Jerzy Tchórzewski[1]
ORCID: 0000-0003-2198-7185

Sebastian Rosochacki[2]
ORCID: 0009-0006-3514-0005

University of Siedlce
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

[1]jerzy.tchorzewski@uws.edu.pl,[2]s.rosochacki@protonmail.com

Selection of the programming environment for neural modelling of the power and electricity demand generation systems in terms of unmanned factories

DOI: 10.34739/si.2023.29.07

Abstract. The work concerns the selection of the programming language and environment for the needs of neural modeling of the power and electricity demand generation system in terms of uninhabited factories. Therefore, the main goal of the conducted research is to obtain the best possible Artificial Neural Network, i.e. to teach it a model of a real system, which is a system for generating demand for power and electricity based on numerical data on parts of the power system operation in terms of uninhabited factories. The learning capabilities of artificial neural networks were checked by comparing the MSE error and the Regression Index R2. In each of the examined programming languages and related programming environments, i.e. Matlab, Python and Wolfram, an Artificial Neural Network with the same structure and properties was designed and implemented, i.e. with the same number of input and output neurons, the number of

hidden layers and the number of neurons in them, the activation function of neurons and the learning method. In addition to the ANN training of the system model, testing and validation as well as comparative studies were carried out.

Keywords: National Power System, MATLAB, Simulink, Python, Artificial Neural Network, Wolfram Mathematica

1 Introduction

Currently, research on the functioning and development of intelligent systems and the information and communication technologies used to model them using stochastic methods, machine learning and artificial intelligence methods are extremely important [16-19, 22-24, 26, 28-29, 37-39 , 41-42]. Intelligent systems are developing in various areas of the economy, administration and society with the use of the whole wealth of artificial intelligence methods [3-4, 19, 26, 33-34, 37, 42]. These include systems such as smart grid, smart power grid, smart metering, unmanned systems or even uninhabited factories [29, 38-39]. One of the models of such systems are neural models obtained in the form of artificial neural networks. The process of creating an artificial neural network requires many design activities, including the selection of a set of pairs of input quantities and output quantities related to them, on the basis of which the structure of the Artificial Neural Network is then designed, i.e. its layers of neurons and connections between them together with assigning them appropriate parameters such as weights and biases as well as neuron activation functions. The next step is the selection of the learning method and the selection of the programming language and the associated programming environment, which is to lead to ANN learning with the use of learning pairs in accordance with the given quality measure. The process is repeated until the ANN quality measure is met [1, 4, 22-24, 33-34, 42]. In recent years, many practical methods of learning and assessing ANN quality have been developed, including system models [3, 19, 26, 29, 37], and even algorithms for automatic selection of ANN structure [22-23]. However, scientifically justified methods of selecting the ANN learning method for the designed task, as well as the selection of the programming language and the programming environment associated with it, have not been published. This article addresses this research problem using two ANN learning methods, i.e. the Levenberg-Marquardt (LM) method and the ADAM method [2, 4-5, 9, 13-16, 24-25, 30, 32, 42]. and three programming environments: MATLAB with Matlab, PYTHON with Python and Mathematica with Wolfram [6-8, 10-11, 17, 20-21, 31, 40].

2 Research experiments

2.1 The structure of numerical data

To conduct a research experiment consisting in learning the Perceptron Artificial Neural Network a model of a smart system, numerical data from the National Power System recorded every hour of the day in the period from 01/04/2020 to 30/04/2022 was used, i.e. numerical data concerning: the volume of electricity [MWh] by generating units Centrally Dispatched Generating Units (CDGU, Polish: JWCD), which are subject to the supervision of the Transmission

System Operator; volume of electricity produced [MWh] by generating units generating units which are not Centrally Dispatched Generating Units (nCDGU, Polish: nJWCD), which are not subject to the supervision of the Transmission System Operator; Domestic Power Demand (DPD) [MW] used by all participants of the National Power System (NPS) [12, 27, 36, 38]. The data was subjected to special preprocessing, e.g. an equal and repeating number of hours in daily reports was adopted, making the experiment immune to the change of time from summer to winter and vice versa from winter to summer. In addition, data for the purposes of learning, testing and validating artificial neural networks were normalized by dividing individual numerical values for a given hour by the sums resulting from all hours of the day. Then, for the purposes of research experiments, the data was further divided into 10 periods of 365 days (contractual annual periods) with the structure listed in Table 1 [27-28, 38].

Table 1. Division of the National Power System data into research experiments in a rolling system. Source: own elaboration using works [27-28, 38]

Experiment number	Beginning data	End data
E1	01.04.2020	30.03.2021
E2	01.05.2020	30.04.2021
E3	31.05.2020	30.05.2021
E4	30.06.2020	29.06.2021
E5	30.07.2020	29.07.2021
E6	29.08.2020	28.08.2021
E7	28.09.2020	27.09.2021
E8	28.10.2020	27.10.2021
E9	27.11.2020	26.11.2021
E10	27.12.2020	26.12.2021

2.2 Artificial Neural Network Architecture

For the purpose of designing the Artificial Neural Network, 48 input quantities and 24 output quantities were adopted. The input values are: daily summary generation of CDGUs separately for each hour of the day (24 quantities) and daily summary generation of nCDGUs separately for each hour of the day (24 quantities), and the output values are daily domestic power demand separately for each hour of the day (24 values). In this way, an ANN with 48 inputs and 24 outputs was obtained [27-28].

Then, after the preliminary analysis, one hidden layer was assumed, and the number of neurons in this layer was selected using the Sartori method [30] and the method of Taimura and Tateishi [35], setting their number as 2/3 of the neurons of the input layer, i.e. 32 neurons. As a result of preliminary research experiments in the MATLAB and Simulink environments, their number was finally set at 34 neurons. In addition, the hyperbolic tangent was assumed for the neuron activation function in the hidden layer, and the linear neuron activation function in the output layer.

2.3 Purpose of conducting research experiments

It was assumed that the main purpose of the conducted research experiments is to compare the performance of programming languages and programming environments using two algorithms for learning artificial neural networks, i.e. the Levenberg-Marquardt algorithm, used e.g. electricity consumption in Greece [16] and the ADAM algorithm, which is an algorithm used in the TensorFlow environment and in the MATHEMATICA environment [7, 10-11, 18, 31, 40]. Comparative analysis using the Levenberg-Marquardt algorithm was carried out using the Matlab language and the Python language, while the comparative analysis using the ADAM algorithm was performed using the Python language and the Wolfram language. Due to the lack of a default implementation of the Levenberg-Marquardt algorithm in the TensorFlow library, an implementation of the algorithm by Fabio Di Marco [10] was used.

Learning using the Levenberg-Marquardt algorithm was carried out using the maximum number of epochs set at 760 and the early stop of the so-called early-stopping set to 5 epochs. When learning using the ADAM algorithm, the time required to process one epoch was clearly shorter compared to learning using the Levenberg-Marquardt algorithm, however, training brought smaller changes in network quality. For this reason, 760 epochs turned out to be insufficient to achieve satisfactory results, so the number was increased to 200,000 epochs, and the number of early-stopping to the limit of 1,500 epochs. The learning speed coefficient in the experimental way was set to 0.001, because its higher values caused a very unstable learning process. Particular attention was paid to the courses of changes in the Regression Index R2 during learning, which are illustrated in Figure 1.

In addition, the sample size was increased to 256 to compensate for the impact of low-quality data on the weights of the learning Artificial Neural Network due to the fact that their impact needed to be strengthened due to dividing the data into smaller sets for further research experiments.

3 Comparison of programming languages due to the learning method

A comparison of three programming languages was carried out due to the mean square error MSE and due to the Regression Index R^2. The following programming environments were selected for comparison: MATLAB, PYTHON and MATHEMATICA. Comparative analysis of learning artificial neural networks was carried out for two learning methods, i.e. for learning using the Levenberg-Marquardt algorithm and using the ADAM algorithm. In the case of the Levenberg-Marquardt algorithm, a comparison was made of ANN learning designed in Matlab and Python, and in the case of the ADAM algorithm, ANN learning in Wolfram Language and Python was compared. In both cases, 10 research experiments were carried out for the assumed 10 periods of NPS operation (Table 1). The ANN learning results obtained in the case of the Levenberg-Marquardt algorithm are presented in Table 2, and in the case of the ADAM algorithm - in Table 3. In addition, selected waveforms of the MSE error obtained in the epoch after the sharp decline and the Regression Index R2 after the end of the learning process for the Levenberg-Marquardt algorithm are presented in Figures 2-3 and for the ADAM algorithm in Figures 4-5.

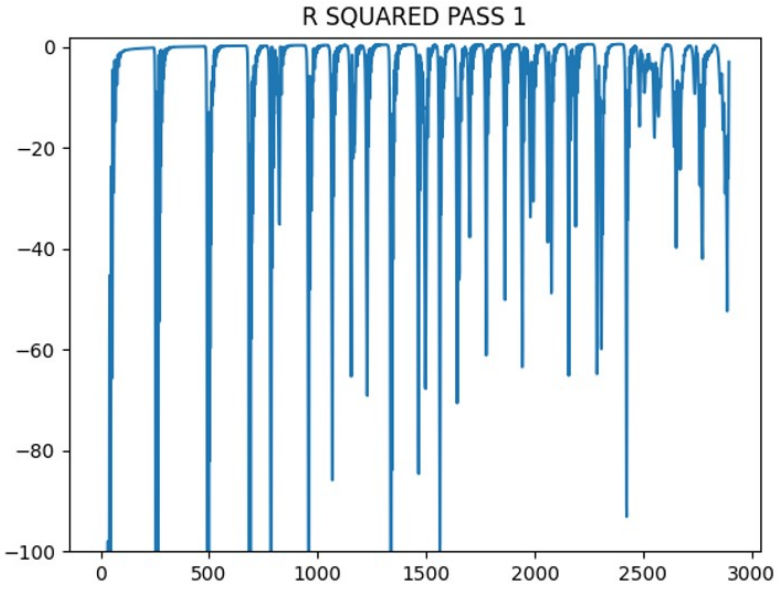

Figure 1. Courses of the Regression Index R^2 during the learning process using the ADAM algorithm at the learning coefficient lr=0.01 during the experiment 1. Designations: axis Y - Regression Index R^2; axis X - learning epoch number. Source: own elaboration using Matplotbib [20, 28].

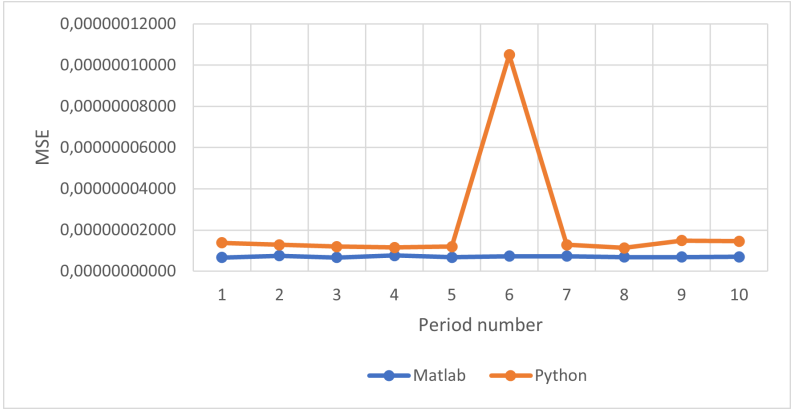

Figure 2. The course of the MSE error after a sharp decrease in the ten periods of NPS operation under the study for the LM method designed in Matlab and Python. Source: Own elaboration using [28].

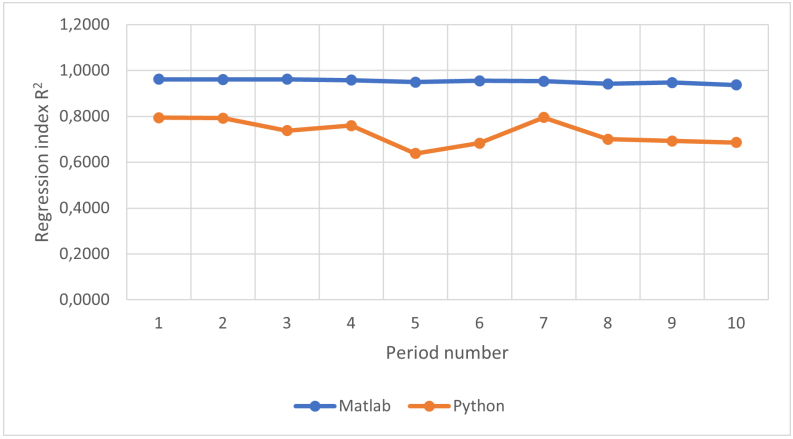

Figure 3. The course of Regression Index R^2 for the LM method designed in Matlab and Python. Source: Own elaboration using [28].

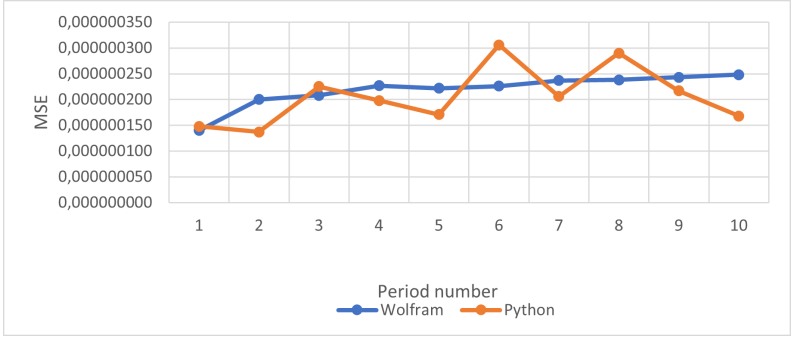

Figure 4. The course of the MSE error after a sharp decrease in the ten periods of NPS operation under the test for the ADAM method designed in Wolfram and Python. Source: Own elaboration using [28].

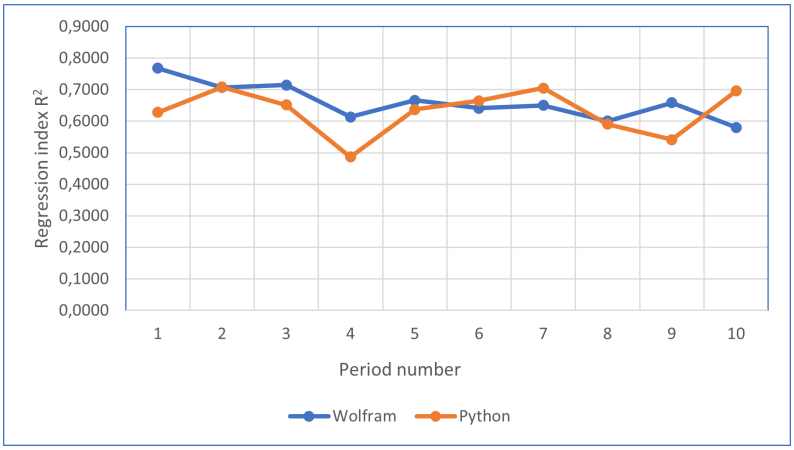

Figure 5. The course of the Regression Index R^2 for the ADAM method designed in Wolfram and in Python. Source: Own elaboration using [28].

It turned out that in the case of the LM method, the ANN learning of the NPS operation is at a similar MSE error level both in the case of implementation in Matlab and in Python. However, for the ANN implementation in Matlab, training is associated with a lower MSE error after a sharp decline of $7.06 \cdot 10^{-9}$ vs. MSE error of $22.1 \cdot 10^{-9}$ for the Python implementation. Moreover, in Experiment 6 in Python, the MSE error turned out to be much higher, amounting to $1.05 \cdot 10^{-7}$, which in Matlab was $7.34 \cdot 10^{-9}$.

In addition, the ANN in Matlab needed only 8 epochs to achieve the required MSE error rate, and the ANN in Python as many as 41 epochs. It is also worth noting that in the case of the ANN implementation in Matlab, the Regression Index R^2 was much better, at an average level of 0.9577, while in Python it was only 0.7278 on average.

On the other hand, in the case of the ADAM method, the ANN learning of the NPS functioning implemented in Python and Wolfram was at a similar level of MSE error, but in Python the learning was associated with a slightly lower MSE error after a sharp decrease of $2.07 \cdot 10^{-7}$, which for the implementation in Wolfram was $2.27 \cdot 10^{-7}$, taking place in the 9th epoch for Python, compared to an average of 11 epochs for the implementation in Wolfram.

4 Conclusions and directions for further research

The obtained research results summarized in Tables 2 and in Tables 3 indicate that the Levenberg-Marquardt algorithm turned out to be a better algorithm for solving the problem of learning the neural model of the examined part of the NPS system on selected 10 research experiments in the case of ANN implementation in Python, for which the average MSE error was $2.21 \cdot 10^{-8}$. It was an order of magnitude lower than in the case of the ADAM Algorithm, for which it was $2.07 \cdot 10^{-7}$.

Table 2. MSE error and Regression Index R^2 results obtained in the case of learning the Perceptron ANN design in Matlab and in Python using the Levenberg-Marquardt algorithm for 10 data periods of the operation of the National Power System. Source: own elaboration using paper [28].

No.	Language	MSE · beginning	MSE after a sharp drop		Stopping the algorithm		R^2
			epoch	MSE	epoch	MSE	
E1	Matlab	$7.28 \cdot 10^{-7}$	8	$6.69 \cdot 10^{-9}$	19	$7.05 \cdot 10^{-10}$	0.9616
	Python	$2.00 \cdot 10^{-6}$	36	$1.38 \cdot 10^{-8}$	134	$5.16 \cdot 10^{-9}$	0.7940
E2	Matlab	$8.60 \cdot 10^{-7}$	8	$7.48 \cdot 10^{-9}$	18	$8.10 \cdot 10^{-10}$	0.9613
	Python	$2.26 \cdot 10^{-6}$	33	$1.28 \cdot 10^{-8}$	133	$3.65 \cdot 10^{-9}$	0.7921
E3	Matlab	$7.16 \cdot 10^{-7}$	8	$6.62 \cdot 10^{-9}$	21	$6.25 \cdot 10^{-10}$	0.9619
	Python	$2.48 \cdot 10^{-6}$	42	$1.21 \cdot 10^{-8}$	129	$3.80 \cdot 10^{-9}$	0.7377
E4	Matlab	$9.66 \cdot 10^{-7}$	8	$7.62 \cdot 10^{-9}$	22	$6.77 \cdot 10^{-10}$	0.9577
	Python	$3.43 \cdot 10^{-6}$	41	$1.15 \cdot 10^{-8}$	200	$5.14 \cdot 10^{-9}$	0.7598
E5	Matlab	$9.04 \cdot 10^{-7}$	8	$6.82 \cdot 10^{-9}$	19	$7.84 \cdot 10^{-10}$	0.9498
	Python	$5.17 \cdot 10^{-6}$	41	$1.20 \cdot 10^{-8}$	139	$6.73 \cdot 10^{-9}$	0.6377
E6	Matlab	$1.05 \cdot 10^{-6}$	8	$7.34 \cdot 10^{-9}$	22	$6.00 \cdot 10^{-10}$	0.9549
	Python	$6.31 \cdot 10^{-6}$	42	$1.05 \cdot 10^{-7}$	149	$4.46 \cdot 10^{-9}$	0.6829
E7	Matlab	$3.52 \cdot 10^{-6}$	8	$7.28 \cdot 10^{-9}$	24	$6.32 \cdot 10^{-10}$	0.9532
	Python	$3.00 \cdot 10^{-6}$	43	$1.28 \cdot 10^{-8}$	133	$6.81 \cdot 10^{-9}$	0.7949
E8	Matlab	$7.60 \cdot 10^{-7}$	8	$6.88 \cdot 10^{-9}$	20	$6.84 \cdot 10^{-10}$	0.9416
	Python	$3.71 \cdot 10^{-6}$	45	$1.13 \cdot 10^{-8}$	114	$5.11 \cdot 10^{-9}$	0.7003
E9	Matlab	$6.60 \cdot 10^{-7}$	8	$6.90 \cdot 10^{-9}$	21	$7.19 \cdot 10^{-10}$	0.9471
	Python	$3.55 \cdot 10^{-6}$	36	$1.49 \cdot 10^{-8}$	113	$4.00 \cdot 10^{-9}$	0.6928
E10	Matlab	$1.17 \cdot 10^{-6}$	8	$6.98 \cdot 10^{-9}$	22	$6.95 \cdot 10^{-10}$	0.9372
	Python	$4.70 \cdot 10^{-6}$	50	$1.46 \cdot 10^{-8}$	166	$2.30 \cdot 10^{-9}$	0.6855
Values average	Matlab	$\mathbf{1.13} \cdot 10^{-6}$	**8**	$\mathbf{7.06} \cdot 10^{-9}$	**20.80**	$\mathbf{6.93} \cdot 10^{-10}$	**0.9577**
	Python	$\mathbf{3.66} \cdot 10^{-6}$	**40.90**	$\mathbf{2.21} \cdot 10^{-8}$	**141.00**	$\mathbf{47.16} \cdot 10^{-10}$	**0.7278**

For the LM algorithm, the Regression Index R^2 was also higher, which amounted to 0.7278, and in the case of the implementation using the ADAM algorithm, it was only 0.6309. It also turned out that achieving these results was achieved at the expense of a smaller number of epochs for the Levenberg-Marquardt algorithm, which reached a sharp decline on average in epoch 9, while for the ADAM algorithm it took place in epoch 41. It is also worth adding that in the case of the LM algorithm ANN training lasted only 141 epochs, and in the case of the ADAM algorithm, on average, as many as 163,729. In turn, the Levenberg-Marquardt algorithm turned out to be the stronger point of the Matlab language, which after a deep

decline reached the MSE error level of $7.06 \cdot 10^{-9}$, which is an order of magnitude lower than in the case of Python, which after a deep decline amounted to $22.1 \cdot 10^{-9}$. It is worth noting that the MSE error after a sharp drop in the case of ANN implementation using the ADAM method was comparable in the case of Python, which was $2.07 \cdot 10^{-7}$. Meanwhile, for Wolfram it was slightly higher than in Python, which was $2.27 \cdot 10^{-7}$ (it was slightly higher).

Table 3. The results of the MSE error and the Regression Index R^2 obtained in the case of learning the Perceptron ANN deisgned in Wolfram Language and in Python with use of the ADAM algorithm for 10 data periods of the operation of the National Power System. Source: own elaboration using paper [28].

No	Language	MSE · beginning	MSE·after·a·sharp·drop		MSE·after·stopping·the·algorithm		R^2
			epoch	MSE	epoch	MSE	
E1	Wolfram	$1.60 \cdot 10^{-6}$	13	$1.40 \cdot 10^{-7}$	200000	$6.34 \cdot 10^{-9}$	0.7679
	Python	$3.00 \cdot 10^{-6}$	15	$1.48 \cdot 10^{-7}$	80568	$8.48 \cdot 10^{-9}$	0.6277
E2	Wolfram	$1.59 \cdot 10^{-6}$	7	$2.00 \cdot 10^{-7}$	200000	$6.64 \cdot 10^{-9}$	0.7070
	Python	$2.13 \cdot 10^{-6}$	11	$1.37 \cdot 10^{-7}$	197434	$7.49 \cdot 10^{-9}$	0.7084
E3	Wolfram	$1.62 \cdot 10^{-6}$	7	$2.08 \cdot 10^{-7}$	170874	$7.20 \cdot 10^{-9}$	0.7147
	Python	$3.3 \cdot 10^{-6}$	12	$2.25 \cdot 10^{-7}$	200000	$6.62 \cdot 10^{-9}$	0.6518
E4	Wolfram	$1.64 \cdot 10^{-6}$	7	$2.27 \cdot 10^{-7}$	154558	$7.75 \cdot 10^{-9}$	0.6135
	Python	$3.95 \cdot 10^{-6}$	15	$1.98 \cdot 10^{-7}$	116194	$8.24 \cdot 10^{-9}$	0.4864
E5	Wolfram	$1.68 \cdot 10^{-6}$	7	$2.22 \cdot 10^{-7}$	200000	$6.85 \cdot 10^{-9}$	0.6664
	Python	$1.98 \cdot 10^{-6}$	9	$1.71 \cdot 10^{-7}$	123775	$6.60 \cdot 10^{-9}$	0.6368
E6	Wolfram	$1.72 \cdot 10^{-6}$	7	$2.26 \cdot 10^{-7}$	200000	$7.88 \cdot 10^{-9}$	0.6408
	Python	$2.70 \cdot 10^{-6}$	10	$3.06 \cdot 10^{-7}$	200000	$6.17 \cdot 10^{-9}$	0.6644
E7	Wolfram	$1.76 \cdot 10^{-6}$	7	$2.37 \cdot 10^{-7}$	154558	$8.21 \cdot 10^{-9}$	0.6501
	Python	$3.49 \cdot 10^{-6}$	13	$2.06 \cdot 10^{-7}$	200000	$6.12 \cdot 10^{-9}$	0.7051
E8	Wolfram	$1.78 \cdot 10^{-6}$	7	$2.38 \cdot 10^{-7}$	200000	$8.38 \cdot 10^{-9}$	0.6001
	Python	$4.27 \cdot 10^{-6}$	15	$2.90 \cdot 10^{-7}$	200000	$5.66 \cdot 10^{-9}$	0.5906
E9	Wolfram	$1.77 \cdot 10^{-6}$	7	$2.43 \cdot 10^{-7}$	173444	$8.81 \cdot 10^{-9}$	0.6586
	Python	$3.25 \cdot 10^{-6}$	13	$2.17 \cdot 10^{-7}$	119234	$7.21 \cdot 10^{-9}$	0.5415
E10	Wolfram	$1.79 \cdot 10^{-6}$	7	$2.48 \cdot 10^{-7}$	200000	$6.85 \cdot 10^{-9}$	0.5803
	Python	$2.40 \cdot 10^{-6}$	12	$1.68 \cdot 10^{-7}$	200000	$6.57 \cdot 10^{-9}$	0.6960
Average	Wolfram	**1.17** $\cdot 10^{-6}$	**11.45**	**2.27** $\cdot 10^{-7}$	**185443.40**	**7.49** $\cdot 10^{-9}$	**0.6594**
	Python	**3.05** $\cdot 10^{-6}$	**9.00**	**2.07** $\cdot 10^{-7}$	**163729.50**	**6.92** $\cdot 10^{-9}$	**0.6309**

From the course of the MSE error during the training process, as well as from the course of the Regression Index R^2, it can be seen that increasing the capacity of the ANN by adding further hidden layers and/or increasing the number of neurons in the hidden layers may have a beneficial effect on improving both the MSE error and the Regression Index R^2. In such situations, there was no visible effect of ANN overtraining even when using a very large number of epochs, which is worth examining in more detail in the next research experiments. The research experiments presented in this work show that in the case of smart grid systems, and even more so in unpopulated systems with a high degree of automation, it is very important to precisely select both learning algorithms as well as programming languages and related programming environments, hence this type of research is worth continuing.

References

1. Aribib M.A.: The Handbook of Brain Theory and Neural Networks, The MIT Press, Cambridge, pages 1134, 2003.
2. Bahi M., Batouche M.: Deep Learning for Ligand-Based Virtual Screening in Drug Discovery, 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), Tebessa, Algeria, pp. 1-5, 2018.
3. Beale M., Hagan M., Demuth H.: Deep Learning Toolbox User's Guide, The MathWorks, Natick 2020-2022.
4. Błaszczyk P.: Sztuczna Inteligencja, Uniwersytet Śląski (in Polish), http://books.icse.us.edu.pl/runestone/static/ai/index.html (access: 28.02.2023).
5. Capellman J.: Hands-On Machine Learning with ML.NET, Packt Publishing, Birmingham, pages 296, 2020.
6. Chollet F.: Deep Learning with Python, Manning's Publications Co., Shelter Island, pages 478, 2021.
7. Chonacky, N., Winch, D.: Reviews of Maple, Mathematica, and Matlab: Coming Soon to a Publication Near You. Computing in Science & Engineering, IEEE XPlore Digital Library, Vol. 7, No. 2, pp. 9-10, March-April 2005.
8. Ciskowski P.: Poznawanie własności sieci neuronowych w środowisku MATLAB (in Polish), Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, pages 102, 2012.
9. Demuth H., Beale M.: Neural Network Toolbox User's Guide, The MathWorks, Natick 2002-2019.
10. Di Marco F.: TensorFlow implementation of Levenberg-Marquadt training algorithm: https://github.com/fabiodimarco/tf-levenberg-marquardt(access: 15.02.2023r.)
11. Freeman J.A.: Simulating Neural Networks with Mathematica, Addison-Wesley Publishing Company, Reading, pages 341, 1994.
12. Giełdowa Platforma Informacyjna, Słownik pojęć (in Polish), http://gpi.tge.pl/informacje/slownik-pojec (access: 16.01.2023).
13. Guesmi L., Fathallah, H., Menif M.: Modulation Format Recognition Using Artificial Neural Networks for the Next Generation Optical Networks, pp. 2-19. 2018.
14. Haykin S.: Neural Networks, A Comprehensive Foundation, Pearson Education, New Delhi, pages 842, 1999.
15. Jain L., Fanelli A.M.: Recent advances in Artificial Neural Networks Design and Applications, CRC Press, Boca Raton, pages 372, 2017.
16. Karampelas P., Vita V., Pavlatos C., Mladenov V., Ekonomou L.: Design of Artificial Neural Network Models for the Prediction of the Hellenic Energy Consumption, 10th Symposium on Neural Network Applications in Electrical Engineering NEUREL-2010, Faculty of Electrical Engineering, University of Belgrade, pp. 41-44, 2010.
17. Kasabov N.K.: Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering, The MIT Press, Cambridge, pages 581, 1998.
18. Kingma D.P., Ba J.L.: Adam: A Method for Stochastic Optimization, International Conference on Learning Representations, San Diego, pages 13, 2015
19. Marlęga R.: Correction of the parametric model of the Day-Ahead Market system using the Artifical Neural Network, Studia Informatica. Systems and Information Technology, Vol. 1(26)2022, pp. 85-105.
20. MathWorks, Perceptron Neural Networks, MATLAB Help Center 2022, https://www.mathworks.com/help/deeplearning/ug/perceptron-neural-networks.html (access: 05.02.2023)
21. MathWorks, The: Hyperbolic tangent sigmoid transfer function, MATLAB Help Center, 2022, https://www.mathworks.com/help/deeplearning/ref/tansig.html (access: 05.02.2023).

22. Obuchowicz A.: Optimization of Neural Network Architectures, Chapter 9, [in:] Intelligent Systems, [eds] Wilamowski B. M., Irvin J. D., The Industrial Electronics Handbook, Second Edition, Taylor and Francis Group, LLC, pp. 9.1-9.24, 2011.
23. Obuchowicz, A. 2000. Optimization of neural network architectures. In Biocybernetics and Biomedical Engineering (in Polish), Neural Networks, [eds.] W. Duch, J. Korbicz, L. Rutkowski, and R. Tadeusiewicz, Academic Publishing House EXIT, Warsaw, pp. 323-368, 2000.
24. Osowski S.: Sieci neuronowe do przetwarzania informacji (in Polish), OW PW, Warszawa, pages 422, 2013
25. Patterson J., Gibson A.: Deep Learning. Parctitioner's Aproach, O'Reilly Media, Sebastopol, pages 83, 2017.
26. Płaczek S., Płaczek A.: Uczenie wielowarstwowych szerokich sieci neuronowych z funkcjami aktywacji typu ReLU w zadaniach klasyfikacji (in Polish), Poznań University of Technology Academic Journals, pp. 47-58, Poznań 2018.
27. Polskie Sieci Elektroenergetyczne SA, Wielkości podstawowe raportów dobowych z pracy KSE (in Polish), https://www.pse.pl/dane-systemowe/funkcjonowanie-kse/raporty-dobowe-z-pracy-kse (access: 26.01.2023, 08.05.2022)
28. Rosochacki S.: Komparatystyka języków programowania do modelowania sztucznej sieci neuronowej z wykorzystaniem danych dotyczących Krajowego Systemu Elektroenergetycznego, Praca inżynierska napisana w Instytucie Informatyki pod kierunkiem dr hab. inż. Jerzego Tchórzewskiego, prof. uczelni, Wydział Nauk Ścisłych i Przyrodniczych, UPH, Siedlce, pages, 67, 2023.
29. Ruciński D.: Modeling of the Day-Ahead Market on the Polish Power Exchange on the Example of Selected Artificial Neural Networks, Chapter 4, [in:] Theory and Application of Artificial Neural Intelligence Methods, [eds.:] Tchórzewski J., Świtalski P., Series Intelligent Systems and Information Technologies, Wydawnictwo UPH, Siedlce, pp. 85-117, 2022.
30. Sartori M.A., Antsaklis P.J.: A simple method to derive bounds on the size and to train multilayer neural networks. IEEE Transaction of Neural Network, 2(4), pp. 467-71, 1991.
31. SciSharp: Tensorflow.NET, https://scisharp.github.io/tensorflow-net-docs/ (access: 02.02.2023)
32. Shah M.: Fundamentals of Computer Vision, University of Central Florida, Orlando, pages 133, 1997.
33. Tadeusiewicz R.: Sieci neuronowe (in Polish), Akademicka Oficyna Wydawnicza, Warszawa, pages 195, 1993.
34. Tadeusiewicz R.: Archipelag sztucznej inteligencji (in Polish), OW EXIT, Warszawa, pages 126, 2022.
35. Tamura S, Tateishi M.: Capabilities of a four layered feedforward neural network: four layers versus three. IEEE Transaction of Neural Network, 8(2), pp. 251 -155, March 1997.
36. Tauron Polska Energia, Karta Aktualizacji nr 16/2020 https://www.tauron-dystrybucja.pl/-/media/offer-documents/dystrybucja/uslugi-dystrybucyjne/iriesd/26-05-2020/projekt_karty-_aktualizacji_16_2020_iriesd.ashx (access: 26.01.2023)
37. Tchórzewski J.: Metody sztucznej inteligencji i informatyki kwantowej w ujęciu teorii sterowania i systemów (in Polish), Wydawnictwo Naukowe UPH, Siedlce, pages 343, 2021.
38. Tchórzewski J.: Rozwój systemu elektroenergetycznego w ujęciu teorii sterowania i systemów (in Polish), Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, pages 190, 2013.
39. Tchórzewski J. Wielgo A.: Neural model of human gait and its implementation in MATLAB and Simulink Environment using Deep Learning Toolbox, Studia Informatica. Systems and Information Technology, Vol. 1-2(25)2021, pp. 39-65.
40. TensorFlow Team, The.: Introduction to TensorFlow Datasets and Estimators, Google Developers Blog, 2017, https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html (access: 28.02.2023).

41. Yang T., Blom H. A., Mehta P. G.: Interacting Multiple Model-Feedback Particle Filter for Stochastic Hybrid Systems, Proceedings of the IEEE Annual Conference on Decision and Control (CDC), pp. 7065-7070, 2013.
42. Żurada J., Barski M., Jędruch W.: Sztuczne Sieci neuronowe. Podstawy teorii i zastosowania (in Polish), Wydawnictwo PWN, Warszawa, pages 375, 1996