Finding input-output dependencies of feed forward neural networks

Vladimír Stenchlák[0000-0001-9993-7554], *Ivan* Kuric[0000-0003-0267-786X], and *Andrej* Bencel[0000-0002-6917-4290]

University of Žilina, Faculty of Mechanical Engineering, Department of Automation and Production Systems, Univerzitná 1, 010 26 Žilina, Slovak Republic

Abstract. In this paper we are finding input-output dependencies of feed-forward neural network which usually behaves as black box. It is very important and difficult to find or evaluate those dependencies especially for multi-input/output data approximation. We will use small neural network which will be trained on a given data in MATLAB Mathworks. Network will be simulated in standalone .NET application.

1 Introduction

Artificial neural networks are representing powerful tools, which are specific, by its behavior, they can adapt its behavior to the given data. On the other side artificial neural networks are also models which are storing their behavior in some parameters like biases and weights. Those models can have many parameters – tens to thousands of parameters. In the case that the model has thousands of weight and bias parameters it is impossible to determine which input neuron affects the certain output neuron the most. In this paper we will work with simple finding and dependency drawing of multiple input quantities and one output quantity. Delineation of this input-output dependency is impossible, because in this case we would need to delineate dependency in *n*-dimension space, where *n* is summation of input and output neurons. In this paper we used simulation data which we generated manually. The data generation algorithm for this paper is not the subject for this research.[1]

The computational mechanism of the artificial neural network is quite simple, more complicated is to find optimized parameters of the network – train the model. For better understanding how the network calculates the output[1,2], we can define weighted input *in*as the sum of the weights multiplied by the inputs to the artificial neuron (eq. 1)

$$in = \sum_{i=1}^{n} w_i x_i \tag{1}$$

In this calculation of weighted input *in*, we are filling the artificial neuron model with *n* inputs values. We can notate those inputs as x_i. The significance and amount of impact of every input is expressed by their weights w_i. Every input has his own weight parameter*w*[2]. When the weighted input value is calculated it is necessary to calculate the activation of the neuron. This operation is simple, all what is important in this step is that

weighted is transferred via the activation function. There are various types of activation functions such as hyperbolic tangent, logical sigmoid, linear activation function etc. Those functions can differ in every layer of artificial neural network. When backpropagation method is chosen for training it is necessary to have differentiate activation functions. Activation function equation (eq. 2) is given below [3].

$$out = a(in) \tag{2}$$

We decided to use hyperbolic tangent activation functions in hidden layers, because of stronger gradient (in comparison with logical sigmoid)[3].

2 Simulation model

For this paper we designed a simulation model which will simulate technological process. The acting of this model is described in the generated data. Also, there was added some noise into the training data for experimental purposes. There are several inputs which are affecting output quantity of product. We will design a simple feed forward neural network which will approximate the behavior function from the training data. This model will be trained in MATLAB by using neural network toolbox. After the training, the model will be simulated in standalone application written in .Net Framework. You can see the simulation model and the physical quantity description in the figure below [4,5].

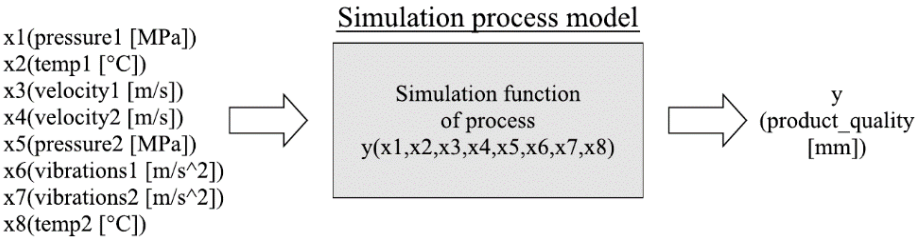

Fig. 1. Simulation model and physical quantities used in this paper.

2.1 Creating the structure of the simulation model in MATLAB

In this case artificial neural network is performing a task called function approximation. The structure of this model is dependent from the function features (number of inputs, outputs etc.). It is obviously that the input layer will correspond to input parameters of the function which is in this case 8. Output layer will have 1 neuron, which represents and stores the values for the product quality[5, 6]. Created structure of the neural network model is shown in the Fig. 2. Also, you can see used activation functions in every layer, connections between layers and sizes of every layer [7].

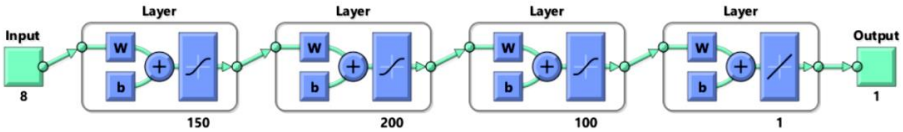

Fig. 2. Simulation model designed in MATLAB.

For the hidden layers we performed some tests which results of those tests are that we need to design 3 hidden layers. The minimum size of the first hidden layer is 150 neurons, second hidden layer should contain 200 neurons and the last hidden layer should contain 100 neurons. We´ve had two options how to achieve this. Firstly, we could use nntool toolbox for creating this type of feed-forward neural network [7]. Alternatively, you can write this MATLAB script which creates desired neural network with defined structure:

```
simulationNet = network;
simulationNet.numInputs = 1;
simulationNet.inputs{1}.size = 8;
simulationNet.numLayers = 4;
simulationNet.outputConnect = [0 0 0 1];
simulationNet.layers{1}.size = 150;
simulationNet.inputConnect(1,1) = 1;
simulationNet.biasConnect = [1;1;1;1];
simulationNet.layers{1}.transferFcn = 'tansig';
```

With this script we have created neural network with 51300 weight parameters and 451 bias parameters which we need to properly optimize – train. We want to find global optima of error function (given by training dataset) in *n* dimension space (n=51752) [8,9]. Also MATLAB can offer various optimization methods for training the neural network for example Levenberg-Marquardt, Quasi-Newton, Scaled Conjugate Gradient, Conjugate Gradient with Powell/Beale Restarts, Resilient Backpropagation, One Step Secant etc. [7,9].

2.2 Training the designed model in MATLAB

We decided to use SCG (Scaled Conjugate Gradient method) for this paper. This optimization method can train any neural network which uses differentiable function. The training is ended when several preconditions[7] are met:

- the maximum epochs or iterations were reached during the training,
- the maximum amount of time was elapsed during the training,
- performance goal of the trained model was reached,
- the performance gradient falls below defined minimal value,
- maximum count of validation check-fail was reached.

In comparison with Levenberg-Marquardt optimization algorithm in SCG there is not used line search per learning iteration. We need to write all parameters of the network in an appropriate way – as a vector in real euclidean space (eq. (3)). The reason why we need this is for the optimization of this algorithm [7]. SCG algorithm is performing matrix mathematical operations and that is why we need to have trained parameters in this vector format[9].

$$\vec{w} = \left(\dots, w_{ij}^{(l)}, w_{i+1j}^{(l)}, \dots, w_{Nj}^{(l)}, \theta_j^{l+1}, w_{ij+1}^{(l)}, w_{i+1j+1}^{(l)}, \dots\right) \quad (3)$$

where $\vec{w}$ - vector of the weight and bias parameters, $w_{ij}^{(l)}$ - i weight parameter of the l layer from j node of previous layer. Before the learning algorithm starts, it is necessary to set up all weight and bias parameters to random small values near 0 so we get initialized vector $\vec{w}$. After that, some specific optimization strategy needs to be performed during the training which is completely different as in Levenberg-Marquardt backpropagation algorithm. Another approach (Levenberg-Marquardt or Gauss-Newton training methods) is to use gradient descent. In this case we have defined error function E (also known as cost function C). Often this function is defined like mean squared error MSE, mean average error MAE or RMSE [10] (eq. 4).

$$E = \frac{1}{N}\sum_{N=1}^{N}(Y_i - Out_i)^2 \tag{4}$$

where, N is number of training samples, Y is the target output from the network, Out is the output from network after training and the i is the index of the training sample. Out represents the complete calculation entry of the neural network. Levenberg-Marquardt or Gauss-Newton approach use this error function E for weight and bias adjustment. Generated data used in this paper were randomly divided for testing, training, and validating. In the figure below you can see the correlation dependency between training (horizontal axis) and real (vertical axis) output from the model. Every circle on this plot represents one sample (model output and target).

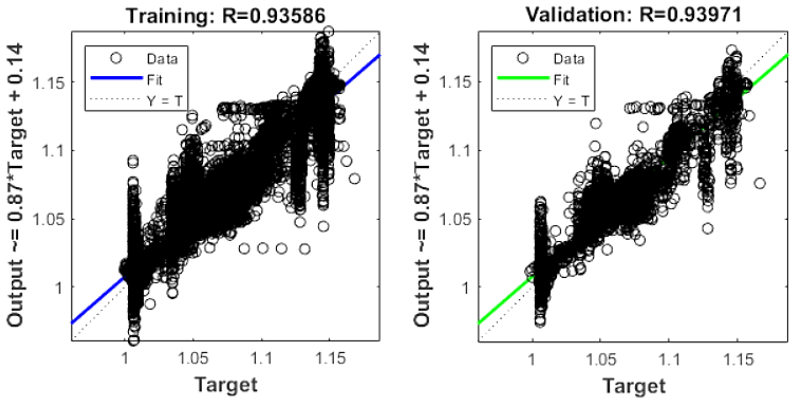

Fig. 3. Regression plot of neural networks performance (training, validating).

The correlation coefficient R (fig. 3) which describes how precisely the model describes the abstracted dependency. We chose the MSE performance function[10]. The overall training took 954 iterations and 9 minutes and 55 seconds. After the training, we should have a model with trained parameters. These parameters should influence the behavior of the model. In this point we have a model on which we can simulate various situations. The other results are shown in figure below. In the figure below you can see the output from the model when we fed the network with the 8 generated training data. The possible way how to increase the correlation coefficient R is to add some filter for smooth the signal and eliminate added noise. There are several methods how to solve or eliminate all noises. Easiest solution for this is to use the exponential smoothing. By adding this filter, you can eliminate the peaks on the signal and simplify the training process for the neural network. But you can see on the Fig. 4, that the neural network can approximate the function very well also when there is implemented noise.

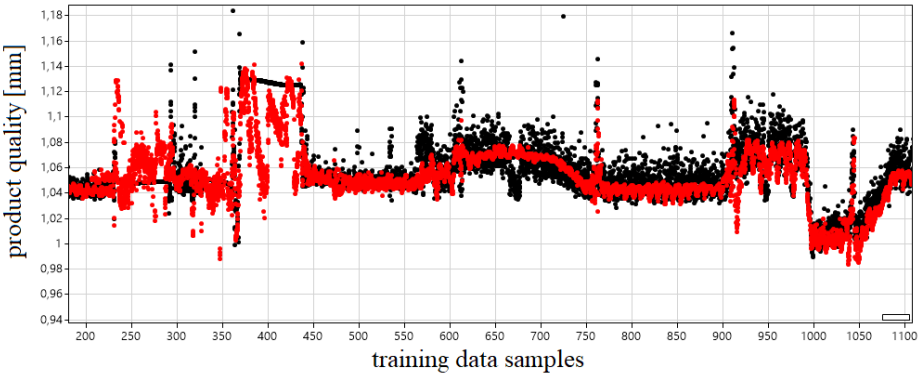

Fig. 4. Output from simulation model (trained data).

As you can see after the training process, we have got a model which can approximate the function y which describes how the product quality depends on physical quantities.

3 Evaluating dependencies

After the training, we have a specific model, and we can simulate various situations and conditions. If we want to evaluate dependency of one physical quantity to the output quantity of the neural network, we need to set other inputs to the neural network to constant values. We can choose and test various parameters from statistics for example mean average, median, modus etc. [10]. In this paper we choose the mean average parameter for the constant input values. When the inputs are set to the constant values, we can simply increment the investigated input by small value and calculate the output from the neural network. You can see the functionality of this process in the C# script below:

```
double investigatedValue;
double[] approxValues =newdouble[1];
inputLayer = constantValues;
inputLayer[variableInputIndex]=variableRange[0];
for(int i =0; i < numOfPoints; i++)
{
  investigatedValue = investigatedValue +((variableRange[1]-
  variableRange[0])/(numOfPoints -1));
  inputLayer[variableInputIndex]= investigatedValue;
  approxValues = neuralNetworkCalc(inputLayer);
  if(i ==0)
  {
    formsPlot1.plt.PlotPoint(investigatedValue + variableRange[0],
    approxValues[0], plotColor, plotLegend);
  }
  formsPlot1.plt.PlotPoint(investigatedValue + variableRange[0],
  approxValues[0], plotColor);
  formsPlot1.Render();
}
```

In this case we want to investigate how the vibrations affect the dependency of the product quality and the speed quantity. Firstly, we will set necessary inputs to constant values. After that we will evaluate the product quality – speed curve when the vibrations quantity has constant value for example 550 m/s^2. When the curve is evaluated, we will simply increment the value by 250. We will do this increasing multiple times – the range of the vibrations will be 550-2050 m/s^2. After the evaluation process we will get total of 7 quality-speed dependency curves. The dependency of those quantities is given in the figure below.

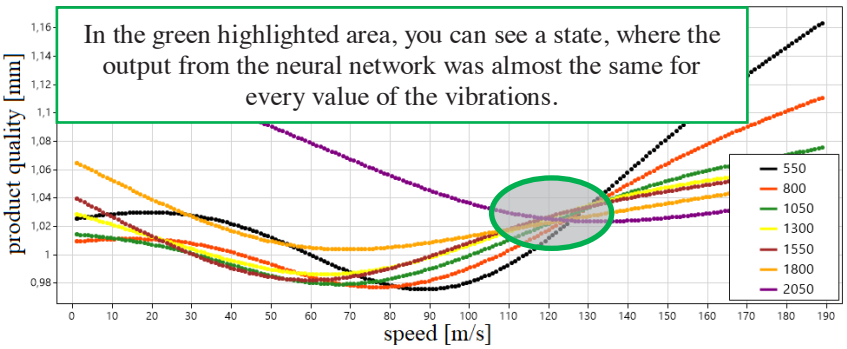

Fig. 5. Dependency evaluation by using feed-forward neural network.

When the dependency was evaluated and plotted on the graph (Fig. 5), we can say that the vibrations in the range of 550 m/s^2 – 1800 m/s^2 don´t affect the quality of the product as much as when the vibrations reached values around 2050 m/s^2. The interesting behavior occurs when the speed values are reaching values around 120 m/s – 130 m/s (green highlighted state on the Fig. 5.). In this state when the speed reaches those values, we see that the vibrations are not affecting the quality of product.
In the next test we will investigate how the vibrations are affecting the dependency of the product quality and pressure dependency. The range of the vibrations will be the same as in the previous test. The evaluated dependencies are shown in the Fig. 6. below.

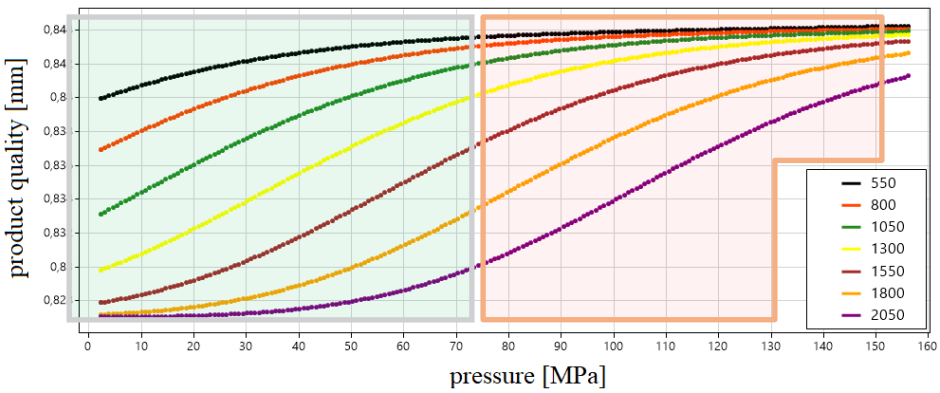

Fig. 6. Dependencyof the vibrations and product quality – pressure.

We can see in the Fig. 6. that when the pressure is low (in the green range of 0 MPa – 80 MPa), the vibrations can affect the product quality much more than in the case of the higher pressure range (red range - 80 MPa – 160 MPa). Also, the product quality increases with the increasing pressure.

4 Conclusion

Neural networks can be used as function approximator. This approximation of the function depends on the given training data. In this paper we created a simulation data set containing 8 inputs and 1 output. After successful training in MATLAB we have a model which stores his behaviour in the weight and bias parameters. Those parameters were exported from MATLAB after training. After that we created .NET application where we could test and evaluate required situations and conditions of the simulation model. There were created several dependency graphs in this application. We analysed two dependency graphs in this paper. We can easily evaluate dependencies of the model as we did in chapter 3. We choosethe vibration variable for investigation how this variable affects the dependency of quality-speed and quality-pressure.
As we can see, these algorithms of artificial intelligence are also very powerful for data analysis. If there is a large data set which contains records of some technological process you can use feed-forward neural networks and train them on this dataset. By training on the dataset, you will get a good approximated simulation model. The accuracy of this model is given by the regression plot and the correlation coefficient. This regression plot is automatically generated during the training process in MATLAB.
Disadvantage of this method is that the training data should contain relevant data – that means that the variables in the training dataset should be in correlating relationship. Also, another disadvantage is that in the training dataset should contain every possible situation and condition of the technological process. It should also contain error conditions where the output has bad, not desired results. This is probably the biggest disadvantage of this method, because collecting and storing large amount of data can be difficult in terms of required time.

Acknowledgement

This article was made under the support of APVV project – APVV-160283. Project title: research and development of multi-criteria diagnosis of production machinery and equipment based on the implementation of artificial intelligence methods.

References

1. M. Nielsen Using neural nets to recognize handwritten digits. http://neuralnetworksanddeeplearning.com/chap1.html (2019) - access 20.7.2020.
2. V. Stenchlak Impact of weights and biases on the output of neurons.Projektowanie, badania i eksploatacja Tom 1. Bielsko-Biala: Wydawnictwo Naukowe Akademii Techniczno-Humanistycznej w Bielsku-Bialej, pp. 335-343 (2020).
3. A. Sharma Understanding Activation Functions in Neural Networks.[online]Available onInternet:https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.1.6.2020 (2017).
4. M. Saga, M. Vasko, M. Handrik, P., Kopas Contribution to random vibration numerical simulation and optimisation of nonlinear mechanical systems, Scientific Journal of Silesian University of Technology - Series Transport 103, 143-154 (2019).
5. I. Kuric New methods and trends in product development and planning. 1st International Conference on Quality and Innovation in Engineering and Management (QIEM). ClujNapoca, 17.3.-19.3. pp. 453-456 (2011).

6. I. Kuric, M. Cisar, V. Tlach et al. Technical Diagnostics at the Department of Automation and Production Systems. Book Series: Advances in Intelligent Systems and Computing, Volume: 835, pp. 474-484 (2019).
7. H. Demuth Neural Network Toolbox: For Use with MATLAB. http://cda.psych.uiuc.edu/matlab_pdf/nnet.pdf (2004) - access 15.07.2020 .
8. C. Sestilli *De*ep Learning: Going Deeper toward Meaningful Patterns in Complex Data. [online] Available on Internet: https://insights.sei.cmu.edu/sei_blog/2018/02/deep-learning-going-deeper-toward-meaningful-patterns-in-complex-data.html (2018) - access 15.06.2020.
9. M. Moller A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks. [online] Available on Internet: https://www.researchgate.net/publication/222482794_Moller_MF_A_Scaled_Conjugate_Gradient_Algorithm_For_Fast_Supervised_Learning_Neural_Networks_6_525-533 (1993) - access 02.06.2020.
10. S. Basu The mean, median, and mode of unimodal distributions: a characterization. Theory of Probability & Its Applications. Publisher: Society for Industrial and Applied Mathematics, pp. 210-223 (1997).