# UK Energy Demand Forecasting.

Comparing normalised radial basis function and multilayer perceptron networks.


## Keywords

Normalised radial basis function (NRBF), Multilayer perceptron (MLP), generalisation



## Abstract

Neural networks can be used to perform in generalisation tasks, this mean creating a network that learns a rule as to which it can apply to unknown data a produce an output that is correct or within a given tolerance. By using different network, such as normalised radial basis functions (NRBF) and multi-layered perceptron (MLP), a different rule can be generated leading to completely different outputs when applied to unknown data samples. These networks can be used to solve many problems and the one written about in this paper is that of forecasting the energy consumption of the UK every hour.


## 1. Introduction

### 1.2 NRBF

A normalised radial basis function (NRBF) is an artificial neural network that contains an input, hidden and output layer as shown in figure 1. A NRBF network is very similar to a regular radial basis function, only there is a slight difference in how the final output is calculated. They can have multiple inputs and outputs and as many hidden nodes as necessary. The connections between layers are the weightings that are used to calculate an eventual output. The hidden layer uses a Gaussian function to generate an activation function, ϕ, which produces an output between -1 – 1 (figure 2). These nodes have a set size, σ, a range of 0.44*σ and are centred at a point, $X_{ci}$, which is stored in the input weights to the node. These are all used when calculating the activation function as shown in equation 1. If there is a data pattern that is presented to the network and does not sit within the range of a node, a new node is recruited, which has its centres set to be equal to the data that recruited it and the output weight set to a random number between 0-1.

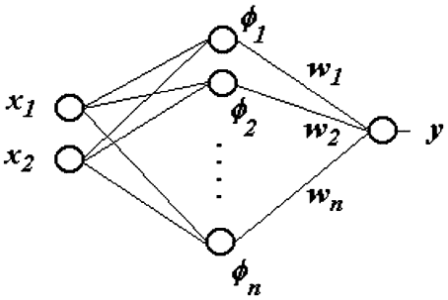


*Figure 1 Architecture of NRBF network (Bugmann, 1998)*

$$\Phi_n = \exp(-\frac{1}{2\sigma}\sum_{i=1}^{j}(x_i - x_{ci})^2) \qquad \text{(Equation 1)}$$

*Where X is the input vector, j is the number of inputs and n is the node id.*

After a few iterations of training, the centres of the nodes will stabilise and stop moving. At this point the centres of the nodes are fully trained.

With a RBF network to generate an output, the activation values are multiplied by the weights connecting them to the output layer, and then summed together as shown:

$$y_i(X) = \sum_{j=1}^{n} w_j . \Phi_j \qquad \text{(Equation 2)}$$

*Where $w_j$ is the output weight.*

Over a range of X this will produce outputs that resemble a Gaussian curve. For a NRBF, the equation is slightly different and produces a different shaped function. The function produced by that of an NRBF function is a "step like" function as shown in figure 3. The equation for this is as shown:

$$y_i(X) = \frac{\sum_{j=1}^{n} w_j.\Phi_j}{\sum_{j=1}^{n} \Phi_j} \qquad \text{(Equation 3)}$$

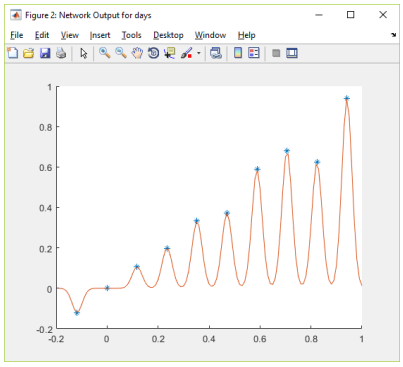


Figure 2 Output of RBF network

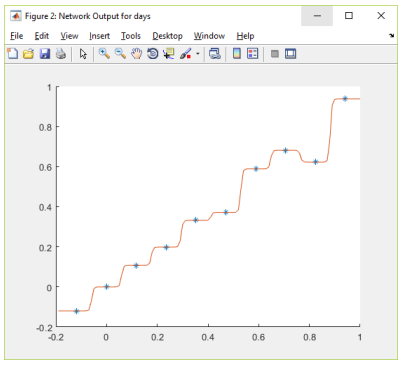


Figure 3 Output of NRBF network

## 1.3 MLP

A MLP network is a network that similarly to NRBF as in it contains input, hidden and output layers. The difference is in that the hidden layer nodes use a sigmoid function rather than a Gaussian as shown by figure 4. Each input is multiplied by its corresponding weight, summed, with the summed value being represented as z, and then passed through the sigmoid function to produce the activation function. The activation function of a sigmoid is as show:


*Figure 4 Sigmoid activation function (Bugmann, 2017)*

$$\Phi_i = \frac{1}{1+\exp(-z_i)} \qquad \text{(Equation 4)}$$

The slope of the sigmoid function is determined by the inputs weights to that node, the higher the weights, the steeper the slope. A steep sloped sigmoid is more suited to classification tasks as the output will be either 0 or 1. Smaller weight values are more suited to generalisation tasks, which is what will be used later on to attempted to generate a forecast for the UK energy consumption. Another difference is that the architecture of a MLP contains a bias node that is passed into every layer of the network (figure 5). The value from this bias node is always 0, but the weight into the next layer is what has an effect on the network. The bias node shifts the activation function along the X-axis. The effectiveness of the bias is also determined by the weights of the other inputs.

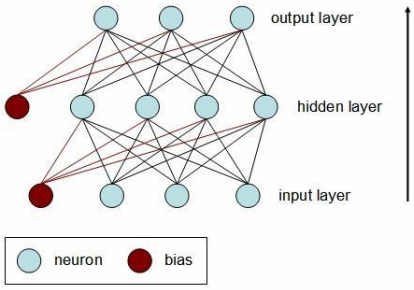


*Figure 5 Mutilayer perceptron architecture (Bugmann, 2017)*

## 2. Task 1

The first task was programming a NRBF neural network using the MATLAB programming language. MATLAB excels at vectors and matrices manipulations, which is how all the data and weights will be stored. All the data is stored into a n by m dimension matrix, with n being the number of inputs + the number of outputs, in this first case the number of inputs being 1 and the number of outputs being 1. m being the number of data samples, which in this first case is 10. The weights are stored in similar matrix, with dimensions p and q, p is the number of inputs + the number of outputs and q is the number of hidden nodes in the network. The number of nodes within the hidden layer changes depending on the size of each node and the range of the input space. With smaller nodes, more are needed to cover the entirety of the input space. A new node is created when there is an input that is presented to the network that does not currently lie within an already existing node, however if the input does lie within a node, the node which is closest to this input is moved slightly closer to that input using the equation:

$$newInWeight = 0.8 * w_i + 0.2 * x_i \quad \text{(Equation 5)}$$

The weights connecting the hidden layer to the output layer is trained using error and the delta rule. On each presentation of a data set the network produces and output, which is compared to the what the actual value should be. The difference in the values is used to calculate an error and then changes are made to the weights. The delta rule for a NRBF network is:

$$\Delta w_i = w_i + learningRate * (yDesired - networkY) * \Phi_i \quad \text{(Equation 6)}$$

## 3. Task 2 overview

This task involved using the previously coded NRBF network and the MLP net fitting application that is supplied by MATLAB to generate a forecast for UK Energy demand. The data used was collected from www.gridwatch.templar.co.uk, which keeps a record of the UK energy demand for every 5 minutes of every hour of every day. All data during the period of 2012-2016 is used to train the network. Data for the 2017 period will then be used to calculate the performance of the networks. This data will not be presented to the network during the training. These two networks will then be compared to see which network best suits the problem of energy forecasting. Both networks will be presented with the same input fields, which will be the days, 1-31, months, 1-12 and hours, 0-23.

### 3.1 Methodology

#### 3.1.1 Data manipulation

The data is initially in the form of two columns inside a CSV file, a column containing a string of the data and time, and one column containing the energy demand. Before being able to present this data to the networks the data must first be split into numerical columns of which will be used later presented. To do this I created a program that went through the CSV file line by line and split the text time and date field into separate numerical fields, a choice can then be made as to what fields can be entered to the network. These fields are then multiplied by the largest number in the column to produce an output that is between 0 and 1. By limiting the size of the values, the inputs have the same distribution over the input spaces (Bugmann, 2017). Data for the MLP is split into separate files, one file containing inputs, and another containing the outputs. The data set is then randomised then split into training and testing data with a 75, 25% split.

### 3.1.2 Optimising the networks

To optimise the NRBF network the σ value was changes ranging from 0.1 to 1 increasing by 0.1 each time. The optimum node size was 0.5. This value produces a low training and more importantly low test error, while also being able to complete training in a reasonable time. Using a σ of 0.1 has a lower training and testing error, but only by 0.004 on testing which can be seen as insignificant. Because of the much smaller size, means there are thousands of nodes, which greatly reduce run time and increase the time needed to compute the 2017 results.

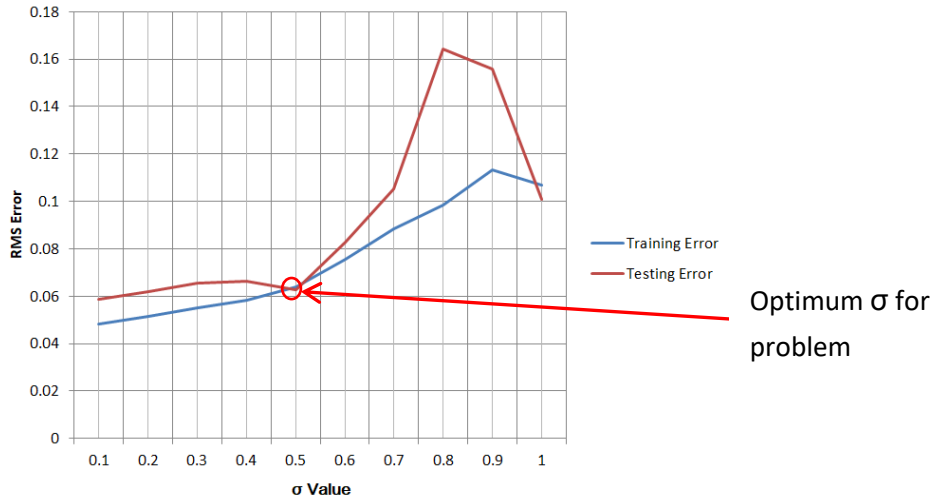


*Figure 6 RMS Error over entire dataset at the end of training*

To optimise the MLP network the number of nodes is changed to optimise the network. Once again, the optimum network was one with a low error rate for the training and testing data as shown in figure 7. The optimum number of nodes for this example is 15, as has a performance of 0.0189 which is similar to the others, but instead using less nodes.

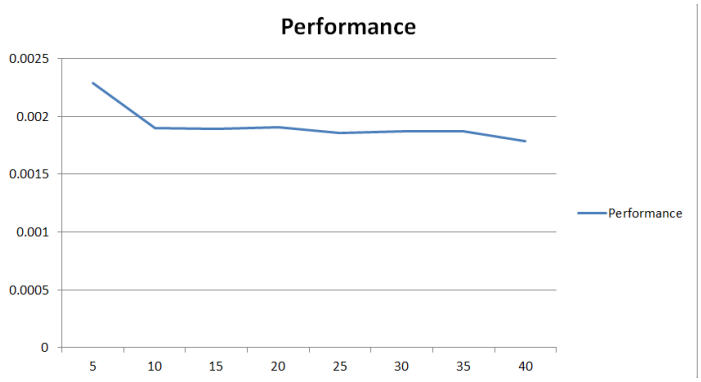


*Figure 7 Performance of MLP with difference size nodes.*

### 3.1.3 Determining the effectiveness of the network

The performance of the networks will be determined by comparing the output of the networks with 2017 data. The entirety of 2017 data will be presented to the optimised networks and the RMS error calculated. The lower the RMS error, the better the learning rule produces, meaning the more effective network for this particular problem.

## 4. Results

### 4.1 NRBF

When presented with the 2017 data set, the network is capable of performing with a RMS error of 0.063. As shown in figure 8, the output from the network generally overlaps with the actual UK energy demand. From the data, there is a clear learning rule that has been generated from the training data. The data appears to be constantly varying, but twelve clear curves can be seen which represent each month. The graph shows that at the beginning of the month the energy demand is high, but decreases though until the middle of the month where it then begins to increase again. It can also be seen that at the beginning and end of the year the energy demand is higher than any other time, also showing a similar shape to the demand each month, but just at a larger scale. When applied to just a single day (Figure 9), the energy demand can be seen peaking at midday.

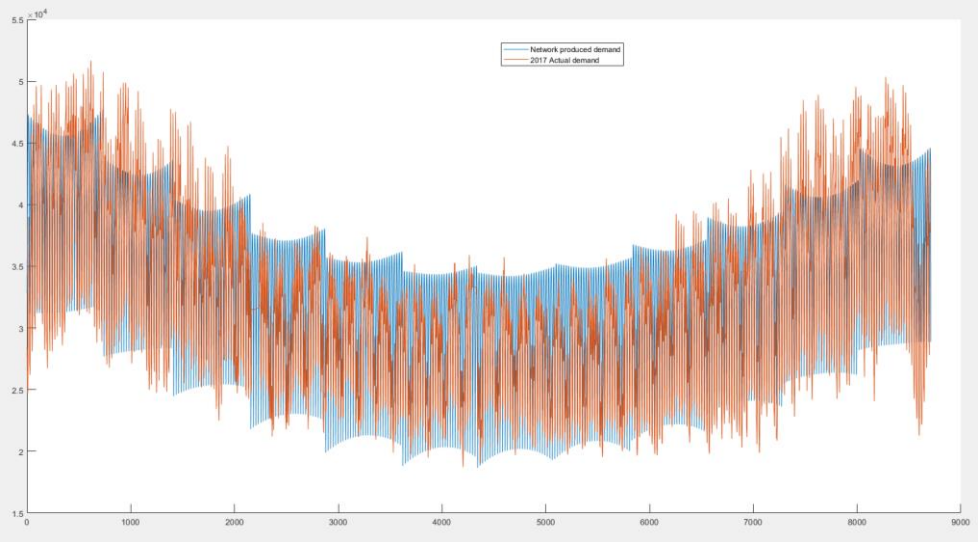


*Figure 8 NRBF network output and actual 2017 data, starting from January 1st 2017, ending December 31st. (X -axis displaying the number of data samples, not dates)*

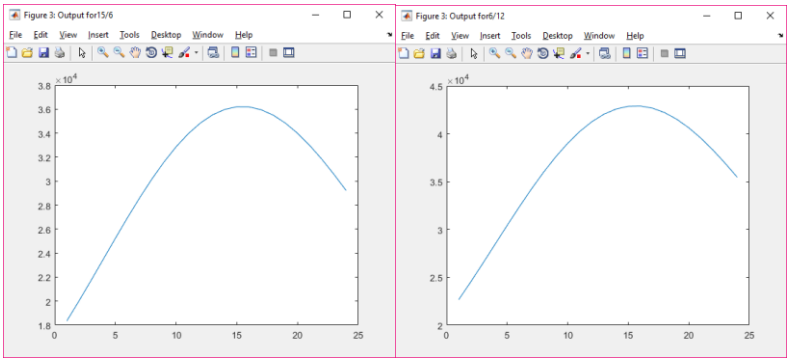


*Figure 9 Generated learning rule applied to a single day of the year*

### 4.2 MLP

The MLP is capable of performing on the 2017 data set and generating a RMS error of 0.003. As seen in figure 9, the network output and the actual 2017 data overlap. The learning rule as shown in figure 9 shows that the energy demand at the beginning and end of the year is at its highest, the data also shows a decrease in energy demand during the middle of the year.

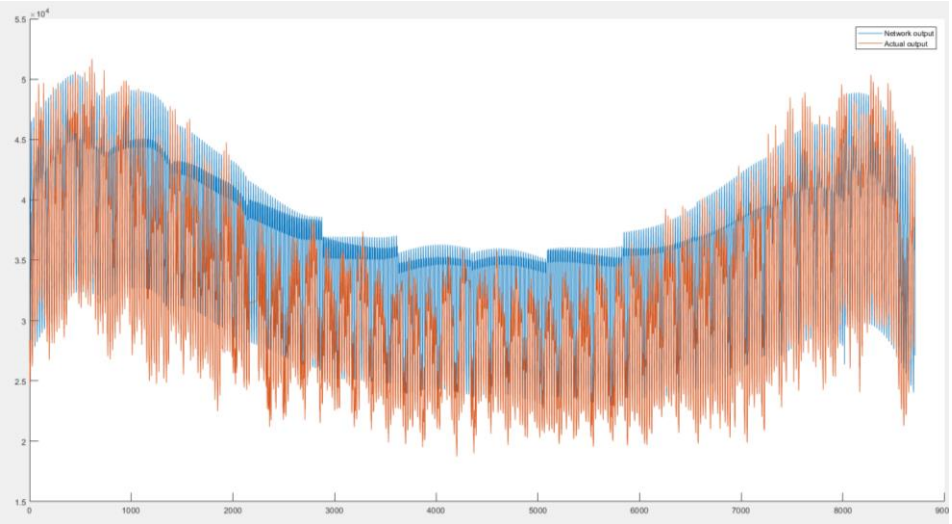


*Figure 9 MLP network output and actual 2017 data, starting from January 1st 2017, ending December 31st. (X a-axis displaying the number of data samples, not dates)*

## 5. Discussion

Due to the number of data samples and the high dimensionality of the data, the time taken to train the NRBF network was sometimes that of hours with low σ values as the network would have to go through a higher number of nodes therefore increasing the computing time. Ideally, more testing could take place to test whether different combinations of inputs performs better or worse, for example, presenting the network with inputs for the hour, day date, month date, and the previous hours energy demand, or the previous year's current date average hour demand.

During the year, there are significant increases in the energy demand due to events such as the world cup of the Olympics. These anomalies are hard to account for as they happen once every four years, and do not land on the exact same days every year. This could possibly be accounted for by creating a new input field containing whether or not there is a significant event-taking place during that hour, with a value of either 0 or 1. This would require an external program or manual entry as grid watch does not supply this information.

## 6. Conclusion

For this problem, it appears that a MLP network is better suited as it is capable of performing with a significantly lower error rate compared to that of the NRBF. It is also capable of performing at much quicker rate than the NRBF. This could be due to the MLP being an application that has been released by the creators of MATLAB, therefore it would have gone through a much more rigorous testing process, and better optimised than the NRBF that was used. This could be taken further by better testing the NRBF optimising the code to perform at a similar speed to the MLP, also by training with different input fields.

## References

Bugmann, G. (2017). *Function Mapping Neural Networks: Issues*. [PDF] p.8. Available at: https://dle.plymouth.ac.uk/pluginfile.php/889409/mod_resource/content/0/3-MAPPING1.pdf [Accessed 9 Jan. 2018].

Bugmann, G. (2017). *Function Mapping Neural Networks: Multilayer Perceptron Architecture*. [PDF] pp.1-4. Available at: https://dle.plymouth.ac.uk/pluginfile.php/889408/mod_resource/content/0/4-MAPPING2.pdf [Accessed 10 Jan. 2018].

Bugmann, G. (2017). Function mapping with Radial Basis Function (RBF) nets. [PDF] p.23. Available at: https://dle.plymouth.ac.uk/pluginfile.php/889406/mod_resource/content/0/6-RBF.pdf [Accessed 9 Jan. 2018].

Bugmann, G. (1998). Normalized Gaussian Radial Basis Function networks. *Neurocomputing*, 20(1-3), pp.97-110.