Imperial College London

CONTROL AND POWER RESEARCH GROUP

ANN Simulation for Continuous Time Operation On-board a Vehicle

Theo Franquet

15 September 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 Management article, written by Johan Tillmar [1], in which a complete artificial neural network has been designed and tested, intended to predict a vehicle's future velocity, based on diverse driving parameters. The network will be explained in more detail over the course of this article.
This article will be explaining the strategy to simulate an established neural network in the general powertrain control system model. A Matlab script will be given at the end of the report.
The neural network has been designed within Matlab's environment which allows for easier matrices manipulations and offers rich online resources. Matlab is also very effective for data extraction and analysis. The simulation script is also done in a Matlab environment to easily interact with the available ANN.

1.2 Background

Artificial Neural Networks (ANN) are defined as a computer systems modeled 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. Similarly to the human brain, in order to obtain accurate outputs, the network has to be trained facing multiple possible input scenarios.
The given ANN is functional and has been tested independently, using a given list of parameter samples. The test has been done regardless of any timeline, meaning the network is training based on past, present, and future measurements, which is not representative of how the network would be used in application. We are looking to operate the network following a moving window across time, where parameters are sampled gradually and where the ANN trains based on its performance in time.

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.

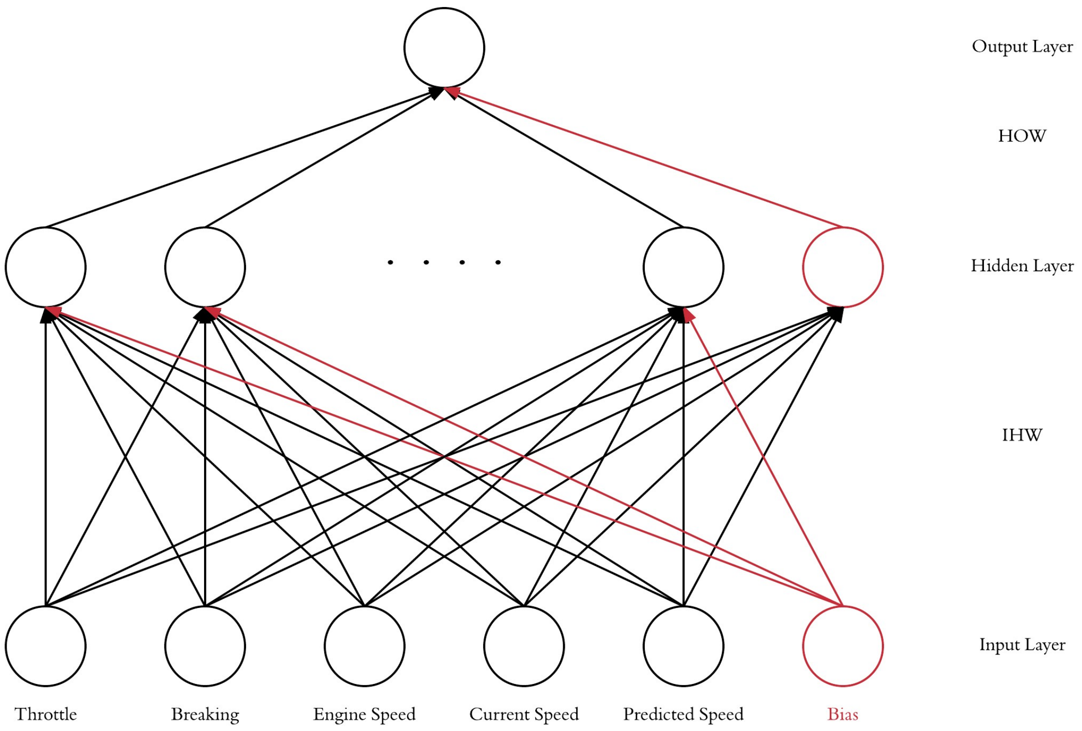


Figure 1: General ANN arrangement

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 functioning. 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 extension 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 training methods and explain which algorithm performs better for our specific network. Detailed explanations of the algorithms have been made in Johan Tillmar report.

3.1 Cost function

The chosen training method uses a cost function, which evaluates the mean squared error between target and output for each neuron. Training a network aims at minimizing the cost function for a certain set of inputs by re-evaluating the weights of the netwok. 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 repsonse 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.
It has been shown that within Matlab's built-in neural network functions, the Levenberg-Marquardt method is the quickest and performs with great accuracy for networks with a structure similar to the one used, and that this algorithm is a viable option looking towards an onboard implementation of the network.

3.2 Levenberg-Marquardt method

The Levenberg-Marquardt (LM) method is a 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 $\mathbf{w}$ the weights, $\mathbf{H}$ the Hessian matrix and $\mathbf{g}$ the n-dimentional gradient vector:

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

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

4 Ideal ANN operation onboard a vehicle

The objective of the program is to simulate an ideal operation of the network, embedded in the vehicle's powertrain's control system. This section describes how the neural network interacts with the rest of the control system, and explains its ideal operation to increase the powertrain's overall efficiency.

4.1 High level operation

The following diagram describes the ANN's role and interactions with the rest of the control system

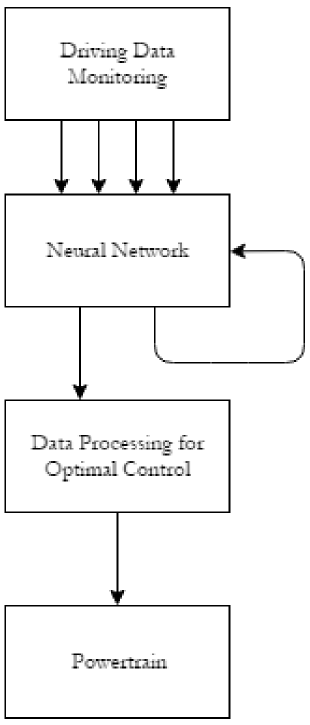


Figure 2: Control System Topology

An array of driving parameters are monitored continuously using relevant sensors. This data is sampled every 0.1 millisecond for the sake of our simulation but is flexible depending on the sensors used. All of this data is then fed to the ANN. The ANN's output is used twice. On one end it will be fed back to the ANN's input to be used for training, on the other it will be sent to a data processing block that will compute control signals to be sent to the powertrain's components.

4.2 Low level operation

Changing to a lower level perspective, the neural network's ideal operation along time can be descibed as follows:

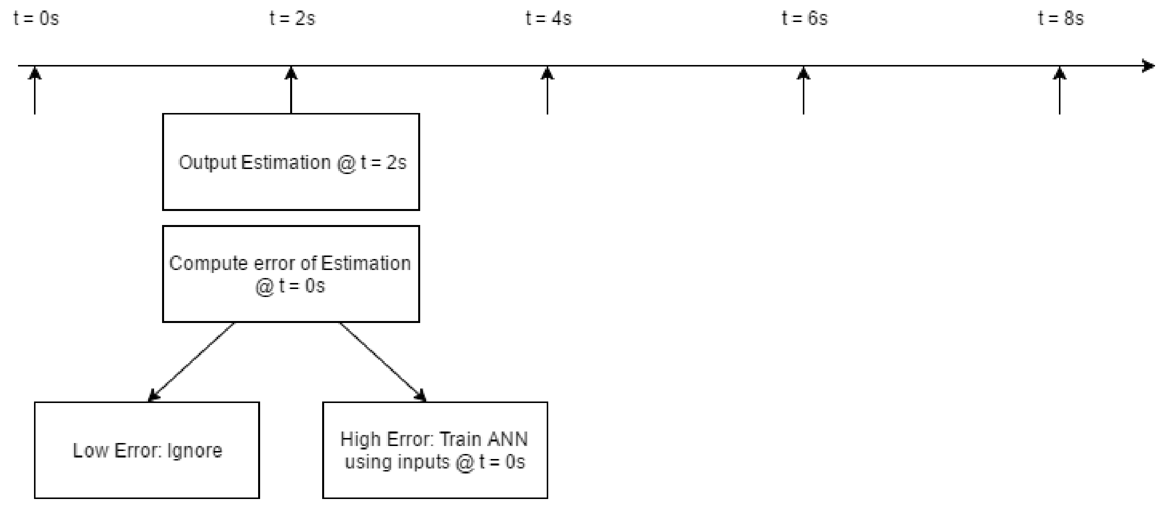

t = 0s
t = 2s
t = 4s
t = 6s
t = 8s
Output Estimation @ t = 2s
Compute error of Estimation @ t = 0s
Low Error: Ignore
High Error: Train ANN using inputs @ t = 0s

Figure 3: Ideal ANN Operation

where the operation protocol is repeated every two second for the sake of the simulation but is a flexible parameter.
Periodically, the neural network will start by propagating giving an output using inputs recorded between t = 0s and t = 2s. This output corresponds to the velocity estimation of the vehicle in 2 seconds. The second step involves comparing the vehicle velocity estimated at t = 0s with the true velocity recorded at t = 2s, allowing us to compute an error factor using the following expression:

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

with N the output array's length, i the output array index, ς the target and O the output of the network.

This error is then compared to a threshold set by the user: if the error is too large, probably meaning the ANN has never come across this input arrangement in the past, training will take place, using data monitored between t = -1s and t = 0s only. If the error is low, training will not take place. This decision making gives a dynamic dimension to the ANN's training. Changing the error threshold has important impact of the ANN's operating speed and accuracy. Comparative results will be shown later in the article.

5 Drive cycle and data set

5.1 Data Set Parameters

A 108 seconds drive cycle is used for the ANN simulation, working with four different driving parameters: Throttle, Breaking, Engine Speed as well as Actual Vehicle Velocity. Data is recorded using a sampling rate of 0.1ms and stored in one array for each parameter. The inputs used for simulation are illustrated bellow:

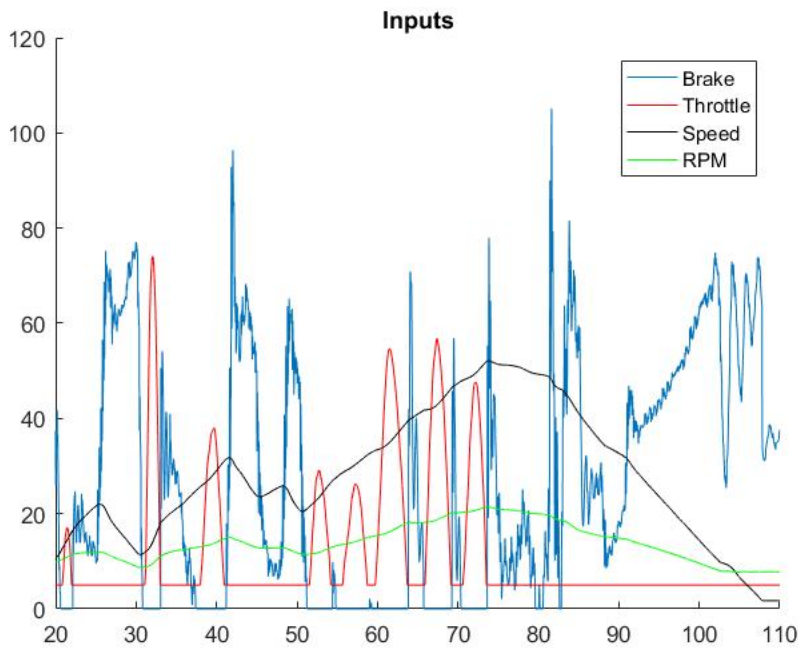


Figure 4: Data Set Used

Note that the first 20 seconds of the drive cycle are removed due to unreliable data. To simulate the network operation described in section 4.2, the data set has to be gradually recorded by the control system, a moving window of two seconds is used to go over the drive cycle following a continuous timeline which is comparable to how the system would be operating onboard a vehicle.

5.2 Dynamic index

In order to create a moving window, a dynamic index is used in the simulation script. This dynamic index, which can be viewed as a subarray, will set boundaries within the 108 seconds drive cycle array, between which a 2 seconds equivalent period will be established. All ANN operations will be using data belonging in the window set by the dynamic index. This index will be incremented as the simulation script is executed, hence making it a dynamic index. This will mirror the system recording new data along time.
It is important to note that two windows are used in the simulation. The first one will include most recently recorded data, used to estimate future vehicle velocity. The second one includes data recorded further in the past, used to train the network if the estimation made at the time is seen to be faulty.

6 Script structure

This section describes the script's structure and content, which aims at simulating the ANN's operation following a continuous timeline as described in section 4.2. The code is written on a single `TestMovingWindow.m` file.

6.1 Simulation parameters

The Matlab script is made as flexible as possible when it comes to changing some of the simulation parameters. The first section of the code allows the user to set five constants used in the simulation:

Variable	Unit	Details
`seconds_to_remove`	s	Removing a starting section of the data set
`sampling`	s^{-1}	Sampling rate of used data set
`max_error`	n/a	Error threshold used for training decision
`time_window`	s	Length of moving window
`time_before_interupt`	s	Time before simulation halts and presents output plots

Figure 5: Simulation input parameters

The amount of iterations done for each training sequence can also be modified at the start of the `trainLM.m` file's code.

6.2 Formatting inputs and creating dynamic index

The second section of the script (lines 17 to 48) aims at formatting the inputs for the rest of the simulation. All driving parameters are initially stored in a `.mat` file where each one is described as a two dimensional vector: a `.Data` column stores the parameter amplitude and a `.Time` column stores time. All this data is extracted from the `.mat` file and fed into one dimensional arrays within the Matlab environment. This section also eliminates any starting section of the drive cycle according to the user's setting.
The third section (lines 48 to 67) is where the dynamic index is created. It is first initialized: its length changes with respect to settings set by the user (`time_window` and `sampling`). It is important to note that the dynamic index does not start at the very beginning of the data set. This offset is eplained by the need of a second moving window further in the past constituting training data. This offset also directly depends on the constants set by the user in the first section of the script.
Some variables are then initialized. These are used throughout the simulation to monitor certain parameters and are used to present the ANN's performance at the end of the simulation.

6.3 Creating two moving windows

As described in section 5.2 and 6.2, two moving windows are put to use in the simulation: one is set in "present time" while the other is set in the past and encloses "old data". In this section (lines 67 to 94) dimensional subarrays, described by the dynamic index, are created for present and old data, for each driving parameter. This also involves manipulation using constants set by the user.

6.4 Training logic

The next section of script (lines 94 to 128) aims at initializing the ANN. This is only done once, any future manipulation of the ANN will keep its structure unchanged. Three I/O arrays are created: `inputs_present`, `inputs_old` and `targets` for ANN operating and training. The network creating function `BackPropNetV2` is called once for the rest of the simulation.
To implement the logic described in section 4.2 of this article, an if statement is used to compare the error made over the recently made estimation with the threshold set by the user. If the error is too high, the network will train using the old data and recently recorded target speeds. The network is then operated and inputs propagate towards the output layer to give out the estimated velocity in two seconds time (lines 128 to 145).

6.5 Dynamic index incrementation

Lastly, the dynamic index is incremented simulating a moving window following a continuous timeline (lines 145 to 149). All code described from section 6.3 to section 6.5 is set inside a loop simulating driving time. Each loop incrementation creates and processes different subarrays, as the dynamic index is changed accordingly. The ANN outputs an estimated velocity and can potentially be trained over each loop incrementation. The loop will break after a certain time has elapsed, as requested by the user (`time_before_interupt`).

7 Simulation results

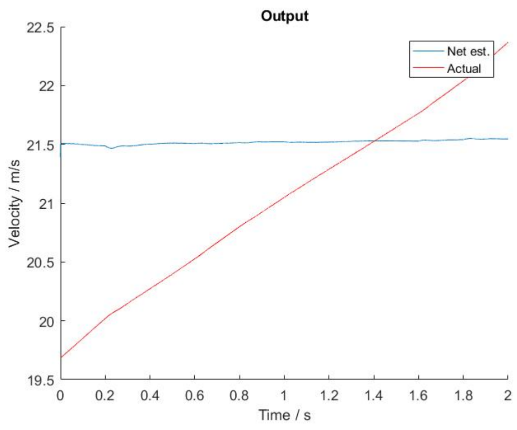
This section will present different aspects of the simulation's results. The first one being the output accuracy and performance of the ANN, followed by the training rate.

7.1 Output accuracy

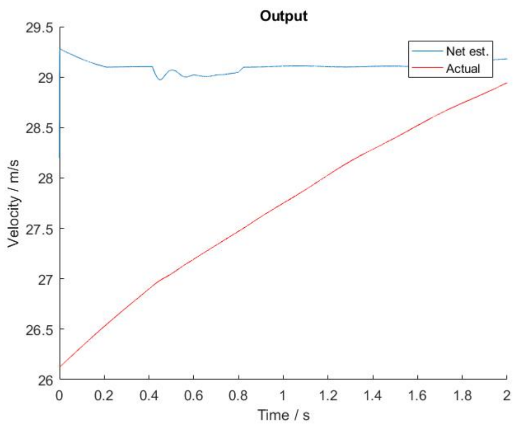
Output accuracy can be evaluated and compared at many levels and timeframes. This section will describe the ANN's performance over a 2 second window, before looking into the full drive cycle output accuracy.

7.1.1 Two second timeframe

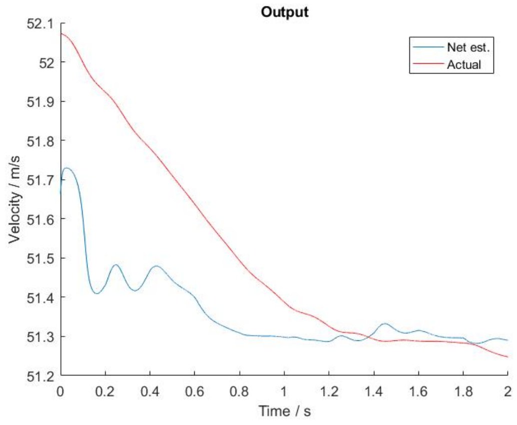
Here are four 2 second windows after simulations under different conditions:



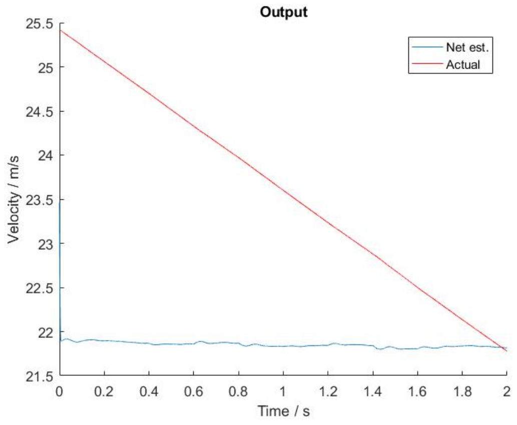
(a) 10 seconds limit



(b) 30 seconds limit



(c) 50 seconds limit



(d) 70 seconds limit

Figure 6: ANN output comparaison for various simulation times

Using the following parameters:
`time_window` $= 2$
`max_error` $= 2$
`sampling` $= 10^{-4}$
`seconds_to_remove` $= 20$
`limit` $= 4$ (training iterations)

Although it would be expected to obtain greater accuracy as the simulation duration increases and as the network goes over more training, these measurements show that output accuracy remains similar across all four cases. The velocity estimated stays mostly constant over the 2 seconds window, leading to visible error. However, this flat step is not mandatory and theoretically, as the network goes over several similar scenarios, its accuracy will increase and the output curve will tend to get closer to the target. The inaccuracy seen here can be explained by the fact that the network only goes over each 2 second scenario once and that each training is only done over four iterations.

7.1.2 Full drive cycle timeframe

Changing timeframe and observing the ANN's performance over the full drive cycle is also important for its specific application. The goal is to have a velocity estimation accurate enough to improve the overall powertrain efficiency. A plot of the cumulated ANN output throughout the entire drive cycle is shown below.

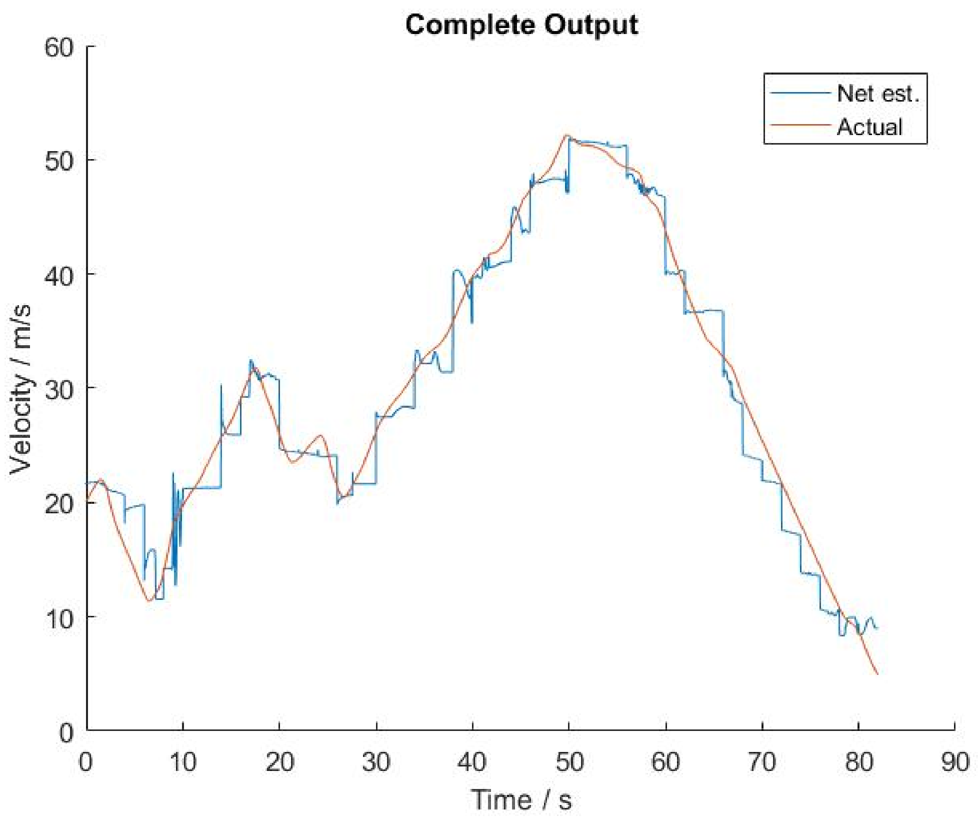
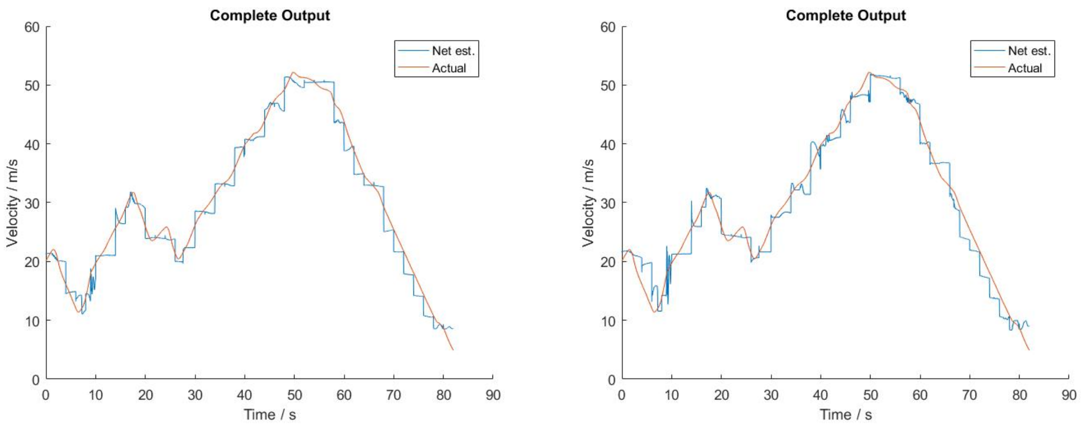


Figure 7: ANN output over entire drive cycle

Using the following parameters:
`time_window` = 2
`max_error` = 2
`sampling` = 10^{-4}

`seconds_to_remove` = 20
`limit` = 10 (training iterations)
`time_before_interupt` = 80

We can observe that the estimated speed does fit to the target overall, but keeping each two second estimation constant creating steps on the plot above. However these steps are not as flat towards the end of the simulation than during the first 30 seconds. It seems that the output accuracy is generally improving throughout the simulation. To study the importance of increasing the number of iterations for each training, the simulation has been done with 4 and 10 iterations per training, with all parameters remaining the same.



(a) 4 iterations per training (b) 10 iterations per training

Figure 8: Performance comparaison for different training methods

Using the following parameters:
`time_window` = 2
`max_error` = 2
`sampling` = 10^{-4}
`seconds_to_remove` = 20
`time_before_interupt` = 80

The difference in accuracy from one ANN version to the other is not extremely important as using four iterations per training already outputs a rather accurate estimated velocity. However difference is observable looking back at the angle of each step: using 10 iterations per training leads to a more dynamic output comparing it to the other method where the ANN's output stays mostly constant.

7.2 Training rate

As seen previously, the ANN can potentially train for every loop incrementation. However, as the network goes through more training, its output becomes increasingly accurate for a certain set of inputs, making it less likely to train in the future. Training rate changes depending on certain simulation conditions. A first comparison has been done, observing training frequency when changing the maximum error threshold.

max_error	Training rate
0.5	100%
1	90.9%
1.5	81.8%
2	81.8%
2.5	72.7%
3	72.7%

Figure 9: Taining rate against threshold

Using the following parameters:
`time_window` = 2
`time_before_interupt` = 20
`sampling` = 10^{-4}
`seconds_to_remove` = 20
`limit` = 4 (training iterations)

We can confirm that as the error threshold rises, the network does not have to train as often as the system has a greater tolerance for output error. However, we should note that the training rate remains very high over the measurements, which is due to the network being still very recent as well as stopping its operation after only 20 seconds. We can expect that training rate will significantly decrease over a much longer time of simulation.

8 Conclusion

The objective of this report has been to give a short introduction on the neural network that is being used for future vehicle velocity estimation, as well as explaining the strategy and methods used to simulate the given ANN's operation onboard a vehicle. The ideal progression of the ANN is to work continuously, recording driving parameters and using this data to output a speed estimation and train itself. A Matlab script has been written for that purpose and its structure and content has been covered in section 6 of the report. A complete version of the code is given in the appendix.

The ANN's performance overall following various simulations is satisfying, characterised by a good output accuracy and learning rate.
Further work has to be done on two sectors. The first one being training speed. Simulating the ANN operation on the Matlab environment, training times were in the order of ten seconds (for 4 training iterations), however the ANN should be able to train itself in under 2 seconds to function continuously. Changing the simulation environment and reducing sampling speed to diminish input load should improve overall ANN speed in that case. More research also has to been done to confirm which simulation parameters constitute an ideal compromise between ANN speed, accuracy and viability for the end application.

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.

Appendix A Matlab Code: `TestMovingWindow.m`

```
clear;
for i = 1:15
    clf(figure(i));
end

load('C:\Users\User\Desktop\NeuralNetwork\IDC _Pure_IC_Engine.mat')

%------------------------ Simulation Parameters ------------------------%

seconds_to_remove    = 20;
sampling             = 10^-4;
max_error            = 2;
time_window          = 2;
time_before_interupt = 90;


%-------------------------- Formating Inputs --------------------------%

%Brake
brake_data = Brake_Time.Data;
brake_time = Brake_Time.Time;

brake_data(1:seconds_to_remove/sampling) = [];
brake_time(1:seconds_to_remove/sampling) = [];

%Throttle
throttle_data = Throttle_Time.Data;
throttle_time_1 = Throttle_Time.Time;

throttle_data(1:seconds_to_remove/sampling) = [];
throttle_time_1(1:seconds_to_remove/sampling) = [];

%Clutch/IP
clutch_data = Clutch_IP_ClutchSpeed.Data;
clutch_time = Clutch_IP_ClutchSpeed.Time;

clutch_data(1:seconds_to_remove/sampling) = [];
clutch_time(1:seconds_to_remove/sampling) = [];

%Speed
speed_data = VSpeed_khr.Data;
speed_time = VSpeed_khr.Time;

speed_data(1:seconds_to_remove/sampling) = [];
speed_time(1:seconds_to_remove/sampling) = [];


%-------------------- Dynamic Index & Initialization --------------------%

%Create dynamic index, for 0.1  millisecond sampling%
%Initialize index%
in = 2*(time_window/sampling)+1;
out = 2*(time_window/sampling) + (time_window/sampling);
dynamic_index = in:out;
%Initialize error%
error = 1000;
```

```
%Initialize counter%
train_count = 0;
%Initialize error sum%
error_sum = 0;
%Initializing total targets%
total_outputs = [];
%Initializing total output%
total_targets = [];


%---------------------------- Two windows -----------------------------%

%First window (present) used for speed estimation
%Second window (old) used for network training

for i = 0:(time_before_interupt/2)

    brake_data_present    = brake_data(dynamic_index);
    brake_time_present    = brake_time(dynamic_index);
    throttle_data_present = throttle_data(dynamic_index);
    throttle_time_present = throttle_time_1(dynamic_index);
    clutch_data_present   = clutch_data(dynamic_index);
    clutch_time_present   = clutch_time(dynamic_index);
    speed_data_present    = speed_data(dynamic_index);
    speed_time_present    = speed_time(dynamic_index);


    brake_data_old    = brake_data(in - 2*(time_window/sampling):in - (time_window/sampling));
    brake_time_old    = brake_time(in - 2*(time_window/sampling):in - (time_window/sampling));
    throttle_data_old = throttle_data(in - 2*(time_window/sampling):in - (time_window/sampling));
    throttle_time_old = throttle_time_1(in - 2*(time_window/sampling):in - (time_window/sampling));
    clutch_data_old   = clutch_data(in - 2*(time_window/sampling):in - (time_window/sampling));
    clutch_time_old   = clutch_time(in - 2*(time_window/sampling):in - (time_window/sampling));
    speed_data_old    = speed_data(in - 2*(time_window/sampling):in - (time_window/sampling));
    speed_time_old    = speed_time(in - 2*(time_window/sampling):in - (time_window/sampling));


%--------------------------- Creating Network ---------------------------%

    %Create input and target for the network
    %nbrSample = zeros(nbrSample,nbrIn);
    inputs_present  = [brake_data_present throttle_data_present clutch_data_present speed_data_present (thr
    inputs_old  = [brake_data_old throttle_data_old clutch_data_old speed_data_old (throttle_data_old.*clut
    targets = speed_data_present;


    %Create training inputs and targets that uses less data
    reductionProcent = 1;

    nbrSamples = length(targets);

    trainInputs = inputs_old(1:nbrSamples*reductionProcent,:);
    trainTargets = targets(1:nbrSamples*reductionProcent,:);

    %Create net and train it
    s       = size(inputs_old);
    nbrIn   = s(2);
    s       = size(targets);
    nbrOut  = s(2);

```

```matlab
    nbrHidden  = 10;
    nbrDelays  = 4; % 1 is normal, just input, no dependence from previous values
    nbrFBDelay = 4;

    if i == 0
    net = BackPropNetV2(nbrHidden,nbrIn,nbrOut,nbrDelays,nbrFBDelay);
    end


%-------------------------- Training Decision --------------------------%

    if error > max_error
    [net, Performance] = net.trainNetwork(trainInputs,trainTargets);
    train_count = train_count + 1;
    end
    output = net.output(inputs_present);
    targets(1:nbrDelays-1) = [];
    total_outputs = [total_outputs; output];
    total_targets = [total_targets; targets];
    clearvars sum
    %Actual error
    error = sum((targets-output).^2)/(nbrSamples);

    error_sum = error_sum + error;


%--------------------- Dynamic Index Incrementation ---------------------%

    in = in + (time_window/sampling);
    out = out + (time_window/sampling);
    dynamic_index = in:out;
end

%---------------------- Post Training Analysation ----------------------%


%Linear regression
p = polyfit(targets,output,1);
x = linspace(0,max(targets),length(output));


total_shifts = time_before_interupt/2 + 1;
disp(['total_shifts = ' num2str(total_shifts)]);
disp(['train_count = ' num2str(train_count)]);
disp(['training_rate = ' num2str((train_count/total_shifts)*100) '%']);
disp(['average error = ' num2str(error_sum/total_shifts)]);


%--------------------------------------------------------------------------
%---------------------------PLOTS------------------------------------------
%--------------------------------------------------------------------------

%------Input plot------
figure(1); hold on;
plot(brake_time,brake_data)
plot(throttle_time_present,throttle_data_present,'r')
plot(speed_time_present,speed_data_present,'k')
plot(clutch_time_present,clutch_data_present,'g')
legend('Brake','Throttle','Speed','RPM')
title('Inputs')
```

```matlab

t1 = linspace(0,nbrSamples/10000,nbrSamples-(nbrDelays-1));
%------Output plot------
figure(2); hold on;
plot(t1,output);
plot(t1,targets,'r')
title('Output')
legend('Net est.','Actual')
ylabel('Velocity / m/s')
xlabel('Time / s')

s1 = length(total_outputs);
t2 = linspace(0,s1/10000,s1);
%------General Output plot------
figure(3); hold on;
plot(t2,total_outputs);
plot(t2,total_targets);
title('Complete Output')
xlabel('Time / s')
ylabel('Velocity / m/s')
legend('Net est.','Actual')

%------Performance plot------
figure(4); hold on;
plot(Performance(:,1));
plot(Performance(:,2));
plot(Performance(:,3));
title('Performance')
legend('Training','Validation','Test');
xlabel('Iterations')
dim = [.2 .5 .3 .3];
str = ['Actual error = '  num2str(error)];
annotation('textbox',dim,'String',str,'FitBoxToText','on');

%--------------------------------------------------------------------------
%--------------------------------------------------------------------------
```