COMPUTER SCIENCES AND INFORMATION TECHNOLOGIES

UDC 519.6:004.032.26(045)
DOI:10.18372/1990-5548.75.17542

[1]V. V. Mamonov,
[2]Y. O. Shatikhin,
[3]Y. O. Tymoshenko

LINEAR ALGEBRAIC SYSTEMS NEURAL NETWORK SOLUTION. PART 1

[1,2,3]Educational and Research Institute for Applied System Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine
E-mails: [1]mamonov.volodymyr@lll.kpi.ua, [2]shatikhin.yevhen@lll.kpi.ua,
[3]yury.alex.tym@gmail.com ORCID 0000-0001-7812-6437

***Abstract*—*In recent years, neural networks have become increasingly popular due to their versatility in solving complex problems. One area of interest is their application in solving linear algebraic systems, especially those that are ill-conditioned. The solutions of such systems are highly sensitive to small changes in their coefficients, leading to unstable solutions. Therefore, solving these types of systems can be challenging and require specialized techniques. This article explores the use of neural network methodologies for solving linear algebraic systems, focusing on ill-conditioned systems. The primary goal is to develop a model capable of directly solving linear equations and to evaluate its performance on a range of linear equation sets, including ill-conditioned systems. To tackle this problem, neural network implementing iterative algorithm was built. Error function of linear algebraic system is minimized using stochastic gradient descent. This model doesn't require extensive training other than tweaking learning rate for particularly large systems. The analysis shows that the suggested model can handle well-conditioned systems of varying sizes, although for systems with large coefficients some normalization is required. Improvements are necessary for effectively solving ill-conditioned systems, since researched algorithm is shown to be not numerically stable. This research contributes to the understanding and application of neural network techniques for solving linear algebraic systems. It provides a foundation for future advances in this field and opens up new possibilities for solving complex problems. With further research and development, neural network models can become a powerful tool for solving ill-conditioned linear systems and other related problems.*

Index Terms—Linear algebraic systems; condition number; ill-conditioned systems; neural network; gradient descent; TensorFlow.

I. INTRODUCTION

Solving linear algebraic systems is crucial in science and engineering. traditional methods, like Gaussian elimination, may struggle with ill-conditioned systems where small input changes cause significant solution variations. Neural networks offer a promising alternative, capable of learning complex patterns and adapting to diverse problems. This paper explores neural network methods for solving various linear equation systems, focusing on ill-conditioned systems. Our goal is to develop a model that directly solves linear equation systems while minimizing overfitting risks. We propose a versatile model and evaluate its performance on multiple linear equation sets, including ill-conditioned ones. The article discusses existing methods, challenges with ill-conditioned systems, and introduces the proposed neural network model. We present the results, providing insights into the model's performance and areas for further refinement.

II. PROBLEM STATEMENT

The linear equations system can be represented in matrix form, where **A** is an $m \times n$ matrix, x is a column vector with n entries, and b is a column vector with m entries, as:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \quad (1)$$

or matrix equation $Ax = b$. Given matrix A is square and has full rank, the system has a unique solution $x = A^{-1}b$, where A^{-1} represents inverse of matrix **A**.

In real life problems, modelled by linear algebraic system, the matrix **A** or vector b may be known only approximately (due to rounding errors, floating-point accuracy or the sensitivity of the

sensor observing some process), thus introducing some error Δb. Therefore,

$$A(x+\Delta x)=b+\Delta b \tag{2}$$

Solving (2) for Δx gives

$$A(\Delta x)=\Delta b \Rightarrow \Delta x=A^{-1}\Delta b \tag{3}$$

(3) is the error in solution (1).

Furthermore, matrix **A** can be practically non-invertible – its determinant may be close but not equal to 0; such matrices are said to be ill-conditioned. For linear equations, condition number measures the rate of change in solution x according to the change in *b*.

Given the Euclidean norm:

$$\|x\|=\left(\sum_{i=1}^{n}|x_i|^2\right)^{1/2} \tag{4}$$

and

$$K=\max_x \frac{\|Ax\|}{\|x\|}=\|A\|, \tag{5}$$

$$k=\min_x \frac{\|Ax\|}{\|x\|}=\frac{1}{\|A^{-1}\|}, \tag{6}$$

then

$$\mathrm{cond}(A)=\frac{K}{k}=\|A\|*\|A^{-1}\| \tag{7}$$

is called the condition number of the matrix **A** [1].

When the cond(**A**) is not significantly larger than one, the matrix is well-conditioned, which means that its inverse can be computed with good accuracy; when the condition number is very large, then the matrix is said to be ill-conditioned. Practically, such a matrix is almost singular, and the computation of its inverse, or solution of a linear algebraic system is prone to large numerical errors.

Classic example of ill-conditioned matrix is Hilbert matrix [2], its entries defined as

$$H_{ij}=\frac{1}{i+j-1} \tag{8}$$

The following is (3×3) Hilbert matrix:

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}. \tag{9}$$

It's $\mathrm{cond}(H)\approx 526.16$, while (5×5) Hilbert matrix has $\mathrm{cond}(H)\approx 4.8\times 10^5$ as calculated by (7).

The goal of this article is to benchmark neural network architecture with stochastic gradient descent optimizer for approximation of the solution of system on several well-conditioned systems and ill-conditioned systems [3]. We are particularly interested in comparing obtained solution with the exact solution of the system.

III. Problem Solution

A) Neural network architecture

We will use neural network with stochastic gradient descent optimizer to solve linear equations. Traditionally, neural networks are built with input, hidden and output layers consisting of neurons. Then, input data, which is a rather large dataset of training examples with labels, is fed into it. After that, we can predict some value or label for another piece of data. In our case, input dataset would be as big as millions of randomly-sized linear algebraic systems, which we could generate by randomizing matrix A and vector x, the dot product of which would be the vector b. We could then theoretically input any new linear algebraic systems and get its solution. However, since neural network doesn't solve the linear algebraic systems, but rather approximates the weights to best fit training examples, this architecture will not solve system it wasn't trained on.

Instead, we implement naive iterative solution method as described in [4]. We won't be using input, hidden and output layers with neurons; activation function is not needed as well. In the training cycle, we will minimize the error function $f(x_0)=\|Ax_0-b\|$. When $f(x)=0$, x is the exact solution. Matrix **A** and vector b are stored in the model. The solution x is stored in the single layer as weights. The loss function for this architecture is the following:

$$\mathrm{loss}(x)=\sum(Ax-b)^2. \tag{10}$$

The value of the loss function at x_0 is called residual and denoted by $\|r\|$. When this model is trained for *N* epochs, loss function is repeatedly evaluated at x_N and minimized using stochastic gradient descent [5] shown on Fig. 1. We set initial approximation $\overrightarrow{x_0}=\overline{(0,0,...,0)}$ and converge on some local minimum of (10) after *N* epochs.

The inputs and the loss function are the only things that we need to code explicitly. We will use popular Python libraries TensorFlow [6] and Keras

[7] to implement this neural network – training, minimizing and predicting are done implicitly by these libraries. We will also use NumPy [8] for numerical operations and Matplotlib [9] for visualizing graphs.

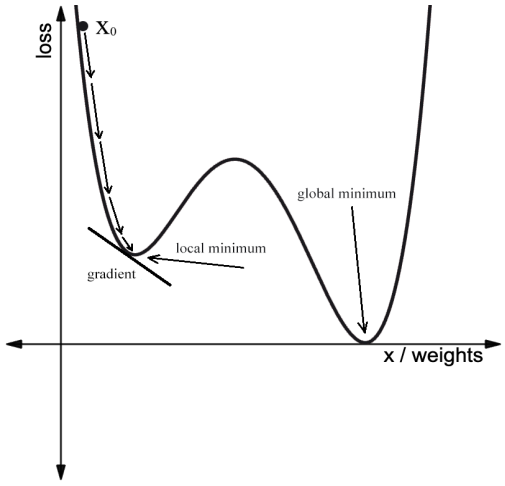

Fig. 1. Minimization of some loss function using stochastic gradient descent. Some local minimum is approximated from x_0 in small incremental steps determined by learning rate of the model

In order to train the model to solve linear algebraic system, we feed matrix A and vector b inside, set initial approximation to all zeroes, choose some learning rate and set the loss function to (10). After the training is complete, we predict the solution as the last approximation of the loss function.

B) Building testing dataset

We begin by generating systems of linear equations. We will generate 3 examples of well-conditioned systems (integers) as well 3 examples of ill-conditioned systems (Hilbert matrices) of sizes (3x3), (5x5) and (10x10). Below are the generated samples:

1) Sample (11)*, a (3×3) system which has* $cond(A) \approx 4.636$

$$\begin{pmatrix} 2 & -1 & -3 \\ -4 & -5 & 4 \\ 4 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} -4 \\ -20 \\ 8 \end{pmatrix}. \quad (11)$$

2) Sample (12)*, a (5×5) system which has* $cond(A) \approx 2.763 \times 10^1$

$$\begin{pmatrix} 3 & -4 & 1 & -4 & -5 \\ 2 & -4 & 0 & -3 & 2 \\ -1 & -3 & -2 & -3 & 4 \\ -5 & -1 & 0 & -1 & -3 \\ -1 & -5 & 1 & -5 & 2 \end{pmatrix} \times \begin{pmatrix} 2 \\ 3 \\ -3 \\ -2 \\ 3 \end{pmatrix} = \begin{pmatrix} -16 \\ 4 \\ 13 \\ -20 \\ -4 \end{pmatrix}. \quad (12)$$

3) Sample (13)*, a (10×10) system which has* $cond(A) \approx 4.534 \times 10^1$.

$$\begin{pmatrix} -5 & -2 & -2 & -5 & 2 & -5 & 1 & 1 & -1 & 3 \\ 2 & -5 & 0 & -3 & -1 & 4 & 3 & 1 & 3 & 4 \\ -1 & -1 & 0 & -3 & -1 & 1 & 4 & 2 & -1 & -4 \\ -5 & -4 & -5 & 4 & -2 & 4 & 4 & 3 & 1 & -5 \\ -5 & 1 & -4 & -2 & 0 & -3 & -4 & 4 & 0 & 1 \\ 4 & 2 & -1 & 2 & 2 & 4 & -1 & 0 & -2 & -4 \\ 3 & 4 & -5 & -5 & 1 & 4 & -1 & 2 & -4 & -1 \\ 2 & -3 & -1 & 0 & 3 & 4 & 4 & 4 & -3 & 1 \\ 0 & 1 & 0 & 4 & 2 & 0 & 4 & 4 & -3 & -4 \\ -2 & -2 & -1 & -3 & -3 & -1 & 0 & 2 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} -4 \\ -2 \\ -1 \\ 4 \\ 1 \\ 1 \\ 2 \\ 3 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 13 \\ 26 \\ -8 \\ 57 \\ 18 \\ -27 \\ -45 \\ 17 \\ 12 \\ 10 \end{pmatrix}. \quad (13)$$

4) Sample (14)*, a (3×3) system which has* $cond(A) \approx 5.262 \times 10^2$

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix} \times \begin{pmatrix} -5 \\ 2 \\ 4 \end{pmatrix} = \begin{pmatrix} -2.6667 \\ -0.8333 \\ -0.3667 \end{pmatrix}. \quad (14)$$

5) Sample (15), *(5×5) system, has* $cond(A) \approx 4.808 \times 10^5$

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix} \times \begin{pmatrix} -5 \\ 3 \\ 3 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} -0.9500 \\ 0.5167 \\ 0.7548 \\ 0.7786 \\ 0.7480 \end{pmatrix}. \quad (15)$$

6) Sample (16), *a (10×10) system, has* $cond(A) \approx 1.633 \times 10^{13}$

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} \\ \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} & \frac{1}{15} \\ \frac{1}{7} & \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} & \frac{1}{15} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} & \frac{1}{15} & \frac{1}{16} & \frac{1}{17} \\ \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} & \frac{1}{15} & \frac{1}{16} & \frac{1}{17} & \frac{1}{18} \\ \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & \frac{1}{14} & \frac{1}{15} & \frac{1}{16} & \frac{1}{17} & \frac{1}{18} & \frac{1}{19} \end{pmatrix} \times \begin{pmatrix} 3 \\ -4 \\ -5 \\ 2 \\ 4 \\ -2 \\ -2 \\ -1 \\ -4 \\ -5 \end{pmatrix} = \begin{pmatrix} -1.0552 \\ -1.5180 \\ -1.4478 \\ -1.3287 \\ -1.2165 \\ -1.1187 \\ -1.0345 \\ -0.9618 \\ -0.8986 \\ -0.8433 \end{pmatrix}. \quad (16)$$

On Fig. 2 shows conditionality of each sample by varying Δb in range $(10^{-5}, 10^{-1})$. Well-conditioned systems in the first row display insignificant change in $\|x\|$ as we add Δb, but ill-conditioned systems in the second row show major change in $\|x\|$ – as big as 10^9 for $\|\Delta b\| = 10^{-2}$.

C) Analyzing solution

On Fig. 3 shows training routine for all samples – after a number of epochs loss function converges on some minimum; the weights are the solution to the system. Results for well-conditioned systems (11), (12), (13) are identical; exact and predicted solutions with $\|\Delta b\| = 10^{-6}$ are perfectly equal and $\|\Delta x\| = 0$. The graph is the same for $\|\Delta b\| = 10^{-4}$ and $\|\Delta b\| = 10^{-2}$. This means that stochastic gradient descent converged to global minimum of the loss functions for the systems, which are the exact solutions, giving $\|\Delta x\| = 0$ and $\|r\| = 0$. This indicates that chosen neural network architecture is capable of solving well-conditioned linear algebraic systems.

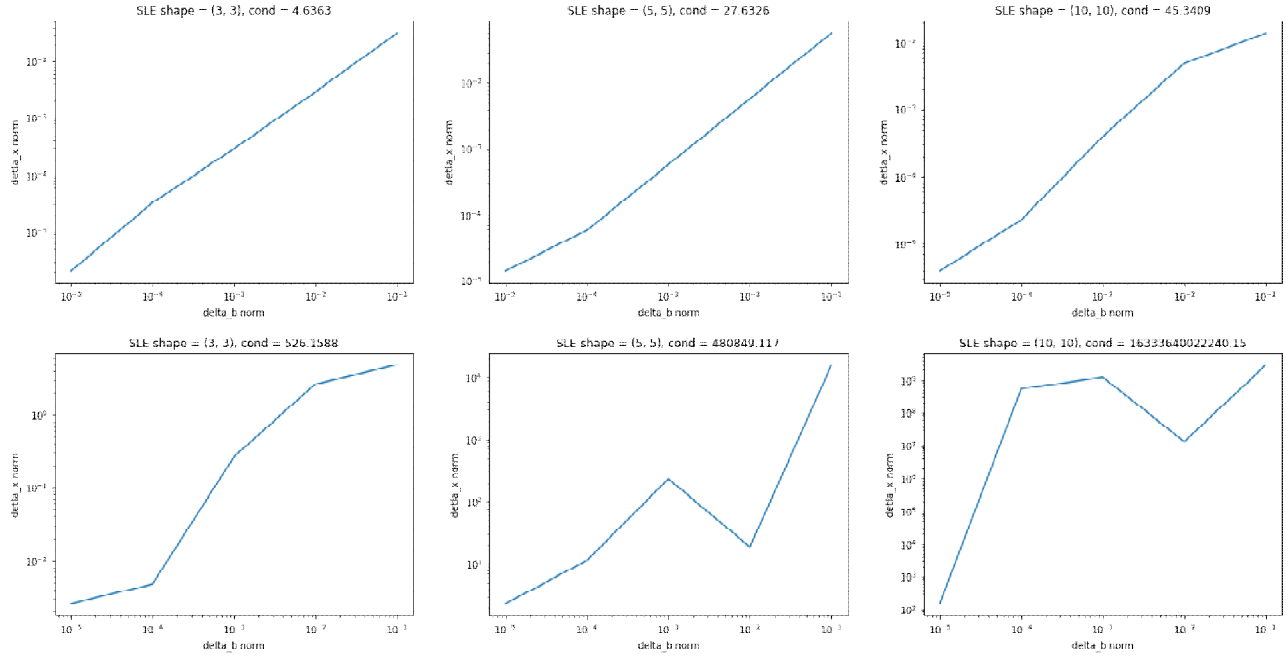

Fig. 2. Small Δb applied to generated samples: Δx is growing as Δb is increased in range $\left(10^{-5}, 10^{-1}\right)$

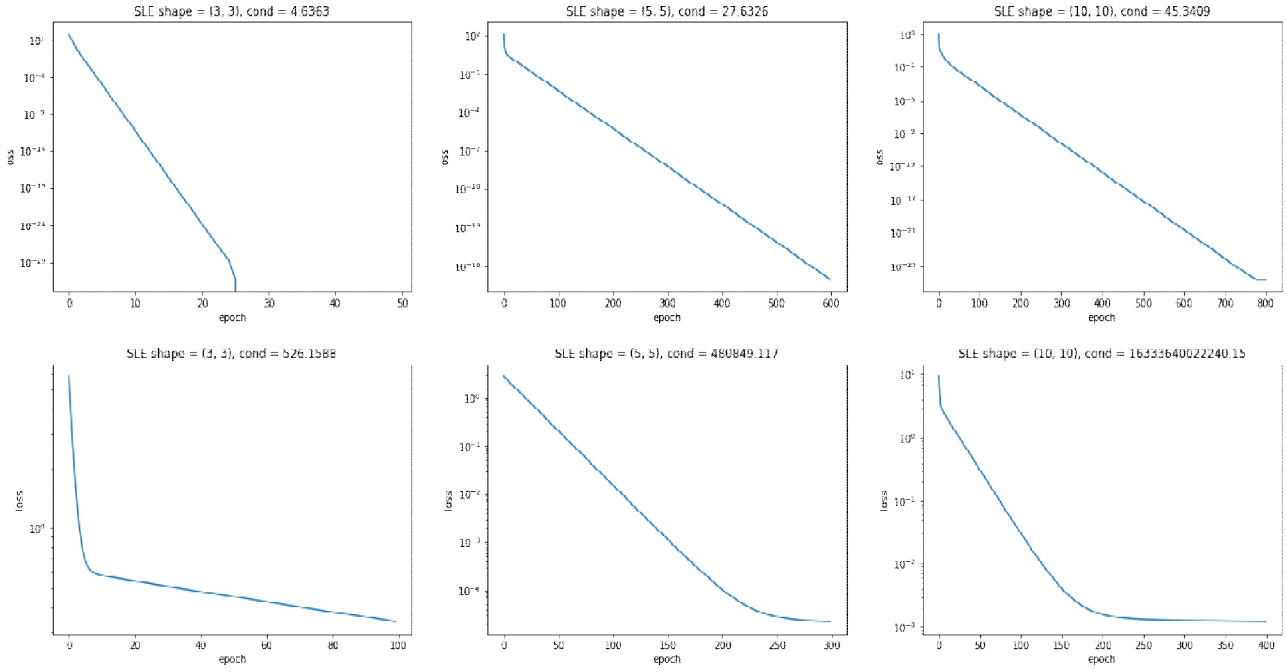

Fig. 3. Loss function converging close to 0 while training on generated samples. Loss function converges faster for well-conditioned systems

Fig. **4** shows components of $\overline{x}$ on x-axis and their values on y-axis for exact (known) and approximated solutions.

The solution for (14), as shown on Fig. 4, has $\|\Delta x\| \approx 22.91$ while $\|r\|$ calculated by (10) is 0.3357. Increasing $\|\Delta b\|$ to 10^{-4} or 10^{-2} in (14) does not change $\|\Delta x\|$ much.

Fig. **5** Fig. **6** show solutions for (15) with $\|\Delta b\| = 10^{-6}$ and $\|\Delta b\| = 10^{-2}$. While in Fig. 5 $\|\Delta x\|$ is comparatively low, in Fig. 6 $\|\Delta x\| \approx 220.21$ with $\|r\| = 6 \times 10^{-6}$.

Finally, Figs. 7, 8 and 9 show results solving (16), a (10×10) linear algebraic system that has $\text{cond}(A) \approx 1.633 \times 10^{13}$. As $\|\Delta b\|$ is increased to

10^{-2}, $\|\Delta x\|$ grows to 1.2659×10^{10} while $\|r\| = 0.0014$. This means that stochastic gradient descent found the local minimum of the loss function for the system (16), which doesn't equal its exact solution and $\|\Delta x\| \approx 1.2659\times10^{10}$. This indicates that demonstrated neural network architecture is not numerically stable and is not capable of approximating solutions for ill-posed problems

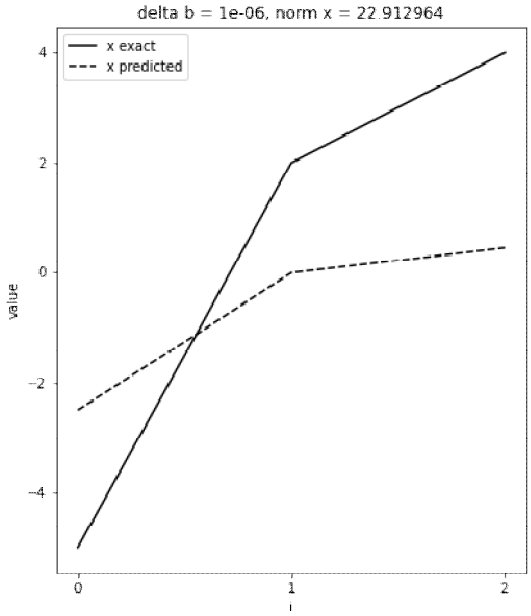

Fig. 4. Exact solution and neural network approximation for (14) with $\|\Delta b\| = 10^{-6}$

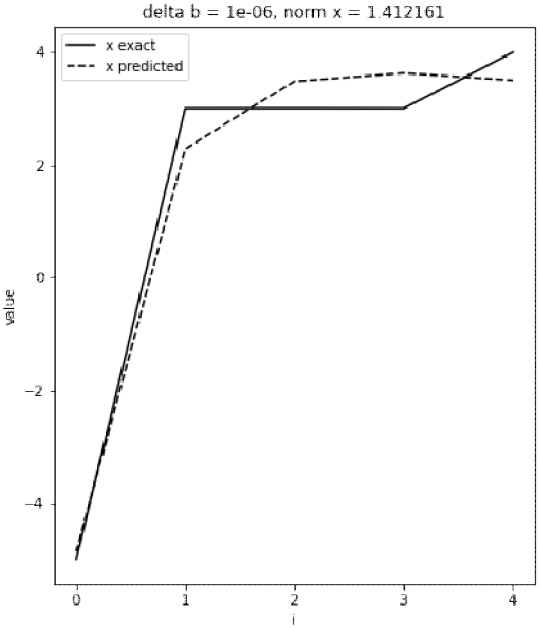

Fig. 5. Exact solution and neural network approximation for (15) with $\|\Delta b\| = 10^{-6}$

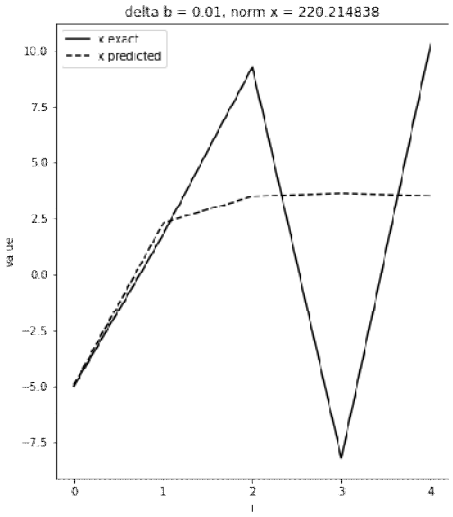

Fig. 6. Exact solution and neural network approximation for (15) with $\|\Delta b\| = 10^{-2}$

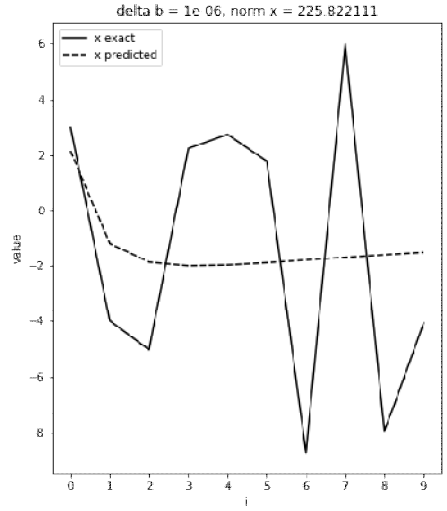

Fig. 7. Exact solution and neural network approximation for (16) with $\|\Delta b\| = 10^{-6}$

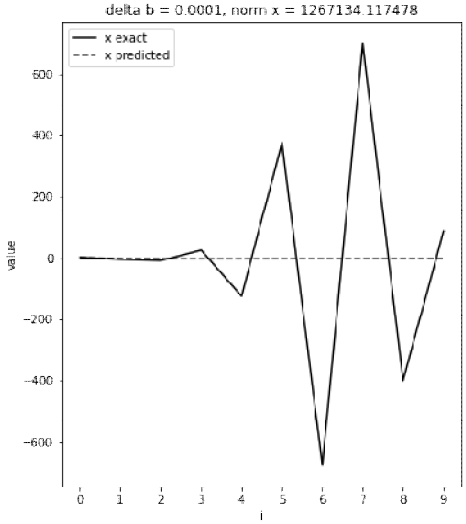

Fig. 8. Exact solution and neural network approximation for (16) with $\|\Delta b\| = 10^{-4}$

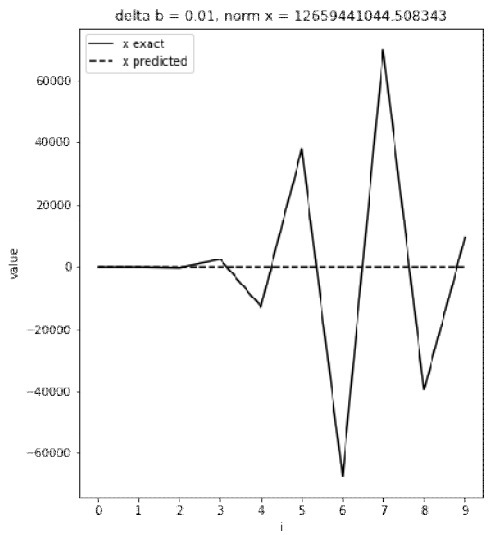

Fig. 9. Exact solution and neural network approximation for (16) with $\|\Delta b\| = 10^{-2}$

IV. CONCLUSIONS

In this article we demonstrated conditionality problem for neural network architecture for solving linear algebraic systems with stochastic gradient descent optimizer. While this architecture performs well for well-conditioned systems, producing exact solution to the system, it is hardly useful for ill-conditioned systems and ill-posed problems. Although the residual is close to 0, the solution itself is very different from the exact solution, having huge norm $\|x\|$. This makes applying this algorithm to some real-world problem impractical. In the next article, we will show some methods and techniques for modifying the proposed solution for better performance on ill-posed problems.

REFERENCES

[1] G. E. Forsythe, M. A. Malcolm, & C. B. Moler, *Computer Solution of Linear Algebraic Systems.* New York: Prentice-Hall, Inc., 1967, 259 p. https://doi.org/10.1002/zamm.19790590235

[2] Hilbert, David, "Ein Beitrag zur Theorie des Legendre'schen Polynoms", (1894) *Acta Mathematica*, 18: 155–159, https://doi.org/10.1007/BF02418278

[3] M. Zgurovsky, V. Sineglazov and E. Chumachenko, *Artificial Intelligence Systems Based on Hybrid Neural Networks. Theory and Applications,* Springer Cham, 2020, 512 p. https://doi.org/10.1007/978-3-030-48453-8

[4] A. Cichocki and R. Unbehauen, "Neural networks for solving systems of linear equations and related problems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications.* vol. 39, no. 2, 1992, pp. 124–138. https://doi.org/10.1109/81.167018

[5] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search,". *Neural Networks for Signal Processing,* Proceedings of the 1992 IEEE Workshop, vol. 2, Sep 1992, pp. 1–11. https://doi.org/10.1109/NNSP.1992.253713

[6] TensorFlow documentation. https://www.tensorflow.org/api_docs/python/tf

[7] Keras documentation. https://keras.io/api/

[8] NumPy documentation. https://numpy.org/doc/stable/reference/index.html#reference

[9] Matplotlib documentation. https://matplotlib.org/stable/api/index.html

Received January 12, 2023

Mamonov Volodymyr. Bachelor's degree student.
Department of Artificial Intelligence. Institute of Applied Systems Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Research area: artificial intelligence, neural networks, distributed computing.
E-mail: mamonov.volodymyr@lll.kpi.ua

Shatikhin Yevhen. Bachelor's degree student.
Department of Artificial Intelligence. Institute of Applied Systems Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Research area: artificial intelligence, neural networks, distributed computing.
E-mail: shatikhin.yevhen@lll.kpi.ua

Tymoshenko Yurii. ORCID 0000-0001-7812-6437. Candidate of Sciences (Engineering). Associated Professor.
Institute of Applied Systems Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: Odesa Polytechnic Institute, Odesa, Ukraine (1974).
Research area: inverse ill-posed problems, artificial intelligence, computer vision, fault-tolerant systems.
Publications: 116 papers.
E-mail: yury.alex.tym@gmail.com

В. В. Мамонов, Є. О. Шатіхін, Ю. О. Тимошенко. Нейромережеве розв'язання систем лінійних алгебричних рівнянь. Частина 1

Останнім часом нейронні мережі стали все популярнішими завдяки своїй універсальності у вирішенні складних проблем. Однією з цікавих областей їх застосування є розв'язання лінійних алгебричних систем, особливо тих, що є погано обумовленими. Розв'язки подібних систем дуже чутливі до невеликих змін їх коефіцієнтів, що призводить до нестійких рішень. Тому розв'язання цих типів систем може бути складною задачею і вимагати спеціальних прийомів. У статті досліджено застосування нейронних мереж для розв'язання систем лінійних алгебричних рівнянь, зосереджуючись на погано обумовлених системах. Основною метою є розробка моделі, здатної безпосередньо розв'язувати лінійні рівняння та оцінювання її продуктивності на ряді систем, в тому числі погано обумовлених. Для вирішення цієї проблеми було побудовано нейронну мережу, яка реалізує ітеративний алгоритм. Функція помилки лінійної алгебричної системи мінімізується за допомогою стохастичного градієнтного спуску. Ця модель не потребує тривалого навчання, окрім налаштування швидкості навчання для особливо великих систем. Аналіз показує, що запропонована модель може добре справлятися з гарно обумовленими системами різних розмірів, хоча для систем з великими коефіцієнтами потрібна нормалізація. Для ефективного розв'язання погано обумовлених систем потрібні покращення, так як досліджуваний алгоритм виявився арифметично нестійким. Це дослідження сприяє розумінню та застосуванню методів нейронних мереж для розв'язання лінійних алгебричних систем. Це забезпечує основу для майбутніх просувань у цій галузі та відкриває нові можливості для вирішення складних проблем. З подальшими дослідженнями та розвитком моделі нейронних мереж можуть стати потужним інструментом для розв'язання погано обумовлених лінійних систем та інших пов'язаних проблем.

Ключові слова: системи лінійних алгебричних рівнянь; число обумовленості; погано обумовлені системи; нейронна мережа; градієнтний спуск; TensorFlow.

Мамонов Володимир Володимирович. Студент бакалаврату.
Кафедра штучного інтелекту, Інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Напрям наукової діяльності: штучний інтелект, нейронні мережі, розподілені обчислення.
E-mail: mamonov.volodymyr@lll.kpi.ua

Шатіхін Євген Олексійович. Студент бакалаврату.
Кафедра штучного інтелекту, Інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Напрям наукової діяльності: штучний інтелект, нейронні мережі, розподілені обчислення.
E-mail: shatikhin.yevhen@lll.kpi.ua

Тимошенко Юрій Олександрович. ORCID 0000-0001-7812-6437. Кандидат технічних наук. Доцент.
Інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Освіта: Одеський політехнічний інститут, Одеса, Україна, (1974).
Напрям наукової діяльності: обернені некоректні задачі, штучний інтелект, комп'ютерний зір, відмовостійкі системи.
Кількість публікацій: 116 наукових робіт.
E-mail: yury.alex.tym@gmail.com