Imperial College
London

Comparative study of Levenberg-Marquardt Method and Backpropagaition Method for ANN Training

Theo Franquet

4 July 2016

Contents

1 Introduction

1.1 Purpose

This paper is made in continuity with the Estimation of Vehicle Velocity for Future Horizon in Powertrain Optimal Energy Managment article, written by Johan Tillmar [1], in which a complete artificial neural network has been designed and tested, intended to predict a vehicule's future velocity, based on diverse driving parameters. The network will be explained in more detail over the course of this article.
This study aims at comparing the performance of two training methods for the established neural network. It will justify a specific testing protocole and present the results of the various experiments.
The neural network has been designed within MATLAB's environment which allows for easier matrices manipulations and offers rich online ressources. MATLAB is also very effective for data extraction and analysis, which is essential in view of our objective to evaluate the network's perfomance.

1.2 Background

Artificial Neural Networks (ANN) are defined as a computer systems modelled on the human brain [2]. They contain underlying entities known as neurons that are interconnected with each other to produce a complex, multi-layered structure with input and output neurons. A neural network is used to apply a non-linear function, without knowing the expression of the function initially. Similarily to the human brain, in order to obtain accurate outputs, the network has to be trained facing multiple possible input scenarios.
Choosing the right training method has a big impact on the overall practical accuracy of a network and multiple algorithms have been developped in recent history to diversify the possibilities and optimise network performances. However, as neural networks are not all identical on many levels, there is no individual stand out algorithm that outperfoms all others for all networks, and several methods have to be implemented and compared.

2 Neural network structure

2.1 Overview

The ANN of interest has been designed and structured as a simple feedforward network, where the signal propagates from the inputs to the output through a hidden layer. Its anatomy is illustrated below.

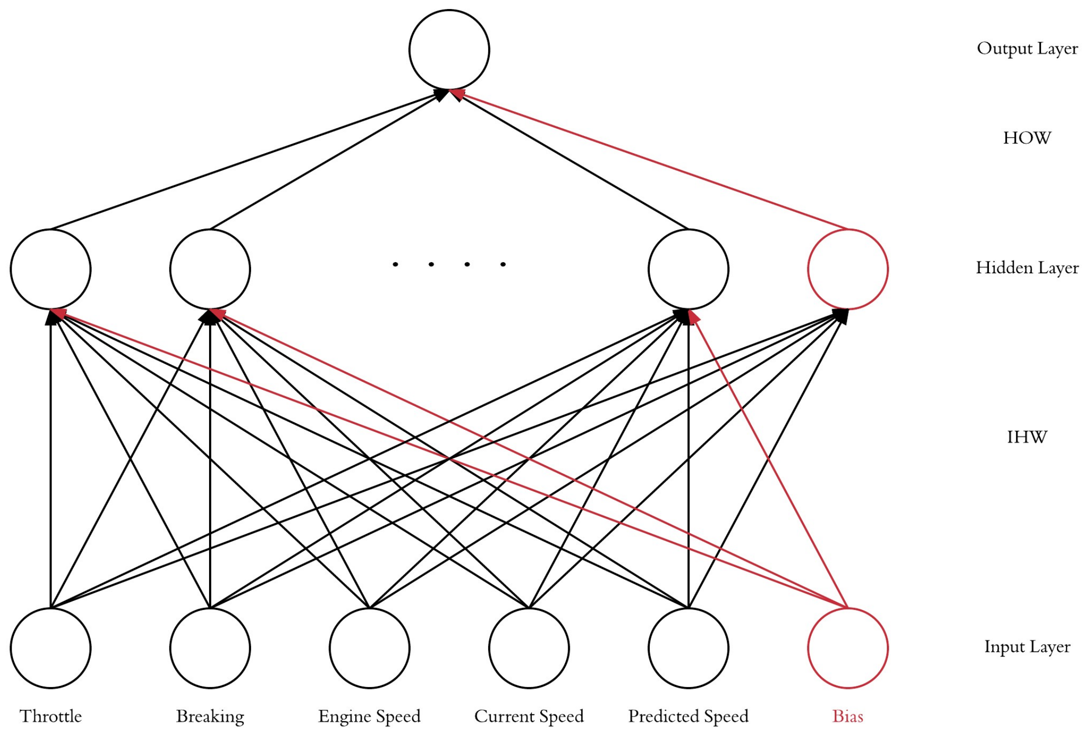


Figure 1: General ANN arrangement

Each connection from one neuron to the other is characterised by a unique **weight** which will influence the overall input perceived by the neurons, which can be expressed as:

$$h_{\mathrm{j}} = \sum_{k} \xi_{\mathrm{k}} W_{\mathrm{jk}}$$

for neurons of the hidden layer, where h is the input value of a hidden layer neuron, j is the hidden layer neuron index, ξ is a normalized input value, W is the branch weight and k is the index of neuron inputs. In Figure 1, IHW and HOW respectively refer to Input-Hidden Weights and Hidden-Output Weights. They are vectors enclosing all weight values for each network layer.

Each neuron of the network can have multiple inputs and outputs, however the outputs of each neurons are determined by a single, non-linear **activation function**, namely the Tansig function:

$$f = \frac{2}{1 + e^{-2x}} - 1$$

The Tansig function has been chosen over other mappings due to its symmetry around zero and output boundaries in the range [-1:1], leading to faster training of the network [6]. To properly use the function, inputs to the network are normalized using a linear min-max mapping method:

$$y = (y_{\max} - y_{\min}) \times \frac{x - x_{\min}}{x_{\max} - x_{\min}} + y_{\min}$$

where x_{min} and x_{max} are minimum and maximum reading values, and y_{min} and y_{max} and minimum and maximum normalized values, set to -1 and 1 respectively.
Biasing is used in the input and hidden layer levels to shift the activation function to the left and right, to achieve a more desirable mapping. Step by step propagation is explained in Johan Tillmar's report in more detail.

2.2 Parameters

Neural networks can be described by several parameters that dictate their structure and functionning. These parameters can be either static, in which case they are entered by the designer initially, and won't be changed during network operation, or parameters can be dynamic and change as the network is active [3].

2.2.1 Static parameters

Number of Layers: 1 hidden layer. **Number of units in each Layer:** 5 input neurons + 1 bias neuron for the input layer. There are 10 units in the hidden layer with another biasing neuron. The output layer only contains 1 neuron. **Number of Inputs and Outputs**: There are 5 parameter inputs and 1 output (see Figure 1). However, it is important to note that each input neuron samples new inputs periodically. These periods are fixed by a constant delay and feedback delay.

2.2.2 Dynamic parameters

Learning Rate: Using adaptive learning rate that increases when the total error is reduced and decreases when error increases. A flexible learning rate allows for a better generalization of the network. **Momentum:** An extention to the backpropagation algorithm that allows the network to escape a local minima by increasing the step taken when changing the weights of the connections.

3 Neural network training

This section will quickly go over the two training methods and explain which algorithm is expected to perform better for our specific network. Detailed explainations of the algorithms have been made in Johan Tillmar report.

3.1 Gradient descent backpropagation method

The gradient descent with adaptive learning rate backpropagation method is a widely used algorithm for ANN training. We first define a cost function, then minimize that cost function for each of the connection between the layers, by evaluating a new weight for each connection. The process is done over many iterations to train the network to face as many scenarios as possible.

3.2 Levenberg-Marquardt method

The Levenberg-Marquardt (LM) method is another useful training algorithm, based on both the gradient descent backpropagation method and the Gauss-Newton method for reduction of least square error. The weights will be updated following:

$$\mathbf{\Delta w} = -\mathbf{H}^{-1}\mathbf{g}$$

with **w** the weights, **H** the Hessian matrix and **g** the n-dimentional gradient vector:

$$\mathbf{g} = \frac{\partial E(\mathbf{x},\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2} & \cdots & \frac{\partial E}{\partial w_\mathrm{N}} \end{bmatrix}^T$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 E}{\partial {w_1}^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_\mathrm{N} \partial w_1} & \frac{\partial^2 E}{\partial w_\mathrm{N} \partial w_2} & \cdots & \frac{\partial^2 E}{\partial {w_\mathrm{N}}^2} \end{bmatrix}$$

3.3 Cost function and theoritical comparaison

Both training methods use a cost function, which evaluates the mean squared error between target and output for each neuron. This cost function can simply be written as:

$$E(w) = \frac{1}{2}\sum_i (\varsigma_\mathrm{i} - O_\mathrm{i})^2$$

with ς the target and O the output of the network.
The network has however been optimized and regularized using an enhanced version of the cost function [4]:

$$E(w) = \gamma(\frac{1}{2}\sum_i (\varsigma_\mathrm{i} - O_\mathrm{i})^2) + (1-\gamma)(\frac{1}{n}\sum_{j=1}^{n}({w_\mathrm{j}}^2))$$

with γ the performance ratio, a value in the range [0:1]. We define the term:

$$\frac{1}{n}\sum_{j=1}^{n}({w_\mathrm{j}}^2)$$

as the mean of the sum of squares of the network weights (MSW).
Including the MSW in the cost function pushes the network to have smaller weights and biases, leading to a smoother reponse and less overfitting. Overfitting, as opposed to generalization, can be described as performance degradation when the network faces inputs that have never been used in training before.
In terms of performance, it is difficult to know which training method is best. It depends on many factors such as the complexity of the problem, the characteristics of the training set, the network's parameters as well as the purpose of the ANN itself. It is also important to know if we wish to priorotize speed or accuracy for our application as both factors are not entierly correlated from one method to the other.
It has been shown that within MATLAB's built-in neural network functions, the LM method is the quickest and performs with great accuracy for networks using a single hidden layer.

3.4 Other training methods

Other algorithms have been compared with the LM method by Mathworks and results of their study are shown below:

Algorithm	Acronym	Mean Time (s)
Levenberg-Marquardt	LM	1.14
BFGS Quasi-Newton	BFG	5.22
Resilient Backpropagation	RP	5.67
Scaled Conjugate Gradient	SCG	6.09
Conjugate Gradient with Powell/Beale Restarts	CGB	6.61
Fletcher-Powell Conjugate Gradient	CGF	7.86
Polak-Ribire Conjugate Gradient	CGP	8.24
One Step Secant	OSS	9.64
Variable Learning Rate Backpropagation	GDX	27.69

Figure 2: Various Training Method Performance

These test have been made using MATLAB's built in ANN functions, by Mathworks [5]. Here, the time to reach an error of 0.002 has been measured 30 times and averaged out. The network is structured following a 1-5-1 scheme, which is comparable to the one used for the project.

4 Comparaison Protocol

The objective of this study is to confirm that the LM training algorithm is indeed the best choice for our application. Its performance will be compared with the one of the gradient descent with adaptive learning backpropagation method which has been described previously.

4.1 Code topology and hardware implementation

The end objective is to implement the neural network onto a microprocessor embedded in the vehicle's control system. The microprocessor is necessary as miniaturisation and cost contraints have to be taken into account. Using such hardware leads to limited computational power and memory capacity. We are therefore interested in a network that operates with as little hardware and power requirements as possible. MATLAB's language cannot directly be used to program a microprocessor, and the equivalent network has to be coded in a lower level language such as C. Although there will be significant changes in syntax and data manipulation functions when converting to C, the training algorithms will remain the same and their memory and computational requirements will be relatively comparable from one language to the other.
To evaluate which training algorithm will be more suitable for hardware implementation, both networks will run on the matlab environment and the computer's memory (RAM) usage and CPU load will be monitored. Each version of the code will run 5 times and results will be averaged for better accuracy.

4.2 Output Accuracy

Accuracy is the most important paramter when it comes to comparing two networks. The objective being to predict the future velocity of the vehicle and managing energy flow accordingly within the powertrain, the network has to be accurate enough to be effective and make a difference in optimizing the overall vehicule efficiency. Johan Tillmar's paper presents two justifications that reflect a network's accuracy: Covariance of output with actual velocity and the comparaison of the network's output and the true velocity of the vehicule.

4.3 Excecution speed

To evaluate excecution speed, we will re-create MATLAB's experiments that measures the time for the network to reach an output error of 0.002. The two versions of the code will run on the same computer under the same conditions.

5 Results and discussion

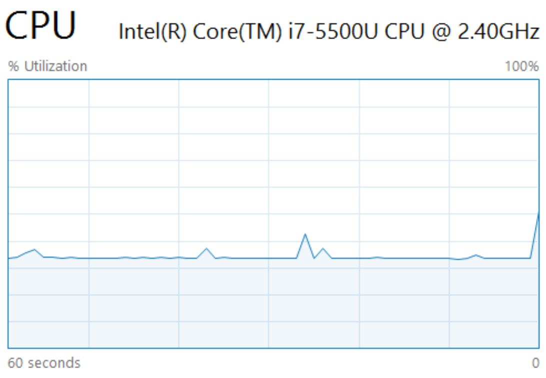
5.1 Hardware and computing requirements

Here are results of the testing for CPU and RAM usage while executing the MATLAB scripts for learning, validating and testing. All measurements stay constant throughout the 100 iterations that the networks works through.

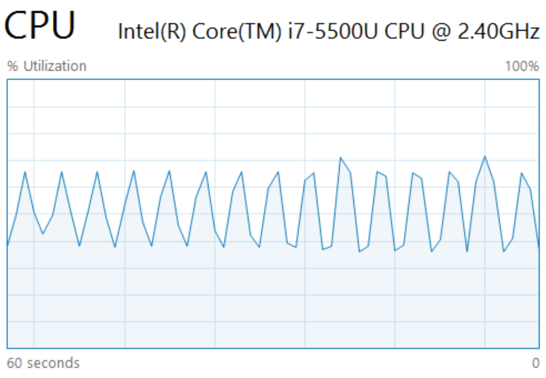
Algorithm	Mean CPU usage (%)	Mean RAM usage (Mb)
Levenberg-Marquardt	50	760
Back Propagation	32	1110

Figure 3: Hardware requirements

Results show that CPU and RAM usage are significantly different from one algorithm to the other. The LM method requires greater computational power and memory capacity. This also implies higher power consumption if the algorithm was to be implemented onto a microprocessor. We can also note that the LM algorithm demands computation in burts, with each peak corresponding to a new iteration:



(a) Backpropagation method



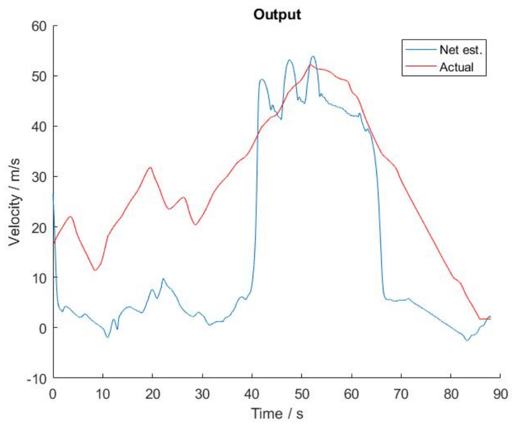
(b) LM method

Figure 4: CPU loading comparaison

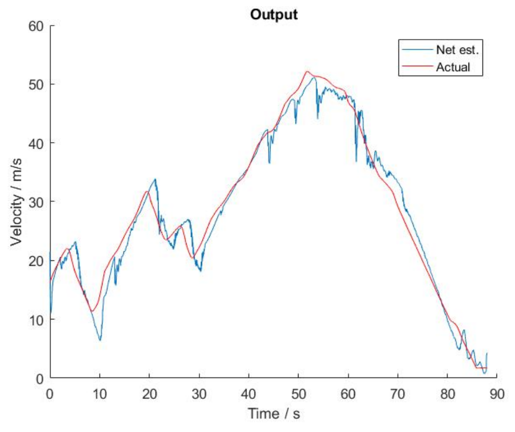
Observing our results, we can state that the LM algorithm is significantly more demanding in terms of processing power and memory allocation. Operating the simple backpropagation algorithm onto a microproccessor would be generally better as computational power and memory are aimed to be as low as possible for a viable end product. It is however important to know if the superior computational need of the LM algorithm will bring a much improved accuracy and excecuting speed next to the backpropagation algorithm.

5.2 Network accuracy

Below are the results for both network output and target output to true output covariance:

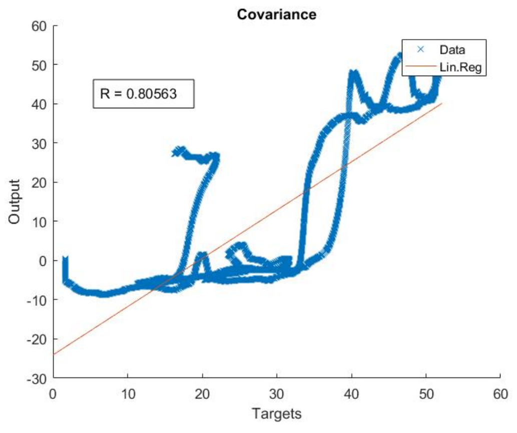


(a) Backpropagation method

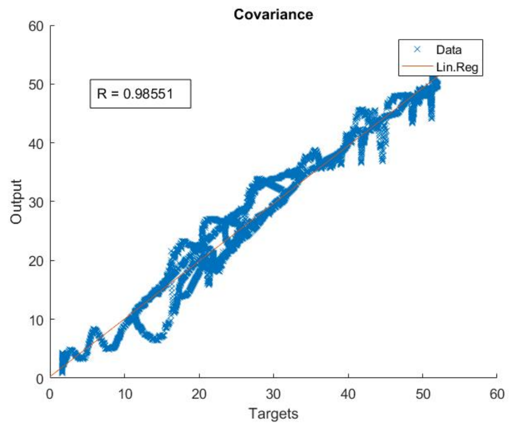


(b) LM method

Figure 5: ANN output vs. actual vehicule velocity



(a) Backpropagation method



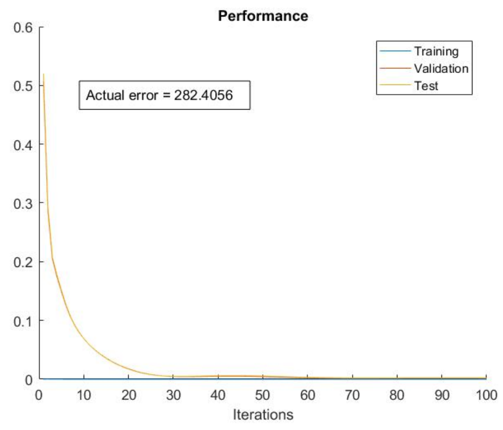
(b) LM method

Figure 6: Covariance between ANN output and target output

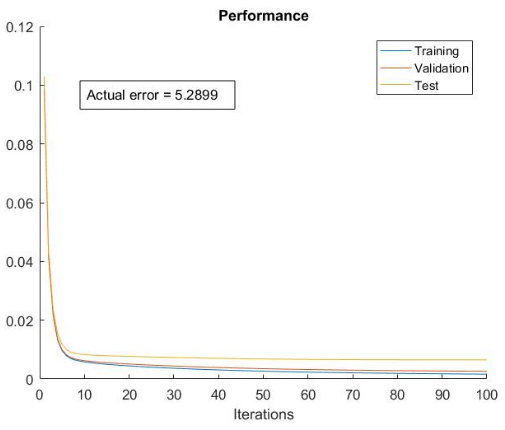
Observing Figure 5, it is easy to see that the LM algorithm really does outperforms the backpropagation algorithm. The error seen using the LM algorithm using the tested data set is minimal. This is statistically reflected by the covariance plot in Figure 6. An ideal prediction would result in a R value of 1. We can clearly observe that the LM method gives a covariance score that is much higher than the backpropagation method. This improvement confirms our hypothesis of a more accurate network using the LM method. The last step to the comparaison is to describe the perfomance of the network over time, and how fast it reaches minimal error.

5.3 Network speed

To evaluate the time for the network to reach an output error of 0.002, we will first count the number of iterations necessary for such accuracy, after which we will measure the time necessary to iterate accordingly. The first step can be done using another script written by Johan Tillmar, which saves the actual error of the network at each iteration and plots the results after 100 iterations. The script was tested for the two versions of the network, leading to the following results:



(a) Backpropagation method



(b) LM method

Figure 7: Output error vs. Iterations

Actual error E is expressed as:

$$E = \frac{\sum_k (\varsigma_\mathrm{k} - O_\mathrm{k})^2}{N}$$

with ς being the target output, O is the actual output and N is the number of output samples taken.

The backpropagation plot, Figure 7 (a), shows that an output error of 0.002 is achieved after about 20 iterations. On the other hand, looking at the LM method's results, we observe a 0.002 error after only 4 iterations.

Here are the times to achieve the required amounts of iterations for an error of 0.002:

Algorithm	Required iterations	Mean time for E = *0.002* (s)
Levenberg-Marquardt	4	17.49
Back Propagation	20	29.74

Figure 8: Algorithm training speeds

The two scripts have been running in the same conditions on the same computer. The LM method results in a much faster learning network, about 58% faster than by invoking the backpropagation method.

6 Conclusion

The objective of this study is to confirm that the Levenberg-Marquardt training algorithm is the best performing method for our network. MATLAB's built-in ANN tools follow this statement, however, as the network designed by Johan Tillmar is built independently of these functionnalities, it is important to confirm which method is preferable for our specific network, in perpective of implementing the ANN onto a microprocessor. The study's results show that even though the backpropagation algorithm is not as exigent as the LM method in terms of computational power, its accuracy and learning speed is significantly degraded next to the LM algorithm. Implementing the network using the LM algorithm will be far more effective in order to accurately predict vehicule velocity and optmimize power managment within the vehicule's powertrain.
Next steps for this network include finiding a suitable equivalent C code for microprocessor implementation as well as performing additional performance tests using supplementary data sets.

References

[1] Johan Tillmar. *Estimation of Vehicle Velocity for Future Horizon in Powertrain Optimal Energy Managment.* 2016.

[2] Oxford Dictionary Online. `http://www.oxforddictionaries.com/definition/english/neural-network?q=neural+network`. Accessed: 8-7-2016.

[3] Johan Tillmar. *Neural Network Flowchart.* 2016.

[4] Mathworks website. `http://uk.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html`. Accessed: 8-7-2016.

[5] Mathworks website.
http://uk.mathworks.com/help/nnet/ug/choose-a-multilayer-neural-network-training-function.html. Accessed: 8-7-2016.

[6] Warren S. Sarle.
ftp://ftp.sas.com/pub/neural/FAQ2.html#A_act. Accessed: 8-7-2016.