COMP3003 Machine Learning

Exercise 2: Supervised machine learning techniques

Reece Davies [10572794]

Plymouth University

2021

Contents

Summary

This document analyses and explains what supervised machine learning is and how these techniques can be applied to real world scenarios to eliminate specific problems that may be present. These examples will be tailored to a specific industry and outlines how supervised machine learning techniques would prove to be beneficial in creating a solution. Supervised learning will be compared to reinforcement learning and explained how these two techniques differ from one another. Furthermore, supervised learning will be implemented in the software 'MATLAB' to demonstrate the process of using a training dataset and testing the algorithm with a separate 'testing' dataset. Findings will be concluded at the end of the report, which also explores how changing the neural network's parameters and the number of neurons affects the training process.

Introduction

What is supervised learning

Supervised machine learning is where an algorithm is given a dataset consisting of labelled input data (X) and their respected output data (Y). This training dataset includes both the correct inputs and outputs, thus allowing the model to learn the correct output in relation to the input over a set period of time. The algorithm measures its accuracy through the loss function, a mathematical function that maps the values onto a real number to evaluate how well the algorithm has modelled the inputted data. The data set is used as the basis for predicting the classification of other unlabelled data, which is generally the testing dataset. If the calculated prediction deviates too much from the actual output, the loss function would create a large number, therefore indicating that the model is not accurate. Gradually, with the use of the loss function, the algorithm learns to reduce the error in its predictions by adjusting its calculations until the error has been sufficiently minimised.

Supervised learning techniques

Supervised learning can be separated into two main categories – classification, and regression. A classification algorithm attempts to determine the class or category of the values within a dataset by recognising key patterns that may cause the different classes to relate to one another. For instance, this technique can be used to classify if a credit card transaction has been identified as fraudulent or genuine. There are different types of classification techniques, which includes but not limited to: logistic regression, linear discriminant analysis, K-nearest neighbour, decision trees, neural networks, and support vector machines.

Regression on the other hand, is a technique typically used for predicting, forecasting, and finding relationships between quantitative data. To summarise, this means regression is used in order to understand the relationship between both the dependent and independent variables in a dataset. For example, this technique might be used to examine if there is a relationship between a business's advertising budget and its profit from sales. Furthermore, the different types of regression in supervised learning are as follows: linear, logistic, ridge, lasso, polynomial, and Bayesian linear.

Supervised learning algorithms attempt to model relationships and any dependencies between the target prediction output and the input features in an effort to predict the output values for a new dataset based on these relationships that the algorithm identified. Reinforcement learning differs from supervised learning in such a manner that supervised learning algorithms are given a key to label the data and therefore have an understanding of how correct its output is. With reinforcement learning, there is no correct answer, but the model must make a decision on how to perform its given task. With the absence of the training dataset, it learns through experience instead of being provided with adequate training that relates to how it must calculate its output. Reinforcement learning is often used in pathfinding algorithms or grid-based systems because it will interact with its environment and learn from the consequences of its actions. By performing different actions, the algorithm can make key decisions on what is the best approach to maximise its efficiency in reaching the end goal. With each calculation, the algorithm receives a reward, which encodes its success of the action's outcome, and the agent seeks to learn from its mistakes.

Literature Review

Real world example 1: Image Classification

One of the most common uses of supervised learning is for image classification; here, the goal is for the algorithm to correctly classify what class a particular image belongs to. In this set of problems, the primary objective is identifying the class label of the image. The input is a single object, such as a photograph; and the output is the class label as to what the image should be classified as. For instance, by making use of image classification, the algorithm can analyse the different images that have been labelled as the class "cat" or "dog". Key aspects per image are analysed which would therefore allow the algorithm to identify the class by searching for these specific features. A dog would have an elongated mouth, whilst cats' mouths are generally much smaller; another key difference between the two is that dogs' ears are shaped different when compared to the sharp nature of a cat's ears. Furthermore, this type of use of supervised learning can be used in comparing and contrasting different images that a human would generally find challenging, as well as being implemented into applications which make use of photographs, images, or the device's camera.

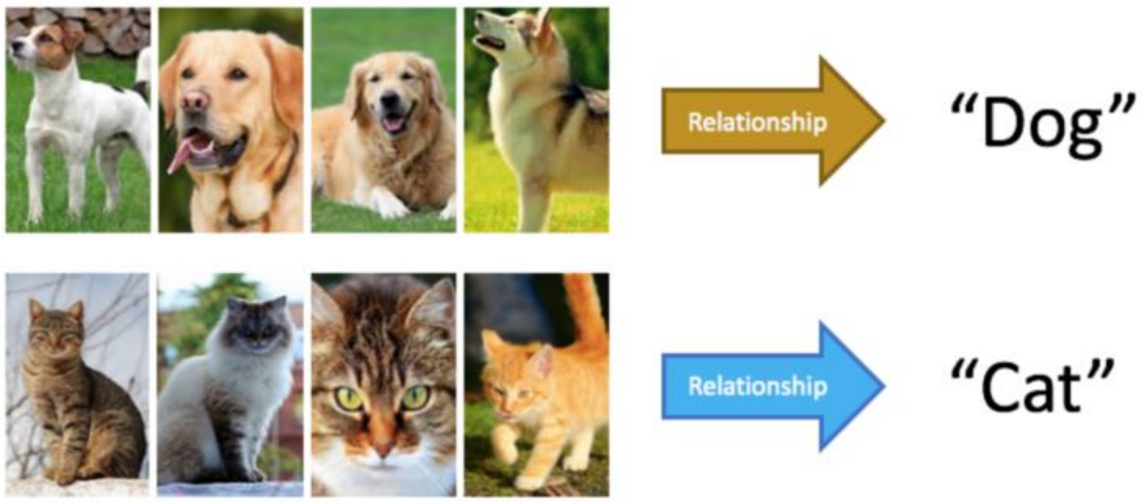


Figure 1: Comparing image classification between cats and dogs

Real world example 2: Medical diagnosis

Supervised learning can also be implemented in diagnosing specific diseases, specifically those that are rare and which doctors do not fully understand when trying to find a diagnosis, as well as diseases that have symptoms that are difficult to identify or are very similar to those of another disease. Furthermore, with the use of supervised learning, the algorithm can assist in recommending what treatments would be most effective for what the patient has been diagnosed with. This is especially beneficial for treating diseases with medical anomalies, such as identifying different types of tumours. The algorithm can be used to examine the publicly available "Breast Cancer Wisconsin Diagnosis Data Set" which includes several instances of tumours. A tumour can either be 'benign' (non-cancerous) or 'malignant' (cancerous); it is in the nature of a benign tumour to grow locally and not spread, therefore being considered non-cancerous. However, these types of tumours still pose a threat to the patient, especially if they are in close proximity to vital organs. In comparison, malignant tumours can spread and therefore considered cancerous; it is vital that the correct type of tumour has been identified to ensure the patient receives the correct treatment. Supervised learning can analyse the findings in the dataset and potentially be more accurate in diagnosing what type of tumour has manifested.

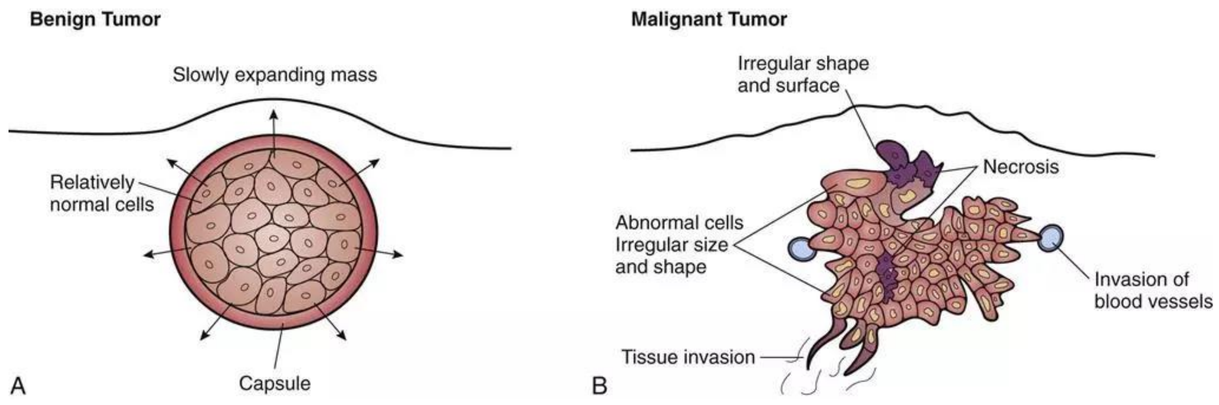
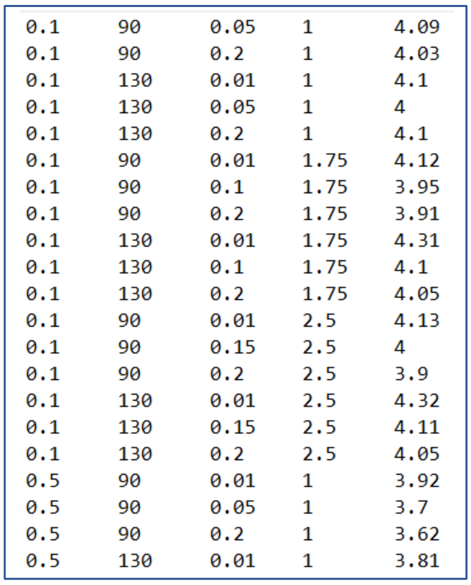


Figure 2: Characteristics of benign and malignant tumours

Implementation of Neural Networks in MATLAB

Loading the dataset

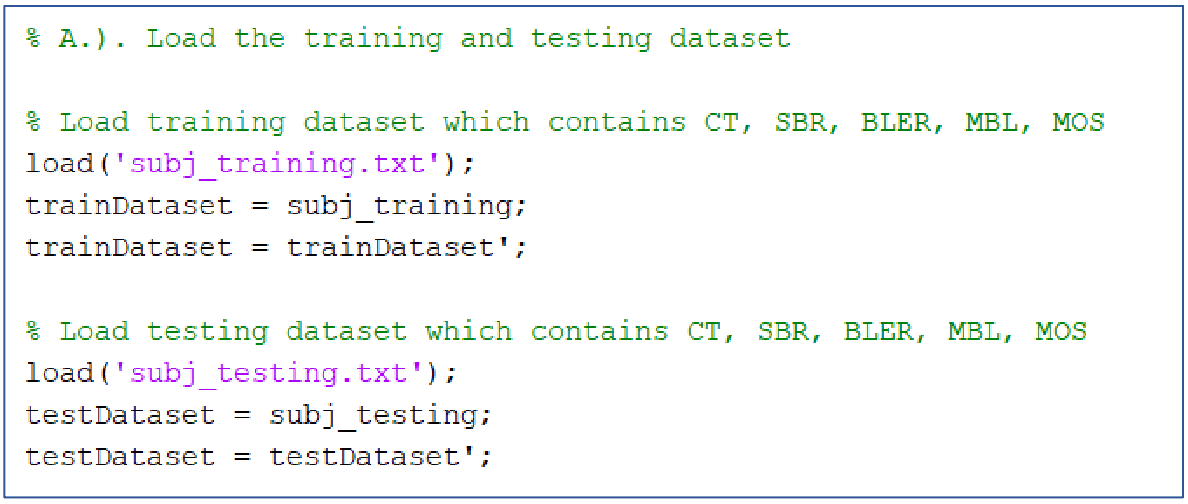


0.1	90	0.05	1	4.09
0.1	90	0.2	1	4.03
0.1	130	0.01	1	4.1
0.1	130	0.05	1	4
0.1	130	0.2	1	4.1
0.1	90	0.01	1.75	4.12
0.1	90	0.1	1.75	3.95
0.1	90	0.2	1.75	3.91
0.1	130	0.01	1.75	4.31
0.1	130	0.1	1.75	4.1
0.1	130	0.2	1.75	4.05
0.1	90	0.01	2.5	4.13
0.1	90	0.15	2.5	4
0.1	90	0.2	2.5	3.9
0.1	130	0.01	2.5	4.32
0.1	130	0.15	2.5	4.11
0.1	130	0.2	2.5	4.05
0.5	90	0.01	1	3.92
0.5	90	0.05	1	3.7
0.5	90	0.2	1	3.62
0.5	130	0.01	1	3.81

Figure 3: Sample of training dataset values

With this given scenario, the goal of implementing MATLAB Neural Networks will be used to predict video quality for mobile video streaming applications. The training and testing dataset consist of four different input features – video Content Type (CT), vide Sending BitRate (SBR), networks Block Error Rate (BLER), and the Mean Burst-loss Length (MBL); all of which have been collected via mobile streaming experiments. The output is the Mean Opinion Score (MOS), which represents quality of the video stream and therefore will reflect the user's experience when streaming a video under different network connection conditions; this output was acquired by averaging the users' opinion score given by 20 individuals. The training dataset consists of 81 different samples, whilst the testing dataset consists of 54.

The datasets are located in two separate files in a plain text format; in order to obtain such data, the MATLAB function "**load**" was used, with the file name as the input argument. This therefore loads all values within the "**subj_training.txt**" and "**subj_testing.txt**" files. Furthermore, once the datasets were loaded into MATLAB, each one needed to be transposed, which is achieved with the colon symbol (**'**). To specify – transposing a matrix interchanges the row and column index for each element, and thus the end result is a matrix where the rows and columns have been replaced with one another.

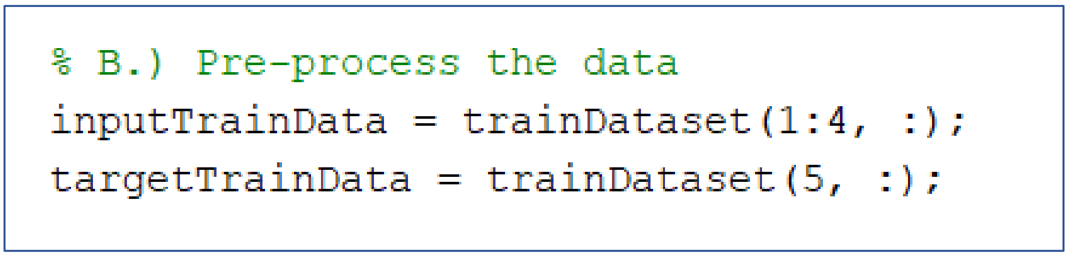


```
% A.). Load the training and testing dataset

% Load training dataset which contains CT, SBR, BLER, MBL, MOS
load('subj_training.txt');
trainDataset = subj_training;
trainDataset = trainDataset';

% Load testing dataset which contains CT, SBR, BLER, MBL, MOS
load('subj_testing.txt');
testDataset = subj_testing;
testDataset = testDataset';
```

Figure 4: MATLAB code for loading datasets

Pre-processing the data

Once the data had been loaded into MATLAB, it needed to be processed for the neural network to understand what the different inputs are and what the output value should be compared to. In this case, the inputs are the four values of CT, SBR, BLER, and MBL, whilst the output target value is MOS, all located within their respected rows. To define which rows the neural network must identify as these values, they have declared with the "**inputTrainData"** and "**targetTrainData**" variables. "**inputTrainData**" specifies that CT, SBR, BLER, and MBL are all the values within the first four rows in the "**trainDataset**" matrix, and "**inputTargetData**" specifies that MOS consists of all values within the fifth row. The code responsible for this can be seen in Figure 5.



```
% B.) Pre-process the data
inputTrainData = trainDataset(1:4, :);
targetTrainData = trainDataset(5, :);
```

Figure 5: MATLAB code for pre-processing the input and output values

Creating the neural network

The specification demanded that the neural network used would be created with the MATLAB function "**feedforwardnet**". This function returns a Feedforward neural network with a hidden layer of a size that is specified within the input argument. A Feedforward neural network consists of numerous layers, with the first layer directly connected to the network's input and each successive layer being connected to the previous layer. The goal of a feedforward network is to approximate some function, such as y=f*(x), where the input X is mapped onto Y. These networks are often referred to as 'multilayer perceptrons' (MLPs) due to their structure of different nodes interconnected among its different layers. An individual node might be connected to several nodes in the layer prior to it, from which is receives data, as well as several nodes to the layer after it, to which it transmits data. Furthermore, the term 'feedforward' is used as the information flows through the function being evaluated from X, through the intermediate computations used to define F.

The nodes all contain a different purpose; there is: an input layer, hidden layer, and output layer. The input layer contains units that receives input from an external source, which will be used to train the network. The hidden layer contains units that are capable of transferring the input into information that can be used to perform specific calculations. And the output layer contains units that respond to the information gained on how to complete a certain task.

This feedforward network will contain one hidden layer, and therefore consist of a total of three separate layers. The behaviour and calculations performed by the network is generally determined by its architecture or structure, such as the number of hidden layers, or nodes within each layer, alongside with other parameters, such as epochs, performance goal, minimum performance gradient, etc.

One hidden layer is generally sufficient for the large majority of problems that a network is used for. Furthermore, the larger the number of hidden layers in a network, the longer it will take to produce the output; however, it will be more capable of solving more complex problems. It is therefore important that the number of hidden layers is justified. With regards to the neurons within each hidden layer, they simply calculate the weighted sum of inputs and weights, add the bias, and execute the activation function. Altering the number of neurons within the hidden layer will affect the performance and ultimately change the Mean Squared Error (MSE).

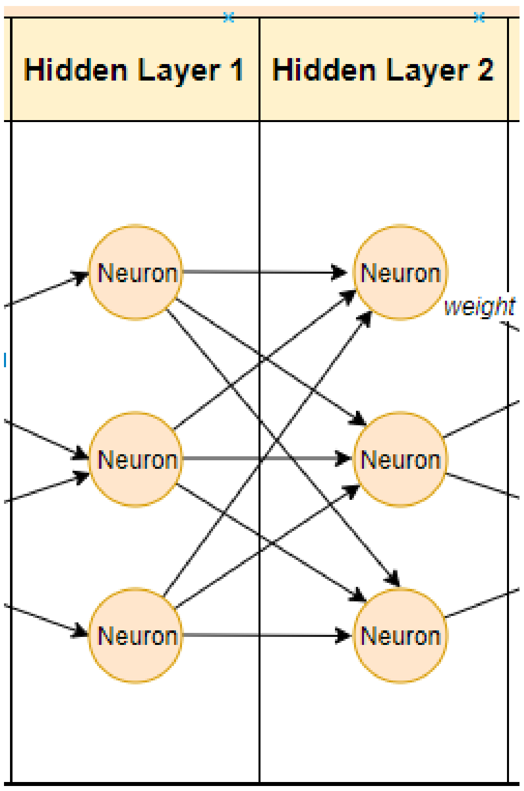
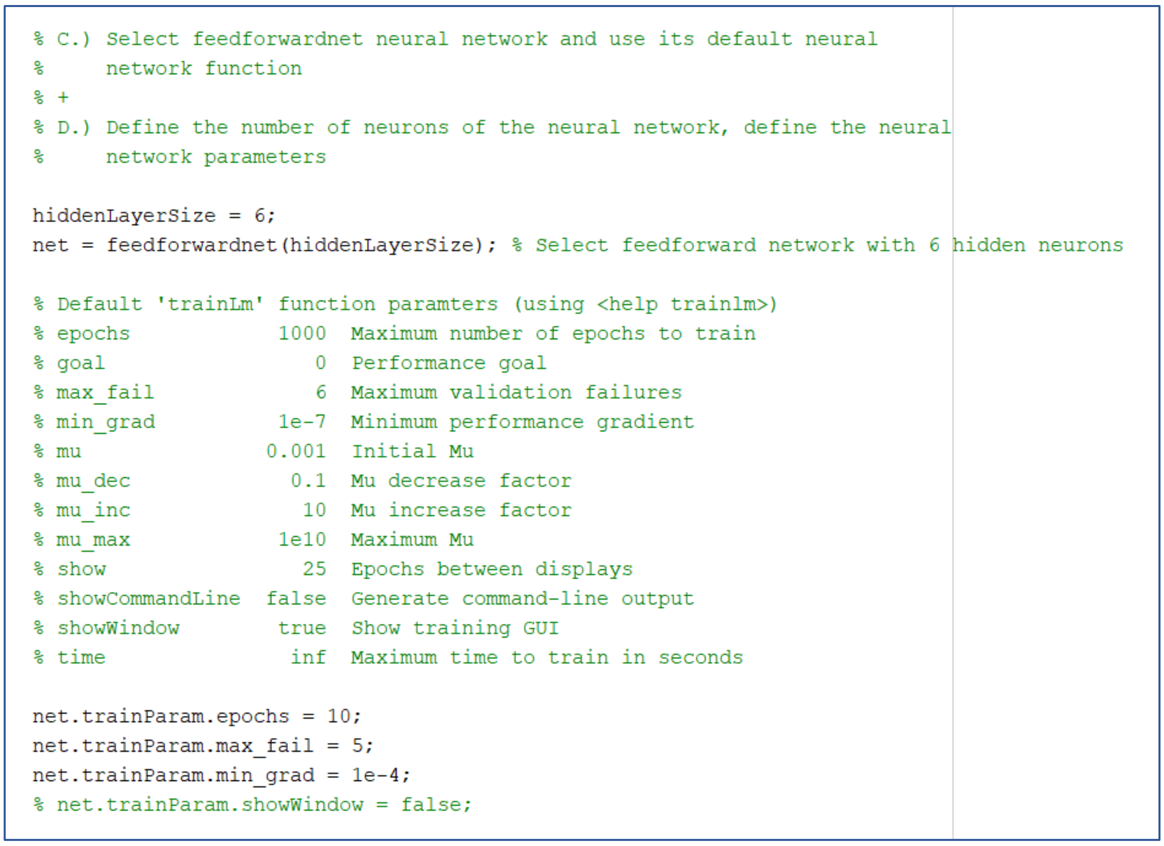


Figure 6: Two hidden layers consisting of 3 neurons per layer. Malik, F. (2019)

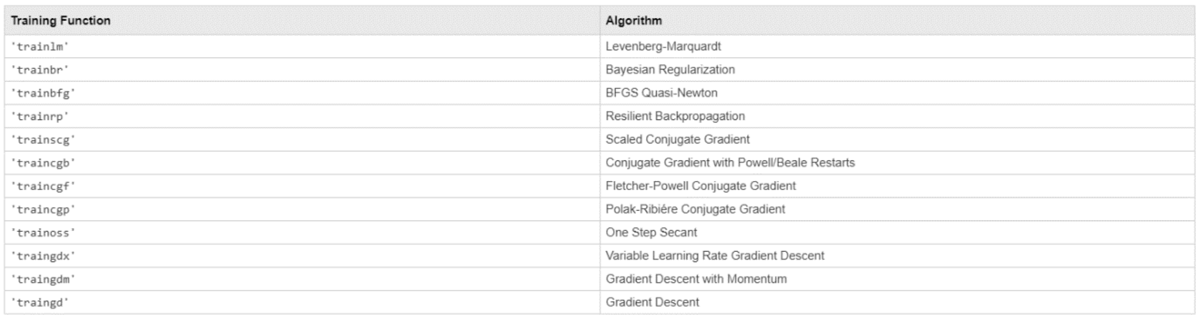


```
% C.) Select feedforwardnet neural network and use its default neural
%     network function
% +
% D.) Define the number of neurons of the neural network, define the neural
%     network parameters

hiddenLayerSize = 6;
net = feedforwardnet(hiddenLayerSize); % Select feedforward network with 6 hidden neurons

% Default 'trainLm' function paramters (using <help trainlm>)
% epochs            1000  Maximum number of epochs to train
% goal                 0  Performance goal
% max_fail             6  Maximum validation failures
% min_grad          1e-7  Minimum performance gradient
% mu               0.001  Initial Mu
% mu_dec             0.1  Mu decrease factor
% mu_inc              10  Mu increase factor
% mu_max            1e10  Maximum Mu
% show                25  Epochs between displays
% showCommandLine  false  Generate command-line output
% showWindow        true  Show training GUI
% time               inf  Maximum time to train in seconds

net.trainParam.epochs = 10;
net.trainParam.max_fail = 5;
net.trainParam.min_grad = 1e-4;
% net.trainParam.showWindow = false;
```

Figure 7: MATLAB code for creating the feedforward network

Figure 7 shows the code that has been used to create the feedforward network; firstly the number of neurons within the hidden layer is controlled with the variable "**hiddenLayerSize**" and is used as an argument in the subsequent statement. The network has been given the name of "**net**" and is created with the MATLAB function "**feedforwardnet**" which takes the hidden layer size as its input argument.

With regards to the different training functions for the feedforward network, the default is used – "Levenberg-Marquardt" backpropagation neural network algorithm; however, if the user wanted to specify this, they would input "**trainlm**". The different training functions used within a network can be specified with the following statements displayed in figure 8. These training functions will ultimately affect how the network behaves, as it will be dependent upon specific parameters that the user can adjust. For instance, by specifying the Levenberg-Marquardt backpropagation algorithm, the network updates the weight and bias values according to Levenberg-Marquardt optimisation, which significantly outperforms gradient descent and conjugate gradient methods for medium sized problems.



Training Function	Algorithm
'trainlm'	Levenberg-Marquardt
'trainbr'	Bayesian Regularization
'trainbfg'	BFGS Quasi-Newton
'trainrp'	Resilient Backpropagation
'trainscg'	Scaled Conjugate Gradient
'traincgb'	Conjugate Gradient with Powell/Beale Restarts
'traincgf'	Fletcher-Powell Conjugate Gradient
'traincgp'	Polak-Ribiére Conjugate Gradient
'trainoss'	One Step Secant
'traingdx'	Variable Learning Rate Gradient Descent
'traingdm'	Gradient Descent with Momentum
'traingd'	Gradient Descent

Figure 8: List of different training functions that can be used in MATLAB networks. Mathworks (2020)

To view the feedforward network's structure, the user can use the MATLAB function "**view**" with the network's name as the argument. This will display a separate window panel for the network's input, hidden layers, and output, as shown in figure 9.

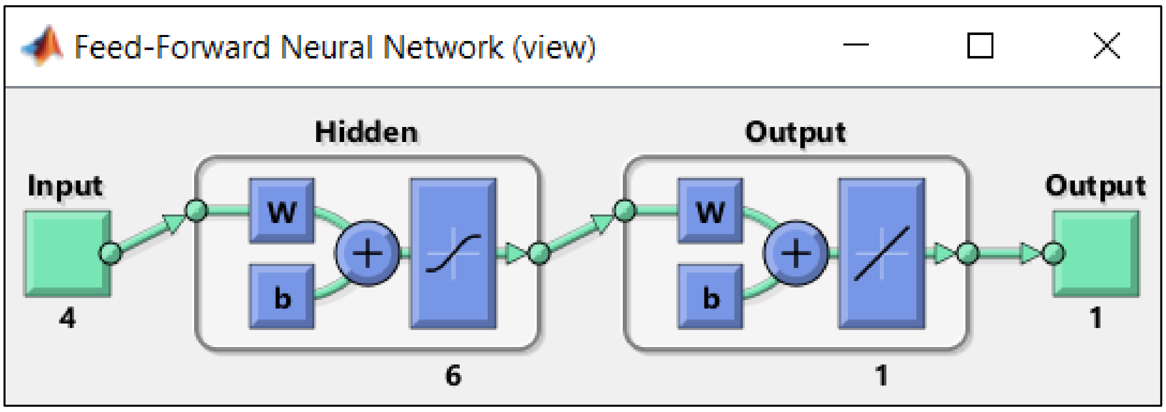
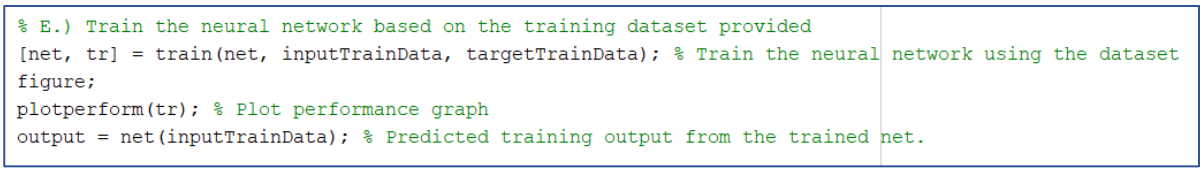


Figure 9: Diagram for the structure of the feedforward network "**net**"

Training the network

Once the feedforward network had been created, the next step is to train it with the appropriate training data. This is done via the MATLAB function "**train**" which takes three arguments – the specified network, the training dataset and the target output values. Is this demonstration "**net**" is being trained with the datasets "**inputTrainData**" and "**targetTrainData**". To specify, the network will perform the calculation for the four inputs per row and compare its output to the associated row in the "**targetTrainData**". This is how the network adjusts its calculation per inputted row for a more accurate output, as the goal is to have an output that equals that to the target value.



```
% E.) Train the neural network based on the training dataset provided
[net, tr] = train(net, inputTrainData, targetTrainData); % Train the neural network using the dataset
figure;
plotperform(tr); % Plot performance graph
output = net(inputTrainData); % Predicted training output from the trained net.
```

Figure 10: MATLAB code for training the dataset.

Furthermore, in order to view the performance of the network in the form of a graph, the MATLAB function “**plotperform**” has been used, which takes the training record as its argument. This figure plots the performance for the training, validation, and test performances for the training record “**tr**” and is demonstrated in figure 11. The graph’s X-axis represents the epochs for the network, whilst the Y-axis represents the Mean Square Error (MSE) per epoch; to bring into context - an epoch refers to one cycle through the full dataset. With the use of the distinct colours for the training (blue), validation (green), and testing (red), it allows the user to easily differentiate one from another with the specified network and thus allows them to have a greater understanding of what parameters to adjust in order to improve the performance.

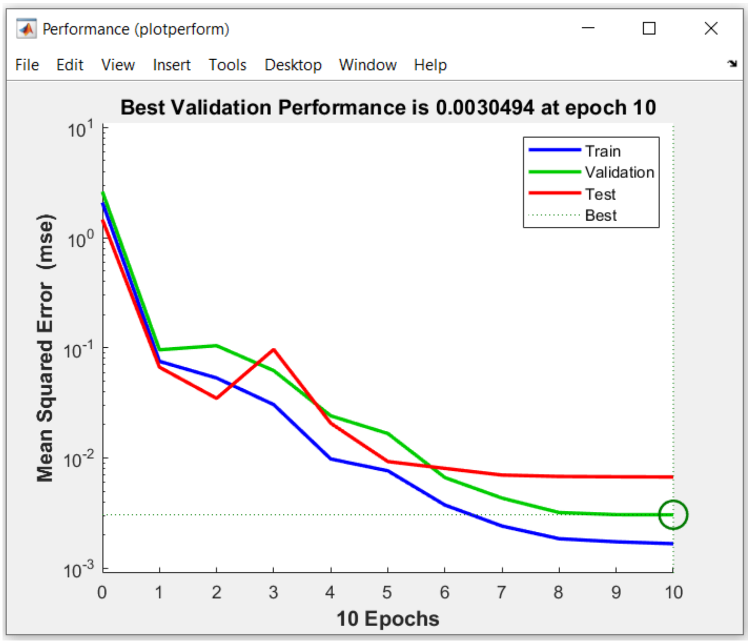


Figure 11: Performance plot for feedforward network

Subsequently, the correlation coefficient is calculated, which measures the strength of the relationship between the relative movements of two variables. In this case, it is measuring the strength between the measured MOS and the predicted MOS for the feedforward network’s training dataset. Given these two variables, the correlation coefficient produces a value between -1 and 1; a value greater than zero represents a positive relationship whilst a value less than zero represents a negative relationship. Either of these relationships will show that the two variables move in the same direction; however, if the correlation coefficient is equal to zero, it therefore indicates that there is no relationship between these two variables. Furthermore, a correlation of exactly -1 or 1 indicates a perfect correlation, but the correlation can never be greater than 1 or less than -1 as this would mean there was an error in calculating the correlation coefficient. A value of 0.1 might represent a positive correlation, but it would be considered inadequate as there is a weak relationship between the two variables. It is common to discard any correlation that does not surpass at least 0.8, with any value greater than 0.9 representing a very strong relationship.

$$\rho_{xy} = \frac{\mathrm{Cov}(x,y)}{\sigma_x \sigma_y}$$

where:

ρ_{xy} = Pearson product-moment correlation coefficient

$\mathrm{Cov}(x,y)$ = covariance of variables x and y

σ_x = standard deviation of x

σ_y = standard deviation of y

Figure 12: Formula for calculating correlation coefficient

In order to calculate the correlation coefficient in MATLAB, the function “**corrcoeff**” is used, with the training data target values as the first argument, and the networks output as the second argument. This value is represented as “**R**” and was converted into a percentage accurate to two decimal values. Next, a graph was generated with the use of the MATLAB function “**scatter**”, which returns a scatter plot with circles at the locations specified by two input vectors. Displayed in figure 14 is the resulting scatter graph generated with the two vectors being the network’s training data target values, and its calculated output. “**R**” is displayed in the title for a better understanding of the relationship between the two vectors.

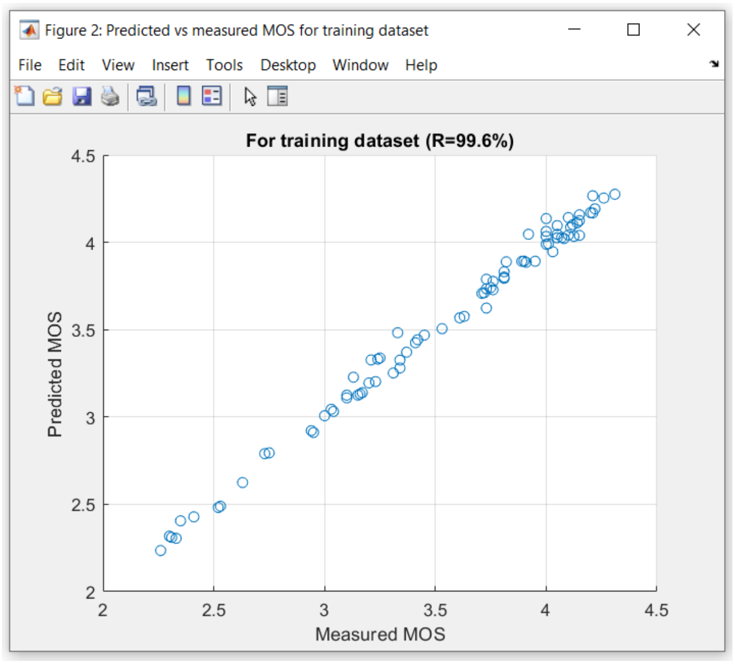


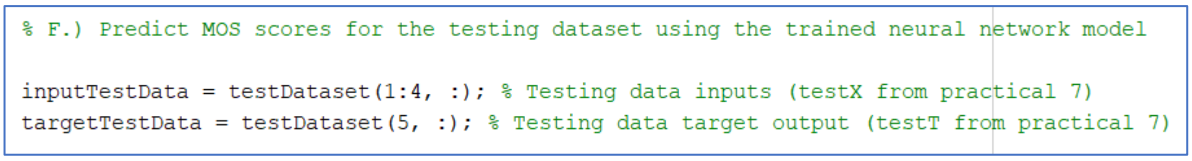
Figure 14: Scatter graph for predicted vs measured MOS for the training dataset

```
% Calculate correlation coefficients (R) for the training dataset
R = corrcoef(targetTrainData, output);
R = R(1,2) * 100; % Turn R into a percentage
R = round(R, 2); % Round R off to 2nd decimal point for better percentage value

% Draw a scatter plot between the target and predicted output)
figure('Name','Predicted vs measured MOS for training dataset');
scatter(targetTrainData, output);
title(['For training dataset (R=',num2str(R),'%)'])
hold off;
xlabel('Measured MOS');
ylabel('Predicted MOS');
grid on;

view(net);
```

Figure 13: MATLAB code for calculating the correlation coefficient and generating a scatter graph between the target and predicted output for the training dataset

Testing the performance with a new dataset

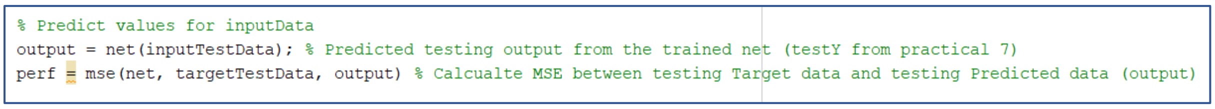
Once the network had been trained, it is now capable of predicting the MOS for a new dataset, referred to as the testing dataset “**testDataset**”. Firstly, the new testing data was required to be pre-processed for the network, similarly to the training dataset.



```
% F.) Predict MOS scores for the testing dataset using the trained neural network model

inputTestData = testDataset(1:4, :); % Testing data inputs (testX from practical 7)
targetTestData = testDataset(5, :); % Testing data target output (testT from practical 7)
```

Figure 14: MATLAB code for pre-processing the testing dataset

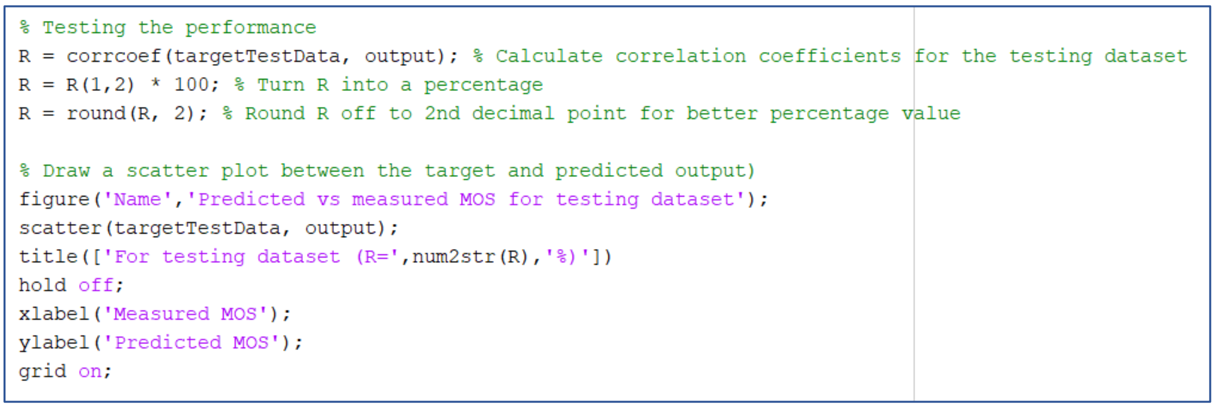
To calculate the network’s output for the testing dataset, the MATLAB function “**net**” is used with “**inputTestData**” as its argument; the values are then stored in the variable “**output**”. Subsequently, the performance is calculated with the MATLAB function “**mse**” which returns the mean square error for network “**net**” between the matrices “**targetTestData**” and “**output**”. To elaborate, the mean square error is the average of the square of the difference between the actual and predicted values. This therefore means the MSE represents the performance of the feedforward network because it compares the test data’s target values and the network’s calculated output in relation to the regression line. The MSE is stored in the variable “**perf**”, which is then output to the command window.



```
% Predict values for inputData
output = net(inputTestData); % Predicted testing output from the trained net (testY from practical 7)
perf = mse(net, targetTestData, output) % Calcualte MSE between testing Target data and testing Predicted data (output)
```

Figure 15: MATLAB code for calculating the network’s performance for the testing dataset

By following the same process as with the training dataset, the correlation coefficient was calculated for the testing data. This ultimately tests the performance of the network as it is predicting the MOS with a new dataset after being trained; the correlation coefficient is then displayed in the form of a scatter graph. The two scatter graphs are displayed side-by-side in figure 17.



```
% Testing the performance
R = corrcoef(targetTestData, output); % Calculate correlation coefficients for the testing dataset
R = R(1,2) * 100; % Turn R into a percentage
R = round(R, 2); % Round R off to 2nd decimal point for better percentage value

% Draw a scatter plot between the target and predicted output)
figure('Name','Predicted vs measured MOS for testing dataset');
scatter(targetTestData, output);
title(['For testing dataset (R=',num2str(R),'%)'])
hold off;
xlabel('Measured MOS');
ylabel('Predicted MOS');
grid on;
```

Figure 16: MATLAB code for calculating the correlation coefficient and generating a scatter graph between the target and predicted output for the testing dataset

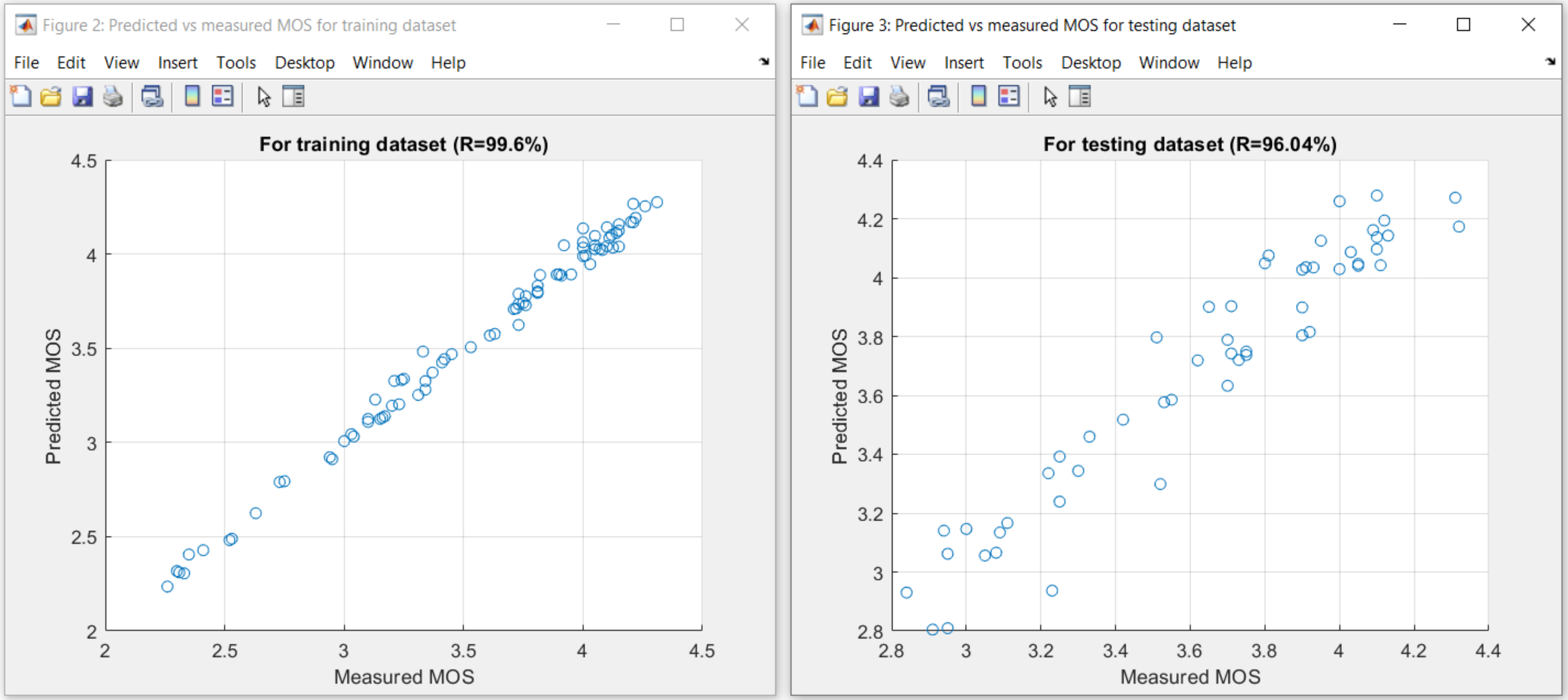


Figure 17: Comparing scatter graphs for the predicted vs measured MOS for the training and testing datasets

The regression of the testing dataset is plotted with the use of the MATLAB function "**plotregression**", which acts very similarly to the scatter graphs previously shown. This function takes two arguments – the target data values, and the network's calculated output. Executing it will produce a two-dimensional plot for the linear regression of the target values relative to the output.

```
% Plot regression (with R value displayed in the title of the graph for the testing dataset)
figure;
plotregression(targetTestData, output);

net % Display the network's specifications in the command window
```

Figure 18: MATLAB code for plotting linear regression of "**targetTestData**" relative to "**output**"

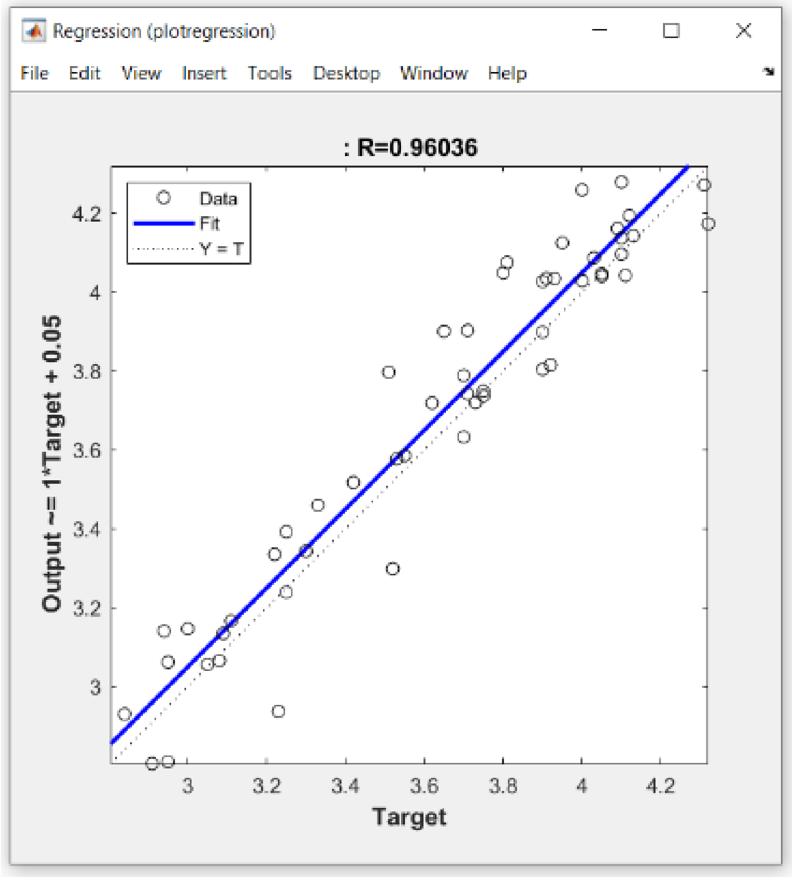
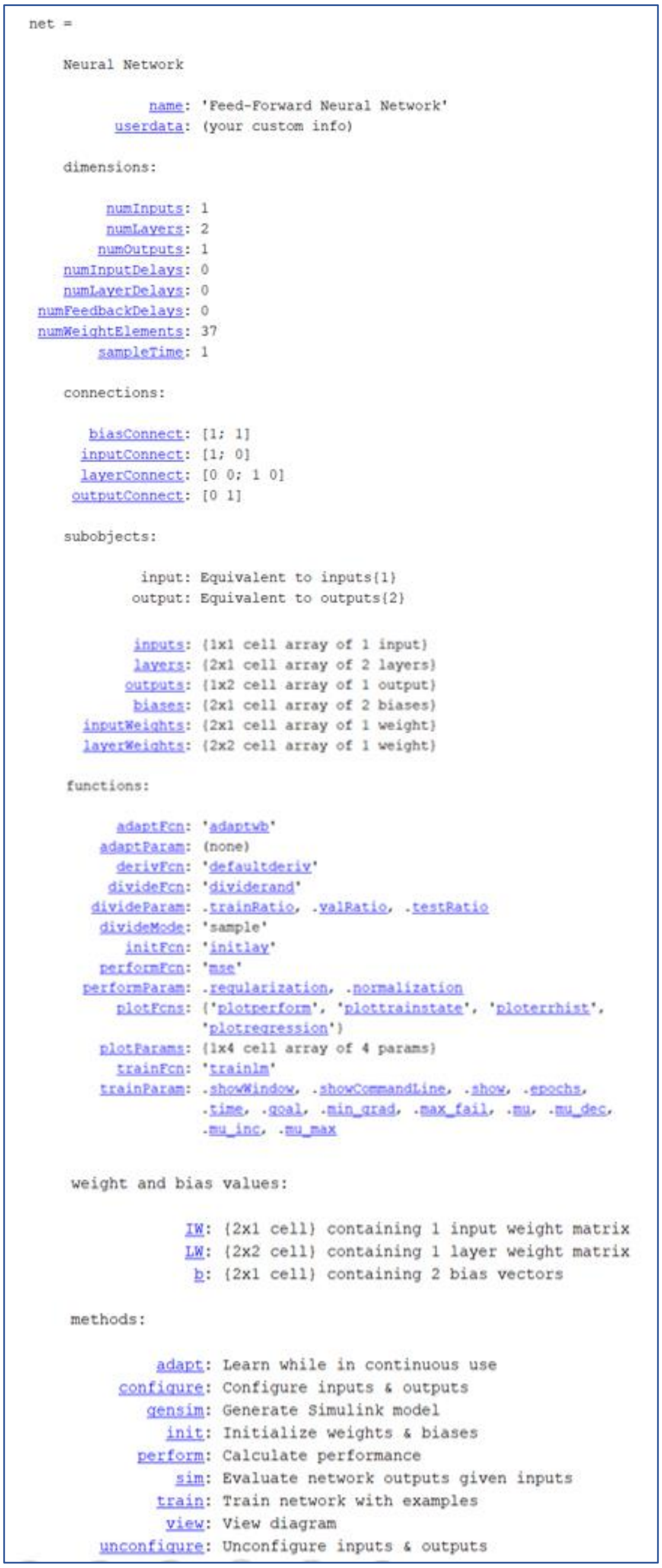


Figure 19: Linear regression plot of "**targetTestData**" relative to "**output**"

Finally, by stating "**net**" in the MATLAB script, it outputs all the specifications of the feedforward network, which can be seen in figure 21.

```
net % Display the network's specifications in the command window
```

Figure 20: MATLAB code used to output the details of the feedforward network "**net**"



```
net =

    Neural Network

              name: 'Feed-Forward Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 37
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression'}
        plotParams: {1x4 cell array of 4 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs
```

Figure 21: MATLAB code used to output the details of the feedforward network “**net**”

Adjusting the network’s parameters for better performance

One may need to adjust the feedforward network’s default parameters in order to achieve a better outcome; this is achieved through either using one of the alternative training functions listed in figure 8, or adjusting the network’s default parameters. In order to change the training function, the user must therefore change the variable “**trainFcn**” to the appropriate function name. For instance, to change to the “Resilient Backpropagation” algorithm, the user must specify the following:

```
trainFcn = 'trainrp';
```

Different training functions use different algorithms and therefore may be best suited for a particular problem. Stated within the research paper “Using Neural Network Algorithms in Prediction of Mean Glandular Dose Based on the Measurable Parameters in Mammography” (2009), they perform tests with different training algorithms and access the performance between one another. Figure 22 presents results of testing for Levenberg-Marquardt and Resilient Backpropagation learning algorithms with correlation R=0.845 and R=0.821, alongside with a summary of results for all training algorithms. As stated in the paper: ‘The coefficient of determination R2 is a measure of how well the regression line represents the data’. These figures show how changing the algorithm can have a major effect on the network’s performance.

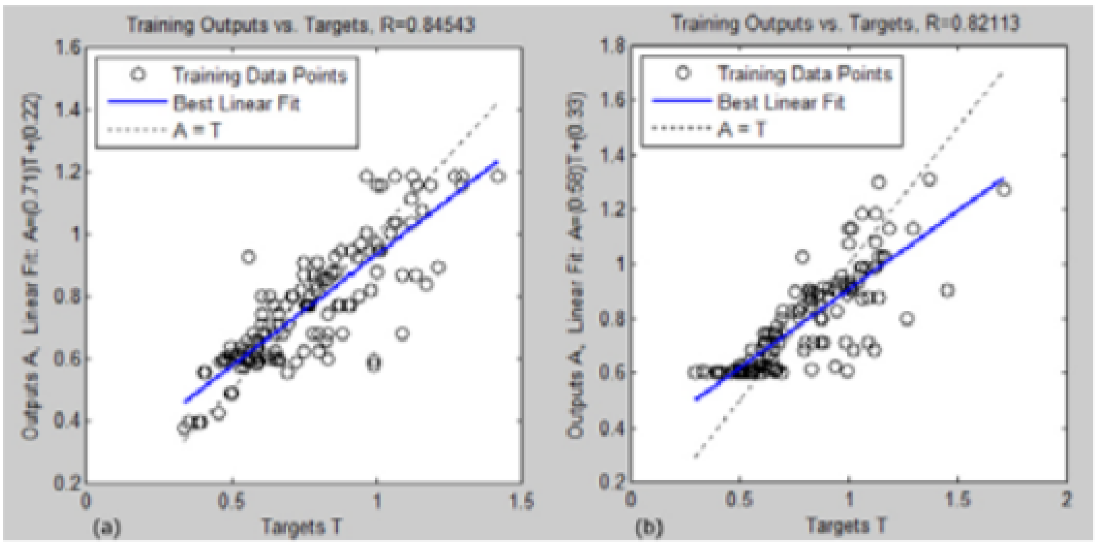
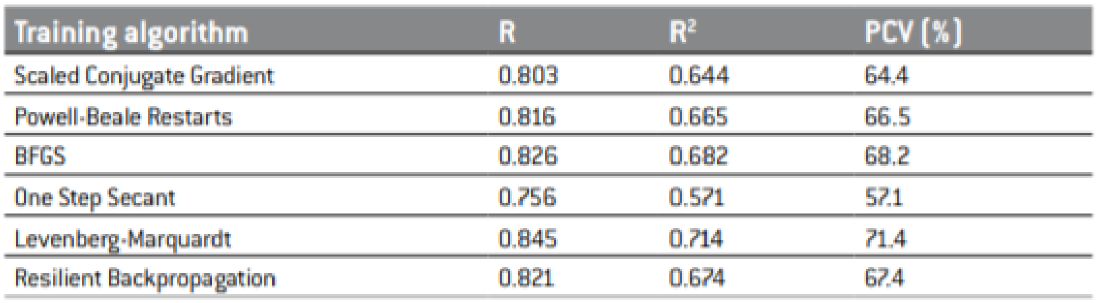


Figure 5. a) Levenberg-Marquardt , b) Resilient Backpropagation.

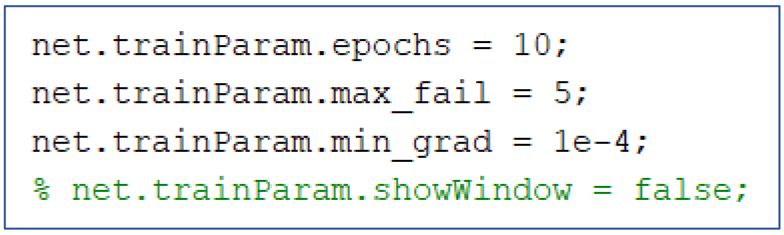


Training algorithm	R	R^2	PCV (%)
Scaled Conjugate Gradient	0.803	0.644	64.4
Powell-Beale Restarts	0.816	0.665	66.5
BFGS	0.826	0.682	68.2
One Step Secant	0.756	0.571	57.1
Levenberg-Marquardt	0.845	0.714	71.4
Resilient Backpropagation	0.821	0.674	67.4

Table 1. Results of training sessions for different learning algorithms.

Figure 22: Comparing performance between different training algorithms. Ceke, D. Kunosic, S. Kopric, M. Lincender, L. (2009)

With regards to the function’s parameters, the user is capable of modifying the default values with the use of “**trainParam**”, as demonstrated in figure 23. The parameters for the feedforward network in this demonstrated have been modified for a better performance, as the default values were not optimal.



```
net.trainParam.epochs = 10;
net.trainParam.max_fail = 5;
net.trainParam.min_grad = 1e-4;
% net.trainParam.showWindow = false;
```

Figure 23: MATLAB code changing network’s default parameters

Changing these parameters makes slight adjustments in the performance; for example changing the epochs will therefore affect the number of cycles that are executed for the dataset, therefore further optimising the network. With more rounds of optimisation, the MSE will progressively reduce; however, if the epochs parameter is too high, there is a possibility that the network may become over-fit, thus reducing performance. The “**max_fail**” (maximum validation failures) parameter was also altered, which is used to prevent the network from performing poorly in validation, testing, and unseen data, but performing well on the training dataset. If the network’s training contiuously degrades, the network’s algorithm is halted. Finally, the “**min_grad**” (minimum performance gradient) is also altered and will prevent the network from plateauing, as it means the network is required to reach a minimum performance criteria; if it does not reach this requirement, the execution is halted.

Conclusion

Due to the small dataset used in this demonstration, changing the parameters may not immediately be visible to the overall performance of the network; however, when applied to a more sophisticated network, the change in results can be detrimental. For example, the default epochs parameter is 1,000 but has been altered to 10; this is because following numerous tests, the best validation performance is between 6-12. Reducing the epochs improved the performance of the network as well as prevented the possibility of over-fitting.

Overall, with the implementation of the Levenberg Marquardt algorithm in a feedforward network, it is clear how a neural network might be useful in the many problems that are faced on a daily basis. By training a network on a large dataset, the algorithm will therefore be capable of recognising hidden patterns and correlations in raw data. With complex scenarios, it may be extremely challenging for a human to detect such patterns, however a neural network may easily identify those on what a human might not necessarily observe. Of course, there is the possibility that a neural network may be over-fit or trained in such a manner that it plateaus, thus hindering its calculations. It is therefore important that the network is properly configured and trained for optimal performance.

References

Akash. (2020). What is Supervised Learning and its different types? Edureka!. Viewed 3rd Jan 2020. <https://www.edureka.co/blog/supervised-learning/>

Ceke, D. Kunosic, S. Kopric, M. Lincender, L. (2009). Using neural network algorithms in prediction of Mean Glandular Dose based on the measurable parameters in mammography. Viewed 19th Jan 2021. p.p.194-197

Fernando, J. (2020). Correlation Coefficient. Investopedia. Viewed 19th Jan 2021. <https://www.investopedia.com/terms/c/correlationcoefficient.asp>

Gharagyozyan, H. (2019). A Practical Application of Machine Learning in Medicine. Macadamian. View 3rd Jan 2021. <https://www.macadamian.com/learn/a-practical-application-of-machine-learning-in-medicine/>

Gupta, T. (2017). Deep Learning: Feedforward Neural Network. Towards Data Science. Viewed 12th Jan 2021. <https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7>

IBM Cloud Education. (2020). What is supervised learning? IBM. Viewed 1st Jan 2021. <https://www.ibm.com/cloud/learn/supervised-learning>

Malik, F. (2019). What Are Hidden Layers?. Medium. Viewed 14th Jan 2021. <https://medium.com/fintechexplained/what-are-hidden-layers-4f54f7328263>

Mathworks. (2020). feedforwardnet. Mathworks. Viewed 11th Jan 2021. <https://www.mathworks.com/help/deeplearning/ref/feedforwardnet.html>

Mathworks. (2020). fitnet. Mathworks. Viewed 11th Jan 2021. <https://www.mathworks.com/help/deeplearning/ref/fitnet.html>

Mathworks. (2020). trainlm. Mathworks. Viewed 12th Jan 2021. <https://www.mathworks.com/help/deeplearning/ref/trainlm.html>

Talabis, M. McPherson, R. Miyamoto, I. Martin, J. Kaye, D. (2014) Information Security Analytics: Finding Security Insights, Patterns and Anomalies in Big Data. Amsterdam, Netherlands. Elsevier Inc. Chapter 1 – Analytics Defined.

Van Beek, P. (2016). For Neural Networks what is the importance of epochs and how is the number of epochs decided?. Viewed 20th Jan 2021. <https://www.researchgate.net/post/For_Neural_Networks_what_is_the_importance_of_epochs_and_how_is_the_number_of_epochs_decided>

Appendix

```
clc;
close all;
clear;

% A.). Load the training and testing dataset

% Load training dataset which contains CT, SBR, BLER, MBL, MOS
load('subj_training.txt');
trainDataset = subj_training;
trainDataset = trainDataset';

% Load testing dataset which contains CT, SBR, BLER, MBL, MOS
load('subj_testing.txt');
testDataset = subj_testing;
testDataset = testDataset';

% B.) Pre-process the data
inputTrainData = trainDataset(1:4, :);
targetTrainData = trainDataset(5, :);

% C.) Select feedforwardnet neural network and use its default 
neural
%     network function
% +
% D.) Define the number of neurons of the neural network, 
define the neural
%     network parameters

% trainFcn = 'trainrp'; % For testing the network with a 
different training function

hiddenLayerSize = 6;
net = feedforwardnet(hiddenLayerSize); % Select feedforward 
network with 6 hidden neurons

% Default 'trainLm' function paramters (using <help trainlm>)
% epochs             1000   Maximum number of epochs to train
% goal                  0   Performance goal
% max_fail              6   Maximum validation failures
% min_grad           1e-7   Minimum performance gradient
% mu                0.001   Initial Mu
% mu_dec              0.1   Mu decrease factor
% mu_inc               10   Mu increase factor
% mu_max             1e10   Maximum Mu
% show                 25   Epochs between displays
% showCommandLine   false   Generate command-line output
% showWindow         true   Show training GUI
% time                inf   Maximum time to train in seconds

net.trainParam.epochs = 10;
net.trainParam.max_fail = 5;
net.trainParam.min_grad = 1e-4;
% net.trainParam.showWindow = false;

% E.) Train the neural network based on the training dataset
provided
[net, tr] = train(net, inputTrainData, targetTrainData); %
Train the neural network using the dataset
figure;
plotperform(tr); % Plot performance graph
output = net(inputTrainData); % Predicted training output from
the trained net.

% Calculate correlation coefficients (R) for the training
dataset
R = corrcoef(targetTrainData, output);
R = R(1,2) * 100; % Turn R into a percentage
R = round(R, 2); % Round R off to 2nd decimal point for better
percentage value

% Draw a scatter plot between the target and predicted output
figure('Name','Predicted vs measured MOS for training
dataset');
scatter(targetTrainData, output);
title(['For training dataset (R=',num2str(R),'%)'])
hold off;
xlabel('Measured MOS');
ylabel('Predicted MOS');
grid on;

view(net);

% F.) Predict MOS scores for the testing dataset using the
trained neural network model

inputTestData = testDataset(1:4, :); % Testing data inputs
(testX from practical 7)
targetTestData = testDataset(5, :); % Testing data target
output (testT from practical 7)

% Predict values for inputData
output = net(inputTestData); % Predicted testing output from
the trained net (testY from practical 7)
perf = mse(net, targetTestData, output) % Calcualte MSE
between testing Target data and testing Predicted data
(output)

% Testing the performance
R = corrcoef(targetTestData, output); % Calculate correlation
coefficients for the testing dataset
R = R(1,2) * 100; % Turn R into a percentage
```

```
R = round(R, 2); % Round R off to 2nd decimal point for better
percentage value

% Draw a scatter plot between the target and predicted output)
figure('Name','Predicted vs measured MOS for testing
dataset');
scatter(targetTestData, output);
title(['For testing dataset (R=',num2str(R),'%)'])
hold off;
xlabel('Measured MOS');
ylabel('Predicted MOS');
grid on;

% Plot regression (with R value displayed in the title of the
graph for the testing dataset)
figure;
plotregression(targetTestData, output);

net % Display the network's specifications in the command
window
```