-Hsien, C., Honavar, V.: Neural network automata. In: Proc. of World Congress on Neural Networks, vol. 4, pp. 470–477 (1994)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Pearson Education, New York (2000)
5. 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, pp. 3–41 (1956)
6. McCulloch, W.S., Pitts, W.H.: A Logical Calculus of the Ideas Immanent in nervous Activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943)
7. Minsky, M.L.: Computation. Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
8. Minsky, M., Papert, S.: Perceptrons. An Introduction to Computational Geometry. MIT Press, Cambridge (1969)
9. von Neumann, J.: First Draft of a Report on the EDVAC (1945), http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf (retrieved October 1, 2012)
10. Noga, A., Dewdney, A.K., Ott, T.J.: Efficient simulation of finite automata by neural nets. J. ACM 38(1991), 495–514 (1991)
11. Piccinini, G.: The First Computational Theory of Mind and Brine: A Close Look at Mcculloch and Pitts, Logical Calculus of Ideas Immanent in Nervous Activity. Synthese 141, 175–215 (2004)
12. Quine, W.V.O.: Mathematical Logic. Harvard University Press, Cambridge (1981)
13. Rojas, R.: Neural Networks. A Systematic Introduction. Springer, Berlin (1996)
14. Searle, J.: Mind: a brief introduction. Oxford University Press, New York (2004)