MATLAB IMPLEMENTATION OF MULTILAYER PERCEPTRON FOR BEARING FAULTS CLASSIFICATION

Martin Doseděl
Doctoral Degree Programme (3), FEEC BUT
E-mail: xdosed04@stud.feec.vutbr.cz

Supervised by: Zdeněk Havránek
E-mail: havranek@feec.vutbr.cz

Abstract: This paper deals with implementation of multilayer perceptron neural network (NN) for bearing faults classification. Neural network has been created from scratch as an M-script with back propagation learning algorithm also, but without using advanced MATLAB packages. Public available bearing dataset from Case Western Reserve University has been used for both training and testing phase, as well as for the final classification process. Problem with sparse input data for training the network has also been addressed. This relatively simple and small neural network is capable to classify the failures of a bearing with very low error rate.

Keywords: Multilayer perceptron (MLP), deep learning, data classification, back-propagation algorithm, bearing faults

1 INTRODUCTION

Artificial intelligence (AI) is nowadays very expanding area of interest for most of the researchers as well as technicians in the industrial area. Connection of AI and technical diagnostics, especially in the predictive maintenance of machines in the industry [4], is very common and popular topic also thanks to Industry 4.0 and Internet of Things (IoT) [1]. E.g. Yin et al. [8] shows an overview of current data-based techniques for modern industrial applications with big data processing. Since the machine learning methods contain mostly statistical methods [10] like SVM, k-NN, PCA, Mahalanobis-Taguchi strategy etc., strong and powerful neural networks are trained for classification of the vibration signals from the time-domain data, such as very popular convolution neural network (CNN, e.g. [7]) and performed on dedicated powerful hardware or graphics cards. Approach presented in this article uses raw signal preprocessing. A key signal features, extracted from the time domain are used as an input of a simple, small-size, but efficient neural network of the MLP type. Aim of this work is to implement the network completely from scratch. Reason for this solution, even there exist procedures for more simple implementation, mainly using object-oriented programming ([2, 6]), was to understand the internal processes inside the network.

1.1 MULTILAYER PERCEPTRON

The idea of a technical perceptron comes from the similarity with the human’s brain and neurons. Single perceptron, which is a base of the network depicted in this work, is shown in Fig. 1.

Formula for output signal considering a bias as the one input is in (1).

\[ y = f \left( \sum_{i=1}^{N} w_i x_i \right) \]  

(1)

where \( w_i \) are the individual inputs weights \( w_1, w_2, ..., w_N \), \( x_i \) are the individual inputs of perceptron \( x_1, x_2, ..., x_N \) and \( f(\cdot) \) is an activation function. There is also a bias expressed in the equation (1) as
the member $x_1$ and its appropriate weight $w_1$. It is good to mention, that in some literature, bias is written outside of the sum as a separate member, usually noted as $\theta$. There exists a lot of activation functions [3], e.g. ReLU, binary step, linear function, TanH, Leaky ReLU, Softmax or sigmoid. Last mentioned, sigmoid function, can be expressed by formula (2)

$$f(x) = \frac{1}{1 + e^{-kx}} \quad (2)$$

where $x$ is an input and $k$ is a scale factor.

MLP has been used in this work as a neural network classifier. The topology is shown in Fig. 2 - one input layer (with eight inputs), one hidden layer (with sixteen neurons) and one output layer (with five outputs representing five output classes).

Since the first (input) layer does not do any computations (serves only for input signals propagation to other layers), the output of hidden layer can be simply expressed (using formula (1)) in a matrix form as:

$$x^{(1)} = a^{(0)} \cdot w^{(0)} \quad (3)$$

and after applying the sigmoid activation function

$$a^{(1)} = \frac{1}{1 + e^{-x^{(1)}}} \quad (4)$$

where $a^{(1)}$ is the vector of outputs from the hidden layer (or inputs to the output layer). Full matrix equation for hidden layer output can be written in the following form (see formula 5).
\[
\begin{bmatrix}
a_0^{(1)} \\
a_1^{(1)} \\
\vdots \\
a_M^{(1)}
\end{bmatrix} =
\begin{bmatrix}
a_0^{(0)} \\
a_1^{(0)} \\
\vdots \\
a_N^{(0)}
\end{bmatrix}^T \cdot 
\begin{bmatrix}
w_{0,0}^{(0)} & w_{0,1}^{(0)} & \cdots & w_{0,M}^{(0)} \\
w_{1,0}^{(0)} & w_{1,1}^{(0)} & \cdots & w_{1,M}^{(0)} \\
\vdots & \vdots & \ddots & \vdots \\
w_{N,0}^{(0)} & w_{N,1}^{(0)} & \cdots & w_{N,M}^{(0)}
\end{bmatrix}
\]  

(5)

The same approach can be applied for the output layer signals \(a^{(2)}\) equation. It is good to note here, that the matrix form of the equation is easy implementable in MATLAB, since it is the matrix-based software.

1.2 BACK PROPAGATION ALGORITHM

To train the network (to minimize the error rate during classification) it is necessary to modify the weights for each neuron’s connection. The weights should be modified according to the equation (6)

\[ w_{j}^{(0)}(t + 1) = w_{j}^{(0)}(t) + \Delta w_{j}^{(0)} \]  

(6)

Back-propagation algorithm is very suitable for this procedure (see e.g. [3]). The goal is to minimize the error, expressed by the following equation (7) as a sum of substracts of calculated (predicted) output values \(a_{j}^{(2)}\) and desired (true) \(d_{j}\) values:

\[ E = \frac{1}{2} \sum_j (a_{j}^{(2)} - d_{j})^2 \]  

(7)

It can be derived, that for back propagation algorithm based on gradient descend method [9] for modification of the weights between hidden and output layer it can be written:

\[ \Delta w_{j}^{(0)} = \frac{\partial E}{\partial w_{j}^{(0)}} = \frac{\partial E_{C}}{\partial a_{j}^{(2)}} \cdot \frac{\partial a_{j}^{(2)}}{\partial z_{k}} \cdot \frac{\partial z_{k}}{\partial w_{j}^{(0)}} = \ldots = (a_{j}^{(2)} - d_{j}) \cdot a_{j}^{(2)}(1 - a_{j}^{(2)}) \cdot a_{j}^{(0)} \cdot \alpha \]  

(8)

where \(a_{j}^{(2)}\) is the output of the NN, \(a_{j}^{(0)}\) is the input of NN, \(d_{j}\) is desired output and \(\alpha\) is a learning rate.

2 EXPERIMENT DESCRIPTION

2.1 INPUT DATA

As an input dataset, Case Reserve Western University [5] bearing dataset has been used. Data for damaged bearing has been used for both, the training (90 % of the total data number) and verification (remaining 10 % of complete dataset) phase. Data represents five output categories – normal state and four degrees of the bearing’s outer ring fault. Since the dataset was relatively sparse for sufficient classification accuracy (only 100 values for each category), new data has been created by adding Gaussian noise to the original data. Number of the input data (10 times higher) was then sufficient. Vibration values for the same bearing, but acquired within 3x higher load, have been used for validation of the NN.

From this extended data, a set of features has been calculated. Eight time-domain features were used – RMS value, kurtosis, skewness, variance, standard deviation, mean value and min and max value.
2.2 Neural network parameters

As above mentioned, MLP NN has been used for this classification task. Network has 8 inputs (for eight data features), one hidden layer with 16 neurons and one output layer containing 5 neurons (each neuron for one class). Sigmoid activation function is used and a learning rate $\alpha = 0.25$ was used. The network as well as back-propagation algorithm was implemented as a pure MATLAB m-file code.

2.3 Sparse input data

For achieve the best performance of the NN while using sparse input data, several measures have been done and implemented:

1. Increased number of input data by adding random values in the range of $\pm 5\%$ to the original signal. This significantly improved the number of epochs (10 times).

2. Full randomization of the input training patterns - a special algorithm implementation, which ensures that no input data from the same output class will be applied to the NN’s input after each other.

3. Initialization of the weights to the value of 0.01 with additional Gaussian noise with the SNR value of 20 dB.

Using these features, the accuracy of NN classification process significantly increased of about 20 %.

3 RESULTS

Minimum square error curve for training process of the resulting NN can be seen in Fig. 3 on the left side. As it can be seen, the error still does not reach its global minimum value – a trend of the curve is still decreasing. This shows to the fact, that the number of training epochs should be higher to reach higher accuracy. Learning ratio was empirically set to the optimal value of $\alpha = 0.25$. Accuracy of

![Figure 3: Mean square error during the training process (on the left) and confusion matrix of validation process (on the right).](image)

the trained network for classification is expressed by the confusion matrix (see Fig. 3 on the right). Total accuracy was calculated to 99.5 %, what is an excellent value considering the size of a network and a size of the training group.
4 CONCLUSION

In this paper, simple MLP for bearing faults classification has been implemented in MATLAB environment. Network has been trained using sparse input data and the accuracy has been verified on the different dataset. Overall accuracy of 99.5% is also a great result considering the fact, that training and validation processes take only 0.35 sec. or 0.05 sec. respectively. This time is with sure strongly dependent on the target hardware.

ACKNOWLEDGEMENT

The completion of this paper was made possible by the grant No. FEKT-S-20-6205 - “Research in Automation, Cybernetics and Artificial Intelligence within Industry 4.0” financially supported by the Internal science fund of Brno University of Technology.

REFERENCES


URL https://linkinghub.elsevier.com/retrieve/pii/S0888327018300748

URL http://csegroups.case.edu/bearingdatacenter


