ALFA BK UNIVERSITY
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY O F COMPUTER SCIENCE
ALFATECH
Proceedings of Conference

Alfa BK Univerzitet

UDK: 004.8
004.415
COBISS.SR-ID 148916745
DOI: 10.5281/zenodo.12594372
Original scientific paper

A MODIFIED VERSION OF GRADIENT DESCENT ALGORITHM AS A SOLUTION OF LOCAL MINIMUM PROBLEM IN ARTIFICIAL NEURAL NETWORK

Goran Kekovi[24], *Rade Božovi*[25], *Negovan Stamenkovi*[26]

Abstract

In this paper, software based on modified gradient descent and backpropagation algorithms for overcoming the local minimum problem in artificial neural networks is proposed. During the training of the artificial neural network, at the end of each epoch, the existence of the global minimum was checked over successive values of the loss function and by determining the percentage of successfully classified samples from the training and test sets. The software is written in the C# programming language in an object-oriented manner. It is written in a modular way in the sense that it has its own mathematical library and can be upgraded with other algorithms of artificial neural networks.

Keywords: *Artificial neural networks, Neuron weights, Global minimum, Gradient descent, Backpropagation algorithm, C# programming language*

Introduction

Artificial intelligence is becoming a part of our everyday life and it has applications in almost all areas of human activity. In this sense, artificial neural networks (ANN) are particularly interesting and numerous algorithms have been developed for them. However, the main breakthrough in this area was the backpropagation algorithm based on the gradient descent method which originate from theory of optimization [1-4]. Its main disadvantages are getting stuck in a local minimum and slow convergence. Various solutions have been proposed for these problems, such as, adaptive learning with momentum and all their variants [5,6]. In these methods, the change of neural weights from the previous epoch multiplied by the moment is added to the neural weights in the next epoch, which achieves regularization and

[24] Goran Kekovi, 1967, PhD, Assistant Professor, Faculty of Information Technology, Alfa BK University, 11070 Belgrade, Serbia, goran.kekovic@alfa.edu.rs https://orcid.org/0000-0003-1429-0582
[25] Rade Božovi, 1983, PhD, Assistant Professor, Faculty of Information Technology, Alfa BK University, 11070 Belgrade, Serbia, rade.bozovic@alfa.edu.rs https://orcid.org/0000-0002-8580-714X
[26] Negovan Stamenkovi, 1979, PhD, Professor, Faculty of Sciences and Mathematics, University of Priština, 38220 Kosovska Mitrovica, Serbia, negovan.stamenkovic@pr.ac.rs https://orcid.org/0000-0003-4025-5342

ISBN: 978-86-6461-074-2

increases the speed of convergence. Variants with variable moment speed up convergence, but at the cost of introducing new parameters, whose values are a matter of numerical experimentation.

The problem of the local minimum has not yet been adequately solved, although some solutions have been proposed here as well. One of them is the early stopping technique [7, 8]. In this technique, the loss function is monitored and if there are no changes after a predetermined number of epochs, it is considered that its global minimum has been reached. The disadvantage is that the number of epochs needed cannot be known in advance and we may still be stuck in a local minimum. Variants of this method include tracking losses on the validation data set, after each or a certain number of epochs. The price of this is an increase in CPU time, but with an increased probability of hitting the global minimum.

All this facts lead to the proposition of this paper. Obviously, there is a trade-off between CPU consuming time and hitting the global minimum in the early stopping method. Namely, the problem can be seen from another perspective by asking two questions. Can a global minimum be reached with high probability and what is the main difference between global and local minimum? According to the the Universal Approximation Theorem any input function can be approximated by an ANN of appropriate configuration. The answer to the first question, could be sought here. This practically means that there is a possibility of the gradient descent passing through the minimum at one point in time. The reason is obvious: after each input sample, neural weights and biases of ANN are corrected. The idea of this paper is to define the mechanism of identification of the global minimum in two steps. In the first step, the minimum is identified by monitoring the loss function and if the threshold is reached (the difference between two consecutive values) procced to step two. Furthermore, the second step is to check if the number of successfully classified samples in both sets (training and validation) is higher than the predefined value. In that case, it can be concluded that it is a global minimum. Otherwise, it is a local minimum and the program can continue with iterations.

Mathematical Background

At the core of the descending gradient algorithm is the correction of the neural weights according to the formula:

$$W_{new}^{(k)} = W_{old}^{(k)} - \alpha \frac{\partial E}{\partial W_{old}^{(k)}} \tag{1}$$

where are, $W^{(k)}_{new}$, $W^{(k)}_{old}$ neural weight matrices of the *k*-th neural layer before and after one epoch, *E* is loss function and α is learning constant. The size of the neural weight matrices *W* is *n x m*, where *n,m* represent the numbers of neurons in the *k*-th and *(k-1)*-th layers, respectively. If the previous, *k-1* layer is input, then *m* respresents number of components of the input vector *X*. Similarly, if *k-th* layer is the output layer, then *n* is equal to the number of components of the output or target vector *Y*. In the first stage or forward propagation the matrix of output vectors or activations *A* is calculated, as:

ISBN: 978-86-6461-074-2

$$A^{(k)} = W^{(k)}O^{(k-1)} + b^{(k)} \tag{2}$$

where $b^{(k)}$ represents the biases matrix of the *k*-th layer. The matrix of variable $O^{(k)}$ is determined according to:

$$O^{(k)} = f(A^{(k)}) \tag{3}$$

In the above expression, *f* is the so-called activation function and there are several different types. The famous *logsig* function is used in this software:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

In the second phase, the loss function gradient matrix is determined, via backpropagation algorithm according to the formula:

$$\frac{\partial E}{\partial W^{(k)}} = \delta^{(k)}(O^{(k-1)})^T \tag{5}$$

where the error matrix $\delta^{(k)}$ of *k-th* layer is calculated by back propagation (from the output layer to the input layer) according to:

$$\delta^{(k)} = \frac{\partial f(O^{(k)})}{\partial A^{(k)}} \odot ((W^{(k+1)})^T \delta^{(k+1)}) \tag{6}$$

In the formula above, the symbol denotes the multiplication between matrices in the manner: $C = A \odot B;\ C_{ij} = A_{ij} \cdot B_{ij}$. The ANN in this software works in inline mode, which means that in each epoch it goes through all the training samples and finally determines the error function, which it then compares with its value from the previous epoch:

$$E_{old} - E_{new} < \varepsilon \tag{7}$$

Software Description

The complete code and all the necessary technical details can be found at [9]. The user of the software defines the value of the constant ε (*double error_limit* in software), which will represents the difference between successive values of the loss function. The value of this constant is usually of the order of magnitude $\sim 10^{-4}, 10^{-5}$.... and is usually $\sim 10^{-4}$ sufficient in most cases. If the mentioned difference is smaller than the value of this constant and if its accuracy of classification $\geq 95\%$, can be assumed that the global minimum has been reached. Otherwise, a small correction is added to the neural weights and biases, so that the network is pushed from the local minimum.

Software architecture

ISBN: 978-86-6461-074-2

The main functionality of the program is separating the local and global minimum. The structure of the software can be best explained by the image shown below. The Fig. 1 shows the typical stages of the backpropagation algorithm. In the first phase - forward, the vector propagates through the network to the output, and after that the phase - backward begins, which is the propagation of the error backward. This leads to the correction of weights and biases for each sample separately, which is characteristic of gradient descent algorithm. After passing through the entire set of input data, loss function is updated and compared with the value from the previous epoch. If the difference is smaller than the set goal, the accuracy in both sets (training and validation) is checked. If the goal is reached, the program ends. Conversely, a small correction is added to the weights and biases, which roughly pushes the network out of the local minimum. The value of this correction is of the order of magnitude $\boldsymbol{\delta \sim 10^{-2}}$. Also, the maximum number of epochs is set to 2000 and can be changed by the user, just like the network configuration.

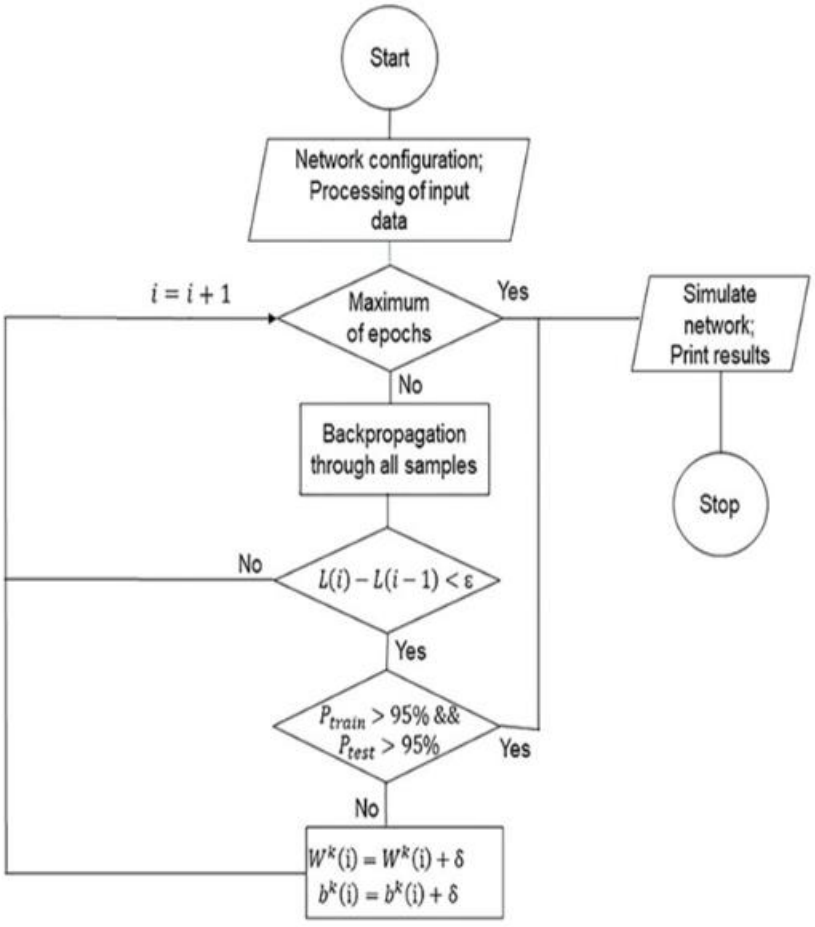

Figure 1: Flowchart of modified gradient descent and backpropagation algorithms, where they are: $L(i)$ – the cost function at *i*–th epoch, P_{train}, P_{test} - classification accuracy on training and test set, $W^k(i), b^k(i)$ - neural weights and biases of the *k*-th layer at *i*–th epoch and $0 \leq \delta \leq 10^{-1}$ is a small random correction.

ISBN: 978-86-6461-074-2

Software functionalities

All the functionalities of this software can be best described by the properties of the most important methods used in the software. For these purposes, we will start from each block.

Data import and processing. Data preprocessing methods:

import_file () - this method imports a file from excel, where types represent samples or input vectors. The columns are properties of the input vectors and it should be noted that the last columns are intended for category values, since the software is used for classification tasks. The corresponding category has a 1, while the others have a 0 (for example, for two categories it will be 1, 0 or 0, 1, etc.) ;

split_test () - as usual, the input data are shuffled and divided into a training set (75% of the input data) and a test set (25% of the input data).

data_process () - input data were converted to [high, low] interval (0,1 or -1,1).

studentize2D () - all columns of the input matrix can be zero-centered, so that their mean value is zero and normalized by dividing it, with their standard deviation. This choice is left to the user, by initializing the variable *int student = 1*. In that case, the method studentize1D (), will automatically apply the same procedure to the input vector for neural network simulation.

Initializing and training network. Initialization and configuration of ANN:

network.init() - this method is accessed by net object which is initialized from the class Network as *net.network.init().* The list of parameters of this method consists of network configuration, number of samples and number of classification categories. The configuration of the neural network is determined by an array, which contains information about the hidden layers. For example, int[] config = {3, 5}, means that there are two hidden layers, size of 3 and 5 neurons each. The number of neurons in the input layer is equal to the number of columns of the data mattrix and the number of neurons in the output layer is equal to the number of categories for classification ;

extend_backprop() – method for network training. It is accessed as in the previous case with the net object: net.extend_backprop(). The list of input parameters consists of: network configuration, training and test data matrices, number of epochs, learning constant, default value for the cost function error, and data mapping interval bounds.

Network simulation. After training, the ANN can be simulated with unseen input data:

simulate() - network simulation for a specific vector from the input takes place via the net object, *net.simulate().* The list of input parameters is: neural weights and biases as part of the output of the method for training, the input vector that can be centered, the network configuration, the number of categories for classification, the interval bounds for mapping the input vector, as well as interval boundaries for input vector mapping and maximum and minimum value of input vector interval. As an output of this method, a vector appears where the unit is in the place of the corresponding category, while there are zeros in the other places.

Network statistics. The statistics of ANN can be determined by the following method:

stats() - finally, network statistics is performed using this method as, *net.stats ()*, with the output being the percentages of successfully classified samples of the training and test data sets.

The software is tested on three different databases. The first database is the well-known iris data set [10], while the others are the cancer set [11] and the data set related to types of wine

ISBN: 978-86-6461-074-2

[12]. In the iris data set, there were three categories for classification and two each for cancer and wine types, respectively. The simulations were performed on a computer with a processor i3 - 8100 3.5 GHz and a RAM of 8 GB. The results are summarized in the table below.

Table 1: Software results on various data sets

Data sets	Number of samples	Classification accuracy (%) Training set	Classification accuracy (%) Test set	Overall classification accuracy(%)	Epochs to reach the goal	Program running time (s)
Iris	150	98,21	97,37	97,58	89	4,6
Cancer	699	95,42	95,43	95,43	1	10,5
Wine	3199	97	95,75	96,06	1169	77,38

Analyzing the results in the table above it can be observed that the execution time of the code is proportional to the number of samples, which is understandable, considering that the network works in inline mode. It can be seen that in the third case, more than 1000 epochs are needed, which means that numerical experiments are needed, with the number of neurons in the hidden layer, the number of layers, etc. It can also be noted that despite the modification, the software is not free from the lack of slow convergence. This can be seen from the last column of Table 1, where it can be seen that the execution time of the program increases rapidly with the increase in the number of samples. This is the result of ANN operation in inline mode. On the other hand, the advantage of this mode is accuracy, although there are algorithms such as Levenberg-Marquardt that work in batch mode and achieve a high degree of accuracy [13, 14].

Impact

The contribution of the software to scientific research and the improvement of deep learning software in general, can be summarized as follows:

- In this software, a way to separate the local and global minimum in the backpropagation algorithm is proposed. This does not eliminate all the weaknesses of the backpropagation algorithm, but it can be a good basis for further research.
- The software is designed in an object-oriented manner, to speed up its execution and to make it easier to use.
- All mathematical procedures of algorithms and details are moved to a separate class *Network*. It also contains its own mathematical library that can be expanded and later used to upgrade the algorithm with new deep learning methods.
- Advanced features of the C# programming language, such as Array lists and jagged arrays are used in this software.

Ilustrative example

In order to illustrate the functioning of the software, this paragraph shows the main program for classifying data in the iris database. The first step is to introduce the required namespaces via the *NuGet* package manager for .NET as shown in Fig. 2. Further, Figure 3 shows the

ISBN: 978-86-6461-074-2

main program with an instantiated net object that accesses the member methods and which are described in the *Software architecture* section.

```
using System;
using System.Data;
using System.Collections;
using System.Collections.Generic;
using Microsoft.Office.Interop.Excel;
using System.Runtime.InteropServices;
using Excel = Microsoft.Office.Interop.Excel;
using System.Diagnostics;
using System.Threading;
```

Figure 2: Necessary namespaces for program operation

ISBN: 978-86-6461-074-2

```
namespace Artifical_inteligence
{
    class Program
    {
        static void Main(string[] args)
        {
            /******* NETWORK PARAMETERS *********/
            int Epochs = 2000; double alpha = 0.1;
            double error_limit = 0.0001; int high = 1, low = 0;
            double training_ratio = 0.75; int[] config = {10};
            /*******************************/
            /*******THE ARRAY LISTS **********/
            ArrayList st = new ArrayList();
            ArrayList net_input = new ArrayList();
            ArrayList net_enter = new ArrayList();
            ArrayList results = new ArrayList();
            /*******************************/
            /*********NET OBJECT************/
            Network net = new Network();
            /*******************************/
            /*****DATA IMPORT AND PROCESSING ******/
            string path = @"C:\Input_data\iris_set.xlsx";
            double[][] allData = net.import_file(path);
            int num_class = 3;
            int xlength = allData[0].Length - num_class;
            int student = 0;
            st = net.split_test(allData, num_class, training_ratio, student);
            net_input = net.data_proces(st, xlength, num_class,high, low);
            /*************************************/
            /*****INITIALIZATION AND TRAINING NETWORK******/
            net_enter = net.network_init(config, xlength, num_class);
            results = net.extend_backprop(config, net_input, net_enter,
                Epochs, alpha, error_limit, high, low);
            /*****************************************/
            /********NETWORK PERFORMANCE ****/
            net.stats(st,results,config);
            /*************************************/
            /******SIMULATE NETWORK***********/
            double maxx = (double)net_input[6];
            double minix = (double)net_input[7];
            double[][,] weights = (double[][,])results[0];
            double[][] bias = (double[][])results[2];
            double[] test_input = {5, 2, 3.5, 1};
            if(student==1) {test_input = net.studentize1D(test_input);} //iris_set
            double[] test_resp = net.simulate(weights, bias, test_input,
               config, num_class, high, low, maxx, minix);
            for (int i = 0; i < num_class; i++)
            {
                Console.WriteLine(test_resp[i]);
            }

        }

    }

}
```

Figure 3: Flow of the main program during classification in the iris data set

Conclusions

In this paper, the possibility of overcoming the problem of the local minimum of the hypersurface of the loss function is presented. This refers to the ability to separate the local and global maxima, thereby addressing the fundamental weakness of the backpropagation algorithm. Program in proposed software works in inline mode, so its reasonable execution

ISBN: 978-86-6461-074-2

time applies to smaller databases up to several thousand samples. In future research, it is necessary to examine the possibilities of this program in batch and semi-batch modes. Also, the software can be expanded with new mathematical methods and upgraded with algorithms. which opens up another space for further research.

References

[1] Ruder Sebastian, An overview of gradient descent optimization algorithms, *arXiv:1609.04747v2*, (2017), DOI:10.48550/arXiv.1609.04747
[2] Schun-ichi Amar, Backpropagation and shtohastic gradient descent method, *Neurocomputing*, 5, (1993), pp:185-196, DOI:10.1016/0925-2312(93)90006-O.
[3] Fan Zhou, Guojing C. On the Convergence Properties of a K-step Averaging Stochastic Gradient Descent Algorithm for Nonconvex Optimization, *arXiv:1708.01012v3*, (2018), DOI: 10.48550/arXiv.1708.01012
[4] Bello-Cruz Junier, Bouza Allende Gemayqzel , Steepest Descent-Like Method for Variable Order Vector Optimization Problems,. *Journal of Optimization Theory and Applications*, 162, (2013), pp:371-391, DOI:10.1007/s10957-013-0308-6
[5] Hameeda Alaa Ali, Karlik B, Shukri Salman Mohammad, Back-propagation algorithm with variable adaptive momentum, *Knowledge-Based Systems*, (2016), pp.1-9. DOI:10.1016/j.knosys.2016.10.001
[6] Phansalkar VV, Sastry PS, Analysis of the Back-Propagation Algorithm with Momentum. *IEEE Transactions on Neural Networks and Learning*, 5, (1994), 3, pp:505-6, DOI:10.1109/72.286925
[7] Zhang Thong, Bin Yu, Boosting with early stopping: convergence and consistency, *Annals of Statistics*, 33, (2005), 4, pp:1538-1579, DOI:10214/009053605000000255
[8] Hagiwara Katsuyuki, Regularization learning, early stopping and biased estimator. *Neurocomputing*, 48, (2002), pp:937-955. https://doi.org.10.1016/S0925-2312(01)00681-6
[9] https://gitlab.com/astra22/ANN
[dataset] [10] Murphy PM, Aha DW. Iris data set, UCI Repository of machine learning databases, (1994), http://www.ics.uci.edu/~mlearn/MLRepository.html.
[dataset] [11] Murphy PM, Aha DW. Cancer data set, UCI Repository of machine learning databases, (1994), http://www.ics.uci.edu/~mlearn/MLRepository.html.
[dataset] [12] Cortez Paulo, Cerdeira António, Almeida Fernando, Matos Telmo, Reis José, Modeling wine preferences by data mining from physicochemical properties, *Decision Support Systems*, 47, (2009), pp:547-553, DOI:10.1016/j.dss.2009.05.016
[13] Levenberg Kenneth, A Method for the Solution of Certain Non-Linear Problems in Least Squares, *Quarterly of Applied Mathematics*, 2, (1944), 2, pp:164–168, DOI:10.1090/qam/10666.
[14] Marquardt Donald, An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics*, 11, (1963), 2, pp:431–441. DOI:10.1137/0111030. hdl:10338.dmlcz

ISBN: 978-86-6461-074-2