Article

Not peer-reviewed version

Symbolic Analysis of Classical Neural Networks for Deep Learning

Maja Lutovac-Banduka [*] , Igor Franc , Vladimir Milićević , Nemanja Zdravković , Nikola Dimitrijević

Posted Date: 7 November 2023

doi: 10.20944/preprints202311.0446.v1

Keywords: machine learning; visual programming language; closed-form expression; theoretical justification; feature extraction; complexity; optimization and validation

Article

Symbolic Analysis of Classical Neural Networks for Deep Learning

Maja Lutovac-Banduka [1,*], Igor Franc [2], Vladimir Milićević [3], Nemanja Zdravković [2] and Nikola Dimitrijević [2]

[1] RT-RK LLC (former Department of RT-RK Institute, Computer Based Systems), 2v Dunavska, 11158 Belgrade, Serbia; maja.lutovac-banduka@rt-rk.com
[2] Fakultet informacionih tehnologija, Univerzitet Metropolitan, Tadeuša Košćuška 63, 11158 Belgrade, Serbia; igor.franc@metropolitan.ac.rs; nemanja.zdravkovic@metropolitan.ac.rs; nikola.dimitrijevic@metropolitan.ac.rs
[3] Fakultet za mašinstvo i građevinarstvo u Kraljevu, Univerzitet u Kragujevcu, Dositejeva 19, 36000 Kraljevo, Serbia; milicevic.v@mfkv.kg.ac.rs
* Correspondence: majamlutovac@gmail.com; Tel.: +381-62-8132280

Abstract: Deep learning is based on matrix computing with a large amount of hidden parameters that is not visible outside the computing module. Deep learning is a non-linear system and a linear approach is not possible. It is natural for people to visualize the algorithm and to follow some hidden parameters. In this paper, we propose a simple graphical programming of a nonlinear system based on drawing the simplest unit, such as a single neuron model. A more complex scheme of a classical neural network is obtained by commands copy-move-place of one neuron. The number of neurons and layers can be chosen arbitrarily. Once the scheme is complete, the implementation code is evaluated with symbolic parameters and nonlinear activation functions. This cannot be done manually. With the symbolic expression of outputs in terms of inputs and symbolic parameters, including symbolic activation pure functions, some other properties can be derived in closed form. This unique original approach can help scientists and developers and design numerical algorithms for machine learning and to understand how deep learning algorithms work.

Keywords: machine learning; visual programming language; closed-form expression; theoretical justification; feature extraction; complexity; optimization and validation

1. Introduction

Machine learning, as subfield of Artificial Intelligence (AI), refers to computers learning to do things on their own [1]. Machine learning (ML) is the core of AI which solves the problems of classification, clustering, and forecasting complex algorithms and methods, in order to overcome the limitations and achieve expansion of AI and ML applications [2]. There are many applications in several fields of basic science and engineering focused on some important and challenging topics, such as nonlinear problems, numerical methods, analytical methods, error analysis and mathematical models [3]. A review of a variety of experiments on extracting structure from machine-learning data is presented in [4]. Some algorithms do not need a huge set of training data and can be implemented only by applying standard templates and thus provide the high response speed [5]. Some applications with nonlinear process demonstrate the different neural network-based models [6]. Some heuristic procedures can reduce computational complexity with respect to optimal multiple sequence alignment and thus outperform humans in the post-processing of recognition hypotheses [7]. Adaptive network enhancement method for the best least squares approximation using two-layer Rectified-Linear-Unit (ReLU) neural networks is introduce in [8]. Trends for mathematical explanations of the theoretical aspects of Artificial Neural Networks (ANN), with a special attention to activation functions, can be used to absorb the defining features of each design scenario [9]. Some other researchers have developed cost estimation techniques, which are based on statistical and computational techniques [10]. Learning representations of graphs have become a popular learning model for prediction tasks for a practice [11]. Graph Neural Networks (GNNs), neural network architectures targeted to. Symbolic regression, as the task of predicting the mathematical expression of a function from the observation of its values, is a difficult task. Neural networks have recently been tasked to predict the correct skeleton in a single try, but still remain much less powerful [12].

The biggest disadvantage of machine learning is that most of them cannot be explained. Up to now, AI solved problems that are intellectually difficult for humans but relatively straightforward for computers problems. As explained in the previous paragraph, success is possible if we are using a list of formal mathematical rules. The challenge to AI is to solve the tasks that are easy for humans to perform, but formally hard to describe in spite that we can solve intuitively. The hierarchy enables the computer to learn complicated concepts by building them out of simpler ones. The graph is deep when it has many layers. For this reason, we call this approach to AI deep learning [13]. The difficulties facts by systems relying on hard-coded know knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data [13]. This capability is known as machine learning. We can use mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. The powerful root mean square error accuracy comparison of gradient boosting machines (a popular machine learning regression tree algorithm) we can use to avoid the multicollinearity of multivariate input variables with the simulated normal distribution data [14]. For solving decision-making problems in both computer science and engineering, we can use Optimization Machine Learning Toolkit as an open-source software package incorporating neural network [15]. Software tools can simplify training neural networks, using machine learning, from simpler into larger optimization problems.

Recent advances in machine learning have led to increased interest in solving visual computing problems using methods that employ coordinate-based neural networks. These methods we call neural fields. [16]. Machine learning tools are part of Wolfram Language that performs classification, regression, dimensionality reduction and neural network processing. The book [1] takes a "show, don't tell" approach and we use the same examples to show how to use the Wolfram Language [17]. Using a visual programming language (the SchematicSolver application package [18], requires software Mathematica 9 [17]) we can develop own algorithms and understand the architecture of complex networks. Mathematical education at the end of high school should be sufficient to understand mathematical content [1]. A visual programming or GUI (Graphical User Interface) can provide fast interactive design, symbolic analysis, accurate simulation, exact verification, and reports [19]. It helps to skip the gap between theory and practice in continuous-time systems. The mathematical representation of the system can be obtained automatically from the schematic description, and automated symbolic manipulations are possible.

Although accurate, AI models often are “black boxes” which we are not able to understand [20]. Transfer learning is a powerful approach that leverages knowledge from one domain to enhance learning in another domain; it provides practical insights for researchers and practitioners interested in applying transfer learning techniques [21]. The pervasive success in predictive performance comes alongside a severe weakness, the lack of explainability of their decisions, especially relevant in human-centric [22]. From black-box to explainable AI in healthcare: existing tools and case studies [23].

Convolutional Neural Networks (CNNs) constitute a widely used deep learning approach that has frequently been applied to the problem of brain tumor diagnosis. Such techniques still face some critical challenges in moving towards clinic application [24]. The smart visualization of medical images (SVMI) model is based on multi-detector computed tomography (MDCT) data sets that can provide a clearer view of changes in the brain, such as tumors (expansive changes), bleeding, and ischemia on native imaging. The new method provides a more precise representation of the brain image by hiding pixels that are not carrying information and rescaling and coloring the range of pixels essential for detecting and visualizing the disease [25].

The main trends with which machine learning techniques have been applied to robotic manipulation are presented in [26]. Resources in the robotics field are limited for a common user to access and demand deep knowledge of robot programming paradigm. It is presented how to program control algorithms and to control complex robotic systems by using widespread and inexpensive devices such as smartphones and tablets, Android operating system, for simulation and remote control of robotic manipulators [27]. The review paper on deep learning-assisted smart process planning, robotic wireless sensor networks and geospatial big data management algorithms in the internet of manufacturing things configures on deep learning-assisted smart process [28]. The paper [29] presents a system for programming and simulation of the human centrifuge motion with integrated 3D motion simulator. The simulator can be used for verification of newly defined motion commands and testing the new algorithms for the human centrifuge motion.

Analog designs that implement various machine learning algorithms can be useful in low-power deep network accelerators suitable for edge or tiny machine learning applications [30]. Analog circuit design requires large amounts of human knowledge; a special case of circuit design is the synthesis of robust and failure-resilient electronics [31]. An example of dissipation minimization of two-stage amplifier using deep learning is presented in [32] (deep learning is a subset of machine learning). The deep learning module controls the operating mode of the amplifier which is a function of the average input signal. The application of VLSI (Very Large Scale Integrated) circuit in AI is described in [33]. VLSI is the inter-disciplinary science of utilizing advance semiconductor technology to create various functions of computer system such as the close link of microelectronics and AI. By combining VLSI technology, a very powerful computer architecture confinement is possible.

The purpose of this paper is an attempt to explain machine learning models using algorithm visualization. The main goal of the work is the final results as expressions in a closed form where the developer can change and monitor hidden parameters.

2. Materials

Deep learning can involve learning using an artificial neural network. Humans have about one hundred billion biological neurons in their brain and about ten thousand times more connections. A neuron has its inputs x_1, x_2, ..., x_i, via input branches. The neuron then calculates an electrical output y_j, that is sent to other neurons through tiny intercellular connections.

Artificial neural networks use the same basic principles that are much simpler than their biological neurons. For given input values x_1, x_2, ..., x_i, the artificial neuron has only one output y_j, whose value is calculated by the formula:

$$y_j = f\big(w_1x_1 + w_2x_2 + \cdots + w_ix_i + b_j\big). \tag{1}$$

The parameters, w_1, w_2, ..., w_i, change during neuron operation and are interpreted as strengths of the connection between neurons. They are called weights or weight parameters. The parameter b_j is called the bias and is interpreted as the threshold at which the neuron is activated. The nonlinear function f is called the activation function or the transfer function.

An illustration of the calculation of an artificial neuron (which gives the same output expression as it should theoretically be) is given in the Figure 1 (drawn with SchematicSolver ver2.3 [18] that is written using Wolfram language, Mathematica ver. 13 [17]):

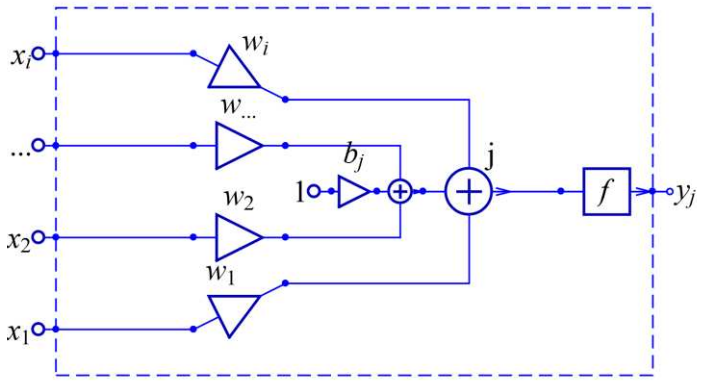


Figure 1. Illustration of the computation made by an artificial neuron.

The first part is a linear combination of the inputs, and then the nonlinearity f is applied. Nonlinearity allows neuron to model nonlinear system. Biological neurons are either activated or not. Therefore, it is tempting to use some kind of activation function. Artificial neurons used logistic sigmoid function, hyperbolic tangent function or Ramp function (rectified linear unit, ReLU) that works pretty well in deep neural networks [1], see Figure 2:

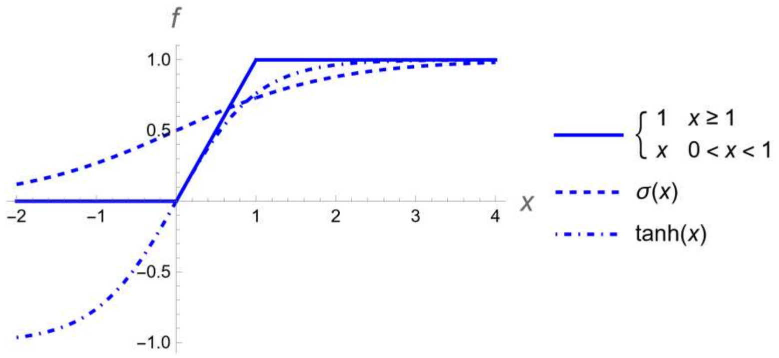


Figure 2. Classic activation functions as rectified linear unit ReLU, logistic sigmoid σ(x), or hyperbolic tangent function tanh(x).

3. Methods

In Figure 3 on the left, the architecture of the artificial neuron is drawn, where the input signals x_1, x_2, ..., x_i, are separated from the parameters that are modified during the operation of the neuron. The same figure on the right shows a block diagram of a neuron that depends exclusively on input signals, and the parameters w_1, w_2, ..., w_i, that are listed as inputs through which the output is calculated. It is important to note that multipliers with constant values are not used as in linear systems, because the parameters w_1, w_2, ..., w_i and b_j, are not constants. They are modified during the operation of the neuron.

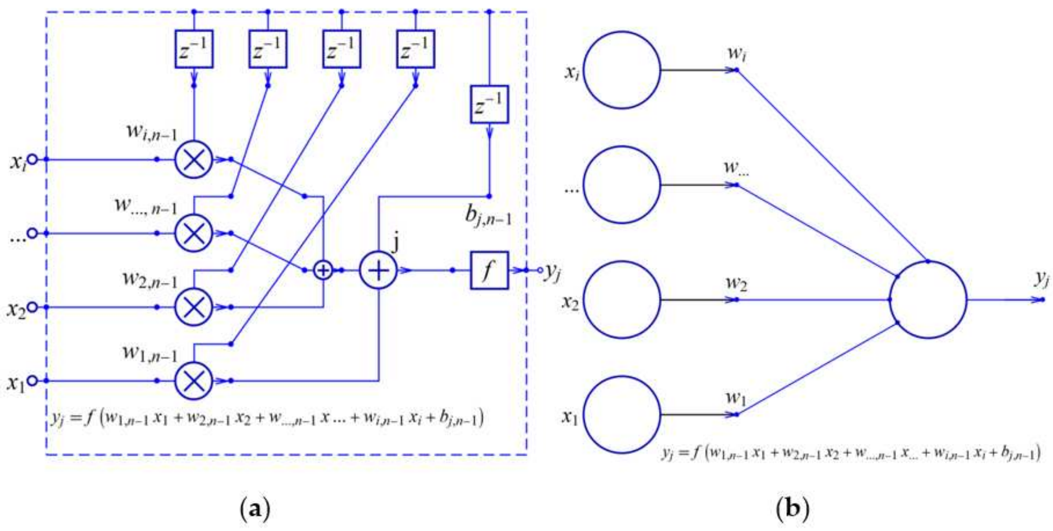


Figure 3. Classic artificial neuron (a) with memory elements, (b) simplified.

By connecting a large number of artificial neurons, networks can be made. Circles represent numerical values called activations. It is understood to be a linear combination of its inputs, plus some bias term, and the result is passed through the nonlinearity. The simplest network has two input values and one output value. The graph is directed and acyclic, so the output can be calculated simply by following the arrows.

This network is a parametric model. There is one weight parameter per input and one bias parameter per neuron. We could train this network in the same way as any other parametric model: by minimizing a cost function calculated from some training data. The architecture does not change; only the numerical values of the parameters (weights and biases) are changed. Familiar architecture is used; neurons are grouped in layers and each neuron in one layer sends its output to each neuron in the next layer. These fully connected layers are also called linear layers.

Let the network consist of several layers, and let there be four neurons in each layer. Four input signals come to the first layer. The output of each neuron generates only one output, and since there are 4 neurons, 4 output signals are generated. The output signals of the first layer become the input signals for the second layer, which also has 4 neurons and generates 4 output signals. The output signals of the second layer are the input signals of the third layer, which also has 4 neurons with 4 output signals. The output signals of the third layer are the outputs of the neural network. The first and second layers are called hidden layers because the outputs of the neurons are not seen outside the network.

The number of input signals to the first layer must not be greater than 4, and to ensure that some of the 4 network inputs do not affect the network outputs, the weight parameters are set to 0 from the network inputs in the first layer to the adders.

Assume that there are only two numeric values as inputs and that the first layer generates four numeric values as the output of the first hidden layer. Now the input of the second hidden layer has 4 numerical values as inputs, and it generates 4 output values which are the values of the third layer neurons. These four inputs generate 4 output numerical values that become the outputs of the neural network, and therefore the third layer is not a hidden layer because the results are visible. The activation function generates one of the 4 classes to which the processing result belongs, and the activation function is unambiguously mapped to the output, that is, the one that has all positive results (all probabilities must be positive) and the sum of all probabilities (outputs) is used as a nonlinear function of the last layer) must be 1 (sum of probabilities of all classes must be 1). Such a network is used to train a classifier that has four possible classes, and the output values would then be the class probabilities. For a regression task, you would have only one output (predicted value) and that is the class that is most likely.

In this example, two hidden layers are included, but there could be many more. A network with two or more hidden layers is called a deep network, hence the name "deep learning". This name emphasizes the importance of using models that can perform several computational steps.

3.1. Matrix representation

As an illustration of how a neural network works, we will analyze a network that has 4 neurons in three layers as a minimal deep learning network. Usually, a mathematical approach is used through matrices, which are easily implemented in numerical programs. The first input layer contains 4 weight coefficients for each neuron, so a total of 2×4=8 parameters for the weights of the first layer to calculate the output values of the first hidden layer. One bias parameter per output is added. Visually, the weight parameters are represented as a 4×4=16 matrix (for four neurons, each with 4 weights), and a 1×4=4 matrix (for four neurons, each with one 1 bias parameter). For input values, we will use a 1×4=4 matrix with numerous values that can be input signals to the network or output signals of the previous layer. The matrix representation is as follows according [x1]:

$$\mathbf{Y} = f(\mathbf{W} \cdot \mathbf{X} + \mathbf{B}) \tag{2}$$

where:

$$\mathbf{Y} = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{vmatrix}, \quad \mathbf{W} = \begin{vmatrix} w_1 & w_5 & w_9 & w_{13} \\ w_2 & w_6 & w_{10} & w_{14} \\ w_3 & w_7 & w_{11} & w_{15} \\ w_4 & w_8 & w_{12} & w_{16} \end{vmatrix}, \quad \mathbf{X} = \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{vmatrix}, \quad f(\mathbf{A}) = \tanh(\mathbf{A}) \tag{3}$$

3.2. Matrix computation

The activation function is chosen to be the hyperbolic tangent. For only two inputs, numerous values were arbitrarily chosen, while the remaining two inputs were set to 0. All parameters were obtained by a random number generator (`RandomReal`) in the Wolfram language. In order to obtain always the same numerical value, the function `SeedRandom[123]` is used. Now the number values for the first layer are rounded to 3 decimal places, according to [x1]:

$$\mathbf{X} = \begin{vmatrix} 1.2 \\ 4.5 \\ 0 \\ 0 \end{vmatrix}, \quad \mathbf{W} = \begin{vmatrix} -0.089 & 0.956 & 0 & 0 \\ 0.887 & 0.924 & 0 & 0 \\ -0.395 & -0.067 & 0 & 0 \\ -0.877 & -0.229 & 0 & 0 \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} -0.140 \\ 0.557 \\ -0.903 \\ 0.257 \end{vmatrix}, \quad \mathbf{Y} = \begin{vmatrix} 0.999398 \\ 0.999981 \\ -0.932446 \\ -0.949308 \end{vmatrix} \tag{4}$$

To make sure that the third and fourth inputs do not affect the result, some weight parameters are set to 0. This is therefore a network with only two inputs.

The output values of the first hidden layer are represented as a matrix Y. These values become the inputs for the next layer, so the following values are obtained for the second hidden layer:

$$\mathbf{X} = \begin{vmatrix} 0.999398 \\ 0.999981 \\ -0.932446 \\ -0.949308 \end{vmatrix}, \mathbf{W} = \begin{vmatrix} -0.444 & -0.820 & 0.753 & -0.782 \\ -0.468 & 0.837 & -0.660 & -0.801 \\ -0.059 & -0.194 & 0.943 & -0.370 \\ -0.748 & -0.455 & 0.211 & 0.344 \end{vmatrix}, \mathbf{B} = \begin{vmatrix} -0.515 \\ -0.016 \\ 0.126 \\ 0.098 \end{vmatrix}, \mathbf{Y} = \begin{vmatrix} -0.940069 \\ 0.938869 \\ -0.575210 \\ -0.925832 \end{vmatrix} \tag{5}$$

The output values of the second hidden layer are shown as the matrix Y. These values become the inputs to the output layer, so the following values are obtained for the third layer:

$$\mathbf{X} = \begin{vmatrix} -0.9400769 \\ 0.938869 \\ -0.575210 \\ -0.925832 \end{vmatrix}, \quad \mathbf{W} = \begin{vmatrix} -0.037 & 0.289 & -0.165 & 0.409 \\ 0.923 & 0.808 & 0.741 & 0.263 \\ -0.215 & -0.127 & 0.137 & -0.034 \\ -0.679 & 0.927 & -0.908 & -0.007 \end{vmatrix}, \quad \mathbf{B} = \begin{vmatrix} -0.008 \\ -0.975 \\ -0.023 \\ -0.843 \end{vmatrix}, \quad \mathbf{Y} = \begin{vmatrix} 0.600 \\ 0.184 \\ 0.031 \\ 0.184 \end{vmatrix} \tag{6}$$

The output values of the last layer do not use the hyperbolic tangent for the activation function, but the so-called **`softmax`** function that calculates the probability of the occurrence of an output:

$$\frac{e^{y_i}}{e^{y_1}+e^{y_2}+e^{y_3}+e^{y_4}}, \; i=\{1, 2, 3, 4\} \tag{7}$$

The fourth outcome has a probability of 0.600, i.e. 60.0%, the first and third outcomes have a probability of 0.184 (18.4%) and the first outcome has a probability of 0.031 (3.1%). All probabilities must be positive, and the sum of probabilities must be 1 (the first decimal place may have a difference due to rounding).

A classifier that has parameters, as in this example, shows that the most likely result is class 4. The weight parameters w_1, w_2, ..., w_i and the bias parameter b_j change during the operation of the neuron. In total, in the analyzed network with two hidden layers and one output layer of 4 neurons, there are 3×4×4=48 weight parameters and 3×1×4=12 bias parameters, i.e. in total the analyzed network has 48+12=60 parameters which are changed during neural network processing.

4. Results

In computing it is natural to use a matrix data structure. It is more natural for people to see an architectural sketch.

4.1. Symbolic computation

Firstly we will draw the architecture of one layer consisting of 4 neurons using SchematicSolver [18] as shown in Figure 4.

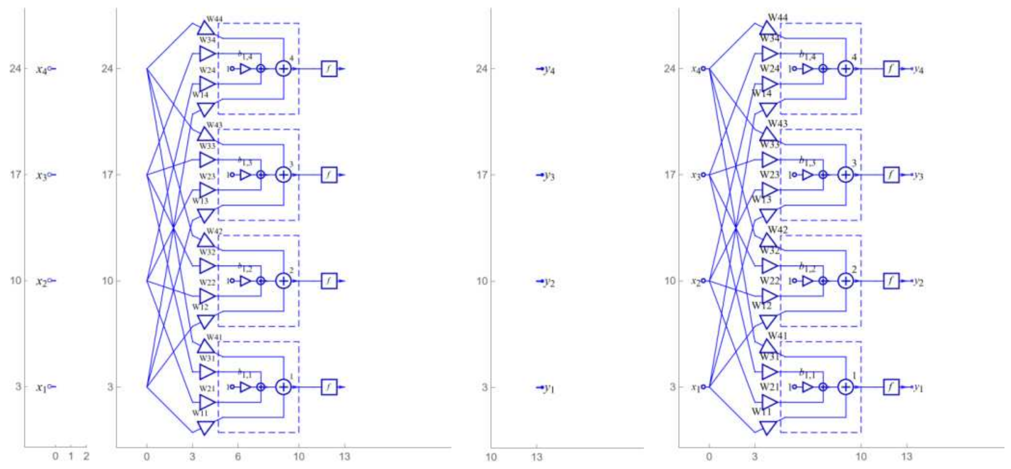

(a) (b) (c) (d)

Figure 4. Classic artificial neural network with single layer: (a) 4 inputs, (b) layer with 4 neurons, (c) 4 outputs, (d) overall schematic.

Inputs and outputs are drawn separately for a neural network with one or more layers, Figure 4(a) and Figure 4(c). Using one layer, Figure 4(b), the neural network is obtained by joining the input, output and one layer shown in Figure 4(d).

To draw a neural network with three layers, an already drawn layer, image in Figure 4(b) is used, which is copied the necessary three times to the right with a raster of 13 (how wide the base layer is) and the output layer is shifted to the right by 3×13=39. Symbolic parameter names and activation functions are changed with each copy. The complete scheme is obtained by connecting the inputs, outputs and all layers and is shown in Figure 5.

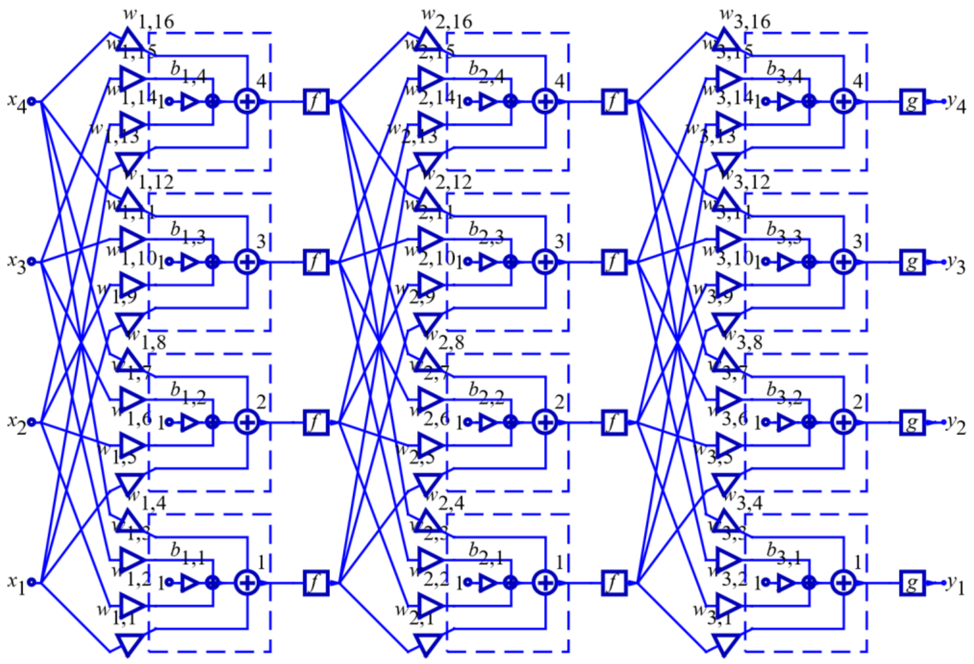
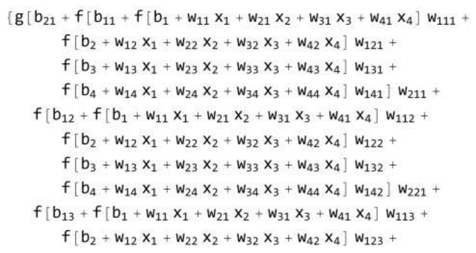


Figure 5. Classic artificial neural network with 3 layers.

It is important to notice that the activation function of the final layer is different, *g* such that the output values are equal to the class probabilities.

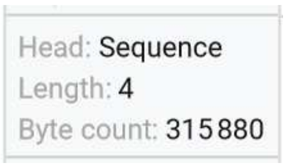
4.2. Symbolic solving

By clicking on the button to implement the schematic from the image drawn in SchematicSolver, the implementation code is obtained, a small part of which is in Figure 6(a).



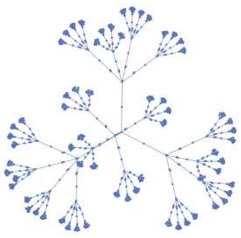
$$
\begin{aligned}
\{ & g[b_{21} + f[b_{11} + f[b_1 + w_{11}\,x_1 + w_{21}\,x_2 + w_{31}\,x_3 + w_{41}\,x_4]\,w_{111} + \\
& \quad f[b_2 + w_{12}\,x_1 + w_{22}\,x_2 + w_{32}\,x_3 + w_{42}\,x_4]\,w_{121} + \\
& \quad f[b_3 + w_{13}\,x_1 + w_{23}\,x_2 + w_{33}\,x_3 + w_{43}\,x_4]\,w_{131} + \\
& \quad f[b_4 + w_{14}\,x_1 + w_{24}\,x_2 + w_{34}\,x_3 + w_{44}\,x_4]\,w_{141}]\,w_{211} + \\
& f[b_{12} + f[b_1 + w_{11}\,x_1 + w_{21}\,x_2 + w_{31}\,x_3 + w_{41}\,x_4]\,w_{112} + \\
& \quad f[b_2 + w_{12}\,x_1 + w_{22}\,x_2 + w_{32}\,x_3 + w_{42}\,x_4]\,w_{122} + \\
& \quad f[b_3 + w_{13}\,x_1 + w_{23}\,x_2 + w_{33}\,x_3 + w_{43}\,x_4]\,w_{132} + \\
& \quad f[b_4 + w_{14}\,x_1 + w_{24}\,x_2 + w_{34}\,x_3 + w_{44}\,x_4]\,w_{142}]\,w_{221} + \\
& f[b_{13} + f[b_1 + w_{11}\,x_1 + w_{21}\,x_2 + w_{31}\,x_3 + w_{41}\,x_4]\,w_{113} + \\
& \quad f[b_2 + w_{12}\,x_1 + w_{22}\,x_2 + w_{32}\,x_3 + w_{42}\,x_4]\,w_{123} +
\end{aligned}
$$

(**a**)



Head: Sequence
Length: 4
Byte count: 315880

(b)



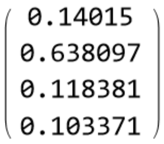
(c)

Figure 6. Classic artificial neural network with 3 layers: (a) smaller part of symbolic response, (b) size of response, and (c) tree graph with different levels at different depths.

Symbolically derived expressions for 4 classes use almost 316KB. Such a performance in closed form is not possible to derive manually. Visually, it can be seen that the graph with all levels is very complex. The symbolically derived expression can be used for further analysis, which is not possible for a purely numerical approach. By changing the symbolic values with the numerical parameters from the matrix approach, the same probabilities are obtained, a 60.0% probability that the classification number is 4 as with matrix computing.

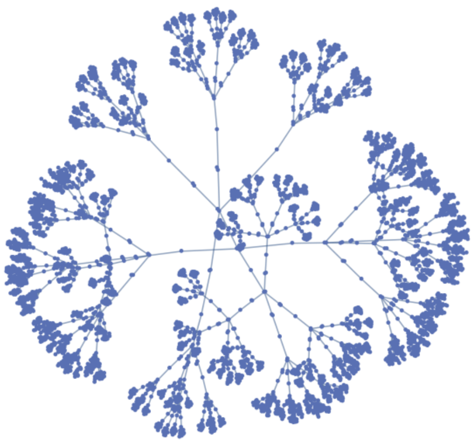
4.3. *Symbolic analysis*

In the same way, a neural network with 4 layers is obtained, of which 3 are hidden layers. With the same values for the parameters as with 2 hidden layers, but with the addition of randomly generated values for the newly added layer, a probability of 63.8% is obtained for class 2. The complexity of the closed-form final expressions in the symbolic notation is shown in Figure 7. This derivation exceeds manual execution.



$$
\begin{pmatrix}
0.14015 \\
0.638097 \\
0.118381 \\
0.103371
\end{pmatrix}
$$

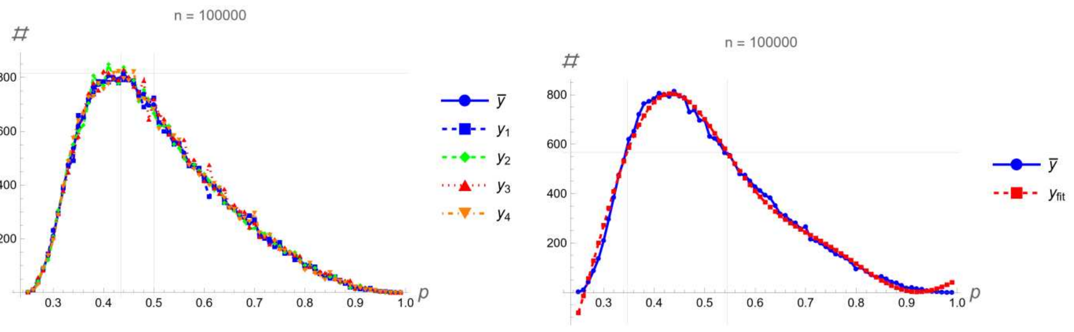
(**a**)



(b)

Figure 7. Classic artificial neural network with 4 layers: (a) probabilities of four classes and (b) tree graph with different levels at different depths.

The symbolically obtained response of the neural network enables further analysis without the use of schematics. With 100,000 randomly generated parameters, probabilities for all outputs were determined as shown in Figure 8. .



(**a**) (b)

Figure 8. (a) Class probabilities of all outputs and (b) mean value of probabilities.

For the mean value, an expression was derived in closed form:

$$y(x) = 96.267 + 86.935 \cos(16.728\,x) - 344.78 \log(x) + 592.72 \log(x) \sin(9.2\,x) \tag{8}$$

The most likely probability was calculated when the maximum number of classes was detected, which is for a probability of 43.3%, when 800 cases were detected. For 0.7071 in relation to the maximum, the range in which there is the largest number of certain classes was calculated, which is for the range between 34.7% to 54.4% probability. Half of all classes were detected for this range. It is important to note that there cannot be a probability of less than 25% when a class is detected, because at least one other class would have a probability of more than 25%. Each class was detected approximately 25,000 times, and no case was detected where all classes were equally likely. In all cases, the same input signal values were used:

$$\mathbf{X} = \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} 1.2 \\ 4.5 \\ 0 \\ 0 \end{vmatrix} \tag{9}$$

For all classes and all probabilities, all parameter values are equally likely, that is, there is no parameter value where the probability of detecting one of the 4 classes would be more pronounced.

5. Discussion

The disadvantage of using machine learning as black box is overcome by designing non-linear system using visual programming. During test or validation process, system parameters can be viewed for verification purposes. The final results as expressions in a closed form can be derived and optimized according the required task. The future development is looking for fully automated design based on required number of neurons in a layer as well as the number of layers. There is no similar approach as the best knowledge of the authors.

Supplementary Materials: All programs are available from the corresponding author as well as from the journal repository.

Author Contributions: Conceptualization, M.L.B. and N.Z.; methodology, M.L.B. and V.M.; software, M.L.B. and N.D.; validation, M.L.B., V.M. and N.D.; formal analysis, M.L.B. and I.F.; investigation, M.L.B. and I.F.; resources, M.L.B. and N.D.; data curation, M.L.B. and N.Z.; writing—original draft preparation, M.L.B. and N.D.; writing—review and editing, M.L.B. and N.Z.; visualization, N.Z. and I.F.; supervision, M.L.B., V.M. and I.F.; project administration, V.M.; funding acquisition, I.F., V.M., N.Z. and N.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All programs are available from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

We use the Wolfram language for programming. The code is in the form of readable text with the extension .nb. The first step is to set the working directory and import the knowledge as a package or text file with the extension .m:

```
mainDir = SetDirectory[NotebookDirectory[]];
Get["SchematicSolver`"]
<< SSknowledgeImp00`
```

The next few lines of the program tell how the scheme is visible in the images:

```
SetOptions[DrawElement, {BaseStyle -> {FontFamily -> "Times", FontSize -> 16}}];
plotStyleDD = {{Dashing[.01], GrayLevel[.5]}, {Dashing[.01], GrayLevel[.5]}};
SetOptions[DrawElement, {ElementSize -> {1, 1},
            PlotStyle -> {{RGBColor[0, 0, 0.7], Thickness[0.004],
            PointSize[0.01]}, {RGBColor[0, 0, 1], Thickness[0.0025`],
            PointSize[0.01]}}, ShowArrowTail -> True, ShowNodes -> True,
            TextOffset->       Automatic,BaseStyle->{FontFamily->"Times",FontSize-
     >16}}];
```

Input, output, and single-layer schemes are defined as separate lists:

```
artNeuronInp04 = {
   {"Input", {0, 0}, XX1, " ", TextOffset -> {1, 0}},
   {"Input", {0, 2}, XX2, " ", TextOffset -> {1, 0}},
   {"Input", {0, 4}, XX3, " ", TextOffset -> {1, 0}},
   {"Input", {0, 6}, XX4, " ", TextOffset -> {1, 0}}};
artNeuronOut04 = {
   {"Output", {13, 3}, YY1, " ", TextOffset -> {-1, 0}}};
artNeuronBody04 = {
   {"Adder", {{7, 3}, {5, 2}, {8, 3}, {5, 4}}, {1, 1, 2, 1}, " "},
   {"Adder", {{8, 3}, {5, 1}, {11, 3}, {5, 5}}, {1, 1, 2, 1}, "j"},
   {"Function", {{11, 3}, {13, 3}}, f, " "},
   {"Input", {6, 3}, 1, " ", TextOffset -> {1, 0}},
   {"Line", {{0, 0}, {3, 0}}}, {"Line", {{0, 2}, {3, 2}}},
   {"Line", {{0, 4}, {3, 4}}}, {"Line", {{0, 6}, {3, 6}}},
   {"Multiplier", {{3, 0}, {5, 1}}, W11, " ", ElementSize -> {1, 1}},
   {"Multiplier", {{3, 2}, {5, 2}}, W21, " ", ElementSize -> {1, 1}},
   {"Multiplier", {{3, 4}, {5, 4}}, W31, " ", ElementSize -> {1, 1}},
   {"Multiplier", {{3, 6}, {5, 5}}, W41, " ", ElementSize -> {1, 1}},
   {"Multiplier", {{6, 3}, {7, 3}}, b01, " "},
   {"Polyline", {{0, -1}, {13, -1}, {13, 7}, {0, 7}, {0, -1}}}};
```

Replacement rules are used for drawings (represented in bitmap format):

```
      subsSymbW = {W11 → w1, W21 → w2, W31 → w"...", W41 → wi};
      subsSymbB = {b01 → bj};
In[∘]:= subsSymbX = {XX1 → x1, XX2 → x2, XX3 → "...", XX4 → xi, YY1 → yj};
      mySchematic = artNeuronInp04~Join~artNeuronBody04~Join~artNeuronOut04;
      figure001 = Show[{DrawElement /@ (mySchematic /. subsSymbX /. subsSymbB /. subsSymbW)}, Frame → False]
```

Figure 1 was created with the previous code.

Appendix B

Figure 3(a) was created with the next code:

```
artNeuronInp04 = {
   {"Input", {0, 0}, XX1, " ", TextOffset -> {1, 0}},
   {"Input", {0, 2}, XX2, " ", TextOffset -> {1, 0}},
   {"Input", {0, 4}, XX3, " ", TextOffset -> {1, 0}},
   {"Input", {0, 6}, XX4, " ", TextOffset -> {1, 0}}};
artNeuronOut04 = {{"Output", {13, 3}, YY1, " ", TextOffset -> {-1, 0}}};
artNeuronBody04 = {
   {"Adder", {{7, 3}, {5, 2}, {8, 3}, {7, 5}}, {1, 1, 2, 1}, " "},
   {"Adder", {{8, 3}, {5, 0}, {11, 3}, {12, 5}}, {1, 1, 2, 1}, "j"},
   {"Function", {{11, 3}, {13, 3}}, f, " "},
   {"Line", {{0, 0}, {3, 0}}}, {"Line", {{0, 2}, {3, 2}}},
   {"Line", {{0, 4}, {3, 4}}}, {"Line", {{0, 6}, {3, 6}}},
   {"Line", {{5, 6}, {7, 5}}}, {"Line", {{5, 5}, {6, 8}}},
   {"Line", {{5, 4}, {7, 3}}}, {"Line", {{5, 3}, {8, 8}}},
   {"Line", {{5, 1}, {10, 8}}},
   {"Modulator", {{3, 0},{4,-1},{5, 0},{5, 1}},{1, 0, 2, 1}, W11,TextOffset ->
{1,-1}},
   {"Modulator", {{3, 2},{4,-1},{5, 2},{5, 3}},{1, 0, 2, 1}, W21,TextOffset ->
{1,-1}},
   {"Modulator", {{3, 4},{4,-1},{5, 4},{5, 5}},{1, 0, 2, 1}, W31,TextOffset ->
{1,-1}},
   {"Modulator", {{3, 6},{4, 5},{5, 6},{4, 8}},{1, 0, 2, 1}, W41,TextOffset ->
{1,-1}},
   {"Polyline", {{0, -2},{13, -2}, {13, 10}, {0,10}, {0, -2}}}};
```

Notice that we are using **`Modulator`** instead of **`Multiplier`**. The Delay elements are presented in bitmap format:

```
       artNeuronDelay04 = {
          {"Text", {6, -1}, "y_j = f (W_{1,n-1} x_1 + W_{2,n-1} x_2 + W_{...,n-1} x ... + W_{i,n-1} x_i + b_{j,n-1})",
           BaseStyle → {FontFamily → "Times", FontSize → 12}},
          {"Text", {12, 4.5}, b_{j,n-1}},
In[ ]:=   {"Delay", {{4, 10}, {4, 8}}, 1, " "},
          {"Delay", {{6, 10}, {6, 8}}, 1, " "},
          {"Delay", {{8, 10}, {8, 8}}, 1, " "},
          {"Delay", {{10, 10}, {10, 8}}, 1, " "},
          {"Delay", {{12, 10}, {12, 5}}, 1, " "}};
```

Figure 3(b) was created with the similar code.

Appendix C

Figure 4(b) was created with the next code:

```
artNeuronBody04 = {
  {"Adder", {{7, 3}, {5, 2}, {8, 3}, {5, 4}}, {1, 1, 2, 1}, " "},
  {"Adder", {{8, 3}, {5, 1}, {11, 3}, {5, 5}},{1, 1, 2, 1}, "1"},
  {"Adder", {{7, 10},{5, 9}, {8, 10},{5, 11}},{1, 1, 2, 1}, " "},
  {"Adder", {{8, 10},{5, 8},{11, 10},{5, 12}},{1, 1, 2, 1}, "2"},
  {"Adder", {{7, 17},{5, 16},{8, 17},{5, 18}},{1, 1, 2, 1}, " "},
  {"Adder", {{8, 17},{5, 15},{11,17},{5, 19}},{1, 1, 2, 1}, "3"},
  {"Adder", {{7, 24},{5, 23},{8, 24},{5, 25}},{1, 1, 2, 1}, " "},
  {"Adder", {{8, 24},{5, 22},{11,24},{5, 26}},{1, 1, 2, 1}, "4"},
  {"Function", {{11, 24}, {13, 24}}, f, " "},
  {"Function", {{11, 17}, {13, 17}}, f, " "},
  {"Function", {{11, 10}, {13, 10}}, f, " "},
  {"Function", {{11,  3}, {13,  3}}, f, " "},
  {"Input", {6, 24}, 1, " ", TextOffset -> {1, 0}},
  {"Input", {6, 17}, 1, " ", TextOffset -> {1, 0}},
  {"Input", {6, 10}, 1, " ", TextOffset -> {1, 0}},
  {"Input", {6,  3}, 1, " ", TextOffset -> {1, 0}},
  {"Line", {{0, 3}, {3, 0}}},  {"Line", {{0, 10}, {3,  2}}},
  {"Line", {{0, 3}, {3, 7}}},  {"Line", {{0,  3}, {3, 14}}},
  {"Line", {{0, 3}, {3, 21}}}, {"Line", {{0, 17}, {3,  4}}},
  {"Line", {{0, 10}, {3, 9}}}, {"Line", {{0, 10}, {3, 16}}},
  {"Line", {{0, 10}, {3, 23}}},{"Line", {{0, 17}, {3, 11}}},
  {"Line", {{0, 17}, {3, 18}}},{"Line", {{0, 17}, {3, 25}}},
  {"Line", {{0, 24}, {3, 27}}},{"Line", {{0, 24}, {3, 20}}},
  {"Line", {{0, 24}, {3, 13}}},{"Line", {{0, 24}, {3, 6}}},
  {"Multiplier", {{3, 0}, {5, 1}}, W11, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 2}, {5, 2}}, W21, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 4}, {5, 4}}, W31, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 6}, {5, 5}}, W41, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 7}, {5, 8}}, W12, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 14},{5,15}}, W13, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 21},{5,22}}, W14, " ", ElementSize -> {1, 1}},
  {"Multiplier", {{3, 9}, {5, 9}}, W22," "}, {"Multiplier", {{3, 11},{5,11}},
W32," "},
  {"Multiplier", {{3, 16},{5,16}}, W23," "}, {"Multiplier", {{3, 18},{5,18}},
W33," "},
  {"Multiplier", {{3, 13},{5,12}}, W42," "}, {"Multiplier", {{3, 20},{5,19}},
W43," "},
  {"Multiplier", {{3,23}, {5,23}}, W24," "}, {"Multiplier", {{3, 25},{5,25}},
W34," "},
  {"Multiplier", {{3, 27},{5,26}}, W44," "}, {"Multiplier", {{6, 3}, {7, 3}},
b01," "},
  {"Multiplier", {{6, 10},{7,10}}, b02," "}, {"Multiplier", {{6,17}, {7,17}},
b03," "},
  {"Multiplier", {{6, 24},{7,24}}, b04," "},
  {"Polyline", {{4.7, 0}, {10, 0}, {10, 6}, {4.7, 6}, {4.7, 0}}},
  {"Polyline", {{4.7, 7}, {10, 7}, {10,13}, {4.7,13}, {4.7, 7}}},
  {"Polyline", {{4.7,14}, {10,14}, {10,20}, {4.7,20}, {4.7,14}}},
  {"Polyline", {{4.7,21}, {10,21}, {10,27}, {4.7,27}, {4.7,21}}}};
```

The initial layer can be translated and copied 3 times using `Do` command:

```
numberOfStages = 3;
widthOfStages = 13;
systemBody = artNeuronBody04;
b1TOb1 = Thread[Flatten[{b01, b02, b03, b04}] -> Flatten[{b01, b02, b03, b04}]];
…
w1TOw1 = Thread[{W11,W12,W13,W14,W21,W22,W23,W24,W31,W32,W33,W34,W41,W42,W43,W44}
  ->
{W011,W012,W013,W014,W021,W022,W023,W024,W031,W032,W033,W034,W041,W042,W043,W044}
];
…
B1TObK = {b1TOb2, b1TOb3, b1TOb4};
W1TOWK = {w1TOw2, w1TOw3, w1TOw4};
f1TOfK = {f -> f, f -> f, f -> g};
f1TOfK = {f -> f, f -> g, f -> G};
Do[systemBody =
   Join[systemBody, (TranslateSchematic[artNeuronBody04,{k*widthOfStages,0}] /.
   B1TObK[[k]] /. W1TOWK[[k]] /. f1TOfK[[k]])];, {k, numberOfStages - 1}];
```

Each symbol can be presented in different mathematical notation.

$$\text{subsSymb2} = \{\text{W11} \to w_{11}, \text{W12} \to w_{12}, \text{W13} \to w_{13}, \text{W14} \to w_{14}, \text{W21} \to w_{21}, \text{W22} \to w_{22}, \text{W23} \to w_{23}, \text{W24} \to w_{24}, \text{W31} \to w_{31}, \text{W32} \to w_{32}, \text{W33} \to w_{33}, \text{W34} \to w_{34}, \text{W41} \to w_{41}, \text{W42} \to w_{42}, \text{W43} \to w_{43}, \text{W44} \to w_{44}\};$$

$$\text{subsSymb3} = \{\text{b01} \to b_{01}, \text{b02} \to b_{02}, \text{b03} \to b_{03}, \text{b04} \to b_{04}, \text{b11} \to b_{11}, \text{b12} \to b_{12}, \text{b13} \to b_{13}, \text{b14} \to b_{14}, \text{b21} \to b_{21}, \text{b22} \to b_{22}, \text{b23} \to b_{23}, \text{b24} \to b_{24}\};$$

For the previous code we are using bitmap presentation.

Appendix D

The closed-form outputs are derived using the next code:

```
DiscreteSystemImplementation2[mySchematic, ToString[implementationProcedure]];
eqns = DiscreteSystemImplementationEquations2[mySchematic,Verbose -> False];
inputSequence = {{XX1, XX2, XX3, XX4,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}}
initialConditions = 0*eqns[[2]];
systemParameters = eqns[[3]];
{outputSequence,finalConditions} =
   DiscreteSystemImplementationProcessing2[inputSequence,
      initialConditions, systemParameters, procedureName];
y1st = outputSequence[[1]];
y1stIND = y1st /. subsSymb1 /. w1TOw1 /. subsALL1;
% // StandardForm
```

Figure 7(b) is obtained using `ExpressionGraph` command.

References

1. Bernard, E. *Introduction to Machine Learning*; Wolfram Media: Champaign, IL USA, 2022; pp. 1-424.
2. Mukhamediev, R.; Popova, Y.; Kuchin, Y.; Zaitseva, E.; et al. Review of artificial intelligence and machine learning technologies: classification, restrictions, opportunities and challenges. *Mathematics* **2022**, 10(15), #2552, 1-25.
3. Juraev, D.; Noeiaghdam, S. Modern problems of mathematical physics and their applications. *Axioms* **2022**, 11(2), 1-6.
4. He, Y. H. Machine-learning mathematical structures. *IntJDataS.Math.S* **2023**, 1(01), 23-47.

5. Rajebi, S.; Pedrammehr, S.; Mohajerpoor, R. A license plate recognition system with robustness against adverse environmental conditions using Hopfield's neural network. *Axioms* **2023**, 12(5), 1-12.
6. Ren, Y.M.; Alhajeri, M.S.; Luo, J.; Chen, S.; Abdullah, F.; Wu, Z.; Christofides, P. D. A tutorial review of neural network modeling approaches for model predictive control. *Comp.Chem.Eng.* **2022**, 165, 1-71.
7. Gnjatović, M.; Maček, N.; Saračević, M.; Adamović, S.; Joksimović, D.; Karabašević, D. Cognitively economical heuristic for multiple sequence alignment under uncertainties. *Axioms* **2022**, 12(1), 1-15.
8. Liu, M.; Cai, Z. Adaptive two-layer ReLU neural network: II. Ritz approximation to elliptic PDEs. *Comp.Math.App.* **2023**, 113, 103-116.
9. Cabello, J.G. Mathematical neural networks. *Axioms* **2022**, 11(2), 1-18.
10. Refonaa, J.; Huy, D.T.N.; Trung, N.D.; Van Thuc, H.; Raj, R.; Haq, M.A.; Kumar, A. Probabilistic methods and neural networks in structural engineering. *Int.J.Adv.Manuf.Techn.* **2022**, 125(3-4), 1-9.
11. Jegelka, S. Theory of graph neural networks: representation and learning. In Proceedings of the Int. Congress of Mathematicians, (virtual event, originally planned Saint Petersburg, Russia), Helsinki, Finland, 6-14 July 2022.
12. Kamienny, P.A.; d'Ascoli, S.; Lample, G.; Charton, F. End-to-end symbolic regression with transformers. *Adv.Neural Inf.Proc.Syst.* **2022**, 35, 10269-10281.
13. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; The MIT Press: Cambridge, MA USA, 2022; pp. 1–800.
14. Kim, J.M.; Kim, J.; Ha, I.D. Application of deep learning and neural network to speeding ticket and insurance claim count data. *Axioms* **2022**, 11(6), 1-185.
15. Ceccon, F.; Jalving, J.; Haddad, J.; Thebelt, A.; et al. OMLT: optimization & machine learning toolkit. *J.Mach.Learn.Research.* **2022**, 23(1), 15829–15836.
16. Xie, Y.; Takikawa, `T.; Saito, S.; Litany, O.; et al. Neural fields in visual computing and beyond. *STAR.* **2022**, 41(2), 1–36.
17. Wolfram, S. *An Elementary Introduction to the Wolfram Language*, 3rd ed.; Wolfram Media: Champaign, IL USA, 2023; pp. 1–376.
18. SchematicSolver. Available online: https://www.wolfram.com/products/applications/schematicsolver (accessed on 17 10 2023).
19. Lutovac-Banduka, M.; Milosevic, D.; Cen, Y.; Kar, A.; Mladenovic, V. Graphical user interface for design, analysis, validation, and reporting of continuous-time systems using Wolfram language. *JCSC* **2023**, *32*, # 2350244, 1-14.
20. Setzu, M.; Guidotti, R.; Monreale, A.; Turini, F.; et al. GLocalX – from local to global explanations of black box AI models. *Artificial Intelligence.* **2021**, 294(103457), 1–15.
21. Ghelani, D. Explainable AI: approaches to make machine learning models more transparent and understandable for humans. *IJCST.* **2022**, 6(4), 45–53.
22. Swamy, V.; Radmehr, B.; Krco, N.; Marras, M.; Käser, T. Evaluating the explainers: black-box explainable machine learning for student success prediction in MOOCs. In Proceedings of the 15th International Conference on Educational Data Mining, Durham, England, 24-27 July 2022.
23. Srinivasu, P. N.; Sandhya, N.; Jhaveri, R. H.; Raut, R. From black-box to explainable AI in healthcare: existing tools and case studies. *Mob.Inf.Syst.* **2022**, 2022, 1–20.
24. Xie, Y.; Zaccagna, F.; Rundo, L.; Testa, C.; Agati, R.; et al. Convolutional neural network techniques for brain tumor classification (from 2015 to 2022): review, challenges, and future perspectives. *Diagnostics.* **2022**, 12(8), 1-46.
25. Simović, A.; Lutovac-Banduka, M.; Lekić, S.; Kuleto, V. Smart visualization of medical images as a tool in the function of education in neuroradiology. *Diagnostics* **2022**, *12*, 3208, 1–19.
26. Vuong, Q. Machine Learning for Robotic Manipulation. *arXiv e-prints.* **2021**, arXiv-2101.00755, 1-15.
27. Lutovac Banduka, M. Robotics first – a mobile environment for robotics education. *Int.J.Eng.Edu.* **2016**, 32(2), 818–829.
28. Lăzăroiu, G., Andronie, M., Iatagan, M., Geamănu, M., Ștefănescu, R., & Dijmărescu, I. Review deep learning-assisted smart process planning, robotic wireless sensor networks, and geospatial big data management algorithms in the internet of manufacturing things. *IoMT.* **2022**, 11(5), 1-26.
29. Lutovac-Banduka, M.; Kvrgić, V.; Ferenc, G.; Dimić, Z.; Vidaković, J. 3D simulator for human centrifuge motion testing and verification, In Proceedings of Mediterranean Conference on Embedded Computing, Budva, Montenegro, 15-20 June 2013.
30. Liu, S.C.; Strachan, J.P.; Basu, A. Prospects for analog circuits in deep networks. In *Analog Circuits for Machine Learning, Current/Voltage/Temperature Sensors, and High-speed Communication Advances in Analog Circuit Design*, Harpe, P., Makinwa, K., Baschirotto, Eds.; Springer: Heidelberg, Germany, 2022; Volume 1, pp. 49-61.
31. Rojec, Ž.; Fajfar, I.; Burmen, Á. Evolutionary synthesis of failure-resilient analog circuits. *Mathematics* **2022**, 10(1), #156, 1-20.

32. Lutovac-Banduka, M.; Simović, A.; Orlić, V.; Stevanović, A. Dissipation minimization of two-stage amplifier using deep learning. *SJEE* **2023**, *20*, 129–145.
33. Sahu, A.; Behera, S. VLSI techniques used in AI. *Dogo Rangsang Res.J.* **2019**, 9(3), 965–967.