Applying an Almost Optimal Monte Carlo Algorithm for Integral Equations in Neuron Networks

Venelin Todorov, Ivan Dimov
IICT, Bulgarian Academy of Sciences
Acad. G. Bonchev 25 A, 1113 Sofia, Bulgaria

Abstract: *In this paper we discuss Monte Carlo algorithm for integral equation which describes the procedure of teaching of neuron networking. An almost optimal Monte Carlo method for integral equations based on balancing of systematic and stochastic errors is presented. An approach to the problem of controlling the error in non- deterministic methods is shown. Lower bounds for the number of realizations and number of iterations in the algorithm are provided once a preliminary given error is given. Fredholm integral equation of the second kind for neuron networks is presented and numerical results are discussed.*
Key words: *almost optimal monte carlo algorithm, integral quations, neuron networks*

Introduction

Neural networks are promising and convenient data models for computation and representation of complex relations between inputs and outputs. They are developed in order to allow execution of "intelligent" tasks similar to those performed by the biological neural network – the brain. Thus, like the brain neural networks acquire knowledge through learning and store it within inter-neuron connections, known as synaptic weights. The reason for them to gain special attention in engineering and science is because they have ability to represent both linear and non-linear relationships and can learn these relationships directly from the data model. Traditional linear models are inapplicable for modelling nonlinear data. The Monte Carlo method is a powerful tool in solving problems in neuron networking. It is known that the algorithms based on this method give statistical estimates for the functional of the solution by performing random sampling of a certain random variable whose mathematical expectation is the desired functional. An almost optimal Monte Carlo algorithm with probabilities chosen to be proportional to the function from the linear functional under consideration and the kernel has high algorithmic efficiency and this is one of the reasons to use it in order to solve the integral equation describing the procedure of learning neuron networks.

Preliminaries

The human brain was always an inspiration for engineers and scientists. The neuroscientist Warren S. McCulloch and the logician Walter Pitts, developed the first conceptual model of an artificial neural network in 1943. This was published in the paper "A logical calculus of the ideas imminent in nervous activity," where a conceptual model in which, a neuron as a single cell living in a network of cells receives inputs, processes those inputs, and generates an output, is described. The focus on theirs and the works after that, was not on how biological brain works, but rather on development of artificial neural network resembling the brain as a computational model for solving certain tasks. Main strength of neural networks is their ability to perform tasks difficult for machines – pattern recognition. Unlike the ordinary computational systems which are linear – executing code step by step till the last command, neural network does not follow a linear path. Information is processed collectively, in parallel through network of nodes/neurons.

A special advantage of neural networks is their ability to adapt based on learning and changing their internal structure making possible the existence of artificial intelligence. Usually done by adjusting the weights. We consider a connection between two neurons and make the pathway for the flow of information. Each connection has a weight, a number that controls the signal between the two neurons. If the network generates a

"good" output, there is no need to adjust the weights. However, if the network generates a "poor" output- an error, so to speak- then the system adapts, altering the weights in order to improve subsequent results. There are several strategies for learning. *Supervised Learning* -essentially, a strategy that involves a teacher that is smarter than the network itself. For example, let's take the facial recognition example. The teacher shows the network a bunch of faces, and the teacher already knows the name associated with each face. The network makes its guesses, and then the teacher provides the network with the answers. The network can then compare its answers to the known "correct" ones and make adjustments according to its errors. *Unsupervised Learning*-required when there isn't an example data set with known answers. Imagine searching for a hidden pattern in a data set. An application of this is clustering, i.e. dividing a set of elements into groups according to some unknown pattern. *Reinforcement Learning*- a strategy built on observation. Think of a little mouse running through a maze. If it turns left, it gets a piece of cheese; if it turns right, it receives a little shock. Presumably, the mouse will learn over time to turn left. Its neural network makes a decision with an outcome (turn left or right) and observes its environment (yum or ouch). If the observation is negative, the network can adjust its weights in order to make a different decision the next time. Reinforcement learning is common in robotics. At time t, the robot performs a task and observes the results. [7]

Commonly Neural networks are used in optical character recognition, facial recognition, adaptive audio filters, soft sensors, weather forecasting etc. Last years they made possible the emerging of smart systems like self-driving cars, smart microgrids, renewable energy sources (RES) forecasting, battery/energy storage monitoring and forecasting, failure monitoring and anomaly detection, decision making controllers etc.

Monte Carlo algorithms for integral equations

In general, Monte Carlo numerical algorithms may be divided into two classes direct algorithms and iterative algorithms. The direct algorithms provide an estimate of the solution of the equation in a finite number of steps, and contain only a stochastic error. For example, direct Monte Carlo algorithms are the algorithms for evaluating. Iterative Monte Carlo algorithms deal with an approximate solution obtaining an improved solution with each step of the algorithm. In principle, they require an infinite number of steps to obtain the exact solution, but usually one is happy with an approximation to say *k* significant figures. In this latter case there are two errors - systematic and stochastic [2]. The systematic error depends both on the number of iterations performed and the characteristic values of the iteration operator, while the stochastic errors depend on the probabilistic nature of the algorithm. Iterative algorithms are preferred for solving integral equations and large sparse systems of algebraic equations (such as those arising from approximations of partial differential equations). Such algorithms are good for diagonally dominant systems in which convergence is rapid; they are not so useful for problems involving dense matrices.

Define an iterator of degree *j* as

$$u^{(k+1)} = F_k(A, b, u^{(k)}, u^{(k-1)}, ..., u^{(k-j+1)}), \quad (1)$$

where $u^{(k)}$ is obtained from k^{th} iteration. It is desired that

$$u^{(k)} \to u = A^{-1}b, k \to \infty \quad (2)$$

Usually the degree of *j* is kept small because of storage requirements.

The iteration is called stationary if $F_k = F$ for all *k*, that is F_k is independent of *k*.

The iterative process is said to be linear if F_k is a linear function of $u^{(k)}, u^{(k-1)}, ..., u^{(k-j+1)}$.

We shall consider iterative stationary linear Monte Carlo algorithms and will analyse both systematic and stochastic errors. Sometimes the iterative stationary linear Monte

Carlo algorithms are called Power Monte Carlo algorithms. The reason is that these algorithms find an approximation of a functional of powers of linear operators. In literature this class of algorithms is also known as Markov chain Monte Carlo since the statistical estimates can be considered as weights of Markov chains.

Consider a general description of the iterative Monte Carlo algorithms. Let X be a Banach space of real-valued functions. Let $f = f(x) \in X, u_k = u(x_k) \in X$ be defined in $\mathbb{R}^d$ and $L = L(u)$ be a linear operator defined on X.

Consider the sequence $u_1, u_2, ...,$ defined by the recursion formula

$$u_k = L(u_{k-1}) + f, k = 1, 2, ... \tag{3}$$

The formal solution of the above is the truncated Neuman series

$$u_k = f + L(f) + ... + L^{k-1}(f) + L^k(u_0), k > 0, \tag{4}$$

where L^k means the k^{th} iteration of L.

Let's consider the integral functions.

Let $u(x) \in X, x \in \Omega \subset \mathbb{R}^d$ and $l(x, x')$ be a function defined for $x \in \Omega, x' \in \Omega$. The integral transformation

$$Lu(x) = \int_\Omega l(x, x')u(x')dx' \tag{5}$$

maps the function $u(x)$ into the function $Lu(x)$, and is called an iteration of $u(x)$ by the integral transformation kernel $l(x, x')$. The second integral iteration of $u(x)$ is denoted by

$$LLu(x) = L^2u(x) \tag{6}$$

Obviously,

$$L^2u(x) = \int_\Omega \int_\Omega l(x, x')l(x', x'')dx'dx''. \tag{7}$$

In this way $L^3u(x), ..., L^iu(x), ...$ can be defined.

When the infinite series converges, the sum is an element u from the space X which satisfies the equation

$$u = L(u) + f \tag{8}$$

The truncation error of (4) is

$$u_k - u = L^k(u_0 - u). \tag{9}$$

Let $J(u_k)$ be a linear function that is to be calculated. Consider the spaces

$$T_{i+1} = \mathbb{R}^d \times \mathbb{R}^d \times ... \times \mathbb{R}^d, i = 1, 2, ..., k, \tag{10}$$

where $\times$ denotes the Catesian product of spaces.

Random variables $\theta_i, i = 0, 1, ..., k$ are defined on the respective product spaces T_{i+1} and have conditional mathematical expectations:

$$E\theta_0 = J(u_0), E(\theta_1 / \theta_0) = J(u_1), ..., E(\theta_k / \theta_0) = J(u_k), \tag{11}$$

where $J(u)$ is a linear functional of u.

The computational problem then becomes one of calculating repeated realizations of θ_k and combining them into an appropriate statistical estimator of $J(u_k)$.

As an approximate value of the linear functional $J(u_k)$ is set up

$$J(u_k) \approx \frac{1}{N} \sum_{s=1}^{N} \{\theta_k\}_s, \tag{12}$$

where $\{\theta_k\}_s$ is the s^{th} realization of the random variable θ_k.

The probable error r_N of the above is then $r_N = c\sigma(\theta_k)N^{-\frac{1}{2}}$,

where $c \approx 0.6745$ [4] and $\sigma(\theta_k)$ is the standard deviation of the random variable θ_k.

There are two approaches which correspond to two special cases of the operator L: L is a matrix and u and f are vectors; and L is an ordinary integral transform

$$L(u)=\int_{\Omega} l(x,y)u(y)dy \tag{13}$$

and $u(x)$ and $f(x)$ are functions.

In this paper we consider the second case. The equation (8) becomes

$$u(x)=\int_{\Omega} l(x,y)u(y)dy+f(x) \text{ or } u=Lu+f. \tag{14}$$

Monte Carlo algorithms frequently involve the linear functionals of the solution of the following type

$$J(u)=\int_{\Omega} h(x)u(x)dx=(u,h) \tag{15}$$

In fact the last equation defines the inner product of a given function $h(x)\in X$ with the solution of the integral equation (14).

Sometimes the adjoint equation is also used

$$v=L^{*}v+h \tag{16}$$

In our example $X=X^{*}=L_2$. Note also, that if $h(x),u(x)\in L_2$ then the inner product (15) is finite. In fact,

$$\left|\int_{\Omega} h(x)u(x)dx\right|\le\int_{\Omega}|h(x)u(x)|dx\le\left\{\int_{\Omega} h^2dx\int_{\Omega} u^2dx\right\}^{\frac{1}{2}}<\infty. \tag{17}$$

One can also see, that if $u(x)\in L_2$ and $l(x,x')\in L_2(\Omega\times\Omega)$ then $Lu(x)\in L_2$:

$$|Lu(x)|^2\le\left\{\int_{\Omega}|lu|dx'\right\}^2\le\int_{\Omega} l^2(x,x')dx'\int_{\Omega} u^2(x')dx'. \tag{18}$$

Let us integrate the last inequality with respect to x:

$$\int_{\Omega}|Lu|^2dx\le\int_{\Omega}\int_{\Omega} l^2(x,x')dx'dx\int_{\Omega} u^2(x')dx'<\infty. \tag{19}$$

From the last inequality follows that $L^2u(x),...,L^iu(x),...$ also belongs to $L^2(\Omega)$.

If it is assumed that $\|L^m\|<1$, where m is a natural number, then the Neuman series converges.

$$u=\sum_{i=0}^{\infty} L^if \tag{20}$$

It is easy to show that:

$$J=(h,u)=(f,v). \tag{21}$$

Let us multiply (14) by u and (16) by v and integrate. We obtain

$$(v,u)=(u,Lv)+(v,f);(v,u)=(L^{*}v,u)+(h,u). \tag{22}$$

Since

$$\begin{aligned}(L^{*}v,u)&=\int_{\Omega} L^{*}v(x)u(x)dx=\int_{\Omega}\int_{\Omega} l^{*}(x,x')v(x')u(x)dxdx'=\\&=\int_{\Omega}\int_{\Omega} l(x',x)v(x')u(x)dxdx'=\int_{\Omega} Lu(x')v(x')dx'=(v,Lu),\end{aligned} \tag{23}$$

we have

$$(L^{*}v,u)=(v,Lu). \tag{24}$$

Thus $(h,u)=(f,v)$. The kernel $l(x,x')$ is called transposed kernel.

Consider the Monte Carlo algorithm for evaluating the functional (15). It can be seen that when $l(x,x')\equiv 0$. evaluation of the integers can pose a problem. Consider a random point $\xi\in\Omega$ with a density $p(x)$ and let be N realizations of the random point $\xi_i\,(i=1,2,...,N)$. Let a random variable $\theta(\xi)$ be defined in Ω such that

$$E\theta(\xi) = J. \tag{25}$$

Then the computational problem becomes one of calculating repeated realizations of θ and of combining them into an appropriate statistical estimator of J. Note that the nature of the every process realization of θ is a Markov process. We will consider only discrete Markov processes with a finite set of states, the so called Markov chains. An approximation value of the linear functional J is

$$J \approx \frac{1}{N}\sum_{i=1}^{N}(\theta)_s = \theta_N, \tag{26}$$

where $(\theta)_s$ is the s^{th} realization of the random variable θ.

The random variable whose mathematical expectation is equal to $J(u)$ is given by the following expression

$$\theta[h] = \frac{h(\xi_0)}{p(\xi_0)}\sum_{j=0}^{\infty} W_j f(\xi_j), \tag{27}$$

where

$$W_0 = 1, W_j = W_{j-1}\frac{l(\xi_{j-1},\xi_j)}{p(\xi_{j-1},\xi_j)}, j = 1,2,..., \tag{28}$$

$\xi_0, \xi_1, ...$ is a Markov chain in Ω with initial density function $p(x)$ and transition density function $p(x,y)$.

For the case where L is a matrix, the equation can be written as

$$u_k = L^k u_0 + L^{k-1} f + ... + Lf + f = \left(I - L^k\right)(I-L)^{-1} f + L^k u_0, \tag{29}$$

where I is the unit or identity matrix,

$L = \{l_{ij}\}_{i,j=1}^{n}; u_0 = \left(u_1^0, ..., u_n^0\right)^T$

and matrix $I - L$ is supposed to be non-singular.

It is well known that if all eigenvalues of the matrix L lie within the unit circle of the complex plane then there exist a vector u such that

$$u = \lim_{k\to\infty} u_k, \tag{30}$$

which satisfies the equation

$$u = Lu + f. \tag{31}$$

Now we consider the problem of evaluating the inner product

$$J(u) = (h,u) = \sum_{i=1}^{n} h_i u_i, \tag{32}$$

where $h \in \mathbb{R}^{n\times 1}$ is a given vector column.

To define a random variable whose mathematical expectation coincides the above functional for $u = Lu + f$ first consider the integral equation (14) for which $\Omega_i = [i-1, i), i = 1,2,...,n$ such that

$$\begin{cases} l(x,y) = l_{ij}, x \in \Omega_i, y \in \Omega_j \\ f(x) = f_i, x \in \Omega_i \end{cases} \tag{33}$$

Then the integral equation (14) becomes

$$u_i = \sum_j \int_{\Omega_j} l_{ij} u(y) dy + f_i \tag{34}$$

for $u_i \in \Omega_i$. Denote that

$$u_j = \int_{\Omega_j} u(y) dy \tag{35}$$

so that one obtains for $u(x) \in \Omega_i$

$$u(x)=\sum_{j=1}^{n} l_{ij}u_j + f_i . \tag{36}$$

From the last equation it follows that $u(x)=u_i$ and so,

$$u_i=\sum_{j=1}^{n} l_{ij}u_j + f_i \tag{37}$$

or in a matrix form

$$u=Lu+f , \tag{38}$$

where $L=\{l_{ij}\}_{i,j=1}^{n}$.

The above presentation permits the consideration of the following random variable

$$\theta[h]=\frac{h_{\alpha_0}}{p_0}\sum_{v=0}^{\infty} W_v f_{\alpha_v}, \tag{39}$$

where $$W_0=1, W_v=W_{v-1}\frac{l_{\alpha_{v-1},\alpha_v}}{p_{\alpha_{v-1},\alpha_v}}, v=1,2,..., \tag{40}$$

and $\alpha_0,\alpha_1,...$ is a Markov chain on elements of the matrix L created by using an initial probability p_0 and a transition probability $p_{\alpha_{v-1},\alpha_v}$ for choosing the element $l_{\alpha_{v-1},\alpha_v}$ of the matrix L.

Consider the initial density vector $p=\{p_i\}_{i=1}^{n}\in\mathbb{R}^n$, such that $p_i\geq 0, i,j=1,...,n$ and $\sum_{i=1}^{n} p_i=1$.

Consider also a transition density matrix $P=\{p_{ij}\}_{i,j=1}^{n}\in\mathbb{R}^{n\times n}$, such that $p_{ij}\geq 0, i,j=1,...,n$ and $\sum_{j=1}^{n} p_{ij}=1$ for any $i=1,...,n$ [6].

Define sets of permissible densities P_h and P_L.

The initial density vector $p=\{p_i\}_{i=1}^{n}$ is called permissible to the vector $h=\{h_i\}_{i=1}^{n}\in\mathbb{R}^n$, i.e. $p\in\mathrm{P}_h$, if $p_i>0$ when $h_i\neq 0$ and $p_i=0$ when $h_i=0$ for $i=1,...,n$.

The transition density matrix $P=\{p_{ij}\}_{i,j=1}^{n}$ is called permissible to the matrix $L=\{l_{ij}\}_{i,j=1}^{n}\in\mathbb{R}^{n\times n}$ i.e. $p\in\mathrm{P}_L$, if $p_{ij}>0$ when $l_{ij}\neq 0$ and $p_{ij}=0$ when $l_{ij}=0$ for $i,j=1,...,m$.

Consider the following Markov chain

$$T_i=\alpha_0\rightarrow\alpha_1\rightarrow...\rightarrow\alpha_i , \tag{41}$$

where $\alpha_j=1,2,...,i$ for $j=1,...,i$ are natural random numbers.

The rules for defining the Markov chain are:

$$\Pr(\alpha_0=\alpha)=p_\alpha, \Pr(\alpha_j=\beta|\alpha_{j-1}=\alpha)=p_{\alpha\beta}. \tag{42}$$

Assume that $p=\{p_\alpha\}_{\alpha=1}^{n}\in P_h$, $P=\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}\in P_L$.

Now define random variables $W_v, v=1,...,i.$ from (40). One can see that random variables W_v can also be considered as weights on the Markov chain.

From all possible permissible densities we choose the following

$$p=\{p_\alpha\}_{\alpha=1}^{n}\in P_h, p_\alpha=\frac{|h_\alpha|}{\sum_{\alpha=1}^{n}|h_\alpha|}; \tag{43}$$

$$P=\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}\in P_L, p_{\alpha\beta}=\frac{|l_{\alpha\beta}|}{\sum_{\beta=1}^{n}|l_{\alpha\beta}|}, \alpha=1,\ldots,n. \quad (44)$$

Such a choice of the initial density vector and the transition density matrix leads to an Almost Optimal Monte Carlo (MAO) algorithm. The initial density vector $p=\{p_\alpha\}_{\alpha=1}^{n}$ is called almost optimal initial density vector and the transition density matrix $P=\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$ is called almost optimal density matrix. Such density distributions lead to almost optimal. The reason to use MAO instead of Uniform Monte Carlo is that MAO normally gives much smaller variances. On the other hand, the truly optimal weighted algorithms are very time consuming, since to define the optimal densities one needs to solve an additional integral equation with a quadratic kernel. This procedure makes the optimal algorithms very expensive.

In this paper we will combine MAO algorithm described above with the idea of balancing of the errors described in the next section.

Balancing of the errors.

We have two errors in our Monte Carlo algorithm- stochastic and systematic errors. The systematic error depends both on the number of iterations performed and the characteristic values of the iteration operator, while the stochastic errors depend on the probabilistic nature of the algorithm. In order to obtain good results the stochastic error r_N must be approximately equal to the systematic error r_k or

$$r_N=O(r_k) \quad (45)$$

The problem of balancing the errors is closely connected with the problem of obtaining an optimal ratio between the number of realizations of the random variable and the number of iterations in each random trajectory. We will use the optimal ratio obtained in [1].

We obtained the following lower bounds for the two errors [4]:

$$r_N\le\frac{0.6745\|f\|_{L_2}\|\varphi\|_{L_2}}{\sqrt{N}(1-\|K\|_{L_2})}, \quad (46)$$

$$r_k\le\frac{\|f\|_{L_2}\|\varphi\|_{L_2}\|K\|_{L_2}^{k+1}}{1-\|K\|_{L_2}}, \quad (47)$$

where $\|f\|_{L_2}$, $\|\varphi\|_{L_2}$, $\|K\|_{L_2}$ are the L_2 norms of the right hand side, the fraction from the linear functional under consideration and the kernel respectively [5].

In order to solve the integral equation we use Monte Carlo technique balancing of the errors. In [1] are obtained error balancing conditions when the systematic and stochastic errors are approximately equal, once a preliminary given error δ is given.

Theorem1. In Monte Carlo algorithm with a balancing o the errors the lower bounds for the number of iterations and realizations are given by the following inequalities:

$$k \geq \frac{\ln \frac{\delta(1-\|K\|_{L_2})}{2\|f\|_{L_2}\|\varphi\|_{L_2}\|K\|_{L_2}}}{\ln\|K\|_{L_2}} \tag{48}$$

$$N \geq \left(\frac{1.349\|f\|_{L_2}\|\varphi\|_{L_2}}{\delta(1-\|K\|_{L_2})}\right)^2 \tag{49}$$

Theorem 2. In Monte Carlo algorithm with a balancing of the errors one may choose the following exact values for the number of realizations of the random variable and the mean value of the number of steps in each random trajectory:

$$N = \left\lceil \left(\frac{1.349\|f\|_{L_2}\|\varphi\|_{L_2}}{\delta(1-\|K\|_{L_2})}\right)^2 \right\rceil \tag{50}$$

$$k = \left\lceil \frac{\ln \frac{0.6745}{\|K\|_{L_2}\sqrt{N}}}{\ln\|K\|_{L_2}} \right\rceil . \tag{51}$$

Numerical example

We study the following Fredholm integral equation of the second kind which describes the procedure of learning neuron networks:

$$u(x) = \int_{\Omega} k(x,x')u(x')dx' + f(x), \tag{52}$$

where $x, x' \in \Omega \subset \mathbb{R}^d, u(x), f(x) \in L_2(\Omega), k(x,x') \in L_2(\Omega x \Omega)$.

The domain is the interval $\Omega \equiv [-2,2]$ and the kernel, right hand side and the function from the linear functional that we want to evaluate is given by the following expressions:

$$k(x,x') = \frac{0.055}{1+e^{-3x}} + 0.07\,,\; f(x) = 0.02\left(3x^2 + e^{-0.35x}\right),\; \varphi(x) = 0.7((x+1)^2\cos(5x)+20) \tag{53}$$

We want to find the linear functional from the solution (φ, u), where

$$J(u) = \int \varphi(x)u(x)dx = (\varphi, u) \tag{54}$$

The exact solution of the integral equation which describes the procedure of learning neuron networks is 8.98635750518 [3].

Numerical experiments.

We apply (50) and (51) to estimate N and k. In order to do this we evaluate:

$$\|f\|_{L_2} = 0.2510, \|K\|_{L_2} = 0.2001, \|\varphi\|_{L_2} = 27.7782. \quad (55)$$

We make 20 algorithm runs on Intel Core i5-2410M @ 2.3 GHz.

We consider the almost optimal Monte Carlo algorithm based on balancing of the errors, i.e. we choose the initial and transition probabilities in the Monte Carlo algorithm for integral equation to be proportional to the function from the linear functional under consideration and the kernel respectively.

We give the preliminary given error different values and estimate N and k by theorem 1 and theorem 2. We compare the experimental relative error with the expected theoretical error. We can see the results in the table shown below. Theoretical or expected relative error is obtained by dividing the preliminary given error by the exact value. We evaluate experimental relative error by the almost optimal Monte Carlo algorithm for integral equations based on balancing of the two errors. We can see that experimental relative error confirms the expected theoretical error. The results are getting better when the number of realizations increases to 10^6. This shows the strength of the applied almost optimal Monte Carlo algorithm.

Table1. Experimental relative error and computational time for the integral equation describing the procedure of learning networks solved by MAO algorithm based on balancing of the errors

δ	N	k	Theoretical relative error	Experimental relative error	Time, sec.
0.075	24581	4	0.00835	0.0033	69.2
0.04	86415	5	0.00445	0.0030	264.1
0.035	112689	5	0.00391	0.0028	322.3
0.025	221196	6	0.00282	0.0018	640.5
0.012	960050	7	0.00134	0.000991	3952.2
0.01	1382471	7	0.00110	0.000972	4542.3
0.008	2160371	8	0.000890	0.000883	9871.7

Conclusion

An introduction to artificial neuron networking is presented. Learning neuron networks technique is discussed. Fredholm integral equation of the second kind which describes the procedure of learning neuron networks is solved. An almost optimal Monte Carlo algorithm for integral equations based on balancing of the systematic and stochastic errors is applied. Theorems for evaluating the number of realizations of the random variable and number of iterations in the Markov chain are presented. Numerical experiments and results are discussed. We see that experimental relative error confirms the expected theoretical error. It is shown that the presented almost optimal Monte Carlo algorithm gives results which are very close to the expected error when the number of realizations of the random variable is above 10^6. It is shown that the idea of combining MAO algorithm with balancing of the errors gives optimal results for solving this integral equation.

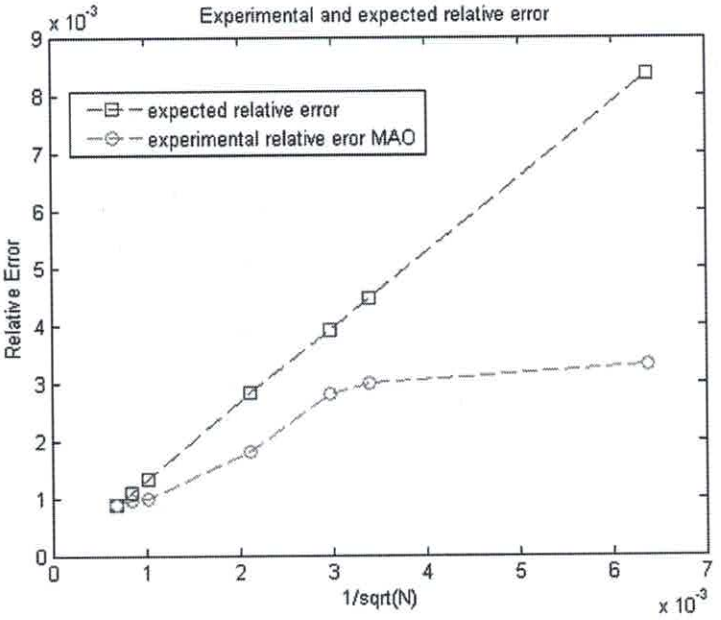

Fig. 1. Expected and experimental relative error for the integral equation describing the procedure of learning of neuron networks obtained with MAO algorithm based on balancing of the errors.

Bibliography

[1] I. Dimov, R. Georgieva, V. Todorov, Balancing of Systematic and Stochastic Errors in Monte Carlo Algorithms for Integral Equations, LNCS89622, Proceeding of 8th International Conference Numerical Methods and Applications, NMA 2014, Borovets, Bulgaria, August 20-24, 2014

[2] I. Dimov, Monte Carlo Methods for Applied Scientists, New Jersey, London, Singapore, World Scientific, 2008, 291p.

[3] R. Georgieva, PhD Thesis: Computational complexity of Monte Carlo algorithms for multidimensional integrals and integral equations, Sofia, 2003

[4] I. Sobol. Numerical methods Monte Carlo. Nauka, Moscow, 1973.

[5] J.H. Curtiss. Monte Carlo Methods for The Iteration of Linear Operators.

[6] I. Dimov, Minimization of the Probable Error for Some Monte Carlo methods. Proc. Int. Conf. on Mathematical Modeling and Scientific Computation, Albena, Bulgaria, Sofia, Publ. House of the Bulgarian Academy of Sciences, 1991, 159-170.

[7] D. Shiiffman, The Nature of Code ISBN-13: 978-0985930806, Ch 10