Application of neural networks to model catamaran type powerboats

Garron Fish Mike Dempsey
Claytex Services Ltd
Edmund House, Rugby Road, Leamington Spa, UK
garron.fish@claytex.com

Abstract

Powerboats in operation represent a system consisting of a number of complex components such as: surface propellers, aerodynamics and hydrodynamics; which interact with each other and with the wind and water surface conditions. By measuring the behaviour of the powerboat it is possible to create a mathematical model using system identification methods. A neural network model has been generated which can be used to predict how the powerboat will perform under different driver inputs for the purpose of optimizing performance.

Keywords: neural networks; system identification; powerboats

1 Introduction

There are many different approaches to mathematical modelling and the decision about the most appropriate method to use is based on what *a priori* knowledge is known about the system. Modelica is typically used for white box modelling, which is based on the application of the universal laws and principles. This paper discusses the use of black box modelling techniques that are entirely based on the use of measurement data to generate the mathematical model [1].

In black box modelling, the inputs and outputs of an unknown system are used to create a model that produces an output “close” to that of the actual system, when supplied with the same inputs. Neural network system identification is one method that can be used to create black box models.

In the case of a powerboat, it is convenient to model the system as a black box, as it is not feasible to model the behaviour of the system as a white box model. Figure 1 shows a class 1 powerboat under race conditions. To create a white box model we would need to create models of the aerodynamic and hydrodynamic effects, and their interaction with the surface propellers and environmental conditions (such as the water surface and wind speed and direction).

The aim of generating a mathematical model of the system was to be able to investigate the effect on the boat performance of variations in driver input and boat setup. A neural network system identification method was selected as the most appropriate way to model the system. Neural network techniques can be very effective at identifying complex nonlinear systems when complete model information cannot be obtained [2].

A Modelica library called ANN_SID has been developed to facilitate system identification using neural networks. The library contains different types of neural network and several training methods and has been applied to study the powerboat system.

Figure 1. View of a powerboat during operation. Image courtesy of Victory Team

2 Neural networks

2.1 An artificial neural network

An artificial neural network is a network of functions called neurons, which are connected by weighted signals (see Figure 2). This architecture is loosely based on a biological neural network. Neural networks can be used for a variety of tasks such as system identification and classification. The ANN_SID library provides neural networks appropriate for system identification tasks.

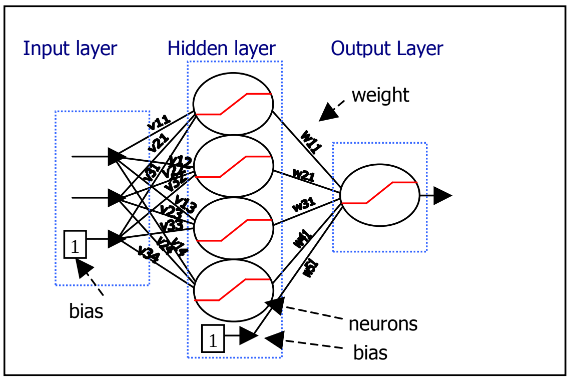

Figure 2. Feedforward neural network

Figure 2 shows a simple diagram of a typical neural network, commonly called a feedforward neural network, which consists of an input layer, a hidden layer and an output layer. In both the hidden and output layers, the weighted sum of the inputs to the layer and the bias, are applied to neuron functions.

The formulae that describe the feedforward neural network in Figure 2 are shown below. Equation (1) calculates the outputs of the hidden layer and equation (2) calculates the output of the neural network.

$$o_j(t) = O_j\left(\sum_i u_i(t)v_{ij} + 1\cdot v_{n+1j}\right) \qquad (1)$$

$$y_k(t) = Y_k\left(\sum_j o_j(t)w_{jk} + 1\cdot w_{m+1k}\right) \qquad (2)$$

where:

o_j is the output of the hidden layer

O_j is the hidden neuron function

u_i is the input

v_{n+1j} is the bias weight for the j neuron (there are n inputs)

y_k is the output of the neural network

Y_k is the output neuron function

w_{m+1k} is the bias weight for the k neuron (there are m hidden neurons)

t is the current sample

Within the ANN_SID library the most common neuron functions such as linear, sigmoid and tanh are available. The user can also easily add their own neuron function by extending from the neuron function base class and implementing the required function.

2.2 Types of neural networks implemented in the ANN_SID library

The ANN_SID library provides pre-defined neural network models for feedforward neural networks and a form of dynamic recursive neural networks called Neural Network Output Error. Within each type of neural network there can be any number of inputs, hidden neurons, neuron layers and output neurons.

In the dynamic recursive neural network, the output of the neural network can be used as an input to the neural network, as shown in Figure 3. This type of recursive network is used for modelling dynamic systems where the next output is affected by the previous output values and previous input values.

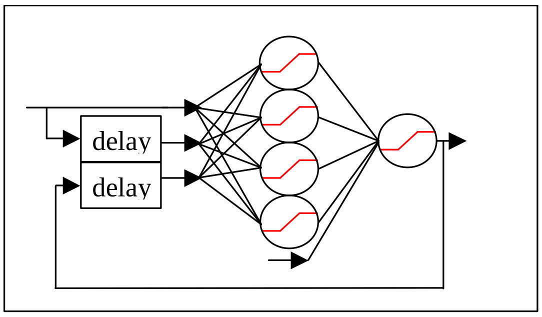

Figure 3. Dynamic recursive neural network

2.3 Training of the neural network

The weights and biases in a neural network have to be trained so that the output of the neural network approximates the actual system well. The mean square error between the actual output and predicted output is the cost function determining the measure of the closeness of the approximation of the neural network to the actual system, as in equation (3).

$$MSE = \frac{1}{2N}\sum_{t=1}^{N}(z(t) - y(t))^2 \qquad (3)$$

where:

MSE is the mean square error

z is the target value (output from the actual system)

y is the output of the neural network

N is the total number of target values

The process of minimising the cost function of the neural network is called training. In the ANN_SID library both backpropagation and the Levenberg-

Marquardt training methods are available. These methods have been implemented in Modelica in both continuous and discrete forms. The choice of method to train a neural network is influenced by the size of the neural network and the amount of data being used to train the network.

The continuous training methods have the advantage that the gradient, which is the rate of change of the weights, is accurate everywhere, not only at the linearization points as with discrete methods. This can result in the search method travelling along the bottom of valleys of the cost function and not oscillating along valley walls.

The continuous method interacts with the variable step solvers to determine the step-size. If the gradient changes suddenly then the solver will reduce the step size to deal with this efficiently. The disadvantage of the continuous method is that it generates huge numbers of equations due to the way that Dymola expands the for loops used in the model. By using Modelica functions and external C functions these problems can be minimized through the reuse of code sections.

Data storage and the manipulation of large matrices in Dymola can also generate problems with large neural networks if the continuous training methods are used. The discrete methods have been implemented to overcome these issues.

2.3.1 Backpropagation

It is possible to train a neural network by calculating the gradient of the cost function with respect to the weights, and to then adjust the weights in the appropriate direction to reduce the cost function. This method is called backpropagation and can be slow to converge to a solution. Appendix A has further information about how the gradient is calculated.

2.3.2 Levenberg-Marquardt

The Levenberg-Marquardt training method generally requires fewer iterations than the backpropagation method to train a neural network. However the LM method is more complex and requires more computation and memory to perform each iteration.

The rules used to calculate the weights are described in Appendix B.

2.3.3 Recursive method

In this method the partial derivative of the neural network with respect to the weights is required. From this partial derivative the gradient and Hessian matrices can be calculated. Once we have determined these matrices either the backpropagation or Levenberg-Marquardt training methods can be used to minimise the cost function.

Modelica provides semantics to define partial derivatives and Dymola is able to utilise these semantics to generate the symbolic derivative of functions. Example 1 shows how the partial derivatives are defined in Modelica. This method was used to help define a function to calculate the partial derivatives of the neural network with respect to the weights.

```
[Example: The specific enthalphy can
be computed from a Gibbs-function as
follows:
function Gibbs
input Real p,T;
output Real g;
algorithm
...
end Gibbs;

function Gibbs_T=der(Gibbs, T);

function specificEnthalpy
input Real p,T;
output Real h;
algorithm
h:=Gibbs(p,T)-T*Gibbs_T(p,T);
end specificEnthalpy;
]
```

Example 1. An example of Modelica code for the generation of the partial derivative of a function. Quoted from Modelica 3.0 Specification [4]

2.3.4 ANN_SID Implementation

An example of training a neural network using the ANN_SID library is shown in Figure 4. The training methods are implemented in the replaceable training component and the user simply selects the required method.

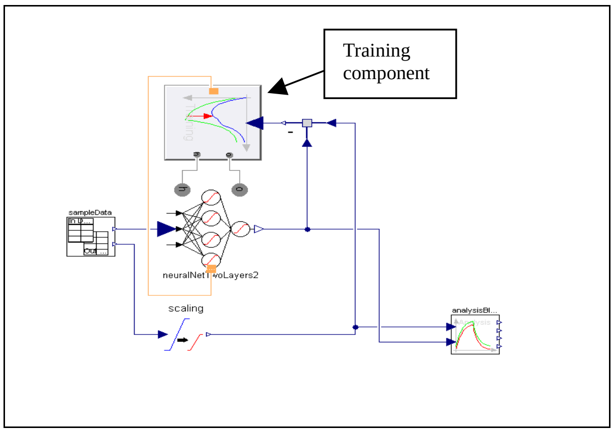

Figure 4. ANN_SID training performed in a model

2.4 Improving the neural network training

When training a neural network it is important to have confidence that the neural network will approximate well with inputs that are not part of the training data. This ability is known as generalisation [5]. One way to investigate this is to divide the data into two sets, one that is used to train the neural network (training data), and one that is used to test the generalisation of the neural network (test data).

Generalisation is likely to be improved by reducing the number of weights used in the neural network [3]. The ANN_SID library supports both weight decay and pruning methods to reduce the number of weights and improve generalisation.

Two pruning methods are available in the library and these are known as Optimal Brain Surgery and Optimal Brain Damage. These algorithms determine which weights to remove from the neural network. The remaining weights are then updated to reduce the errors introduced by removing the weights (for further details refer to [3]).

Weight decay is another approach to removing weights from a neural network. In this method a penalty proportional to the magnitude of the weights is added to the cost function (see [3] for further details). All cost functions should contain a measure of the closeness of the neural network outputs to the desired output. Adding a weight penalty to the cost function generates a trade off between reducing the magnitude of the weights and reducing the closeness measure. As a result of this the weights that have little effect on improving the closeness measure will now be reduced in magnitude.

3 Powerboat operation

The type of powerboat that has been modelled using the ANN_SID library is a Victory Team class 1 offshore powerboat as shown in Figures 1 and 5. These boats have a catamaran hull with two engines and a central rudder. Each engine drives a height adjustable, steerable propeller. The boats are operated by two crewmembers: a throttle man and a driver. Between them they have 5 controls, which are:

- A steering wheel that directly controls the rudder angle. The steered angle for the propellers is also controlled by the steering wheel angle.
- The propeller heights are set using two rocker switches. These control the trim pistons that move the propellers vertically.
- Throttle position is set using the throttle levers for the left and right engine.

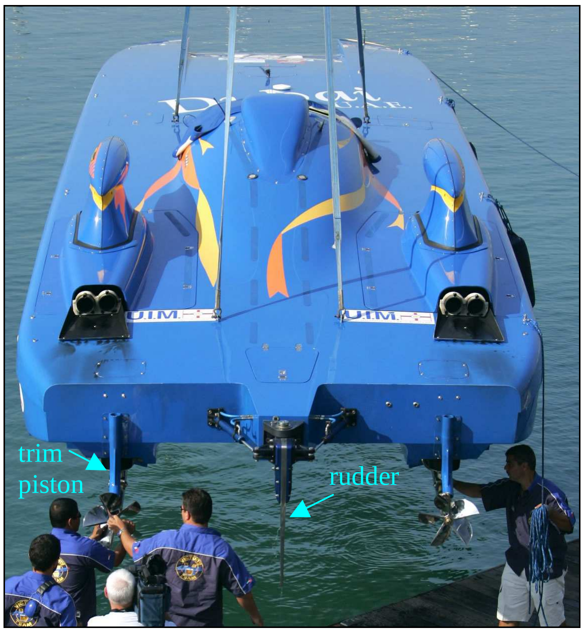

Figure 5. Rear view of a powerboat. Image courtesy of Victory Team

As the boat accelerates it begins to plane and travels higher above the water, i.e. less of the hull is below the water line. As the boat lifts out of the water, the propellers are lowered (trimmed down) to control their depth in the water.

The trim height (or propeller depth) also affects the pitch angle at which the boat travels. In general, the lower the depth of the propellers, the lower the pitch. If the boat is travelling at a pitch angle that is too high for the speed it is doing, it will flip over (a blowover). If the pitch angle of the boat is too low the result will be a larger surface area of the boat in the water and thus an increase in drag.

When cornering, the catamaran powerboat rolls to the inside of the corner (due to the asymmetrical hull design). If the cornering is too severe for the current speed, the boat will begin to roll to the outside of the corner, and will roll over if the drivers do not take correcting action. A typical cornering manoeuvre requires the throttle man to reduce the throttle to slow the boat to a controllable speed before the corner, and the driver to steer the boat along the course, ensuring that the steering angle is not too steep for the current speed.

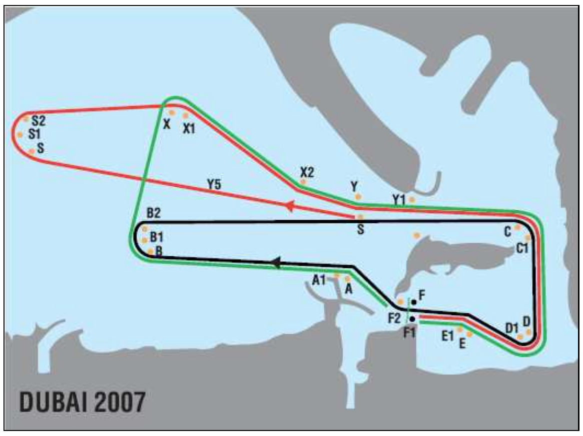

Figure 6. A Racecourse. The powerboats must travel along a course defined by buoys. There are three different types of laps. The start lap in red, the short lap in black and the long lap in green. Supplied courtesy of IOTA.

A race involves the boats travelling around a course defined by buoys laid out in the water (see Figure 6). The drivers try to select good trim height to maximize acceleration while maintaining stability. The drivers also try to find good throttle, rudder and trim positions for cornering that result in fast and stable cornering. The neural network can investigate different possible driver inputs and predict their effects on boat performance over a lap.

4 Powerboat model

4.1 Defining the neural network

To model the powerboat using neural network techniques, a significant amount of data is used to characterise the system. During races and testing sessions the boats are fitted with a data logger that records the data required to train the neural network.

The input data required is:

- The engine throttle positions
- The rudder angle
- The trim height of both propellers

The target output data required is:

- The engine speed of both engines
- The boat speed
- The yaw rate of the boat

Using this input and output data we can train a neural network to represent the powerboat system and then use the neural network to investigate the system performance with different inputs.

As the boat is an example of a dynamic system, the dynamic recursive neural network was chosen. The model has been generated from data recorded by Victory Team from their racing boat number 77 during the 2007 Arendal race. During this race the boat completed 12 laps of the course.

The measured data was filtered using Basel and Chebyshev filters to reduce the amount of noise and high frequency components in the data. The filtered data was then re-sampled from 100Hz to 1.7Hz to reduce the number of duplicate data points and to decrease the amount of time required to train the neural network. Finally the data was divided into training and test data sets.

4.2 Training the neural network

Training a neural network for such a complex system is done in a number of steps. When first training a dynamic recursive neural network it is not known how many past outputs and inputs will result in the model giving a good representation of the powerboat system. It is also not known how many neurons will be required, or which neuron functions should be used. These can only be determined by trying different configurations to find the best setup.

The first step in training this type of neural network is to train it to only predict the next output value from the previous data value. The weights from this training are then used as the initial weights for the recursive training. The recursive training algorithm described in 2.4.3 was used with the Levenberg-Marquardt method to train the neural network. To improve the generalization, weight decay was used.

4.3 Correlation results

After training, the MSE for the neural network using the training data set was 0.0035 and the MSE for the test data set was 0.0064. This means that the neural network has been trained successfully and is able to accurately predict the performance of the powerboat, as shown in Figure 7.

In Figure 7, the recorded driver inputs have been fed in to the trained neural network and the outputs for boat speed, engine speed and the yaw rate of the boat are compared to the measured data. Overall the results show that there is very good correlation between the neural network and the real powerboat.

There are some small deviations which could be due to a number of different factors, such as swell and wind conditions along the course, that are not accounted for in the neural network.

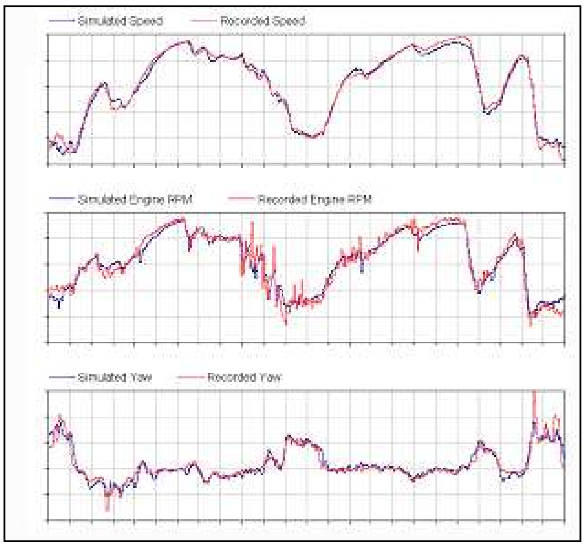

Figure 7. Comparison of simulated neural network with recorded data for a single lap of the course.

4.4 Optimisation of the trim strategy

Section 3 describes how the propeller height (trim height) affects the performance of the powerboat. By using the neural network it is possible to determine what the optimum trimming strategy is for the powerboat.

The model shown in Figure 8 uses the trained neural network to simulate the powerboat accelerating from an initial speed up to its maximum speed.

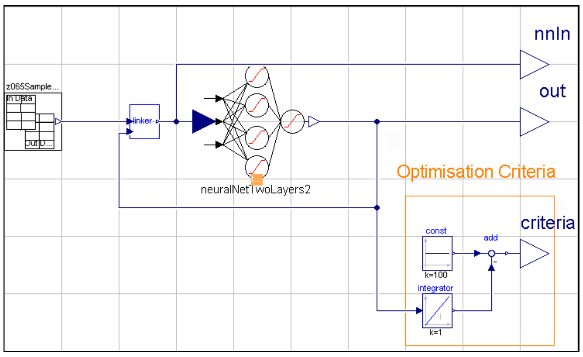

Figure 8. Acceleration model test. The throttle position is set to 100% and the boat is travelling in a straight line.

Figure 9 compares an example trimming strategy extracted from the race data and the optimized trim strategy that has been determined with the use of the neural network. Using the optimised trimming strategy the powerboat would take 1s less to travel along a 2km straight than using the example trim strategy.

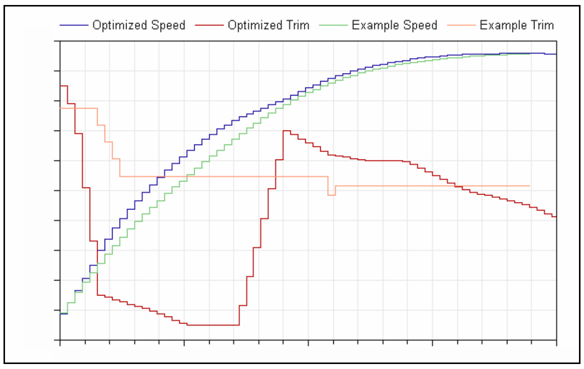

Figure 9. Comparison of a simulated trim strategy with a real trim strategy. The simulated optimal results (Simulated Speed and Simulated Trim) assume the boat is travelling perfectly straight. The Example Trim strategy was taken from the race data and applied to the simulator.

The neural network used in the model can only be expected to accurately model an operating region if this region was sufficiently excited during the data recording stage. In Figure 10 the histogram data identifies what trim position data is available for the operating region of the simulated result. The optimum trim strategy is limited by the availability of data (see Figure 10). The upper bound on the trim data is probably due to driver caution because of the risk of a blowover in this operating region.

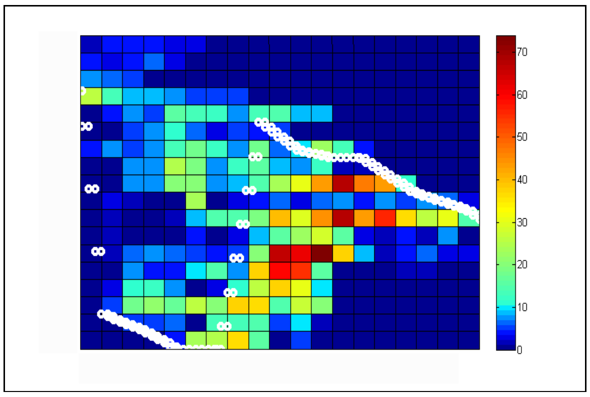

Figure 10. Histogram plot of recorded trim position at the operating state. The optimised trim position is plotted over the histogram as white circles

5 Conclusions

A library called ANN_SID has been developed for the development and training of neural networks for system identification. This library was used to generate a black box model of a powerboat, and this model was then used to determine an improved trimming strategy that should deliver improvements in boat performance.

Acknowledgments

Victory Team have kindly provided the photographs used in this paper and the powerboat data used to carry out this study.

IOTA have kindly provided the course map used in Figure 9.

References

[1] Estrada-Flores S., et-al. Development and validation of “grey-box” models for refrigeration applications: a review of key concepts. International Journal of Refrigeration, pages 931-946, July 2006.

[2] Wen Yu, Nonlinear system identification using discrete-time recurrent neural networks with stable learning algorithms. Information Sciences, pages 131-147, 2004.

[3] M. Nørgaard, O. Ravn, N.K.Poulsen and L.K.Hansen, Neural Networks for Modelling and Control of Dynamic Systems

[4] Modelica 3.0 Specifications
http://www.modelica.org/release_of_modelica_3_0/view

[5] Neural Network FAQ, part 3 of 7, ftp://ftp.sas.com/pub/neural/FAQ3.html

APPENDIX A: Backpropagation algorithm

By calculating the gradient of the cost function (see Section 2.3) it is possible to update the weights in a way that will reduce the cost function. The example below is how backpropagation would be used to update a feedforward neural network using the MSE as the cost function.

The following equations describe a neural network.

$$o_j = O_j(\sum_i u_i v_{ij} + 1\cdot v_{n+1j}) \qquad (1)$$

$$y_k = Y_k(\sum_j o_j w_{jk} + 1\cdot w_{m+1k}) \qquad (2)$$

The cost function is the mean square error (i.e. MSE):

$$V = \frac{1}{2N}\sum_{t=1}^{N}(z(t)-y(t))^2 \qquad (3)$$

The calculation of the partial derivative of MSE with respect to output weight w_{jk} follows:

$$\frac{\partial V}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}(\frac{1}{2N}\sum_{t=1}^{N}(z(t)-y(t))^2)$$

$$\frac{\partial V}{\partial w_{jk}} = \frac{-1}{N}\sum_{t=1}^{N}\varepsilon_k(t)\frac{\partial y_k(t)}{\partial w_{jk}} \qquad (\text{let } \varepsilon = z(t)\text{-}y(t))$$

Substituting in (2):

$$\frac{\partial V}{\partial w_{jk}} = \frac{-1}{N}\sum_{t=1}^{N}\varepsilon_k(t)\frac{\partial Y_k(\sum_j o_j(t)w_{jk} + 1\cdot w_{m+1k})}{\partial w_{jk}}$$

$$\frac{\partial V}{\partial w_{jk}} = \frac{-1}{N}\sum_{t=1}^{N}\varepsilon_k(t)\frac{\partial y_k}{\partial a_k}\cdot o_j(t)$$

(where $a_k = o_j(t)w_{jk} + 1\cdot w_{m+1k}$)

The calculation of the partial derivative of the cost function with respect to hidden weight follows:

$$\frac{\partial V}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}}(\frac{1}{2N}\sum_{t=1}^{N}(z(t)-y(t))^2)$$

$$\frac{\partial V}{\partial v_{ij}} = \frac{-1}{N}\sum_{t=1}^{N}\varepsilon(t)\frac{\partial y(t)}{\partial v_{ij}}$$

(note that ε and y are vectors)

$$\frac{\partial V}{\partial v_{ij}} = \frac{-1}{N}\sum_{t=1}^{N}(\sum_k \varepsilon_k(t)\frac{\partial y_k(t)}{\partial a_k(t)})\frac{\partial(o_j(t)w_{jk} + 1\cdot w_{m+1j})}{\partial v_{ij}}$$

$$\frac{\partial V}{\partial v_{ij}} = \frac{-1}{N}\sum_{t=1}^{N}(\sum_k \varepsilon_k(t)\frac{\partial y_k(t)}{\partial a_k(t)})w_i\frac{\partial o_j(t)}{\partial b_{ij}(t)}u_i(t)$$

where: $b_{ij}(t) = O_j(\sum_i u_i(t)v_{ij} + 1\cdot v_{nj})$

The discrete weight update method is:

$$\Delta w_{jk} = -\eta \frac{\partial V}{\partial v_{ij}}$$

By choosing η sufficiently small, the cost function can be decreased at each iterate.

The continuous method uses the gradient calculated above to update the existing weights continuously.

APPENDIX B: Levenberg-Marquardt algorithm

In the backpropagation algorithm the search direction is calculated from the first order Talyor approximation of the cost function. The Levenberg-Marquardt algorithm makes use of the second order Talyor approximation of the cost function to update the weights. The second order approximation of the cost function follows:

$$\hat{V}(\theta) = V(\theta^*) + (\theta - \theta^*)V'(\theta^*) + \tfrac{1}{2}(\theta - \theta^*)V(\theta^*)''(\theta - \theta^*)$$

$$\hat{V}(\theta) = V(\theta^*) + (\theta - \theta^*)G + \tfrac{1}{2}(\theta - \theta^*)\, H(\theta - \theta^*)$$

where:

θ represents all the weights in the neural network

θ* are the weights at which the Taylor approximation is made.

V' is $dV/d\theta$ and equal to the gradient G

V'' is $d^2V/d\theta^2$ and equal to the Hessian H

In the Levenberg-Marquardt method a further approximation is made; the Hessian is approximated by the following equation:

$$R(\theta) = \frac{1}{N}\sum_{t=1}^{N} \frac{dy(t)}{d\theta}\frac{dy(t)}{d\theta}$$

This is valid when the MSE is the cost function.

Let the approximation of the cost function be:

$$L(\theta) = V(\theta^*) + (\theta - \theta^*)G + \tfrac{1}{2}(\theta - \theta^*)\, R(\theta - \theta^*) \qquad (4)$$

This cost function is minimised using an iterative process; where the next weights are limited to a region around the current weights (see (5)). Limiting the range of the search is often effective as “If the minimum of L is far from the current iterate, $\theta^{(i)}$, a poor search direction may be obtained.” [3].

$$\theta^{(i+1)} = \arg\min_{\theta} L^{(i)}(\theta)$$

$$\text{subject to } |\theta^{(i+1)} - \theta^{(i)}| \le \delta^{(i)} \qquad (5)$$

where:

$\lambda^{(i)}$ has a monotonic relationship with $\delta^{(i)}$ [3]. Where increasing $\lambda^{(i)}$ decreases $\delta^{(i)}$ and visa versa.

The weights are updated using the following rule:

$[R(\theta^{(i)}) + \lambda^{(i)}\mathrm{I}]\Delta\theta = -G(\theta)$

where:

$\Delta\theta = \theta^{(i+1)} - \theta^{\,i)}$

The update rule for the λ value follows:

1. If the $L^{(i)}$ value approximates MSE well, then $\lambda^{(i+1)} = \lambda^{(i)}/2$ and thus increasing the search region.
2. If the $L^{(i)}$ value does not approximates MSE well, then $\lambda^{(i+1)} = \lambda^{(i)}*2$ and thus decreasing the search region.
3. Leave λ^* if neither the 1 or 2 thresholds are true.

To get a more detailed explanation on the update rule for λ^* refer to [3].