Title
Knowledge, experience, generations, and limits in machine learning

Journal
Theoretical Computer Science, 317(1-3)

ISSN
0304-3975

Authors
Burgin, Mark
Klinger, Allen

Publication Date
2004-06-01

Peer reviewed

KNOWLEDGE, EXPERIENCE, GENERATIONS, AND LIMITS IN MACHINE LEARNING

Mark Burgin and Allen Klinger

Department of Computer Science
University of California, Los Angeles
405 Hilgard Ave.
Los Angeles, CA 90095

Abstract: This paper extends traditional models of machine learning beyond their one-level structure by introducing previously obtained problem knowledge into the algorithm or automaton involved. Some authors studied more advanced than traditional models that utilize some kind of predetermined knowledge, having a two-level structure. However, even in this case, the model has not reflected the source and inherited properties of predetermined knowledge. In society, knowledge is often transmitted from previous generations. The aim of this paper is to construct and study algorithmic models of learning processes that utilize predetermined or prior knowledge. The models use recursive, subrecursive, and super-recursive algorithms. Predetermined knowledge includes: a text description, activity rules (e.g., for cognition), and specific structured personal or social memory. Algorithmic models represent these three forms as separate structured processing systems: automata with 1) advice; 2) structured program; and 3) structured memory. That yields three basic models for learning systems: polynomially bounded Turing machines, Turing machines, and inductive Turing machines of the first order.

Keywords: machine learning, computation in the limit, learning in the limit, Turing machine, inductive Turing machine, experience, knowledge

1. Introduction

Cognition in general and learning, in particular, has always been important for the whole society and separate individuals. Without efficient learning strategies, neither an individual nor a society can survive in his/her/its environment. That is why cognition has been and is one of the central topics of philosophy. Development of humanities and social sciences brought cognition in the center of their studies. Development of education directed essential efforts to better understanding learning processes. Creation of computers gave birth to a new area – machine learning.

Learning consists of diverse forms of activity. Further one can consider cognition as a kind of learning. It is learning from the environment or nature. Such traditional forms as testing, verification, and detection, as well as such new forms as data mining, web search, and client monitoring are also kinds of learning.

Traditional models of machine learning have a one-level structure. An algorithm or automaton begins learning without predetermined problem knowledge. Advanced models utilize some kind of predetermined knowledge, involving a two-level structure. However, these models reflect neither knowledge sources nor means of inheritance. At the same time, society mainly acquires predetermined knowledge from previous generations. The whole history of science is a magnificent example how previous generations provided knowledge for further development, great discoveries, and unexpected inventions for next generations. The great Newton wrote, "If I have seen a little farther than others, it is because I have stood on the shoulders of giants."

To see the situation in more detail, let us consider two events of the first importance in the whole history of mathematics: creation of and grounding the calculus.

It is assumed that calculus was created by two great mathematicians: Isaac Newton and Wilhelm Leibnitz. However, as writes Bell [9], it was inevitable after the work of Cavalieri, Fermat, Wallis, Barrow, and others that the calculus should presently get itself organized as an autonomous discipline. The major base of Newton's work on calculus were results of his teacher Isaac Barrow mostly published in the book *Lectiones Geometricae* (cf., [16]). Actually, this book explained all main constructions of calculus application without perceiving the deeper significance and building a formalized mechanism. In his turn, Barrow (explicitly or implicitly) utilized results and constructions of his predecessors: Fermat, Pascal, and even Archimedes. *Lectiones*

Geometricae was the culmination of the 17th century and previous investigations leading toward the calculus.

Leibnitz also wrote that he created calculus under the influence of the works of Pascal and other French mathematicians as Leibnitz got his mathematical education in Paris [16]. It is interesting to remark that there were few if any new applications in Leibnitz's series of manuscripts, where he built the calculus, but a formalism was developed that helped systematize and generalize the diverse geometric results of old and is used even now in contrast to Newton's formalism.

After the calculus had been built, other mathematicians, such as Euler, the Bernullis, and Lagrange, developed it further and found a quantity of important applications in a variety of fields.

Although Newton and Leibnitz built the calculus, they never founded it. The first prominent mathematician to suggest that the theory of limits was fundamental in calculus was d'Alembert (1754). However, only Cauchy did this grounding to a full extent in the 19th century. But this was not the end because although the calculus had been grounded as whole, the main concepts that had been used by Newton (fluxions and fluents) and Leibnitz (differentials, infinitely small and infinite values) were still mathematically ungrounded. Only in the 20th century, Robinson formed a base for these concepts creating the nonstandard analysis.

Taking a broader understanding of experience of generations, we see that one culture can utilize and develop what was gained by another culture. For instance, many cultures were involved in the evolution of arithmetic. Ancient Greeks based their results on the achievements of Egypt and Babylonia. After the decline of science and mathematics in Europe during the middle ages, Greek tradition was cultivated by a school of Arabian scholars who faithfully translated the Greek classics into Arabic. In addition, Arabian mathematics incorporated achievements of Indian mathematics, developed both Greek and Indian results and brought their accomplishments to Europe, where a new stage of the mathematics development has begun.

All these and many other examples necessitate mathematical modeling of such cognitive and learning processes that take into account knowledge accumulated before the process starts. In this paper, we build such models. These models employ predetermined knowledge obtained by other similar processes and utilize different

kinds of algorithms: recursive, subrecursive, and super-recursive. *Recursive algorithms* are algorithms equivalent with respect to their computing power to Turing machines. *Super-recursive algorithms* can do more than Turing machines, while *subrecursive algorithms* (e.g., finite automata) can do less than Turing machines. The main goal of this paper is a comparison of recursive algorithms such as Turing machines with such super-recursive algorithms as inductive Turing machines. The utmost goal of this analysis is learning in the limit modeled by super-recursive algorithms.

2. Theoretical Mechanisms for Modeling Learning Processes

According to *Webster's Dictionary*, "to learn" is "to gain knowledge, understanding, or skill by study, instruction, or experience." This specifies three distinct forms of learning:

1. *Study* as assimilating material from descriptive knowledge sources. This is an active form of learning. Examples of descriptive knowledge sources are: books, people, data- and knowledge bases, Internet, etc.
2. *Learning by instruction* is receiving prescriptive knowledge from outside sources. This is a passive form of learning. Examples of prescriptive knowledge sources are people, instruction manual, computer programs, etc.
3. *Learning by experience* is acquiring knowledge and achieving understanding and skill from person's life and behavior. This comes from observation (passive learning form) and experimentation (active learning form).

There are combinations of different learning forms. They result in interactive learning experiences, including possibility of reference to printed text, computer files, and human experts.

Both understanding and skill can be formulated as knowledge. Understanding is knowledge of relations in a system. Skill is a kind of procedural knowledge that people are able to apply in their activity. Very often skill is implicit knowledge.

Here is a working definition of learning.

Definition 1. *Learning* is a process of taking or receiving some information (learning material) and transforming it. The product is knowledge, state or behavior of a learning system.

Learning proceeds both from particulars to general presentations (induction) and from general rules to particular facts (deduction). For instance, when a teacher wants to learn how to help her student to get better results in the class, she applies the theory of learning styles to this student and learns how better to teach him. This is deduction.

In presenting material to be learned there are two complementary classifications: *wholeness* and *transparency*.

In the *wholeness classification*, material to be learned may be:

1. in a complete form represented as a whole;
2. in an incomplete form given/taken by parts of the whole:
    a. all parts are given after some time;
    b. it is possible to view only some parts.

The *transparency classification* yields that material to be learned may be:

1. represented explicitly;
2. given/taken in a transformed form:
    a. reconstruction of knowledge can be done by known algorithms;
    b. reconstruction of knowledge cannot be done by known algorithms.

Another issue is the learning result form. We distinguish between three kinds of learning:

1. *Factual* or *partial learning*. Here a learner determines or finds whether some element x belongs to a set X (has a property P).
2. *Class* or *enumerative learning*. Here a learner represents or separates or builds elements from some set X (that have a property P).
3. *Mechanism* or *constructive learning*. Here a learner builds or finds a machine (mechanism, rules) to generate or separate a set X (all elements with a property P).

Class learning has the potential of being infinite and becoming or emerging.

Mechanism learning is actually infinite or existing.

Remark 1. Some researchers do not consider factual knowledge as a kind of knowledge, contrasting facts that are given as input to knowledge that is obtained as a result of knowledge acquisition system functioning. This approach contradicts the real situation in science, mathematics, and everyday life. For example, the fact that 2 + 2 = 4 is an important part of our knowledge about the ordinary arithmetic. Important elements of mathematical knowledge are such facts as: *e* is approximately equal to 2.71, the set of all real numbers is a field, the set of all integer numbers is a ring, and not all functions are continuous. Important elements of physical knowledge are such facts as: the velocity of light in the vacuum is a constant, it is impossible to build a *perpetuum mobile*, and that atoms consist of protons, neutrons, and electrons. In everyday life, journalists, detectives, and the majority of other people are seeking fact and not general theories or mechanisms.

People often contrast learning and reasoning (e.g., [4]). However, reasoning is a powerful intellectual tool. It is used for various purposes: learning/cognition, explanation, persuasion, etc. That is why approaches to learning are similar to the kinds of reasoning.

The main reasoning schemes correspond to three major types of learning:

- learning from similarity/by analogy (*abduction*);
- learning from experience or by experimentation (*induction*);
- learning by rules (*deduction*).

Thus, inductive inference is the main cognitive strategy in science and also a popular learning technique. Kelly *et al* [24], describe four basic inductive approaches to learning:

1) A process is organized obeying the rules motivated by considerations other than finding the truth and/or avoiding error (e.g., conformity with practice or intuition);

2) The possibility of errors is neglected if there are not “too many” or if they are all “too remote”;

3) Even significant possibility of error are forgivable if we do the best we can;

4) All errors are avoided in the limit.

This classification reflects three approaches to achievement of a remote goal when there is a measure of success available. We assume that it is impossible to get the result simply from initial data. It also implies the necessity to apply some process of inductive inference.

The first approach, *amelioration*, starts with some initial object, which is then gradually improved.

The second approach, *perfection*, starts with some initial object, from which the procedure moves through some intermediate results to the best outcome.

The third approach, *satisfaction*, also starts with an initial object but then aims at some satisfactory result.

It is possible to realize all these approaches by any of the three learning types. Since the main concern here is inductive learning, we separate three possible situations for a learning process:

1. Complete or desired knowledge/skill is obtained after a finite number of steps and the learner knows it.
2. Complete or desired knowledge/skill is obtained after a finite number of steps but the learner does not know it.
3. No finite number of steps can result in complete or desired knowledge/skill.

This corresponds to three modes of computation [12, 13]:

1. *Recursive computation*.
2. *Inductive computation*.
3. *Limit computation*.

There are abstract automata and algorithms that work in each mode. Thus, we can model the related learning types. In the following the main emphasis is on inductive learning. It provides the most adequate model for learning in nature and society.

3. One-Level Inductive Processes

Limiting recursive functions [19] and their version, trial-and-error predicates [29] were the initial one-level inductive schemes.

Definition 2. A partial function $f(x)$ is called *limiting recursive* if there is a total recursive function $g(x, n)$ such that

$$f(x) = \lim_{n\to\infty} g(x, n) \qquad (1)$$

Definition 3. A partial function $f(x)$ is called *limiting primitive* (*partial*) *recursive* if $g(x, n)$ in (1) is primitive (partial) recursive.

Definition 4. A predicate $P(x_1 , x_2 , x_3 , \dots , x_n)$ is called *trial-and-error predicate* if there is a general recursive function $g(x_1 , x_2 , x_3 , \dots , x_n , y)$ such that

$$P(x_1 , x_2 , x_3 , \dots , x_n) = \lim_{y\to\infty} g(x_1 , x_2 , x_3 , \dots , x_n , y)$$

As in the theory of recursive functions and algorithms, a class of algorithms (functions) defines sets that are decidable (enumerable) with respect to this class.

Definition 5. A set X is called *limiting recursive* (*limiting recursively enumerable*) if its characteristic function $\chi_S(x)$ (its partial characteristic function $C_S(x)$) is limiting recursive.

Definition 6. A set X is called *limiting primitive* (*partial*) *recursive* (*limiting primitive* (*partial*) *recursively enumerable*) if its characteristic function $\chi_S(x)$ (its partial characteristic function $C_S(x)$) is limiting primitive (partial) recursive.

Following results that relate the construction of limiting recursion to the arithmetical hierarchy levels were proved by Gold [19].

Theorem 1. a) A set S is limiting (partial) recursive if and only if S belongs to the level $\mathbf{\Sigma_2 \cap \Pi_2}$ of the arithmetical hierarchy.

b) A set S is limiting recursively enumerable if and only if S belongs to the level $\mathbf{\Sigma_2}$ of the arithmetical hierarchy.

c) The class of limiting partial recursively enumerable sets is contained in $\mathbf{\Sigma_3}$, contains $\mathbf{\Sigma_2 \cup \Pi_2}$, and is not closed under complementation.

However, the exact location of the class of limiting partial recursively enumerable sets in the arithmetical hierarchy was not defined by Gold.

Later the idea of functions computed in the limit was transformed into the construction of algorithmic inductive inference and learning in the limit [18]. The

development of this direction brought researchers to the following concept (e.g., [5]).

Definition 7. An *inductive inference machine* (IIM) M is a generating procedure that requests inputs from time to time. It produces some words as its partial output also from time to time. These words produced by the machine after receiving each portion of data are called *conjectures*. The final result of the inductive inference machine M is the word w to which the computational process of M converges.

Definition 8. If $u(n)$ denotes the conjecture produced by an inductive inference machine M after receiving the portion of input data with the number n, then the computational process of the inductive inference machine M *stabilizes* (or *converges*) to a word w if there exists a number $n_0 \in \boldsymbol{N}$ such that $u(n)$ equals w for any $n > n_0$.

Here procedure means some algorithmic scheme such as Turing machine that is described by effective operations, but for which it is not specified how the result is obtained. This means that an inductive inference machine is a super-recursive algorithm.

All these approaches and constructions are synthesized in the concept of inductive Turing machine. First, we turn to simple inductive Turing machines. They realize *inductive computation of the first level*.

The simplest realistic inductive Turing machine has the same structure as a conventional Turing machine with three tapes and three heads: input, working, and output. This structure is much closer to the architecture of modern computer than that of a Turing machine with one tape.

Both, inductive and ordinary Turing machines go through similar computational steps. Their differences lie how they determine their outputs. We know that a conventional Turing machine produces a result only when it halts. We assume that this result is a word written on the output tape. A simple inductive Turing machine also produces words as its results. In some cases, it stops at its final state and gives a result like a conventional Turing machine. The difference begins when the machine does not stop. An inductive Turing machine can give a result without stopping. To show this, we consider the output tape and assume that the result has to be written on it.

It is possible that in the sequence of computations, the word that is written on the output tape after some step is not changing although the machine continues to work.

Then the last reached (unchanging) word is taken as the result of this computation. Thus, an inductive Turing machine does not halt but it still produces a definite result after a finite number of computing operations. This explains the name "inductive." In induction we also proceed step by step checking if some statement *P* is true for an unlimited sequence of cases. When it is found that *P* is true for each case whatever number of cases is considered, we conclude that *P* is true for all cases.

While working without halting, an inductive Turing machine can occasionally change its output. However, people are able to use machines that occasionally change outputs. They can be satisfied that the result just printed is good enough, even if another (possibly better) result may arrive in the future. And if you continue computing, it will eventually come. Another example is a program that outputs successively better approximations to a number. Once a few digits of accuracy are attained, the user can use the output generated even if the machine is not "done". All these properties essentially extend the possibilities and indicate uses of inductive Turing machines.

Theorem 2. For any Turing machine *T*, there is an inductive Turing machine *M* such that *M* computes the same function as *T*, i.e., *M* and *T* are functionally equivalent.

This theorem demonstrates that Turing machine is, in some sense, a particular case of inductive Turing machine.

To show that inductive Turing machines are more powerful than ordinary Turing machines, we need to find a problem that no ordinary Turing machine can solve and to explain how some inductive Turing machine solves this problem. To do this, consider the halting problem for an arbitrary Turing machine. It was proved unsolvable for Turing machines. Today it is one of the most popular unsolvable problems in the theory of algorithms.

However, there is an inductive Turing machine **M** that solves this problem. This machine **M** contains a universal Turing machine **U** as a subroutine. Given a word *u* and description D(**T**) of a Turing machine **T**, machine **M** uses machine **U** to simulate **T** with the input *u*. While **U** simulates **T**, machine **M** produces 0 on the output tape. If machine **U** stops, and this means that **T** halts being applied to *u*, machine **M** produces 1 on the output tape. According to the definition, the result of **M** is equal to 1 when **T**

halts and the result of **M** is equal to 0 when **T** never halts. In such a way, **M** solves the halting problem.

So, even the simplest inductive Turing machines are more powerful than conventional Turing machines. At the same time, the development of their structure allowed inductive Turing machines to achieve much higher computing power than have the simplest inductive Turing machines described above. This contrasts to such a property of a conventional Turing machine that by changing its structure, we cannot get greater computing power.

Abstract automata are used to model, while physical machines are used to realize learning systems. In this case, the forms of predetermined knowledge correspond, respectively, to separate structural components of the device: infware, software, and hardware of the automaton.

3. Memory in learning processes

The place where information is stored and preserved is traditionally called memory. Thus, memory is one of three central components of any learning system. Functioning of memory influences complexity of different processes in a system.

There are three types of complexity of any process (and learning, in particular):

- the *complexity of separate steps*/*operations*;
- the *complexity of separate subprocesses*;
- the *complexity of the whole process*.

Two first types of complexity exist on three levels: *microlevel*, *macrolevel*, and *megalevel* operations and subprocesses.

For example, in a computer, the operations of elements from its ALU are examples of microlevel operations (e.g., compare two bits, add two bytes) The operations of ALU are macrolevel operations (e.g. compare two 32 digit numbers, add two 32 digit numbers). Operations from programming languages are examples of megalevel operations (e.g., $x + y$, $n!$).

Two latter types of process complexity are related to complexity in inductive inference (and more generally, in learning) considered by Ambainis [2]:

- the complexity of computations necessary for learning;

- the complexity of learning itself;

Some complexity measures better reflect properties of computations, while others better reflect features of learning. Several attempts to separate these two kinds of complexity have been made.

For space (memory) complexity, Freivalds, Kinber, and Smith [17] did such a separation. They proposed considering two kinds of memory: long-term memory and short-term memory, and incorporated this idea in Inductive Inference Machine (IIM), a theoretical device they developed for inductive inference. IIM uses long-term memory to remember portions of input it has seen and, perhaps, other necessary information. In addition, IIM has a short-term memory for computations. Each time when IIM reads new input data, this memory is automatically cleared: it cannot be used to remember information.

The complete triadic stratification of process complexity explicates a more developed three-type-gradation of memory: *operation memory*, *short-term memory*, and *long-term memory*.

Operation memory performs information storage for separate steps/operations. Consequently, it stores information for the least time.

Short-term memory performs information storage for separate subprocesses. Consequently, it stores information for longer time.

Long-term memory performs information storage for the whole process. Consequently, it stores information for the longest time.

Usually, in artificial intelligence (cf., for instance, [27]) and psychology (cf, for instance, [6]), only the two latter types are considered. However, a more advanced model of the mind contains all three parts [7, 20, 34]. The model portrays the mind as containing three memory stores: sensory, short-term, and long-term. Each store is characterized by its function (the role it plays in the overall workings of the mind), its capacity (the amount of information it can hold at any given instant), and its duration (the length of time it can hold any item of information).

Computers also have three levels of memory: registers, which constitute a special holding area of the CPU for the numbers the ALU uses for computation; primary storage, which is electronic circuitry that holds data and program instructions until it is their turn to be processed; and additional storage media and devices as a long-term memory [27]. Today two types of long-term storage media are the main ones in use: magnetic and optical. Primary storage includes: RAM, CMOS memory, and ROM. Magnetic storage includes: floppy disks, hard disks, tapes, and Bernoulli disks. Optical storage includes CD-ROM and WORM.

Memory of a Turing machine is its tape(s). In addition, it is possible to store information in the states of the Turing machine.

Inductive Turing machines have many more possibilities for memory stratification. They are able to realize all three types of computer memory, as well as their subtypes and new types of memory (e.g., quantum memory).

Experience of an individual or society is usually considered as a big data and knowledge base. Such knowledge includes not only declarative knowledge, but also model, procedural, and problem knowledge. Knowledge, according to contemporary understanding is a system in which connections play a central role. Thus, a structured memory of an inductive Turing machine provides efficient means to represent these connections and the knowledge system. At the same time, experience also includes skills and understanding. These attributes cannot be completely reduced to explicit knowledge – they are mainly related to implicit knowledge.

Experience is a kind of implicit knowledge. Examples of implicit knowledge are: weights of neurons in neural networks [21], connections in a structured memory with a dynamic structure, and connections between nodes/automata in virtual or potential grid automata [14, 15].

Implicit knowledge gives an explanation to intuition, at least, for some kind of intuition. It is possible to consider two forms of implicit knowledge: internally implicit and externally implicit.

Externally implicit knowledge of an individual is hidden from others, but the individual is aware of it. In contrast to this, individual is not aware of his/her internally implicit knowledge. Accordingly, there are two types of intuition: explicit intuition

based on externally implicit knowledge and implicit intuition based on knowledge that is internally implicit.

For instance, experimental intuition is an explicit intuition when a person derives a general pattern from a quantity of examples and then uses it to make decisions in similar situations.

4. Stratified Learning

All kinds of memory considered in the previous section are individual. To study learning processes that go through generations, we introduce also *social memory*. If learning is performed by some class of automata **A**, then the results of algorithms from **A** are stored in this memory. Examples of such classes are:

- the class of all inductive inference machines;
- the class of all inductive Turing machines.

Acquired experience and knowledge can be embodied or stored in three forms:

- as texts describing obtained knowledge (*descriptive* or *propositional representation*),
- as activity rules (*prescriptive* or *procedural representation*), and
- as a specific structure of individual (personal) or social memory (*recreative* or *structural representation*).

Using this classification, we now consider three classes of learning system models that utilize experiences of previous generations:

1. *Automata with an advice* [8].
2. *Automata with a structured program* [13].
3. *Automata with a structured memory* [15].

Automata with an advice can preserve in the memory for advice knowledge obtained by previous generations.

The most popular automaton with an advice is the, so-called, advice-taking Turing machine [8, 30].

Definition 9. An *advice function* $f(x)$ is some function, the values of which depend only on the length of x and are written on a special tape of a Turing machine.

Hence, advice functions provide external information to the machines, just as do oracles. However, the information provided by an oracle may depend on the actual input, whereas the information provided by an advice function does not. Indeed, only the length of the input matters. Consequently, advice-taking Turing machines form a subclass of Turing machines with oracles.

Advice-taking automata are important in complexity theory. The definitions and results in this theory are often based on special Turing machines that can determine the result of an oracle "for free", that is, in constant time.

Definition 10. An *advice-taking Turing machine* is a Turing machine enhanced with the possibility to access the advice tape. The access for advice and reading the value of its advice $f(x)$ takes place in constant time.

The fact that the advice value $f(x)$ can be determined in constant time (while $f(x)$ can be an intractable or even undecidable function) essentially increases the power and efficiency of an advice-taking Turing machine in comparison with a regular Turing machine. For example, an advice-taking Turing machine can calculate in polynomial time many functions that a regular Turing machine cannot (including some intractable ones).

Computations performed by Turing machines with oracles and Turing machines with advice are non-uniform because a different advice string may be defined for every different length of input.

Automata with structured programs can preserve knowledge obtained by previous generations in the form of a program.

Automata with structured memory can preserve knowledge obtained by previous generations in the form of the structure of their memory.

Modern neurobiological studies show that automata with structured memory provide relevant models for representation of learning processes in the brain. According to Knudsen [25], learning is a balance of innate and experiential influences. The capacity of a network to learn from experience is limited by innate factors that establish and refine initial patterns of connectivity [23]. These patterns represent specific substructures in the structured memory of a learning system. In biological

systems, these patterns of connectivity can contain remarkable specificity, imparting a high degree of functionality that reflects many generations of selections [33]. This gives evidence to the hypothesis that experience of generations accumulates in structures of the memory.

The structure of any inductive Turing machine, as an abstract automaton, consists of three components, which we can call *hardware*, *software*, and *infware*. We begin with the infware, that is, with a description and specification of information that is processed by an inductive Turing machine. Computer infware consists of information, or more exactly, data processed by the computer. An inductive Turing machine M is an abstract automaton, which works with symbolic information in the form of words of formal languages. Consequently, formal languages with which M works constitute its infware. Usually, these languages are divided into three categories: input, output, and working language(s). In contrast to the languages of everyday life (e.g., English, German or French), inductive Turing machines use formal languages.

A *formal language* **L** consists of three parts: the alphabet A of **L,** a finite set of symbols; the set A^* of all words in A, which are finite strings of symbols; and the subset L of the set A^*. Elements from L are called the words of the language **L**. The set L is often represented by generating rules R_G, i.e., the rules that build words from L, or by selection rules R_S, i.e., the rules that separate words that belong to L from all other words in A^*.

The language **L** of an inductive Turing machine consists of three parts $\mathbf{L} = (\mathbf{L}_I, \mathbf{L}_W, \mathbf{L}_O)$ where $\mathbf{L}_I$ is the *input* language, $\mathbf{L}_W$ is the *working* language, and $\mathbf{L}_O$ is the *output* language of M**.** Each of them has the following structure $\mathbf{L}_X = (A_X, R_X, L_X)$ where A_X is the *alphabet*, R_X is the set of *generating rules*, and L_X is the *set of all words* of the language $\mathbf{L}_X$ where X is one of the symbols I, O or W. Usually the generating rules for formal languages as a whole consist of one operation, which is called concatenation and combines two words into one. For example, if x and y are words, then xy is the concatenation of x and y. Taking the alphabet $A_X = \{1, 0\}$ with two words $x = 1001$ and $y = 001$ in this alphabet, we have 1001001 as the result of concatenation. The set A^* of all finite strings in the alphabet A is also a formal language; it includes the empty word ε that contains no symbols. Because a formal

language is an arbitrary subset of A^*, it is possible to consider the languages of an inductive Turing machine M as one language $\mathbf{L}(M)$, which consists of three parts**:** $\mathbf{L}_I$, $\mathbf{L}_W$, and $\mathbf{L}_O$.

Now let us look at the *hardware* or *device* **D** of the inductive Turing machine M with a structured memory. Computer hardware consists of all devices (the processor, system of memory, display, keyboard, etc.) that constitute the computer. In a similar way, the inductive Turing machine M has three abstract devices: a *control device A*, which is a finite automaton and controls performance of the machine M; a *processor* or *operating device H*, which corresponds to one or several *heads* of a conventional Turing machine; and the *memory E*, which corresponds to the *tape* or tapes of a conventional Turing machine. The memory of the simplest inductive Turing machine consists of three linear tapes, and the operating device consists of three heads, each of which is the same as the head of a Turing machine and works with the corresponding tapes.

The control device A has the *state structure* or *configuration* $S = (q_0 , Q, F)$ where Q is *the set of states* or *the state space* of A and of M, q_0 is an element from Q that is called the *start or initial state*, and F is a subset of Q that is called the set of *final* (in some cases, *accepting*) states of M. It is possible to consider a system Q_0 of start states from Q, but this does not change the computing power of an inductive Turing machine. The automaton A regulates: the state of the whole machine M, the processing of information by H, and the storage of information in the memory E.

The memory E is divided into different but, as a rule, uniform cells. It is structured by a system of relations that provide connections or ties between cells. Each cell can contain a symbol from an alphabet of the languages of the inductive Turing machine M or it can be empty. Formally, $E = (\boldsymbol{P}, \boldsymbol{W}, \boldsymbol{K})$ where $\boldsymbol{P}$ is the set of all cells from E, $\boldsymbol{W}$ is the set of connection types, and $\boldsymbol{K} \subseteq \boldsymbol{P}\times\boldsymbol{P}$ is the binary relation on $\boldsymbol{P}$ that provides connections between cells. In such a way, $\boldsymbol{K}$ structures the memory E. Each of the sets $\boldsymbol{P}$ and $\boldsymbol{K}$ is also structured. The set $\boldsymbol{P}$ is enumerated, that is, a one-to-one mapping ν of $\boldsymbol{P}$ into the set $\boldsymbol{N}$ of all natural numbers is given. A type is assigned to each connection from $\boldsymbol{K}$ by the mapping τ: $\boldsymbol{K} \to \boldsymbol{W}$.

In a general case, cells from the set $\boldsymbol{P}$ also may be of different types. This stratification is represented by the mapping ι: $\boldsymbol{P} \to \boldsymbol{V}$ where $\boldsymbol{V}$ is the set of cell types. Different types of cells may be used for storing different kinds of symbols. For example, binary cells, which have type B, store bits of information represented by symbols 1 and 0. Byte cells (type BT) store information represented by strings of eight binary digits. Symbol cells (type SB) store symbols of the alphabet(s) of the machine *M*. Cells in conventional Turing machines have this type. Natural number cells, which have type NN, are used in random access machines [1]. Cells in the memory of quantum computers (type QB) store q-bits or quantum bits. When different kind of devices are combined into one, this new device has several types of memory cells. In addition, different types of cells facilitate modeling the brain neuron structure by inductive Turing machines.

Likewise, the set of cells $\boldsymbol{P}$ is divided into three disjoint parts $\boldsymbol{P}_{\mathrm{I}}$, $\boldsymbol{P}_{\mathrm{W}}$, and $\boldsymbol{P}_{\mathrm{O}}$, where $\boldsymbol{P}_{\mathrm{I}}$ is the *input* registers, $\boldsymbol{P}_{\mathrm{W}}$ is the *working* memory, and $\boldsymbol{P}_{\mathrm{O}}$ is the *output* registers of *M*. Correspondingly, $\boldsymbol{K}$ is divided into three parts $\boldsymbol{K}_{\mathrm{I}}$, $\boldsymbol{K}_{\mathrm{W}}$, and $\boldsymbol{K}_{\mathrm{O}}$ which define connections between the cells from $\boldsymbol{P}_{\mathrm{I}}$, $\boldsymbol{P}_{\mathrm{W}}$, and $\boldsymbol{P}_{\mathrm{O}}$. Usually, input registers are used only for reading, while output registers are used only for writing. For simplicity, we consider $\boldsymbol{P}_{\mathrm{I}}$ as one register and $\boldsymbol{P}_{\mathrm{O}}$ as one register, which are, as a rule, one-dimensional tapes. Besides, it is possible to consider only such inductive Turing machines that have the read-only input register or tape.

At the same time, to model a modern computer with its advanced hierarchical memory, the set $\boldsymbol{P}$ has to be subdivided into more than three components.

Each cell from a linear two-sided tape has two neighbors left and right. The first cell in a one-sided tape has only one neighbor. The structure of a linear tape (the standard for Turing machines) is realized by the relation ***Lin*** with connections of two types: *R* and *L*. Each cell with the number *i* is connected to the cell with the number *i* + 1 with the connection *R*, and each cell with the number *i* + 1 is connected to the cell with the number *i* with the connection *L* (here *i* = 1, 2, 3, …). To get a two-sided linear tape, we reenumerate the corresponding part of $\boldsymbol{P}$ by integer numbers and use the same connections. To get a two-dimensional tape, we can use ties of four types $\boldsymbol{W}$ = {*R*, *L*, *U* and *D*}. Enumeration of cells by natural numbers is transformed to labeling

the cells by pairs of integer numbers. Then each cell (i, j) is connected to four of its neighbors: to $(i + 1, j)$ by the connection R, to $(i - 1, j)$ by the connection L, to $(i, j + 1)$ by the connection U, and to $(i, j - 1)$ by the connection D.

It is possible to realize an arbitrary structured memory of an inductive Turing machine, using only one linear one-sided tape L. To do this, the cells of L are enumerated in the natural order from the first one to infinity. Then L is decomposed into three parts according to the parts $\boldsymbol{P}_{\mathrm{I}}$, $\boldsymbol{P}_{\mathrm{W}}$, and $\boldsymbol{P}_{\mathrm{O}}$ of the structured memory. After this nonlinear connections between cells are installed according to the relation $\boldsymbol{K}$ and the mapping τ: $\boldsymbol{K} \to \boldsymbol{W}$. When an inductive Turing machine with this memory works, the head/processor is not moving to the right or to the left cell from a given cell, but uses the installed nonlinear connections.

Such realization of the structured memory allows us to consider an inductive Turing machine with a structured memory as an inductive Turing machine with conventional tapes in which additional connections are established. This approach has many advantages. One of them is that inductive Turing machines with a structured memory can be treated as multitape automata that have additional structure on their tapes. Then it is conceivable to study different ways to construct this structure.

In addition, this representation of memory allows us to consider any configuration in the structured memory E as a word written on this unstructured tape.

In a similar way, it is feasible to build, study and utilize Turing machines with a structured memory. They have almost the same hardware (they do not necessarily need the output tape, but always have final states) and the same software as inductive Turing machines with a structured memory. But in contrast to inductive Turing machines, Turing machines have to stop to produce a computational result.

If we look at other devices of the inductive Turing machine M, we can see that the processor H performs information processing in M. However, in comparison to computers, this operational device performs very simple operations. When H consists of one unit, it can change a symbol in the cell that is observed by H, and go from this cell to another using a connection from $\boldsymbol{K}$. This is exactly what the head of a Turing machine does.

It is possible that the processor H consists of several processing units similar to heads of a multihead Turing machine. This allows in a natural way one to model various real and abstract computing systems by inductive Turing machines. Examples of such systems are: multiprocessor computers; Turing machines with several tapes; networks, grids and clusters of computers; cellular automata; neural networks; and systolic arrays. However, such representation of information processing systems is not always efficient. This is why other models of information processing systems have been constructed, and are and will be utilized.

Connections between the control device A and the processor H may be differently organized:

1) The processor H may be rigidly connected to A. In this case, the memory E or its part moves when it is necessary to observe the next cell. This is similar to the work of a floppy disk or CD.

2) The connection between A and H may be flexible, allowing H or its parts to move from one cell to another under the control of A. This structure is virtually realized when data from the RAM of a computer are taken to registers of arithmetic units of the same computer.

3) Another option is that the processor H or its parts function autonomously from A, only sending to A information about the content of cells. In this case, H or its parts contain those instructions from the software that regulate operation of H or its parts. This mode of operation models the intelligent agent approach to computation. There an agent moves to the location of data and performs its operation at this new site.

We know that programs constitute computer software and tell the system what to do (and what to not do). The *software* $\boldsymbol{R}$ of the inductive Turing machine M is also a program; it is in the form of simple rules. The traditional representation assumes that the processor H functions as one unit. The rules for functioning have the following form:

$q_h a_i \to a_j q_k$, (1)

$q_h a_i \to c q_k$ (2)

It is also possible to use only rules of one form:

$q_h a_i \rightarrow a_j q_k c$ (3)

Here q_h and q_k are states of A, a_i and a_j are symbols of the alphabet of M, and c is a type of connection from $\boldsymbol{K}$.

Each rule directs one step of computation of the inductive Turing machine M. The rule (1) means that if the state of the control device A of M is q_h and the processor H observes in the cell the symbol a_i , then the state of A becomes q_k and the processor H writes the symbol a_j in the cell where it is situated. The rule (2) means that the processor H then moves to the next cell by a connection of the type c. The rule (3) is a combination of rules (1) and (2).

Like Turing machines, inductive Turing machines can be deterministic and nondeterministic. For a *deterministic inductive Turing machine*, there is at most one connection of any type from any cell. In a *nondeterministic inductive Turing machine*, several connections of the same type may go from some cells, connecting it with (different) other cells. If there is no connection of this type going from the cell that is observed by H, then H stays in the same cell. There may be connections of a cell with itself. Then H also stays in the same cell. It is possible that H observes an empty cell. To represent this situation, we use the symbol Λ. Thus, it is possible that some elements a_i and/or a_j in the rules from $\boldsymbol{R}$ are equal to Λ in the rules of both types. Such rules describe situations when H observes an empty cell and/or when H simply erases the symbol from some cell, writing nothing in it.

The rules of the type (3) allow an inductive Turing machine to rewrite a symbol in a cell and to make a move in one step. Other rule representations for inductive Turing machines separate these operations. Rules of the inductive Turing machine M define the transition function of M and describe changes of A, H, and E. Consequently, they also determine the transition functions of A, H, and E.

When the processor H consists of several processing units or heads, there are several functioning modes:

Uniform synchronized processing (processor units function synchronously): At each step of M each unit performs one operation; they all are controlled by the same system of rules.

Uniform concurrent processing (processor units function concurrently): Units perform operations independently of one another, but all of them are controlled by the same system of rules.

Specialized synchronized processing: Each processor unit has its own system of rules, but all of them function synchronously, i.e., at each step of M each unit performs one operation.

Specialized concurrent processing: Each processor unit has its own system of rules and they perform operations independently of one another.

In what follows, we consider for simplicity only the case when the processor H consists of one unit and M always starts functioning from the same state. Thus, the functioning of the inductive Turing machine M begins when the control device A is in the start state q_0 , the working and output memories are empty, and the processor H observes such a cell in the input register $\boldsymbol{P}_\mathrm{I}$ that this cell contains some symbol and has the least number of all non-empty cells in the input register. It is possible that nothing is written in the input register $\boldsymbol{P}_\mathrm{I}$. In this case, H observes an arbitrary cell. When H observes an empty cell, we denote the content of this cell by the symbol Λ.

A general step of the machine M has the following form. At the beginning of any step, the processor H observes some cell with a symbol a_i (for an empty cell the symbol is Λ) and the control device A is in some state q_h .

Then the control device A and/or the processor H choose from the system $\boldsymbol{R}$ of rules the rule r with the left part equal to $q_\mathrm{h}a_\mathrm{i}$ and perform the operation prescribed by this rule. If there is no rule in $\boldsymbol{R}$ with such left part, the machine M stops functioning. If there are several rules with the same left part, M works as a nondeterministic Turing machine (e.g., [22, 26]) performing all possible operations. When A comes to one of the final states from F, the machine M also stops functioning. In all other cases, it continues operation without stopping.

For an abstract automaton, as well as for a computer, two things are important. Specifically, not only how it functions, but also how it obtains its results. In contrast to Turing machines, inductive Turing machines obtain results even in the case when their operation is not terminated. This results in essential increase of performance abilities of systems of algorithms.

The computational result of the inductive Turing machine M is the word that is written in the output register $\boldsymbol{P}_O$ of M: when M halts while its control device A is in some final state from F, or when M never stops but at some step of computation the content of the output register $\boldsymbol{P}_O$ becomes fixed and does not change although the machine M continues to function. In all other cases, M gives no result.

Theorem 3. Any (inductive) Turing machine T with a recursive memory can be simulated by a (inductive) Turing machine D with one conventional tape, i.e., the machine D computes the same function as T, imitating all moves of T.

Proof here is similar to that for equivalence of different classes of Turing machines. It is done by the standard procedure in which D writes in a special tape consequent instantaneous descriptions of the machine T (e.g., [22]).

Some object that, in contrast to Turing machines, an inductive Turing machine does not always inform a user that a result was obtained. This is the cost that we have to pay for its higher computational power. However, mathematicians and computer scientists encountered similar situation with Turing machines. Having the class of all Turing machines or any other class of recursive algorithms, one never knows whether the given machine will produce the necessary result or not. In contrast to this, the condition that an algorithm always gives a result is often demanded. Trying to limit ourselves to recursive algorithms that always give a result brings us to the following situations: either we have a sufficiently powerful class but one cannot distinguish algorithms from this class from others or we can build all such algorithms but they have insufficient computational and decision power. Thus, we have to make a choice: either to use more powerful algorithms or to know more about algorithms that are used. From this perspective, an inductive Turing machine is the next step in the evergoing trade off between knowledge and power.

5. Learner models and learning potency

As the basic models for learning systems, we take: a polynomially bounded Turing machine (PBTM), which gives the most general model for solving tractable problems;

Turing machine (TM), which gives the most general conventional model for computations; and inductive Turing machine of the first order (ITM1), which gives the most general computational model that is the closest to Turing machines [15].

Definition 11. A Turing machine T is called polynomially bounded if there is a polynomial $P(n)$ and whenever T is given a word x of length n as input, computation of T halts after making at most $P(n)$ steps.

Thus, the function $P(n)$ gives a boundary for time complexity of T.

Definition 12. The memory E is called *recursive* if the relation $\boldsymbol{K} \subseteq \boldsymbol{P}\times\boldsymbol{P}$ that provides connections between cells and all mappings ν: $\boldsymbol{P} \to \boldsymbol{N}$, τ: $\boldsymbol{K} \to \boldsymbol{W}$, and ι: $\boldsymbol{P} \to \boldsymbol{V}$ are recursive.

Here recursive means that there are some Turing machines that decide/build all naming mappings and relations in the structured memory. In addition, we consider potentially achievable knowledge for a given model, which performs without any restrictions on accessible resources.

Theorem 4. Experience of previous generations does not add to potentially accessible knowledge for TM learners.

Proof. According to our model, experience of generations is accumulated in the structure of the memory of the learning model. Here we take a Turing machine M_0 with a structured memory as a model of a learner. This memory is built, i.e., all memory connections are established, by another Turing machine M_1 with a structured memory. If the memory of M_0 is developed by n generations, then the memory of M_1 is developed by $(n - 1)$ generations, and we can prove our statement by induction in the number n of generations.

1. $n = 1$. It means that there is only one generation and the learner receives no extra knowledge to build its memory. Thus, the memory of the learner is one or several linear tapes of an ordinary Turing machine, and the statement of the theorem is true by the definition.

2. $n > 1$ and we assume that that the statement of the theorem is true for $(n – 1)$. It means that the machine M_1 that develops the structure of the memory of the machine M_0 is an ordinary Turing machine.

To prove the necessary result, it is sufficient to show that functioning of M_0 can be modeled by an ordinary Turing machine T. In this case, in spite of its structured memory containing the experience of n previous generations, M_0 can do (and in particular, can learn) no more than the ordinary Turing machine T.

To model M_0, T has a working tape L_0 that is used for simulation of the memory of the machine M_0 and the working tape L_1 that is used for computation of all connections in the memory of M_0.

The machine T functions in the following way. Simulation of one step of the machine M_0 consists of three stages: at first, T finds an instruction u that corresponds to the relevant instruction v of M_0, then (stage 2) T computes the necessary connection that is prescribed by the instruction v, and only after this (stage 3), T performs the instruction u. Performance of u may demand several operations of T.

Standard methods of Turing machine modeling described, for example, in [22, 26], allow one to build a machine T that realizes operations of the machine M_0 step by step.

This completes the proof of Theorem 4.

Theorem 5. Experience of previous generations does not add to potentially accessible knowledge for PBTM learners.

Proof. To prove this statement, we take the proof of Theorem 4 and check that if time complexity of the machine M_0 is bounded by a polynomial $P_0(n)$ and time complexity of the machine M_1 is bounded by a polynomial $P_1(n)$, then time complexity of the modeling machine T is bounded by a polynomial $Q(n) = P_0(kP_0(n))$ where k is some number that depends only on M_0. The description of functioning of the machine T allows us to show that this is true and in such a way to prove Theorem 5.

As remarked Paul Stelling, if we take a fixed length m of input, experience of n generations with n much bigger than m allows one to achieve exponential, or even higher complexity, to solve problems that are not polynomially bounded for inputs with the length less than or equal to m. However, for all inputs, any fixed number of generations does not take us outside polynomial boundaries.

For inductive models of learners, we consider only learning of facts.

Definition 13. Inductive Turing machines with recursive memory are called *inductive Turing machines of the first order* (ITM1).

Theorem 6. In terms of arithmetical hierarchy, each generation adds one level to potentially accessible knowledge for ITM1 learners.

Proof. Elements of the arithmetical hierarchy are relations of natural numbers [30]**.** The set of all recursive relations is taken as the base for building the arithmetical hierarchy. Levels in the arithmetical hierarchy are labeled as $\mathbf{\Sigma_n}$ if they consist of all relations $\exists\, x_1\ \forall\, x_2\ \exists\, x_3\ \ldots\ \forall\, x_{n-2}\ \exists\, x_{n-1}\ \forall\, x_n\ R(x_1, \ldots, x_n, z_1, \ldots, z_m)$ limited to $n - 1$ pairs of alternating quantifiers starting with $\exists$ and having recursive $R(x_1, \ldots, x_n, z_1, \ldots, z_m)$. Similarly the class of all relations $\forall\, x_1\ \exists\, x_2\ \forall\, x_3\ \ldots\ \exists\, x_{n-1}\ \forall x_n\ R(x_1, \ldots, x_n, z_1, \ldots, z_m)$ that start with $\forall$ and have $n - 1$ alternations of quantifiers is labeled as $\mathbf{\Pi_n}$ and recursive $R(x_1, \ldots, x_n, z_1, \ldots, z_m)$. The classes $\mathbf{\Sigma_0}$ and $\mathbf{\Pi_0}$ are defined as having no quantifiers, consist of all recursive relations and thus, are equivalent. The classes $\mathbf{\Sigma_1}$ and $\mathbf{\Pi_1}$ are defined as having a single quantifier: relations from $\mathbf{\Sigma_1}$ have the form $\exists\, x\ R(x, y)$ and relations from $\mathbf{\Pi_1}$ have the form $\forall x\ R(x, y)$ where $R(x, y)$ is a recursive relation and y is an arbitrary vector of natural numbers. By the definition, we have inclusions $\mathbf{\Pi_m} \subseteq \mathbf{\Sigma_n} \cap \mathbf{\Pi_n}$ and $\mathbf{\Sigma_m} \subseteq \mathbf{\Sigma_n} \cap \mathbf{\Pi_n}$ for any $n > 1$ and any $m < n$.

Results from [29] show that it is sufficient to prove the theorem only for relations with one free variable, which have the form $\exists\, x_1\ \forall\, x_2\ \exists\, x_3\ \ldots\ \forall\, x_{n-2}\ \exists\, x_{n-1}\ \forall\, x_n\ R(x_1, \ldots, x_n, z)$ or $\forall\, x_1\ \exists\, x_2\ \forall\, x_3\ \ldots\ \exists\, x_{n-1}\ \forall x_n\ R(x_1, \ldots, x_n, z)$ and are some sets of natural numbers.

For the proof, we use induction on the number n of learning generations.

1. As the base for induction, we take one generation. According to the statement of the theorem this generation has to be able to learn sets from $\mathbf{\Sigma_1}$ and $\mathbf{\Pi_1}$. Results of of Gold [18, 19] and Burgin [10] show that ITM1 learners can learn, that is, decide or compute, any set from $\mathbf{\Sigma_n} \cup \mathbf{\Pi_n}$.

This completes the first step of our induction.

2. To make a general inductive step, we assume that if we have $(n - 1)$ learning generations, then the $(n - 1)^{th}$ generation can learn any relation from the class $\mathbf{\Sigma_{n-1}}$ and the class $\mathbf{\Pi_{n-1}}$. Let us consider an arbitrary relation $Q(z)$ that belongs to the class $\mathbf{\Sigma_n}$. It means that $Q(z) = \exists\, x_1\ \forall\, x_2\ \exists\, x_3\ \ldots\ \forall\, x_{n-2}\ \exists\, x_{n-1}\ \forall\, x_n\ R(x_1, \ldots, x_n, z)$, where $R(x_1, \ldots$

, x_n , z), is a recursive relation. Then, by the definition, the relation $K(x_1 , z) = \forall x_2 \exists x_3 \ldots \forall x_{n-2} \exists x_{n-1} \forall x_n R(x_1 , \ldots , x_n , z)$ belongs to the class $\mathbf{\Pi_{n-1}}$.

By our assumption, a learner from the $(n-1)^{th}$ generation can decide whether a given pair (x, z) belongs to the relation $K(x_1, z)$ or not. Utilizing this knowledge, an ITM1 learner M checks if the pair $(1, z)$ belongs to $K(x_1, z)$. Then it checks if the pair $(2, z)$ belongs to $K(x_1, z)$. Then it repeats the same for the pair $(3, z)$ and so on. After each test with a negative result, M writes 0 on its output tape. If one of these tests gives the positive result, then M writes 0 on its output tape and continues to do so forever.

In such a way, the result of M is 1 when z belongs to $Q(z)$ and is 0 when z does not belong to $Q(z)$. It means that M learns the fact whether z belongs to $Q(z)$ or not.

In a similar way, we can prove that some ITM1 learner M decides whether an arbitrary number z belongs to $Q(z)$ or not for an arbitrary relation $Q(x)$ that belongs to the class $\mathbf{\Pi_n}$. The difference in the proof is that in this case M outputs 1 when before it gave 0 and outputs 0 when before it gave 1.

This completes the general step of our induction.

By induction the statement of the theorem is true. Theorem 6 is proved.

5. Conclusion

The paper analyzed how power grows when a learner uses the experience of previous generations. It is demonstrated that neither recursive nor subrecursive algorithms give any increase of potentially learnable knowledge. Only super-recursive algorithms allow achieving better results in stratified learning.

These results explain why the historical approach to education is good for history, art, and literature, but is bad for mathematics and natural sciences. The reason is that mathematics and natural sciences do not achieve independent results. Due to their quest for unification, they build shortcuts. This is possible only using experience and knowledge of previous generations. Consequently, long ways by which past generations reached essential results become unreasonable.

To conclude, we formulate some open problems. An interesting problem is to consider how experience of generations influences other learner models, for instance, such as Turing machines with oracles or with advice and inductive Turing machines with oracles or with advice.

As it has been demonstrated (cf., [31]), learning/cognition in teams is more powerful and efficient than individual learning. That is why another interesting problem is to consider how experience of generations influences learning and cognition in a group.

Acknowledgments

The authors are grateful to Paul Stelling for helpful remarks.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ulman, *The Design and Analysis of Computer Algorithms*, Reading, Mass., Addison-Wesley P.C. 1976

[2] A. Ambainis, Relations between Two Types of Memory in Inductive Inference (unpublished manuscript)

[3] D. Angluin, Computational Learning Theory: Survey and Selected Bibliography, in *Proc. 24th ACM Symposium on Theory of Computation* (1992) 319-342

[4] D. Angluin, C. Smith, Inductive Inference: Theory and Methods, *Comput. Surveys*, 15:3 (1983) 237—269

[5] K. Apsïtis, S. Arikawa, R. Freivalds, E. Hirowatari, C. H. Smith, On the inductive inference of recursive real-valued functions. Computability and complexity in analysis, *Theoretical Computer Science* , 219:1-2 (1999) 3-17

[6] R.L. Atkinson, R.C. Atkinson, E.E. Smith, D.J. Bem, *Introduction to Psychology*, Harcourt Brace Jovanovich, Inc., San Diego/New York/Chicago, 1990

[7] R.C. Atkinson, R.M. Shiffrin, Human Memory: A Proposed System and its Control processes, in *The Psychology of Learning and Motivation*, Academic Press, New York, 1968

[8] J.L. Balcazar, J. Diaz, J. Gabarro, *Structural Complexity*, Springer-Verlag, Berlin/Heidelberg/ New York, 1988

[9] E.T. Bell, *Men of Mathematics*, Simon and Schuster, New York, 1965

[10] M. S. Burgin, M. Nonlinear Phenomena in Spaces of Algorithms, *International Journal of Computer Mathematics*, v. 80, No. 12 (2003) 1449-1476

[11] M. Burgin, Super-recursive Algorithms as a Tool for High Performance Computing, *Proceedings of the High Performance Computing Symposium*, San Diego (1999) 224-228

[12] M. Burgin, How We Know What Technology Can Do, *Communications of the ACM*, 44:11 (2001) 82-88

[13] M. Burgin, Topological Algorithms, in Proceedings of the ISCA 16th International Conference "*Computers and their Applications*", ISCA, Seattle, Washington (2001) 61-64

[14] M. Burgin, Cluster Computers and Grid Automata, in Proceedings of the ISCA 17th International Conference "*Computers and their Applications*", International Society for Computers and their Applications, Honolulu, Hawaii (2003) 106-109

[15] M. Burgin, *Super-recursive Algorithms*, Springer, New York/Berlin/Heidelberg, 2004

[16] D.M. Burton, *The History of Mathematics*, The McGrow Hill Co., New York, 1997

[17] R. Freivalds, E. Kinber, C. Smith, On the impact of forgetting on learning machines, *Proceedings of the 6-th ACM COLT* , San Jose, California (1993) 165-174

[18] E.M. Gold, Language Identification in the Limit, *Information and Control* 10 (1967) 447-474

[19] E.M. Gold, Limiting Recursion, *Journal of Symbolic Logic*, 30:1 (1965) 28-46

[20] P. Gray, *Psychology*, Worth Publishers, New York, 1994

[21] S. Haykin, *Neural Networks: A Comprehensive Foundation*, New York, Macmillan, 1994

[22] J.E., Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory*, *Languages, and Computation*, Addison Wesley, Boston/San Francisco/New York, 2001

[23] L.C., Katz, C.J. Shatz, Synaptic activity and the construction of cortical circuits, *Nature* 274 (1996) 1133-1138

[24] K.T. Kelly, O. Schulte and C. Juhl, Learning Theory and the Philosophy of Science, *Philosophy of Science* 64 (1997) 245-267

[25] E.I. Knudsen, Instructed learning in the auditory localization pathway of the barn owl, *Nature*, 417 (2002) 322-328

[26] J.C. Martin, *Introduction to Languages and the Theory of Computation*, McGrow-Hill, New York, 1991

[27] M. Minsky, *The Society of Mind*, Simon and Schuster, New York, 1986

[28] J.J. Parsons and D. Oja, *New Perspectives on Computer Concepts*, Course Technology, Inc., Cambridge, Massachusetts, 1994

[29] H. Putnam, Trial and Error Predicates and the Solution to a Problem of Mostowski, *Journal of Symbolic Logic*, 30:1 (1965) 49-57

[30] H. Rogers, *Theory of Recursive Functions and Effective Computability,* MIT Press, Cambridge, Massachusetts, 1987

[31] U. Schöning, Complexity theory and interaction, in *The Universal Turing Machine - A Half-Century Survey,* Oxford University Press, Oxford (1988) 561-580

[32] C.H. Smith, A Brief Survey of Team Learning, in Proceedings of the International Workshop *Quantum Computation and Learning*, Riga, Latvia, 1999

[33] N. Tinbengen, *The Study of Instinct*, Oxford University Press, New York, 1976

[34] N.C. Waugh, D.A. Norman, Primary Memory, *Psychological Review* 72 (1965) 89-104