UDC 004:37

Oksana Markova[1], Serhiy Semerikov[2], Maiia Popel[3]
[1] State Institution of Higher Education «Kryvyi Rih National University», Kryvyi Rih, Ukraine
[2] Kryvyi Rih State Pedagogical University, Kryvyi Rih, Ukraine
[3] Institute of Information Technologies and Learning Tools of NAES of Ukraine, Kyiv, Ukraine

COCALC AS A LEARNING TOOL FOR NEURAL NETWORK SIMULATION IN THE SPECIAL COURSE "FOUNDATIONS OF MATHEMATIC INFORMATICS"

DOI: 10.14308/ite000674

The role of neural network modeling in the learning content of special course "Foundations of Mathematic Informatics" was discussed. The course was developed for the students of technical universities – future IT-specialists and directed to breaking the gap between theoretic computer science and it's applied applications: software, system and computing engineering. CoCalc was justified as a learning tool of mathematical informatics in general and neural network modeling in particular. The elements of technique of using CoCalc at studying topic "Neural network and pattern recognition" of the special course "Foundations of Mathematic Informatics" are shown. The program code was presented in a CofeeScript language, which implements the basic components of artificial neural network: neurons, synaptic connections, functions of activations (tangential, sigmoid, stepped) and their derivatives, methods of calculating the network`s weights, etc. The features of the Kolmogorov's theorem application were discussed for determination the architecture of multilayer neural networks. The implementation of the disjunctive logical element and approximation of an arbitrary function using a three-layer neural network were given as an examples. According to the simulation results, a conclusion was made as for the limits of the use of constructed networks, in which they retain their adequacy. The framework topics of individual research of the artificial neural networks is proposed.

***Keywords**: CoCalc, cloud technologies, neural network simulation, foundations of mathematical informatics.*

1 INTRODUCTION

One of the necessary condition of fundamentalizing of computing education in higher educational and technical educational institutions is reorientation of basic information training from study rapid-changing technologies to a stable scientific basis of informatics. The leading role is played by computer modeling and numerical experiment [8], which simultaneously can be both methodological basis of informatics and learning methods of computing disciplines.

In the work [10] it is shown that effective way of fundamentalizing of informatic training students of pedagogical institutions is a *Mathematical Informatics* – direction of scientific researches, which, on the one hand is a component of theoretical computer science, where mathematical models and tools used to modeling and studying information processes in different spheres of human activity, and, on the other hand, deals with the use of information systems and technologies for solving applied tasks. As an academic discipline *Mathematical Informatics* aims at mastering the basic models, methods and algorithms for solving problems arising in the field of

intellectualization of information systems and considers the problem of the use of information, in particular mathematical models and information technologies for their research.

We have developed a special course "Foundations of Mathematical Informatics" which is intended for students of technical universities – future specialists in information technologies [9]. The content of the course is a combination of two interrelated components: theoretical and practical. The theoretical component is aimed to develop the students' ideas about data structures and algorithms that are the foundation of modern methodology of software development; methods for solving engineering and scientific tasks using numerical methods; the basic principles of coding and modulation of signals during the data transmission, signal processing, increase of noise immunity during data transfer via communication channels; basic methods of signal acquisition, decoding and detection errors by using various error-correcting codes; algorithmic aspects of number theory and their applications in modern cryptography. The practical aspect associated with the acquisition of skills to analyze, evaluate and select existing algorithms; to use methods and techniques of developing and evaluating the algorithms, develop new algorithms related to the design of hardware and software components of computer systems and networks; use existing and develop new mathematical methods for solving problems related to the design and using of computer systems and networks; to choose methods of computation that are resilient to errors; to solve linear and non-linear algebraic equations and their systems; to apply interpolation and approximation; to make a selection of the method for integration of differential equations; to formulate and solve optimization problems; to apply the Kolmogorov's theorem to approximation arbitrary functions by three-layer neural network; to build the rings for the specified module; to apply the methods of error-correcting coding to data recovery when their injury; to build block ciphers; to build linear Bose-Chaudhuri-Hocquenghem codes; to build generating and testing polynomial for encoding and decoding cyclic codes; to apply Reed-Solomon's codes for data transmission in computer networks; to apply methods and tools to ensure the security of programs and data in the design and operation of computer systems and networks; to consider the requirements of data protection; to create a software and hardware subsystem of cryptographic protection of data; to use the RSA algorithm and digital signatures for data transmission in computer networks; to create and manage by key information for the subsystems of the authentication; to use cloud technology for practical implementation of the basic methods of Mathematical Informatics.

There are 4 thematic modules in the content of the course.

In *the first substantive module* "Theory of algorithms" basic concepts and methods are discussed which related to the analysis of algorithms (a machine with random memory access; analysis of the sorting algorithm by the inclusion; comparison of functions), algorithmic strategies (asymptotic analysis of upper and average complexity estimates of the algorithms; compare the best, average and worst estimates; O-, o-, ω0 and θ-notations; empirical measurements of the algorithm' efficiency; the overhead of algorithms by time and memory; recurrence relations and analysis of recursive algorithms; comparison of algorithms; the impact of the data structures and programming features on the algorithm efficiency; methods of algorithm development), algorithms design (value, classification and characteristics of sorting in the implementation of algorithms; simple sorting, their advantages and disadvantages; complex sorting and their advantages and disadvantages; comparison of simple and complex sorting).

In *the second substantive module* "Numerical methods" covers the basics of computer simulation (the concept of models and modeling; properties and classification of models; computer simulation features; statistical modeling features), tasks of linear and nonlinear algebra, approximation technique, methods of solution 1^{st}-order ordinary differential equations; optimization technique (random search method, chord method, Golden section method; Fibonacci method; simplex search), neural networks and the task of pattern recognition (mathematical model of a neuron; the use of Kolmogorov's theorem to approximate arbitrary functions by three-layer neural network).

In *the third substantive module* "Coding theory" the mathematical foundations of coding theory, basic concepts of the error-correcting coding, linear codes, cyclic codes, Bose-Chaudhuri-Hocquenghem codes, Reed-Solomon codes, convolutional codes are discussed.

In *the fourth substantive module* "Basics of cryptography" the basic cryptographic system (symmetric and asymmetric) and their use for the management of cryptographic keys and digital signatures are discussed.

Special course final control of knowledge is a credit by the results of the current and the module control and presentation of individual education and research projects on the artificial neural networks building [2]. They was chosen due to the fact that, firstly, they are based on fundamental mathematical apparatus, and secondly, neural network modeling is one of the modern research directions in the field of mathematical informatics, and thirdly, the results which obtained during simulation can be applied in all substantive modules of the proposed special course.

2 THE AIM AND OBJECTIVES OF THE STUDY

Therefore, the aim of the study is to develop the individual components of the methodic of using cloud technologies as learning tool for neural network simulation in the special course "Foundations of Mathematic Informatics".

To accomplish the set goal, the following tasks had to be solved:

1. justify the choice CoCalc as a learning tools of the foundations of mathematical Informatics for students of technical universities;
2. to develop demonstration models of artificial neural networks using various CoCalc components.

3 LITERATURE REVIEW AND PROBLEM STATEMENT

One of the most powerful cloud technologies tools [1] is CoCalc (formerly known as SageMathCloud [6]) – a cloud based integrated version of the computer mathematics system Sage, hosted on Google's servers. CoCalc is not only the cloud based computer mathematics system, but also the system of support learning the mathematical and CS subjects. The main components of CoCalc are:

1) Sage Worksheets – provides the ability to interactively run commands of Sage or programming (e.g., object-oriented and imperative) languages, such as C++ and HTML;
2) IPython notebooks (since 2016 – Jupyter Notebook) – timed session in Python programming language, the part of SciPy, scientific and engineering computing library. CoCalc provides the ability to multiple users to communicate through IPython note-books in synchronous and asynchronous modes;
3) the workflow system in LaTeX with full support for sagetex, bibtex, etc.;
4) backup system – full save of all edited project files of the user every 2 minutes;
5) the replication system implies the preservation of each project in three physically separated data centers [7].

CoCalc provides opportunities of:

– interactive study of mathematics, natural and computer science;
– real time users collaboration;
– training: adding students, creating projects, monitoring of student's development, etc. using a cloud based educational materials;
– creating and editing of educational and academic texts using LaTeX, Markdown or HTML;
– adding your own files, data processing, presentation of results etc.

The presence of the 'Besides Sage Worksheets' tool in the composition of Jupyter Notebooks provides to the users of full access to classical Linux terminal [4].

The main CoCalc unit is a project. The user can create any number of independent projects of personal workspaces where the user stores resources of different types. The user can also invite others to collaborate in a joint project to provide open access to files or folders.

Each project is executed on the server CoCalc where it divides disk space, CPU and RAM with other projects. Free service plan provides using only those server resources that are free currently. In addition, when the user's project on the free service plan is not used for a few weeks, it is moved to secondary storage in order to free server resources and his restarting will take significantly much more time than the user who paid for the service plan.

Project participants can combine their own computing and storage resources to improve the capabilities of the project as a whole and the reallocation of resources among themselves. To organize joint work with the resources of the CoCalc project is possible either at the level of individual resource, in particular of the worksheet, or project as a whole.

Opening of the share access at the level of individual resource is a web publication of the resource content in a read-only mode for all Internet users, which have link to this resource. The disadvantage of such publication is that the read-only user has no way to control the worksheet calculations, even if the author used the standard controls in it. However, if it necessary, the published worksheet can be copied or downloaded.

Organization of joint work at the level of the whole project is possible without/with the 'course' resource type. The first method involves connecting the participants to the project participants, who will have the ability to work together on existing educational resources of the project, or add new ones, invite other participants to communicate via text and video chats within the joint project. The contribution of each participant of the joint project in the solution of its tasks may be revised in the pages of history of the project or in the pages of his backups [3].

As a cloud subject-oriented environment, CoCalc in its composition contains both a computer mathematics systems and programming environments. The choice of a particular tool is carried out through binding to the file type or through the command of programming environment selection. At the stage of creating new files at project home directory, the user can choose the programming language. According to the choice, the environment is booting with internal compiler (interpreter).

The easiest way of handling CoCalc files is a Linux terminal mode. So, it is necessary to compile and run the developed program to test it. Files created as a result of program execution, become part of the student project in the CoCalc. Another method of executing programs in the CoCalc is directly on Sage worksheets. To do this, in the beginning of the cell, it necessary to specify one of the so-called "magic commands" (%magic below provides a full list of them). For example, %coffeescript executes the CoffeeScript code; the CoCalc is additionally define the printing function print. CoffeeScript code translates to JavaScript and runs directly in the browser, so the CoffeeScript program performance does not depend on the computing power of cloud servers.

4 METHODIC OF USING COCALC AS A LEARNING TOOL FOR NEURAL NETWORK SIMULATION

In the special course of the foundations of mathematical informatics using CoffeeScript can be considered such calculating-intensive tasks as creating and customizing of a neural network. Given the significant time required for this and the importance of the topic "Neural networks and pattern recognition" for the special course in general (such as topic which brings together computing and intelligent content lines), students are offered individual research task – development of an artificial neural network [5].

Artificial neural network is a mathematical model and also its software and hardware implementation, based on the principles of functioning of biological neural networks – networks of nervous cells of a living organism. This concept appeared in the study of processes that occur in the brain, and when we try to simulate these processes. After the development of the learning algorithms the models were used for practical purposes: in problems of prediction, pattern recognition, control problems, etc.

Artificial neural network is a system of interacted artificial neurons, interconnected through synapses. The input of the artificial neuron receives a set of signals, each of which is an output of another neuron. Each input is multiplied by weight coefficient of the synapse, all the components are summed, determining the activation level of a neuron as a scalar product of a vector input on the

weight vector. The resulting value is measured by activation function, which normalizes the value in a given range: for polar activation function is [0; 1], bipolar [–1; +1].

Three-layer neural network is most commonly used; it architecture is presented in Fig. 1.

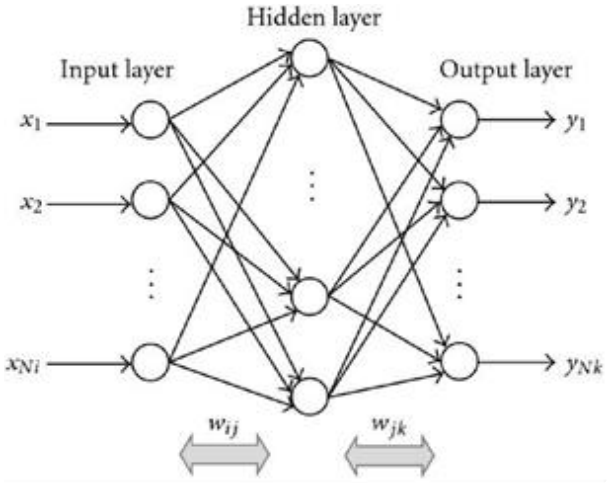

Fig. 1. The architecture of three-layer neural network

To develop a neural network, we offer students the following code in CoffeeScript:

```
    %coffeescript

# Artificial neural network (based on Phillip Wang`s code
# https://github.com/lucidrains/coffee-neural-network)

class Synapse # synapse -connects two neurons
   constructor: (@source_neuron, @dest_neuron)->
      #initial weight is a random value within [-1;+1]
      @weight = @prev_weight = Math.random() * 2 - 1

class TanhGate # tangential activation function
   calculate: (activation)->
      math.tanh(activation)
   derivative: (output)-> # it's derive
      1 - output * output
class SigmoidGate # sigmoidal activation function
   calculate: (activation)->
      1.0 / (1.0 + Math.exp(-activation))
   derivative: (output)-> # it's derive
      output * (1 - output)

class ReluGate # Heaviside step activation function
   @LEAKY_CONSTANT = 0.01
   calculate: (activation)->
      if activation < 0 then activation *
         ReluGate.LEAKY_CONSTANT else activation
   derivative: (output)-> # it's derive
      if output > 0 then 1 else ReluGate.LEAKY_CONSTANT
```

The first of activation function is bipolar and corresponds to the hyperbolic tangent, the second is polar and corresponds to the logistic function. The latest activation function describes a polar stepped function.

```
class Neuron # artificial neuron
  # constants of learning rate and momentum
  @LEARNING_RATE = 0.1
  @MOMENTUM = 0.05

  constructor: (opts={})->
     gate_class = opts.gate_class || SigmoidGate
     @prev_threshold = @threshold = Math.random() * 2 - 1
     @synapses_in = []
     @synapses_out = []
     @dropped = false
     @output = 0.0
     @error = 0.0
     @gate = new gate_class()
  dropout: ->
     @dropped = true
     @output = 0
  calculate_output: -> # calculates the neuron response
     @dropped = false
     activation = 0
     for s in @synapses_in
        activation += s.weight * s.source_neuron.output
     activation -= @threshold
     @output = @gate.calculate(activation)#
derivative: ->
     @gate.derivative @output
#calculation of weight coefficients of the output layer
output_train: (rate, target)->
     @error = (target - @output) * @derivative()
     @update_weights(rate)
#calculation of weight coefficients of the hidden layer
hidden_train: (rate)->
     @error = 0.0
     for synapse in @synapses_out
          @error +=
             synapse.prev_weight * synapse.dest_neuron.error
     @error *= @derivative()
     @update_weights(rate)
update_weights: (rate)->
     for synapse in @synapses_in
          temp_weight = synapse.weight
          synapse.weight += (rate * Neuron.LEARNING_RATE *
               @error * synapse.source_neuron.output) +
               (Neuron.MOMENTUM * ( synapse.weight -
               synapse.prev_weight))
          synapse.prev_weight = temp_weight
     temp_threshold = @threshold
     @threshold += (rate * Neuron.LEARNING_RATE * @error *
          -1) + (Neuron.MOMENTUM * (@threshold -
          @prev_threshold))
    @prev_threshold = temp_threshold
```

The network learning goal is to find the coefficients of neurons interconnections. During the learning process the neural network is able to identify complex dependencies between the input and

output data, and perform generalization. It means that in case of successful learning the network will be able to return the correct result based on the data, which are absent in the training input, as well as incomplete and/or noisy, partly distorted data.

```
class NeuralNetwork # neural network
    @DROPOUT = 0.3

    # the constructor arguments is the type of the
    # activation function, the number of neurons on the
    # input layer, the number(s) of neurons on the hidden
    #  layer(s),  the  number  of  neurons  on  the  output  layer
    constructor: (gate_class, input, hiddens..., output)->
      opts = {gate_class}
      @input_layer = (new Neuron(opts)
          for i in [0...input])
      @hidden_layers = for hidden in hiddens (
          new Neuron(opts) for i in [0...hidden])
      @output_layer = (new Neuron(opts)
          for i in [0...output])
      for i in @input_layer
          for h in @hidden_layers[0]
               synapse = new Synapse(i, h)
               i.synapses_out.push synapse
               h.synapses_in.push synapse
          for layer, ind in @hidden_layers
               next_layer = if ind==(@hidden_layers.length-1)
                 @output_layer
               else
                 @hidden_layers[ind+1]
               for h in layer
                 for o in next_layer
                    synapse = new Synapse(h, o)
                    h.synapses_out.push synapse
                    o.synapses_in.push synapse

    train: (input, output)-> # neural network training
      @feed_forward(input)
      for neuron, ind in @output_layer
        neuron.output_train 0.5, output[ind]
      for layer in @hidden_layers by -1
        for neuron in layer
          neuron.hidden_train 0.5

# feed the input signal through all network layers feed_forward:
(input)->
    for n, ind in @input_layer
      n.output = input[ind]
    for layer in @hidden_layers
      for n in layer
        if Math.random() < NeuralNetwork.DROPOUT
           n.dropout()
        else
           n.calculate_output()
      for n in @output_layer
        n.calculate_output()
```

```
# calculation result is on the output layer
current_outputs: ->
    (n.output for n in @output_layer)
# cloning data
clone = (obj) ->
    return obj if obj is null or typeof (obj) isnt "object"
    temp = new obj.constructor()
    for key of obj
        temp[key] = clone(obj[key])
    temp
```

As an example, firstly we propose to examine a neural network for Boolean functions of two variables “OR”. A feature of this example is that the input network served only two polar values 0 and 1, the output is also one of the two values.

Due to the Kolmogorov’s theorem, in order for three-layer neural network reproduced any function of multiple variables, the dimension of the hidden layer should be at least more than 1 for twice dimension of the input. For this example, it is possible to reduce the dimension of the hidden layer from 5 to 2 due to the fact that there are only two possible values:

```
#    Creating a three-layer neural network:
#    2 input neurons, 2 hidden and 1 output
nn = new NeuralNetwork(SigmoidGate, 2, 2, 1)
#    training sequence consists of pairs "input - output"
pairs = [ #for example, for a disjunction
    [[0,0], [0]],
    [[0,1], [1]],
    [[1,0], [1]],
    [[1,1], [1]]
    ]

#    Training limited by the number of iterations
numiter = 150000
for i in [0...numiter]
    err=0
    for pair in pairs
       nn.train pair[0], pair[1]
       nn.feed_forward pair[0]
       out=nn.current_outputs()
       for k in [0...pair[1].length]
          err+=(out[k]-pair[1][k])*(out[k]-pair[1][k])
    err=Math.sqrt(err/4)
    if i%1000==0
       print "Epoch ", i, ", error = ", err

#    Network testing #1
for i in pairs
    nn.feed_forward i[0]
    print "Input ", i[0], ", calculated ",
       nn.current_outputs(), ", must be ", i[1]
```

The results of neural network testing is shown in Fig. 2.

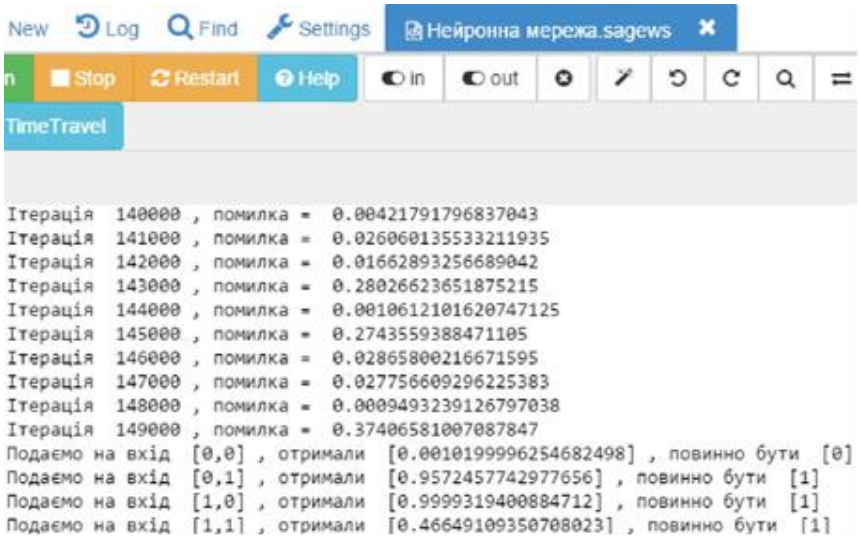

Fig. 2. Testing of neural networks for logic functions "OR" (in Ukrainian)

From Fig. 2 we can see that the number of iterations (150000) for network learning is too big, so that the computing process could be stopped while reducing the error to predetermined value beforehand. It is necessary to pay attention to the results of a calculation when the input network is supplied a pair (1; 1) – unlike the previous three tests, the obtained value is significantly differed from the needful. To resolve this error we offer to choose a different polar function of the activation – stepped.

The following example shows how to build neural network for random values of input and output:

```
#    Creating a three-layer neural network:
#    3 input neurons, 7 hidden and 1 output
nn = new NeuralNetwork(SigmoidGate, 3, 7, 1)

#    generate a new training sequence
data = []
count = 1000 #the number of pairs "input - output"

for i in [0...count]
    x1 = Math.random()*100-60
    x2 = Math.random()*100-40
    x3 = Math.random()*100-50
    data.push([[x1,x2,x3], [x1 + x2 - x3]])
```

The network architecture corresponds to the Kolmogorov`s theorem, but the value at which it is proposed for training are not polar, as required by the logistic function. Bringing them to the desired range requires normalization, which is necessary to find limit values for input and output:

```
#    find limit values for input and output mininput =
data[0][0][0]
    maxinput = data[0][0][0] minoutput = data[0][1][0] maxoutput =
data[0][1][0]
    for i in [0...count]
    for j in [0...data[i][0].length]
    if data[i][0][j]<mininput
```

```
    mininput=data[i][0][j]
    if data[i][0][j]>maxinput
    maxinput=data[i][0][j]
    for j in [0...data[i][1].length]
    if data[i][1][j]<minoutput
    minoutput=data[i][1][j]
    if data[i][1][j]>maxoutput
    maxoutput=data[i][1][j]
```

Network training is performed on the normalized data (Fig. 3):

```
#    training sequence, normalized in the range [0; 1]
normdata=clone(data)
    for i in [0...count]
       for j in [0...data[i][0].length]
          normdata[i][0][j]=
               (data[i][0][j]-mininput)/(maxinput-mininput)
       for j in [0...data[i][1].length]
          normdata[i][1][j]=
               (data[i][1][j]-minoutput)/(maxoutput-minoutput)
#    Training to complete iteration limit
numiter = 100000

for i in [0...numiter]
    err=0
    for pair in normdata
       nn.train pair[0], pair[1]
       nn.feed_forward pair[0]
       out=nn.current_outputs()
       for k in [0...pair[1].length]
          err+=(out[k]-pair[1][k])*(out[k]-pair[1][k])
    err=Math.sqrt(err/4)
    if i%1000==0
       print "Epoch ", i, ", error = ", err
```

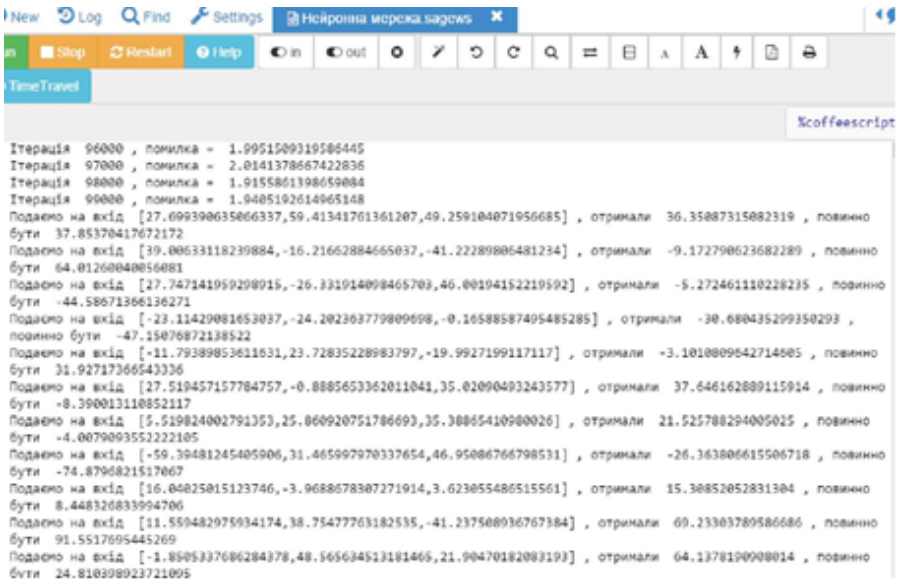

Fig. 3. The results of testing the neural network for adding function (in Ukrainian)

While testing a network, perform the reverse process of denormalization:

```
#    Network testing #2
for i in [0...count]
    nn.feed_forward normdata[i][0]
    res=nn.current_outputs()
    print "Input ", data[i][0], ", calculated ",
      res[0]*(maxoutput-minoutput)+minoutput,
      ", must be ", data[i][1][0]
```

In Fig. 3 shows the test results.

While discussing the test results, it is advisable again to pay attention to the values that differ significantly from the etalons. We first recall that the input values generated randomly in the following ranges: $x_1 \in [-60; 40)$, $x_2 \in [-40; 60)$, $x_3 \in [-50; 50)$. Analysis of the results shows that, then closer the input values to the range limits, then greater the difference of the result from the etalons. This provides an opportunity to do conclusion on the boundaries of application of the constructed network in which it maintains adequacy.

5 CONCLUSIONS

1. The special course "Foundations of Mathematic Informatics" for students of technical universities – future IT-experts aimed and directed to breaking the gap between theoretic computer science and it's applied applications: software, system and computing engineering. In this regard, their fundamental foundations are implemented using modern programming languages and cloud technologies tools.
2. One of the leading cloud technology learning tools of the special course is CoCalc – the mathematical software system that provides the ability to support all sections of the special course in an unified mobile mathematical environment. Despite the fact that Python is the most often used programming language in CoCalc, a program realization examples represented by the extension of browser-based JavaScript language, which provides to developed software a higher level of mobility.
3. The central theme of the special course is the neural network simulation – a traditional technique for modeling natural neural networks, which had a significant impact on all stages of the development of computer and software engineering. The most versatile neural networks architectures and their application to the problems of modeling the basic logical elements of the computer system and identifying hidden dependencies are discussed in paper.
4. The final evaluation for the special course includes the presentation of individual education and research projects on the artificial neural networks building. The framework project's topics involves modeling continuous, discrete-continuous and discrete neural networks for solving problems of circuit synthesis, time series forecasting, pattern recognition, functions approximation, dependency identification, medical diagnostics, decision-making under conditions of incomplete data, data compression, unknown data restoration, clustering, automated control etc.

REFERENCES

1. Markova, O.M., Semerikov, S.O. & Striuk, A. M. (2015). The cloud technologies of learning: origin. *Information Technologies and Learning Tools*, 46(2), 29-44.
2. Permiakova, O.S. & Semerikov, S.O. (2008). *The use of neural networks in forecasting problems.* Materials of the International Scientific and Practical Conference "Young scientist of the XXI century", KTU, Kryviy Rih, 17-18 November 2008.
3. Popel, M.V. (2015). *Organization of learning mathematical disciplines in SageMathCloud*. Publishing Department of the Kryviy Rih National University, Kryviy Rih.
4. SageMath, Inc. (2018). *CoCalc - Collaborative Calculation in the Cloud*. Retrieved from https://cocalc.com.

5. Semerikov, S., Teplytskyi, I. & Yechkalo, Yu. (2018). Computer Simulation of Neural Networks Using Spreadsheets: The Dawn of the Age of Camelot.*Proceedings of the 14th Interna-tional Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Kyiv, 14-17 May 2018*. CEUR Workshop Pro-ceedings.
6. Shokaliuk, S.V., Markova, O.M., Semerikov, S.O. & Soloviov, V.M. (Ed.) (2017). SageMathCloud as the Learning Tool Cloud Technologies of the Computer-Based Studying Mathematics and Informatics Disciplines. *Modeling in Education: State. Problems. Prospects*, 130-142.
7. Stein, W. (2014). *What can SageMathCloud (SMC) do? Sage: open source mathematics software.* Retrieved from http://sagemath.blogspot.com/2014/05/what-can-sagemathcloud-smc-do.html.
8. Teplytskyi, I.O. & Semerikov, S.O. (2007). Computer simulation of absolute and relative motions of the planets the Solar system. *Collection of scientific works of the Kamyanets-Podilsky National University named after Ivan Ogienko. Series: Pedagogical*, 13, 211-214.
9. Turavinina, O.M. & Semerikov, S.O. (2012). *Contents of the learning of the foundations of mathematical informatics of students of technical universities.* Proceedings of the International scientific and methodical conference on Development of intellectual abilities and creative abilities of students and students in the process of teaching disciplines of the natural sciences and mathematics cycle, Sumy State Pedagogical University named after A. S. Makarenko, Sumy, 6-7 December 2012.
10. Turavinina, O.M. (2012). Mathematical informatics in the system fundamen-talization learning the students of technical universities. *Collection of scientific works of the Kamyanets-Podilsky National University named after Ivan Ogienko. Series: Pedagogical*, 18, 189-191.

Стаття надійшла до редакції 20.09.2018.
The article was received 20 September 2018.

Маркова О. М.[1], Семериков С. О.[2], Попель М. В.[3]
[1]Державний вищий навчальний заклад «Криворізький національний університет», Кривий Ріг, Україна
[2] Криворізький державний педагогічний університет, Кривий Ріг, Україна
[3]Інститут інформаційних технологій і засобів навчання НАПН України, Київ, Україна

COCALC ЯК ІНСТРУМЕНТ ПІДГОТОВКИ ДЛЯ МОДЕЛЮВАННЯ НЕЙРОННИХ МЕРЕЖ У СПЕЦІАЛЬНОМУ КУРСІ "ОСНОВИ МАТЕМАТИЧНОЇ ІНФОРМАТИКИ"

У статті розглянута роль моделювання нейронної мережі в навчальному процесі спеціального курсу "Основи математичної інформатики". Курс був розроблений для студентів технічних університетів - майбутніх спеціалістів з інформаційних технологій та спрямований на подолання розриву між теоретичною інформатикою та її прикладними програмами: програмною, системною та комп’ютерною інженерією. CoCalc розглядається як навчальний інструмент математичної інформатики в цілому та, зокрема, для моделювання нейронних мереж. Показані елементи методики використання CoCalc при вивченні теми "Нейронні мережі та розпізнавання образів" спеціального курсу "Основи математичної інформатики". Код програми був представлений на мові CofeeScript, в якій реалізуються основні компоненти штучної нейронної мережі: нейрони, синаптичні з'єднання, функції активації (тангенціальні, сигмоїдні, ступінчасті) та їх похідні, методи розрахунку ваги мережі та ін. Обговорювалися особливості застосування теореми Колмогорова для визначення архітектури багатошарових нейронних мереж. В якості прикладів було наведено реалізацію диз'юнктивного логічного елемента та наближення довільної функції за допомогою тришарової нейронної мережі. Згідно результатів моделювання, було зроблено висновок щодо меж використання побудованих мереж, в яких вони зберігають свою адекватність. Запропоновано основні теми окремих досліджень штучних нейронних мереж.

Ключові слова: CoCalc, хмарні технології, моделювання нейронних мереж, основи математичної інформатики.

Маркова О. М.[1], Семериков С. А. [2], Попель М. В. [3]

[1]Державний вищий навчальний заклад «Криворізький національний університет», Кривий Ріг, Україна

[2] Криворізький державний педагогічний університет, Кривий Ріг, Україна

[3]Інститут інформаційних технологій і засобів навчання НАПН України, Київ, Україна

COCALC КАК ИНСТРУМЕНТ ПОДГОТОВКИ ДЛЯ МОДЕЛИРОВАНИЯ НЕЙРОННЫХ СЕТЕЙ В СПЕЦИАЛЬНОМ КУРСЕ «ОСНОВЫ МАТЕМАТИЧЕСКОЙ ИНФОРМАТИКИ»

В статье рассмотрена роль моделирования нейронной сети в учебном процессе специального курса "Основы математической информатики". Курс был разработан для студентов технических университетов - будущих специалистов по информационным технологиям и направлен на преодоление разрыва между теоретической информатикой и ее приложениями: программной, системной и компьютерной инженерией. CoCalc рассматривается как учебный инструмент математической информатики в целом и, в частности, для моделирования нейронных сетей. Показаны элементы методики использования CoCalc при изучении темы "Нейронные сети и распознавания образов" специального курса "Основы математической информатики". Код программы был представлен на языке CofeeScript, в которой реализуются основные компоненты искусственной нейронной сети: нейроны, синаптические соединения, функции активации (тангенциальные, сигмоидни, ступенчатые) и их производные, методы расчета веса сети и др. Обсуждались особенности применения теоремы Колмогорова для определения архитектуры многослойных нейронных сетей. В качестве примеров были приведены реализацию дизъюнктивного логического элемента и приближения произвольной функции с помощью трехслойной нейронной сети. Согласно результатам моделирования, был сделан вывод о границах использования построенных сетей, в которых они сохраняют свою адекватность. Предложены основные темы отдельных исследований искусственных нейронных сетей.

Ключевые слова: CoCalc, облачные технологии, моделирование нейронных сетей, основы математической информатики.