BULGARIAN ACADEMY OF SCIENCES
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS • 2023 • Vol. **80**, pp. 13-22
p-ISSN: 2738-7356; e-ISSN: 2738-7364
https://doi.org/10.7546/PECR.80.23.02

Multilayer Perceptron with Backpropagation, HDL Coder, and FPGA Technology: An Integrated Approach for Efficient Neural Network Implementation

Krasimir Markov

Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Acad. Georgi Bonchev Str., bl. 2, 1113 Sofia, Bulgaria
E-mail: krasikasi@abv.bg

Abstract: Multilayer perceptron (MLP) and the backpropagation method are two fundamental components in the field of artificial neural networks (ANNs) that have gained significant attention and applications in various areas of computer science and machine learning. This article provides an overview of MLP and the backpropagation method, discussing their underlying principles and functioning. Initially, we provide a general overview of MLP and its architecture. MLP is a form of feedforward neural networks that includes one or more hidden layers between the input and output layers. Each neuron computation in MLP is performed in a forward pass, applying a nonlinear activation function to the weighted sum of inputs and neuron activations from the previous layer. Next, the backpropagation method, which is used for training MLP, is discussed. This method primarily relies on minimizing the error between the predicted and true outputs of the network using gradient descent. The error gradient is propagated backward through the network, updating the weights of each connection, contributing to its effectiveness and learning capability. Various alternative variations and enhancements of MLP and the backpropagation method are then explored, including the use of different activation functions, regularization, architectural modifications, and optimization methods. Significant research has been conducted in recent years to develop more efficient and powerful MLP models. In conclusion, we summarize the importance and applications of MLP and the backpropagation method. MLP and backpropagation are widely used tools for tasks such as classification, regression, function approximation, and others, and they continue to be the subject of active research and development in the field of machine learning and neural networks.

Keywords: *MultiLayer Perceptron, Backpropagation, Artificial Neural Networks, Machine Learning, MATLAB*.

1. Introduction

The MultiLayer Perceptron (MLP) and the Backpropagation method are fundamental concepts in the field of the Artificial Neural Networks (ANNs). MLP is a type of ANN architecture that utilizes multiple input and hidden layers for information processing. It is particularly useful in the areas of data classification and regression.

Backpropagation is a training method for MLPs that employs the Stochastic Gradient Descent (SGD) algorithm to adjust the weights of the connections between neurons. This method utilizes the chain rule to compute gradients of the error function with respect to the network's weights. This allows the MLP to adapt and improve over time.

The Hardware Description Language Coder (HDL) is a tool provided by MathWorks that enables the automatic generation of hardware description language code from MATLAB or Simulink models. This is especially useful for designing digital systems on programmable logic integrated circuits (FPGAs). The Field Programmable Gate Arrays (FPGA) module with AMD Artix™ 7 50T-2I, TE0714-04-52I-7-B represents a specific hardware module based on FPGA technology. It utilizes an FPGA chip from the Artix™ 7 series by AMD and offers various capabilities for developing digital systems. This module can be used to implement MLPs or other digital systems generated by HDL Coder.

2. Structure of a Multilayer Perceptron

An example structure of a multilayer perceptron (MLP) is shown on Fig. 1.

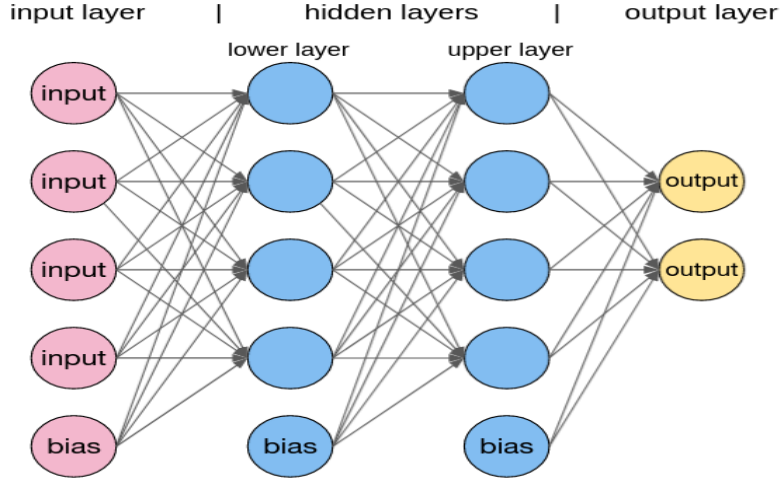

Fig. 1. Multilayer perceptron

The multilayer perceptron (MLP) consists of an input layer, hidden layers, and an output layer, [4, 5, 6]. The input layer receives the input data and passes the signals to the hidden layers. The hidden layers process the information by passing through an activation function. The output layer generates predicted outputs based on the information from the hidden layers. Each connection between neurons has associated weights that are updated during training.

In the backpropagation algorithm for training neural networks, activation functions play a crucial role. They define the output of neurons in each layer of the network based on their input. In the following lines, I will explain how different types of activation functions used in the backpropagation algorithm work.

Activation Functions:

- *Sigmoid Function:* The sigmoid function has an S-shaped curve and is commonly used as an activation function in neural networks. The mathematical formula of the sigmoid function is: $f(x) = 1 / (1 + exp(-x))$. The output of a neuron with a sigmoid activation function is in the range (0, 1), making it suitable for tasks like binary classification.
- *Rectified Linear Unit (ReLU):* ReLU is a simple and effective activation function that returns x for positive values and 0 for negative values. Mathematically, the ReLU function is $f(x) = max (0, x)$. ReLU neurons are computationally efficient and help alleviate the vanishing gradient problem.
- *Tanh Function:* The tanh function is similar to the sigmoid function but has an output range of (-1, 1). Mathematically, the tanh function is $f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))$. The tanh function is continuous and differentiable, making it suitable for training neural networks.
- *Softmax Function:* The Softmax function is used in the last layer of neural networks for multi-class classification. The Softmax function takes a vector of neuron outputs and generates probabilities for the classes. Mathematically, the Softmax function for an input vector x is defined as $f(x) = exp(x\_i) / sum (exp(x\_j))$, where i is the index of the class, and j iterates over all classes.
- *Linear Activation Function:* The linear activation function performs simple scaling operations on the input data. It returns an output that is linearly proportional to the sum of the input data. Mathematically, the linear activation function can be expressed as $f(x) = wx + b$, where $f(x)$ is the output of the activation function, x is the input or input data, w is the weight or scaling coefficient of the input, and b is the bias or function offset. The linear activation function lacks nonlinearity or changes in the output value. This means that the function performs only linear scaling

operations on the input. So, if the input is multiplied by a constant or a constant is added, the output is proportionally increased or shifted.

It is important to note that the linear activation function is not suitable for neural networks with more than one hidden layer. This is because the combination of multiple linear functions remains linear and cannot model complex nonlinear relationships.

These are just some of the most popular activation functions used in the backpropagation algorithm. The choice of activation function depends on the nature of the problem being solved and the specific requirements of the network architecture. It is important to select an appropriate activation function that will ensure stable and effective training of the neural network.

3. Backpropagation Algorithm for MLP Training

The backpropagation algorithm is a training algorithm for MLP (multilayer perceptron), [5, 6]. It consists of two phases: forward propagation and backward propagation. In forward propagation, the input data is passed through the network, and predictions are generated. Then, the error between the predictions and the desired outputs is calculated. In backward propagation, the error is propagated back through the network, and gradients of the error with respect to the weights are computed. These gradients are used to update the weights using an optimization method such as stochastic gradient descent.

The process of forward and backward propagation is repeated for multiple iterations until the desired level of accuracy is achieved. The backpropagation algorithm is one of the most popular algorithms for training neural networks. It is used to adjust the weights of the connections in the network, minimizing the error between the predicted outputs of the network and the desired outputs.

The basic steps in the backpropagation algorithm are as follows:

- *Weight Initialization:* The initial weights of the connections in the network are initialized to random values. The weights determine the strength of the connections between neurons and their influence on the predicted outputs of the network.
- *Forward Propagation:* The input data is passed through the network, and the outputs of each neuron in all layers are computed. This is done sequentially from the input layer to the output layer.
- *Error Calculation:* After obtaining the predicted outputs from the network, the error between the predicted outputs and the desired outputs is calculated. The error can be measured using different error functions, such as mean squared error (MSE).
- *Backward Propagation:* The error is propagated back through the network, and the error for each neuron in each layer is computed. This is

done using the chain rule from differential calculus, which allows the calculation of the gradient of the error with respect to the weights.

- *Weight Update:* After computing the gradients of the error with respect to the weights, the weights of the connections are updated in a direction that reduces the error. This is done using optimization methods such as stochastic gradient descent or its variations.
- *Iteration:* The above steps are repeated multiple times (called epochs) for the entire training dataset. During the iterations, the weights of the connections are updated, reducing the error, and the network adapts to the provided data.

These are the basic steps in the backpropagation algorithm. It allows the neural network to learn the representations and interconnections in the data, leading to achieving the desired outputs when predicting or classifying new data.

4. Data Applications of Multilayer Perceptron

The Multilayer Perceptrons (MLPs) have wide applications in various fields, including computer vision, speech recognition, natural language processing, classification and prediction, and signal processing [14-18].

Feedforward neural networks, such as MLPs, can be used for noise filtering in amplitude-modulated signals. Noise filtering is a common task in signal processing, and neural networks can be a powerful tool for performing this task. One possible architecture of a neural network for noise filtering is shown on Fig. 2 and may include the following layers:

1. *Input Layer*: The input data could be the amplitude-modulated signal, including the noise you want to filter.
2. *Hidden Layers:* After the input layer, you may have one or more hidden layers. These layers can learn complex dependencies and correlations between the signal and the noise, allowing the network to separate the identification of the noise from the desired signal.
3. *Output Layer:* The output layer of the network represents the filtered signal without noise. This layer generates the results of the signal processing by the network.

The process of training the network involves preparing a training set that contains the amplitude-modulated signal and the corresponding noise. The network is trained to understand the relationship between this data and generate the filtered signal by minimizing the difference between the filtered signal and the desired outcome.

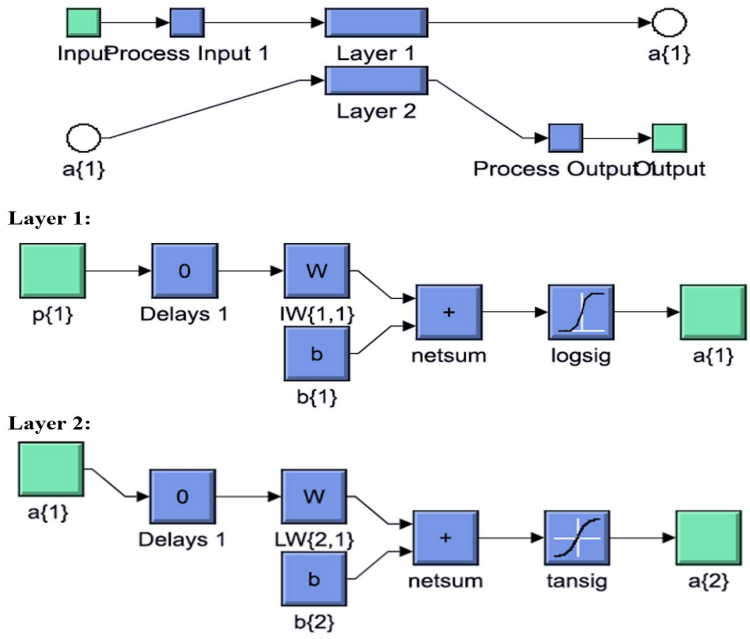

Fig. 2. Neural Network Architecture

5. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization algorithm used for training neural networks and other machine learning models, [6, 7]. It is a variation of gradient descent, which is used to find the minimum of the error function.

The main idea behind SGD is to use a random subset of training data (a single instance or a small batch) to compute the gradients of the error function and update the model parameters. This allows for efficient parameter updates at each step of the training process.

Here are the basic steps of the stochastic gradient descent algorithm:

1. Parameter initialization: Start with random or predefined values for the model parameters.
2. Random sample selection (or small batch): Choose a random instance or a small batch of instances from the training set (also known as a "minibatch").
3. Pass through the model: Feed the selected samples through the model to obtain predictions.
4. Compute gradients: After obtaining the predictions, compute the gradients of the error function with respect to the model parameters. This is done using the chain rule to calculate partial derivatives.
5. Parameter update: Once the gradients are computed, update the model parameters using a formula similar to the one used in gradient descent: $new\ parameter = old\ parameter - learning\_rate * gradient$.

6. Repeat the process: Repeat the above steps multiple times for all instances in the training set or for a certain number of epochs until the optimal solution is reached.

The advantages of stochastic gradient descent include faster training and lower memory requirements compared to full gradient descent. It is particularly useful for large datasets and when training needs to be performed in real-time.

6. Chain Rule

The chain rule is a fundamental rule in differential calculus used for calculating the derivative of a composition of functions, [8]. If we have two functions, for example, $f(x)$ and $g(x)$, and their composition, $h(x) = f(g(x))$, the chain rule allows us to calculate the derivative of h(x) with respect to x in terms of the derivatives of f and g.

Formally, if $y = f(u)$ and $u = g(x)$, then according to the chain rule, the derivative of y with respect to x (dy/dx) is expressed as: $dy/dx = dy/du * du/dx$. In other words, the derivative of y with respect to x is calculated as the product of the derivative of y with respect to u and the derivative of u with respect to x. The chain rule is of crucial importance in backpropagation in neural networks as it allows us to compute the gradient of the error with respect to the weights in the network. Therefore, we can update the weights using the gradient obtained using the chain rule.

7. Synthesis and Implementation

The HDL code generated by HDL Coder can be used in a synthesis and implementation environment, such as Xilinx Vivado or Intel Quartus, [1, 2, 3, 9]. This process involves compiling the VHDL code, creating networks of logic elements, specifying timing constraints, and generating the physical configuration files (bitstreams) for programming the digital device, such as an FPGA, shown on Fig. 3 [10-13].

Fig. 3. Main board

1 step. MATLAB script

```
% Generating data for a neural network
input_data = noisy_signal; % Noisy amplitude-modulated signal
target_data = modulated_signal;% Clean amplitude-modulated signal
(message)
% Creating a neural network
hidden_units = 100; % Number of hidden neurons
 net = feedforwardnet(hidden_units);
net.layers{1}.transferFcn = 'logsig'; % Activation function for
the first hidden
layer net.layers{2}.transferFcn = 'tansig'; % Activation function
for the second hidden layer
% Setting training parameters
net.trainParam.epochs = 100; % Number of epochs net.trainParam.
lr = 0.01; % Learning rate net.trainParam.
mu = 0.5; % Value of mu % Training the neural network
net = train(net, input_data, target_data);
```

2 step. Transfer MATLAB to VHDL.

3 step. Bitstreams file. To create bitstreams for the code provided above, you will need an installation of a synthesis and implementation environment such as Xilinx Vivado or Intel Quartus. Here is an example of creating bitstreams using Xilinx Vivado:

- Launch Xilinx Vivado and create a new project.
- Select a directory for the project and give it a name.
- When adding files to the project, include the HDL code that you wish to synthesize and implement.
- Choose the target platform (FPGA) and configure the relevant settings for the target device.
- Perform synthesis on the project. This will convert the HDL code into an internal representation of the logic elements on the target device.
- Verify the synthesis results and ensure there are no errors or warnings.
- Generate the bitstream file by executing the implementation and configuration process.
- As a result, you will obtain a bitstream file that can be used to program the digital device (e.g., FPGA) and execute the desired functionality.

4 step. FPGA module with AMD Artix™ 7 50T-2I, TE0714-04-52I-7-B by Trenz Electronic is an industrial FPGA module that integrates AMD/Xilinx Artix™ 7 50T, 16 MByte QSPI Flash memory for configuration and operation, as well as powerful switching power supplies for all board voltages. The module features a large number of configurable input/output ports through robust high-speed connectors. All of this in a very small form factor, smaller than half a credit card,

and at a highly competitive price. All components are rated for an industrial temperature range of -40°C to +85°C.

Specifications:

- Rugged for high shock and vibration
- 16 MB QSPI Flash memory
- Differential MEMS oscillator for MGT clocking
- MEMS oscillator for PL clocks (optional)
- Plug-on module with 2 × 100-pin high-speed hermaphroditic strips
    - 138 FPGA I/O's (max 68 differential) +5 I/O's (QSPI or user I/O's)
    - XADC Analog Input
    - 4 GTP (high-performance transceiver) lanes
    - GT reference clock inputs
    - Optimized I/O and power pins for good signal integrity
- On-board high-efficiency DC/DC converters
- Power supply for all on-board components
- eFUSE bit-stream encryption (AES)
- One user configurable LED
- Size 3 cm x 4 cm

8. Conclusion

The Multilayer perceptron and the backpropagation method are powerful tools in the field of artificial neural networks. They allow us to model complex relationships and perform classification and regression on data. By employing the gradient descent algorithm and the chain rule, we can train MLPs and adapt them to specific problems. HDL Coder and FPGA modules like the AMD Artix™ 7 50T-2I, TE0714-04-52I-7-B provide the means to transform MLPs and other digital systems into real hardware solutions. This enables fast execution and efficient utilization of FPGA resources.

The combination of multilayer perceptron, backpropagation, SGD, chain rule, HDL Coder, and FPGA modules offers a wide range of tools and possibilities for developing complex digital systems and solving various tasks in the field of machine learning and data processing.

Acknowledgments

This work was supported by the Bulgarian Ministry of Education and Science under the National Research Programme „Smart crop production”, approved by Decision of the Ministry Council No 866/26.11.2020 (No D01-65/19.03.2021).

References

1. MATLAB – HDL CODER-User's Guide. Retrieved from www.mathworks.com.

2. MATLAB – Signal Processing Toolbox-User's Guide. Retrieved from www.mathworks.com.
3. MATLAB – Neural Network Toolbox -User's Guide. Retrieved from www.mathworks.com.
4. Lagnese, M. C.: Neural Networks with MATLAB. Pearson, (2003).
5. Hu, Y. H., Hwang, J. N.: Neural Network Signal Processing. CRC Press., (1999).
6. Ian Goodfellow, Yoshua Bengio, Aaron Courville: Deep Learning, MIT Press., (2016).
7. Jorge Nocedal, Stephen J. Wright: Numerical Optimization, Springer, (2006).
8. Riley K. F., Hobson, M. P., Bence S. J.: Mathematical Methods for Physics and Engineering, University of Cambridge, (2006).
9. Cem Unsalan, Bora Tar: Digital System Design with FPGA: Implementation Using Verilog and VHDL, McGraw-Hill Education, (2017).
10. Kyovtorov, V., Kabakchiev, C., Behar, V., Kuzmanov, G., Garvanov, I., Doukovska, L.: FPGA implementation of low-frequency GPR signal algorithm using frequency stepped chirp signals in the time domain. In: Proc. of International Radar Symposium – IRS'08, Wroclaw, Poland, pp. 297-300, (2008).
11. Kyovtorov, V., Kabakchiev, C., Garvanov, I., Doukovska, L., Behar, V.: FPGA implementation of FSCS GPR signal algorithm. NATO Advanced Study Institute "Unexploded Ordnance Detection and Mitigation", Il Ciocco, Italy, CD Proc., (2008).
12. Kabakchiev, C., Doukovska, L., Garvanov, I.: Comparative losses analysis of constant false alarm rate processors in conditions of pulse jamming. IIT-BAS Working Papers, 111, (2000).
13. Kabakchiev, C., Doukovska, L., Garvanov, I.: Comparative analysis of losses of CA CFAR processors in pulse jamming. Cybernetics and Information Technologies, 1(1), 21-35, (2001).
14. Toskova, A., Toskov, B., Doukovska, L., Daskalov, B., Radeva, I.: Neural networks in the intelligent educational space. In: Proc. of IEEE International Workshop on Advances in Neural Networks and Applications – ANNA 2018, Berlin, 107-112, (2018).
15. Shahpazov, V., Velev, V., Doukovska, L: Design and application of artificial neural networks for predicting the values of indexes on the Bulgarian Stock market. In: Proc. of Signal Processing Symposium – SPS'13, Jachranka Village, Poland, (2013).
16. Shahpazov, V., Doukovska, L.: Forecasting financial markets with artificial intelligence. In: Proc. of International Workshop on Advanced Control and Optimisation: Step Ahead – ACOSA'14, Bankya, Bulgaria, 67-74, (2014).
17. Terziyska, M., Doukovska, L.: Semi fuzzy neural networks, Part 1: Nonlinear system identification. In: Proc. of International Workshop on Advanced Control and Optimisation: Step Ahead – ACOSA'14, Bankya, Bulgaria, 18-23, (2014).
18. Terziyska, M., Doukovska, L.: Semi fuzzy neural networks, Part 2: Predictive control. In: Proc. of International Workshop on Advanced Control and Optimisation: Step Ahead – ACOSA'14, Bankya, Bulgaria, 24-28, (2014).