CHALMERS
UNIVERSITY OF TECHNOLOGY

Vehicle speed-profile prediction without spatial information

An exploratory study of recurrent neural networks and their ability to predict time-series

Master's thesis in Computer Science – Algorithms, Languages and Logic

ERIK THORSELL

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis EX030/2018

Vehicle speed-profile prediction without spatial information

An exploratory study of recurrent neural networks and their ability to predict time-series

Master's thesis in Computer Science – Algorithms, Languages and Logic

ERIK THORSELL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Vehicle speed-profile prediction without spatial information

An exploratory study of recurrent neural networks
and their ability to predict time-series

ERIK THORSELL

Supervisor: Rickard Arvidsson, Volvo Cars
Supervisor: Miroslaw Staron, Dep. of Software Engineering
Examiner: Tomas McKelvey, Dep. of Electrical Engineering

Master's Thesis EX030/2018
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Vehicle speed profile prediction without spatial information
An exploratory study of recurrent neural networks
and their ability to predict time-series
ERIK THORSELL
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Starting 2019, all cars produced by Volvo Cars will be equipped with an electric engine and by 2025 the company's goal is for 50% of their sales to constitute of fully electric vehicles. Due to battery technology limitations, the main difficulty for Volvo Cars' engineers is to develop vehicles with sufficient range. One potential technique for increasing the range of (hybrid) electric vehicles is to predict the future speed-profile of a driver and set vehicle control parameters accordingly. The thesis explores to what extent recurrent neural networks can be used for performing speed-profile predictions. The cell units: RNN, LSTM, and GRU are evaluated as part of two different models (one to many and many to many).

The thesis aims to explore recurrent neural networks' ability to predict speed-profiles. The two research questions of the thesis entails: (RQ 1) evaluating multiple types of neurons and model structures and (RQ 2) evaluating the effect of preprocessing of data and tuning of the neural network configurations.

The thesis follows the Design Science Research methodology. Proof of concept configurations were implemented, trained, and evaluated on their ability to correctly predict the next 60 samples (seconds) of a vehicle's speed (speed-profile). RQ 1 was trained using only vehicle speed as data feature, and evaluated using ≈ 100 speed-profiles. RQ 2 was trained using both vehicle speed and vehicle battery charge as data features. It was evaluated using ≈ 100 and ≈ 800 speed-profiles.

The thesis shows that several neural network configurations are capable of achieving predictions which are better than that of a trivial predictor. The trivial predictor "predicts" that the driver will keep its speed constant for the next 60 seconds. The best configuration developed to answer RQ 1 is a many to many-model with RNN cell units. Its RMSE is 0.77 times that of the trivial predictor. The evaluation of RQ 2 shows that configurations which make use of fewer preprocessing/regularisation parameters achieve lower RMSE. For ≈ 100 speed-profiles, the best configuration used no additional preprocessing or regularisation and achieved an RMSE 0.80 times that of the best configuration from RQ 1. With ≈ 800 speed-profiles, the best configuration made use of smoothening and aligning of the speed-profiles. Its RMSE was 0.62 times that of the best configuration from RQ 1.

Keywords: machine learning, time-series prediction, recurrent neural networks

Acknowledgements

I want to thank Volvo Cars for suggesting the thesis topic and for providing me with the Volvo C30 Electric log data. In particular I want to thank my industrial supervisor Rickard Arvidsson for helping me with the technical details regarding the vehicles and the data. I also want to thank Christoffer Nilsson for providing me with initial hardware and software support for training my configurations; without him I would not have been able to meet the deadline.

Additionally, I want to thank: Prof. Dr. Miroslaw Staron for the academic guiding and thorough critiquing of my work, and Prof. Tomas McKelvey for accepting my thesis proposal and the role as my examiner.

Lastly, I want to thank my fiancé Daniella for putting up with me through the ups and downs of this past semester.

Erik Thorsell, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

Volvo Cars announced — in the summer of 2017 — that every Volvo sold from 2019 and onwards will have an electric engine [1]. Additionally — during spring of 2018 — the company set as their goal for 50% of their vehicles sold by 2025 to be fully electric [2]. Including an electric powertrain in a vehicle is a non trivial task and according to Conti *et al.*, the largest limitation of Electric Vehicles (EVs) and Plug in Hybrid EVs (PHEVs) as of today is battery technology [3]. The energy and power density of currently available batteries make EVs/PHEVs unable to match the range and refuel rate of vehicles with conventional powertrains. The state of battery technology forces engineers to look elsewhere in order to make their vehicles more efficient. Better body aerodynamics, reduced kerb weight, smart battery charging, and smart driving are a few of the techniques affecting the overall efficiency of the vehicle.

This thesis is concerned with the last of the techniques mentioned above. If a vehicle is provided with information about how its driver is likely to accelerate and decelerate over time, it is possible for the vehicle to make smart decisions about — for instance — which powertrain to use in a PHEV and what gear to use when driving. Unfortunately information from the future is not available at the present. It might however be possible to make accurate *predictions* about the future vehicle speed and use these predictions as a foundation for making the aforementioned decisions.

Volvo Cars is currently developing a system which is capable of making such predictions. Said system is, however, dependant on several factors which requires connectivity to additional services. It is of interest to investigate whether it is possible to perform accurate predictions utilizing only the means available locally in a Volvo Cars vehicle.

1.1 Project aim

The aim of this thesis is to investigate whether it is possible to utilize various time dependent machine learning techniques to predict the future speed-profile for an arbitrary route and vehicle. The novelty of the project lies in implementing a solution which uses these time dependent techniques and performs training and prediction only with local vehicle data (i.e. the velocity or energy consumption of the vehicle). Additionally, the solution must not use any spatial information about the vehicle's movement.

1.2 Related work

Neural networks have previously been used for various prediction tasks related to the automotive industry.

Froehlich and Krumm have developed algorithms for predicting the end-to-end route of a vehicle, based on previous GPS observations [4]. In their report, Froehlich and Krumm conclude that a large number of a typical driver's trips are repeated and their algorithms exploit this fact when mapping the first part of an arbitrary route to one of many previously seen routes. The authors' goal is to reduce the energy consumption in HEVs but the approach used by Froehlich and Krumm differs from that of this thesis, as the former uses GPS data.

Fotouhi *et al.* use multilayer perceptron models for predicting the next 10 seconds of a vehicle's speed [5]. During driving, the model yields a speed-profile prediction for the next 10 seconds, and by comparing the prediction to stored 10 second segments the algorithm determines control parameters for the vehicle. The algorithm acquires a new prediction — and updates the vehicle's parameters — every 10 seconds. The approach presented by Fotouhi *et al.* does not make use of recurrent neural networks, nor is it capable of predicting more than 10 seconds ahead.

Park *et al.* attempt to predict the speed of a vehicle between two predetermined locations [6]. Their algorithm does, however, use both geographical data as well as past and current traffic information.

Jeon *et al.* have developed an algorithm similar to the one by Fotouhi *et al.*, but they use a neural network to determine which out of six predefined driving patterns the current driver most closely follows [7]. Depending on which driving pattern the network selects, a multi-mode driving control algorithm adapts the vehicles' control parameters to achieve better fuel and energy efficiency.

1.3 Scope

The scope of this thesis concerns exploring the applicability for predicting future speed-profiles using various recurrent neural network techniques. Hence, the thesis does not intend to find an optimal way of modelling speed-profile predictors in an unconstrained space of models, but focuses solely on the use of recurrent neural networks.

1.4 Research questions

The number of different neural network configurations (combinations of network models and cell units) available offers a very large search space, should one attempt to implement and evaluate all of them. Additionally — even with a small number of configurations — preprocessing, hyper parameter tuning, regularisation, optimisation algorithm selection, and loss function selection still yields too many alternatives for a project of this size. Hence, this thesis is divided into two parts: (1) a screening process and (2) an in depth tuning process for a selected configuration.

These parts yields two research questions:

Research question 1 Which, out of six, neural network configuration yields the best predictions, with respect to lowest summed root mean squared error, after minimal preprocessing of data and minimal tuning?

Research question 1 constitutes the screening process. Six neural network configurations, which are defined in Section 3.2.2.1, are evaluated. The root mean squared error (RMSE) is an apt measurement of performance as it decreases when the prediction of a configuration gets closer to the actual (true) value. When research question 1 has been answered the best configuration is used as a basis[1] to answer research question 2.

Research question 2 How should the data be preprocessed, and how should the configuration — based on the best configuration according to research question 1 — be tuned, in order for its predictions to achieve as low summed root mean squared error as possible?

Research question 2 aims to use the best configuration from research question 1 as a basis and increase its performance[2]. Better predictions leads to a more solid foundation on which additional algorithms can be implemented. These additional algorithms can be used to assist the driver or vehicle to reduce the energy used while driving and in turn extend the range of the vehicle.

1.5 Thesis report structure

Chapter 2 presents the theory needed to understand the content of the thesis. The chapter introduces time-series, machine learning, neural networks, recurrent neural

[1]Research question 1 will answer which cell units and what neural network model seems best suited for the task at hand.

[2]The performance of a configuration is said to increase when the summed RMSE of the configuration's predictions decreases.

networks, and how to train neural network configurations.

Chapter 3 presents the methodology used throughout the thesis, by which the research questions are answered.

Chapter 4 outlays the implementation details regarding the neural network configurations used to answer the research questions. Chapter 4 also justifies the frameworks used to develop the neural network configurations.

Chapter 5 presents the result of the thesis. The two research questions are evaluated in accordance to the research methodology. Thorough discussions about the results are also presented.

Finally, Chapter 6 concludes the thesis with a reflection regarding the applicability of using recurrent neural network techniques to perform time-series predictions. Additionally, future work and improvements are suggested.

2
Theoretical background

This chapter gives a theoretical background to machine learning in general, and the machine learning techniques used in this thesis in particular. Firstly, however, time-series as a concept is explained. Following this, an overview of the supervised machine learning field is provided and thorough descriptions of the specific cell units and models used in the thesis are given. Additionally, the chapter explains the concept "preprocessing of data" and exemplifies different regularisation techniques which are later used in an attempt to improve the performance of the evaluated neural network configurations.

2.1 Time-Series

Time-series are discrete sequences of an arbitrary, time-valued, function ordered over time [8]. The interval between the values in the sequences must not be equidistant, but time-series are commonly created by sampling some function at a fixed interval of time t. An example of a time-series is visualised in Figure 2.1 where the speed of a vehicle has been sampled every second during one route (start to stop). This is referred to as the *speed-profile* for one *route* or *drive cycle*.

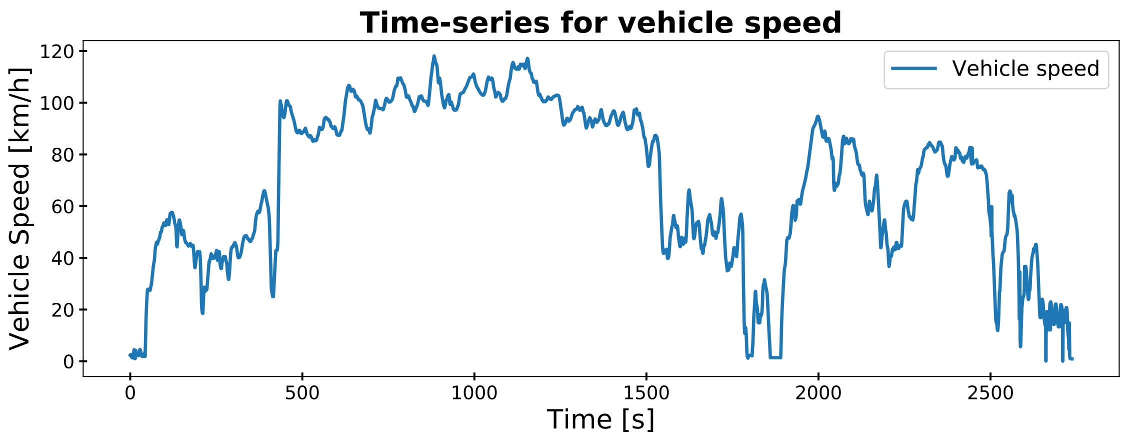

Figure 2.1: Plot showing an example time-series; more specifically the speed profile for one car and one route.

One distinguishes between *deterministic* and *stochastic* time-series, where the former is a time-series which future values are determined by some mathematical function, such as: $f(t) = \sin\left(\frac{3}{4}\pi t\right)$, and the latter describes some stochastic phenomenon, such as the speed-profile in Figure 2.1 [9].

2.1.1 Stationary and non-stationary processes

If a process is assumed to be based on a particular state of *statistical equilibrium*, the process is referred to as a *stationary stochastic processes* [9]. A time-series is said to be stationary if the properties of said time-series does not depend on the time at which the time-series was observed, i.e. the time-series does not show any trend or seasonal behaviour.

Formally, if Z_t is some stochastic process and $F_Z\left(z_{t_1+\tau}, \ldots, z_{t_k+\tau}\right)$ represents the cumulative distribution function of the unconditional joint distribution of Z_t at times $t_1 + \tau, \ldots, t_k + \tau$. Then, Z_t is said to be strictly stationary if, for all k, for all τ, and for all $t_1, \ldots, t_k$, Equation 2.1 holds.

$$F_Z\left(z_{t_1+\tau}, \ldots, z_{t_k+\tau}\right) = F_Z\left(z_{t_1}, \ldots, z_{t_k}\right) \tag{2.1}$$

Results from time-series predictions on non-stationary data are often less accurate than would the data be stationary [10]. It is, however, possible to transform a non-stationary time-series into a stationary representation. One such transformation is called *differencing*, by which a time-series is described in terms of the difference between its consecutive samples. Figure 2.2 shows an example of a differencing transformation.

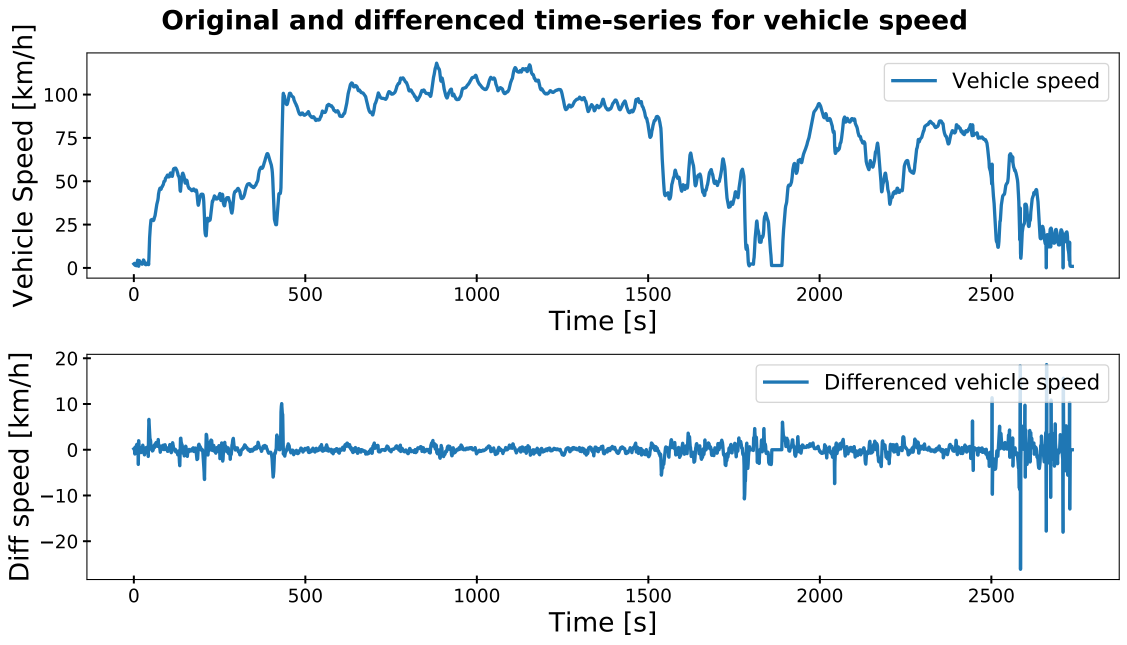

Figure 2.2: Comparison between the speed-profile time-series — as seen in Figure 2.1 — (top plot) and its differenced representation (bottom plot).

Visually inspecting a time-series for trends or seasonal behaviour, in order to classify said time-series as stationary or not, is however a bit ambiguous. There are several methods which attempts to determine whether a time-series depicts a stationary process or not. Many methods make use of a *unit root test* and one such method is the Augmented Dickey-Fuller (ADF) test [10]. The null-hypothesis for an ADF test is that the time-series is non-stationary. As the ADF-statistic decreases it is more likely that the time-series is stationary. According to Hyndman and Athanasopoulos it is customary to accept the null-hypothesis of the ADF test if the p-value is greater than 0.05.

Revisiting the time-series in Figure 2.2, the ADF test for the original and differenced representation of the time-series are given in Table 2.1.

Representation	**ADF-statistic**	**p-value**
Original	-2.25	0.188
Differenced	-10.44	0.000

Table 2.1: ADF-statistic and p-value for the original and differenced representations of the time-series from Figure 2.2.

Due to the respective p-values for the original and differenced time-series, it the ADF test concludes that the differenced representation of the time-series is stationary while the original representation is not stationary.

2.2 Machine learning

The term *machine learning* was coined by Samuel in the 1950's [11]. In the introduction to his work, Samuel describes how his studies were concerned with: "the programming of a digital computer to behave in a way which, if done by human beings or animals, would be described as involving the process of learning". More than half a century later, this notion of machine learning is frequently used across multiple fields.

Machine learning is usually divided into two subfields: *supervised* and *unsupervised* learning [12]. The former technique focuses on prediction while the latter is concerned with description. The techniques used in this thesis are based on supervised learning and a rigorous explanation of the technique is given in Section 2.2.1.

2.2.1 Supervised learning

Imagine a set of pictures depicting various objects, for instance cats and dogs. If each picture is labeled in accordance to the *class* it belongs to (cat or dog), the condition to use supervised learning — to train an algorithm to categorize the pictures —

is met. The supervised learning-algorithm is deemed successful if it generalises to similar but unseen pictures; meaning that it can accurately classify pictures *like the pictures in the original set* not previously seen [13].

More formally, given a dataset of tuples $\mathcal{D} = \{(\mathbf{x}^n, \mathbf{y}^n), n = 1, \dots, N\}$, where $\mathbf{x}^n$ is a data point with label $\mathbf{y}^n$, the goal is for the algorithm to learn each relationship for every pair $\mathbf{x}$ and $\mathbf{y}$ such that, when given an input $\mathbf{x}^*$, from dataset $\mathcal{D}'$ previously not seen, the predicted output $\mathbf{y}^*$ is accurate [12]. The meaning of accuracy is given by a predefined loss function[1] $L(\mathbf{y}^{pred}, \mathbf{y}^{true})$ such that L is small if the prediction is close to the truth.

In the example above an arbitrary tuple $(\mathbf{x}^n, \mathbf{y}^n)$ from $\mathcal{D}$ could be (🐱, $[\,1\ \ 0\,]$), denoting that the picture is of a cat with probability 1 and a dog with probability 0. The loss function L commonly used for image classification is the *softmax function* [12]. The output from the softmax function is a $1 \times C$-dimensional vector, representing a probability distribution over C different possible outcomes. Figure 2.3 shows how an input picture is processed by a neural network which in turn produces an output. The loss L is computed as a function of the predicted label and the true label of the picture and the network parameters are updated accordingly using the backpropagation algorithm explained in Section 2.4.2.

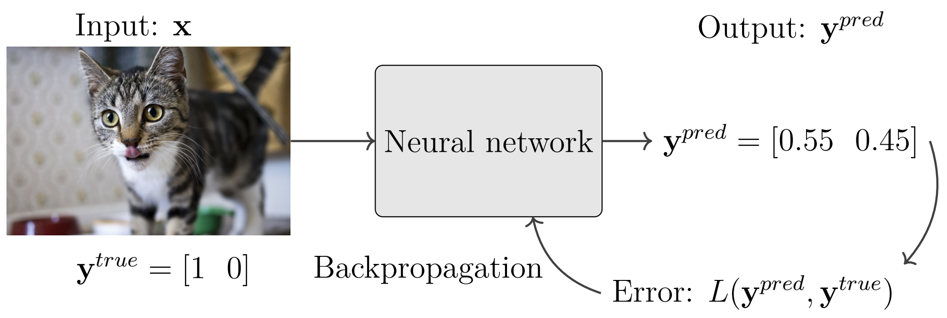
Input: $\mathbf{x}$
Output: $\mathbf{y}^{pred}$
Neural network
$\mathbf{y}^{pred} = [0.55 \ \ 0.45]$
$\mathbf{y}^{true} = [1 \ \ 0]$
Backpropagation
Error: $L(\mathbf{y}^{pred}, \mathbf{y}^{true})$

Figure 2.3: Visual representation of the different steps of the supervised learning algorithm.

The training and evaluation of a neural network are done separately. During training, the process described in Figure 2.3 is repeated for all tuples in $\mathcal{D}$. The parameters in the neural network are updated using the backpropagation algorithm and if the error of the network decreases ($L \to 0$), the performance of the network is improving. After training, $\mathcal{D}'$ is used to evaluate the performance of the network. The evaluation of the network is done using a function commonly referred to as the *metric*. The metric function could be any function which the author of the neural network deems fit. In contrast to the training phase, the parameters of the network are *not* updated during the evaluation phase.

[1]The result of the loss function is also commonly referred to as the *error* of the network.

2.2.1.1 Time-series prediction as a supervised learning problem

Time-series do not have an obvious *input/output-mapping*, like the labeled pictures with corresponding target values in Section 2.2.1, but it is possible to transform a time-series into a form apt for supervised learning.

Table 2.2 shows the first 5 steps of the speed profile time-series depicted in Figure 2.1. Table 2.3 shows the result after shifting all values in the original time-series one step in time. A mapping is created where the values in the original time-series are used as inputs **x** and the values in the shifted time-series are used as outputs **y**.

Time	Value
0	0.00
1	3.75
2	5.25
3	4.00
4	8.50

Table 2.2: The first five values from the speed-profile time-series shown in Figure 2.1.

x	**y**
×	0.00
0.00	3.75
3.75	5.25
5.25	4.00
4.00	8.50
8.50	×

Table 2.3: The values in Table 2.2 as a supervised learning problem.

The transformation[2] preserves order and can be extended to both multivariate time-series and for multi-step predictions (i.e. predicting multiple steps in time). Note that there is no input value for the first target value in Table 2.3. Neither is there any target value for the last input value in the table. During training, both these rows are commonly removed. The result of the transformation is given in Table 2.4.

x	**y**
0.00	3.75
3.75	5.25
5.25	4.00
4.00	8.50

Table 2.4: The resulting table after transforming the values of the time-series in Table 2.2 into a dataset apt for supervised learning.

2.3 Neural networks

The foundation for neural networks was laid when McCulloch and Pitts modeled a simple neural network with electrical circuits, in an attempt to describe how biologi-

[2]This transformation is commonly referred to as the *sliding window* method. In statistics the method is better known as the *lag* method.

cal neurons might work [14]. Today, there are multiple variations of neural networks, but "vanilla neural network" most commonly refers to the *multilayer perceptron* [15].

Multilayer perceptrons (MLPs) are commonly used for classification, as in the picture example in Section 2.2.1 (Supervised learning). The MLP is a *fully connected*, *feedforward* neural network with at least three layers[3]. Being a fully connected neural network implies that every neuron in a layer is connected to every neuron in the next layer. In addition to this, feedforward implies that the connections in the network never cause loops nor skip any layers. Figure 2.4 depicts an MLP with four inputs, two hidden layers, and one output.

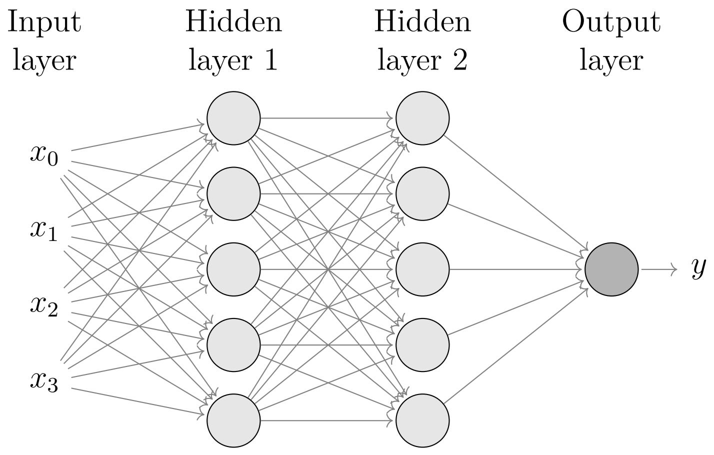

Figure 2.4: A multilayer perceptron with four inputs, two hidden layers (with five neurons in each layer) and a single output neuron in its output layer.

Each line in Figure 2.4 represents a connection between two neurons. By assigning a weight w to each connection in the network, it is possible to control which neurons have more or less influence on the data passed forward. These connections can either inhibit or exhibit the flow of data between the neurons. In addition to the weights, it is also customary to assign a bias b to every neuron of the MLP.

2.4 Training a neural network

All neural networks must be trained before they are useful. At first, the weights and biases in a network are randomly assigned — usually in a small interval around zero — and the predictions of the network will be nothing but random [16]. Training a neural network is an iterative process where each iteration constitutes of two steps: (1) a forward pass of data to retrieve a prediction and (2) a backward pass to update the parameters[4] of the network.

[3]The smallest version of an MLP has one input layer, one hidden layer, and one output layer.

[4]Here, "parameters" refers to the weights and biases of the network, but it is possible to have additional parameters.

2.4.1 Acquiring a prediction from a neural network

The forward pass of an MLP yields a prediction. With the exception of the neurons in the input layer, each neuron in an MLP applies a nonlinear activation function g to a linear combination of the data provided by the previous layer, as is visualized in Figure 2.5.

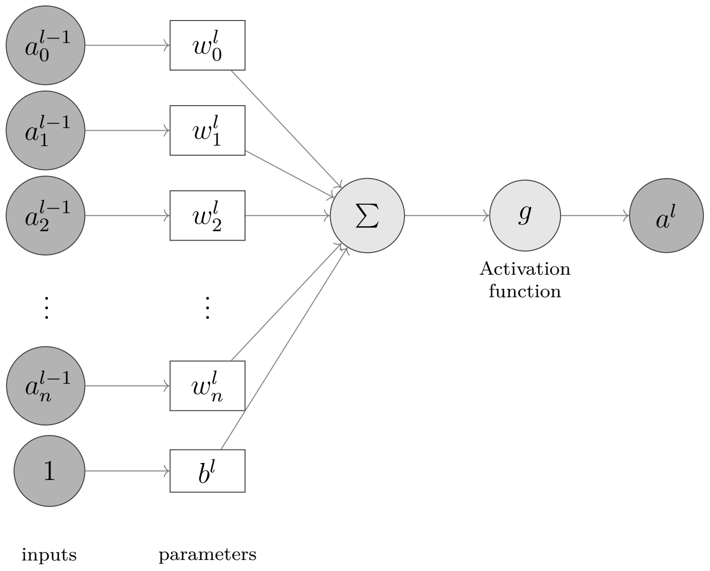

Figure 2.5: The internals of a neuron in a multilayer perceptron.

The activation of the i-th neuron in layer l depends on the parameters between layers $l-1$ and l, as well as the activations from layer $l-1$, in accordance to Equation 2.2. We denote the weight for the connection between neuron k in layer $l-1$ and neuron j in layer l as w_{jk}^l, similarly b_j^l denotes the bias of the j-th neuron in layer l. Finally, a_j^l is the activation of the j-th neuron in the l-th layer. In Figure 2.5 we assume that layer l contains only one neuron, why we may omit the double indices.

$$a_j^l = g\left(\sum_{k=0}^{n}\left(w_{jk}^l a_k^{l-1}\right) + b_j^l\right) \tag{2.2}$$

It is possible that the activation function g differs between layers in an MLP [16]. As was mentioned in Section 2.2.1 (Supervised learning), the softmax function is commonly used as the activation function for the output layer, in order to retrieve a vector which can represent a probability distribution over some classes (y in Figure 2.4). However, it is more common to use the *sigmoid function* (Equation 2.3), the *hyperbolic tangent function* (Equation 2.4) or the *rectified linear unit* (Equation 2.5) as activation functions for the neurons in the hidden layers.

$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{2.3}$$

$$\tanh(t) = \frac{e^{2t} - 1}{e^{2t} + 1} \tag{2.4}$$

$$ReLU(t) = max(0, t) \tag{2.5}$$

2.4.2 Updating the parameters of a neural network

The backward pass of an MLP is performed in order to update the parameters of the neural network, and in turn lower the error L of the network. The process constitutes of two steps: (1) computing partial derivatives and (2) updating the network parameters using the derivatives. The partial derivatives are often referred to as *gradients* and will ultimately be expressed in terms of *errors* and activations. The algorithm used for step 1 is usually *backpropagation*, an algorithm discovered by multiple researches independent of one another over a period spanning the late 1960's to mid 1980's [17], [18]. Step 2 is commonly done using stochastic gradient descent, or a similar method [19].

Intuitively the backpropagation algorithm computes the error contribution of each neuron in a neural network to some loss function L. A neuron's contribution to L depends on its corresponding weights and biases. Hence, what we want to compute is inherently $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for all weights w and biases b in the network, and change the weights and biases some amount Δw and Δb such that $L' < L$ when the forward pass is performed again to acquire L'.

In Section 2.4.1 we defined the activation of a neuron as a_j^l, in accordance to Equation 2.2. It is, however, less algebraically cumbersome to use the *weighted input* of each neuron z_j^l (presented in Equation 2.6) instead of the activation when computing said errors.

$$z_j^l = \sum_{k=0}^{n} \left(w_{jk}^l a_k^{l-1} \right) + b_j^l \tag{2.6}$$

Hence, we start by defining the error of neuron j in layer l, with respect to z, as in Equation 2.7. δ_j^l measures how much the weighted input z_j^l affects the final loss of the network.

$$\delta_j^l = \frac{\partial L}{\partial z_j^l} \tag{2.7}$$

Starting from the back of the network, δ_j^L expresses how much the activation for each neuron in the *output layer* contributes to the loss L, in accordance to Equation 2.8.

$$\delta_j^L = \frac{\partial L}{\partial a_j^L} \sigma' \left(z_j^L \right) \tag{2.8}$$

The first factor of Equation 2.8 measures how L changes depending on the activation of the j-th neuron. The second factor takes the activation function of the output layer into consideration and tells us how fast the activation function is changing at z_j^L. δ_j^L does however use the activation (a_j^L) instead of the weighted input (z_j^L) for computing the error of each neuron. This is only for the last layer of the network and the motivation for using the activation is given in Equation A.1.

After acquiring the errors in the last layer, it is possible to *propagate* the error backwards in the network. The propagation is done using Equation 2.9, which shows how the error for an arbitrary neuron δ_j^l can be computed in terms of the error from the neurons in the layer before. The derivation of the equation is given in Equation A.2.

$$\delta_j^l = \sum_{k=0}^{n} w_{kj}^{l+1} \delta_k^{l+1} \sigma' \left(z_j^l \right) \tag{2.9}$$

It is possible to repetitively apply Equation 2.9 and compute δ_j^{l-1}, δ_j^{l-2}, $\ldots$, all the way back through the network, simply by using the error from the previously calculated layer.

As was referred to earlier in the section, the errors are only part of what is required to update the network parameters. We are ultimately interested in finding out how the weights and biases of the network affects the loss function. The weights' effect on the loss of the network is given in Equation 2.10.

$$\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{2.10}$$

As can be seen in Figure 2.5 in Section 2.4.1, one can think of the biases as parameters whose input is always equal to one. Due to absence of interaction between the biases and the input to the network, the erroneous contribution from the biases are unaffected by the input. Equation 2.11 gives the biases' contributions to the error.

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l \tag{2.11}$$

After the backpropagation algorithm has calculated the gradients, the parameter update is done by subtracting a small fraction of the corresponding gradients from each weight and bias as in Equations 2.12 and 2.13. This is the stochastic gradient method, mentioned earlier in the section.

$$w_{jk}^l = w_{jk}^l - \Delta w_{jk}^l = w_{jk}^l - \eta \frac{\partial L}{\partial w_{jk}^l} \tag{2.12}$$

$$b_j^l = b_j^l - \Delta b_j^l = b_j^l - \eta \frac{\partial L}{\partial b_j^l} \tag{2.13}$$

η is referred to as the *learning rate* of the algorithm and is usually a fixed constant; albeit variations exists, such as *decaying learning rate* [16].

2.4.3 Numerical example for training a neural network

Understanding backpropagation is non-trivial, especially when one has only been presented with the abstract algorithm. Hence, this section will give a numerical example of the backpropagation algorithm for a small MLP in accordance to the network seen in Figure 2.6.

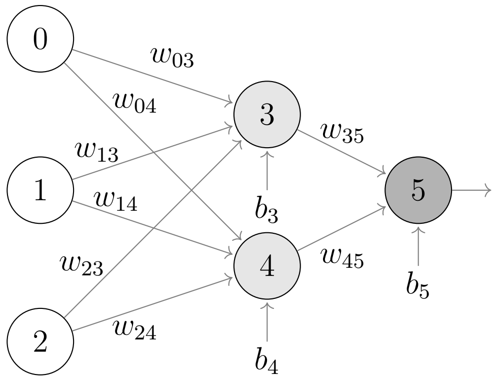

Figure 2.6: An MLP with one hidden layer to be used in the numerical example of the backpropagation algorithm.

The task at hand is that of *binary classification*, hence the output layer of the network (denoted as 5) will have a single neuron with a sigmoid function, as was presented in Equation 2.3. As loss function we use $L = \frac{1}{2}\sum(\mathbf{y}^{true} - \mathbf{y}^{pred})^2$, which is the *sum of squared errors*[5]. Furthermore, we will assume that the activation function for the hidden units (denoted by 3 and 4 respectively) is also the sigmoid function. Finally, we use a learning rate $\eta = 0.1$ to update our parameters[6].

The weights and biases of the MLP are randomly initiated to values in the interval [-0.5, 0.5], in accordance to Table 2.5. We have as input vector $\mathbf{x} = [1\ 0\ 1]^\intercal$ and the label of $\mathbf{x}$ is $\mathbf{y}^{true} = 1$.

Parameter	w_{03}	w_{04}	w_{13}	w_{14}	w_{23}	w_{24}	w_{35}	w_{45}	b_3	b_4	b_5
Value	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Table 2.5: Initial values for the parameters of the MLP presented in Figure 2.6.

The forward pass of the MLP gives rise to the values presented in Table 2.6, and in turn the prediction $\mathbf{y}^{pred} = 0.474$ and the loss $L = \frac{1}{2}(1 - 0.474)^2 = 0.1383$.

Computing the backward pass of the MLP involves taking the derivative of the loss function as well as the sigmoid function, these derivatives are presented in Equations 2.14 and 2.15 respectively.

[5]The factor $\frac{1}{2}$ is added to yield a more pleasant derivative.

[6]Due to the nature of the backpropagation algorithm, the changes made to the weights and biases in each iteration are small which in turn results in the need for several decimal points to show the changes. The example might therefore look a bit cluttered, but using larger numbers (and fewer decimals) would obfuscate the nature of the algorithm.

Neuron	**Weighted Input**	**Activation**
3	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$\sigma(-0.7) = 0.332$
4	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$\sigma(0.1) = 0.525$
5	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$\sigma(-0.105) = 0.474$

Table 2.6: The weighted inputs and activations for the hidden neurons as well as the output neuron in the MLP depicted in Figure 2.6.

$$\frac{\partial L}{\partial \mathbf{y}^{pred}} = \mathbf{y}^{pred} - \mathbf{y}^{true} \tag{2.14}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{2.15}$$

The backward pass of the MLP starts with computing the errors, as was described in Equations 2.8 and 2.9. In total there are three errors, one for the output node and one for each hidden node. These errors are computed and presented in Table 2.7. The gradients are then computed in accordance to Equations 2.10 and 2.11, yielding the result presented in Table 2.8.

Neuron	**Error**
5	$(0.474 - 1)(0.474)(1 - 0.474) = -0.1311$
4	$(-0.2)(-0.1311)(0.525)(1 - 0.525) = 0.0065$
3	$(-0.3)(-0.1311)(0.332)(1 - 0.332) = 0.0087$

Table 2.7: The errors for each neuron in the MLP after the first backward pass of the backpropagation algorithm.

Parameter	**Gradient**	**Updated Value**
w_{03}	$(0.0087)(1) = 0.0087$	$0.2 - (0.1)(0.0087) = 0.19913$
w_{04}	$(0.0065)(1) = 0.0065$	$-0.3 - (0.1)(0.0065) = -0.30065$
w_{13}	$(0.0087)(0) = 0$	$0.4 - (0.1)(0) = 0.4$
w_{14}	$(0.0065)(0) = 0$	$-0.1 - (0.1)(0) = -0.1$
w_{23}	$(0.0087)(1) = 0.0087$	$-0.5 - (0.1)(0.0087) = -0.50087$
w_{24}	$(0.0065)(1) = 0.0065$	$0.2 - (0.1)(0.0065) = 0.19935$
w_{35}	$(-0.1311)(0.332) = -0.0435$	$-0.3 + (0.1)(0.0435) = -0.29565$
w_{45}	$(-0.1311)(0.525) = -0.0688$	$-0.2 + (0.1)(0.0688) = -0.19312$
b_3	0.0087	$-0.4 - (0.1)(0.0087) = -0.40087$
b_4	0.0065	$0.2 - (0.1)(0.0065) = 0.19935$
b_5	-0.1311	$0.1 + (0.1)(0.1311) = 0.11311$

Table 2.8: The gradients, as well as the updated values, for each parameter in the MLP after one iteration of the backpropagation algorithm.

A full forward pass and backward pass has now been performed. Performing a second forward pass (with the same input and label vectors as before) yields the prediction $\mathbf{y}^{pred} = 0.478$ and the loss $L = 0.1359$. Hence, the parameters has changed such that the prediction is closer to the true label and the loss has decreased. The initial and final values are presented in Table 2.9.

	Old value	**New value**	**Target value**
$\mathbf{y}^{pred}$	0.474	0.478	1
L	0.1383	0.1359	0

Table 2.9: The predicted value and the loss before and after one iteration of the backpropagation algorithm has been performed.

2.4.4 Tuning the performance of a neural network

When discussing the performance of a neural network configuration, two common terms are *under-fitting* and *over-fitting*. Under-fitting refers to a configuration that is unable to minimize the cost function sufficiently during training [19]. The resulting predictions, both during training and testing, will likely be poor. Over-fitting occurs when the neural network configuration is *too flexible*. The cost function is properly minimized during training but as the network is evaluated on unseen data the error is significantly larger. That is, if a *training sample* is provided to the configuration, the output will likely be good, but if a previously not seen sample is provided the prediction will be poor.

There are multiple ways in which it is possible to tune the performance of a neural network configuration and this section discusses the importance of appropriate data, the shape of a neural networks model, as well as different regularization techniques used during the actual training of a configuration.

2.4.4.1 Normalizing data

It is common that the raw input data is unsuited for many neural network configurations [19] and transforming the data can greatly improve performance [12].

One common preprocessing technique is *normalization* of data. Normalizing is often done such that the training data is rescaled to the range [0, 1] or [-1, 1], in accordance to what is deemed best suited for the used activation functions. This is done because common activation functions used in most neural network configurations[7] greatly benefit from this [16].

[7]The sigmoid, tanh and ReLU functions were all defined in Equations 2.3, 2.4, and 2.5 respectively.

2.4.4.2 Shaping the neural network model

When building a neural network model, such as the MLP in Figure 2.4, the number of inputs and outputs are often determined by the dimensionality of the data at hand [15]. However, the *shape* — i.e. the number of hidden layers (depth) and the number of units per layer (width) — can be set to whatever the creator deems fit. Considering once again the MLP as an example, as the complexity of the model increases,[8] it is reasonable that the configuration should be able to mimic more complex functions. An MLP with only one neuron (and possibly a bias), and a linear activation function, could not possible mimic a more complex function than $f(x) = wx + b$, since there will be only one weight in the network[9]. As the depth and width of the neural network model is increased, f can fit more complex data.

Figure 2.7 contains three plots, each plot showing the result after one of three different neural network configurations were trained to perform regression. In this particular instance, the task of the neural network was to find the underlying structure in noisy sinusoid data. The input data to the configuration constituted of 20 real values in the interval $[-\pi, \pi]$. For each $x \in [-\pi, \pi]$, the target data was generated as $y = \sin(x) + \epsilon$, for some small, random, ϵ. A successfully trained model would be a model which produces an output that closely resembles a sinusoid. All configurations had two hidden layers with N units in each layer. $N \in [1, 3, 10]$ for the different configurations.

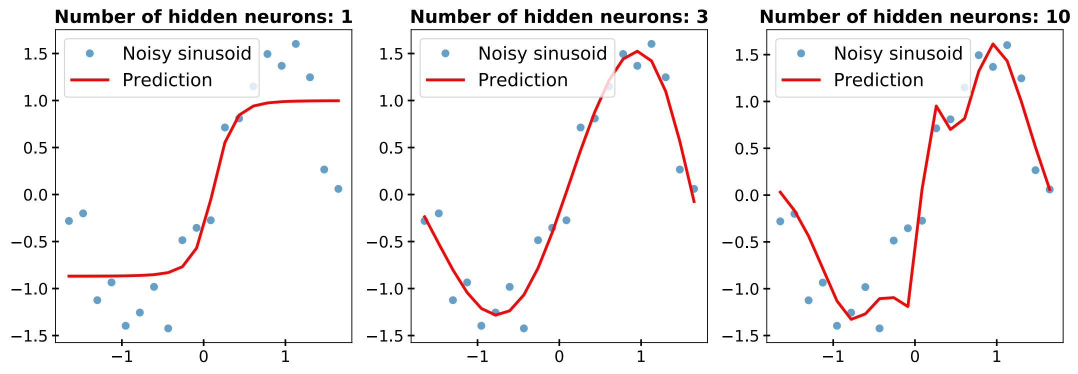

Figure 2.7: Three plots depicting how the number of hidden neurons (1, 3, and 10) can affect the outcome when attempting to solve a regression problem using a neural network configuration.

In each plot in Figure 2.7, the dots correspond to the target data y and the line is the prediction yielded by the respective configuration. The leftmost plot in Figure 2.7 shows an *under-fitted* result. The model used is not complex enough to correctly learn the underlying structure of the data. The rightmost plot in Figure 2.7 shows an *over-fitted* result. The training loss has kept decreasing, but the configuration does not generalise well and instead of finding the underlying structure of the data the configuration has learned the (more or less) exact structure of the target data.

[8]As the model is made deeper and wider.

[9]The weight connecting the input layer to the output layer

According to Goodfellow *et al.*, the best performance of any network is achieved when the shape of the neural network model corresponds to the true complexity of the task it should perform and the amount of training data at hand [19]. Finding this shape, however, is not trivial.

2.4.4.3 Early stopping

If the configuration at hand is sufficiently complex, the cost function should continue to decrease during the entire training phase but the resulting configuration might over-fit. One way to mitigate over-fitting during training is by punishing flexibility. Early stopping does this by keeping track of an additional loss or metric (usually a validation error[10]) and saves the parameters of the configuration only if said loss or metric improves. If the loss/metric stops improving, it is possible to keep training and hope that the configuration will correct itself, but if the loss/metric has remained stationary (or not improved) over a number of iterations it is more common to abort the training entirely.

According to Bishop, early stopping prevents over-fitting by restricting the space from which the network parameters can draw values [20]. More specifically, if the network parameters are initialized within some interval $[-\alpha, \alpha]$, the early stopping strategy prevents the parameters to reach values with much larger absolute values than those they were initiated to.

2.5 Cell units

There are different kinds of supervised learning-tasks and even though the MLP introduced in Section 2.3 (Neural networks) is a common neural network configuration for many supervised learning-tasks it is not the best suited configuration for *all* such tasks. Revisiting the image classification example in Section 2.2.1 (Supervised learning), an image of a cat remains an image of a cat regardless of any changes made to the remaining photos of the dataset. However, two adjacent samples in speed-profile time-series are related by physical limitations concerning acceleration and deceleration, and information about the first of the two samples gives information about the second. Therefore, we need a technique that captures the information conveyed in the *sequence* of data [21]. One such technique is *recurrent neural networks*.

[10]The validation error is acquired by evaluating the performance of the configuration on a small subset of the training data. As with the testing data, the configuration is not allowed to train on samples from the validation data.

2.5.1 Recurrent neural networks

Recurrent Neural Networks (RNNs) are networks with loops. They attempt to capture the information that could possible be found in sequences of data, by allowing for past information to remain in the network after the data has first been processed. One can think of RNNs as networks with memory and Figure 2.8 shows how a single RNN cell[11] is connected both to some succeeding layer and to itself. The data that is passed from the cell to itself gives rise to what is usually called the *hidden state* of the cell.

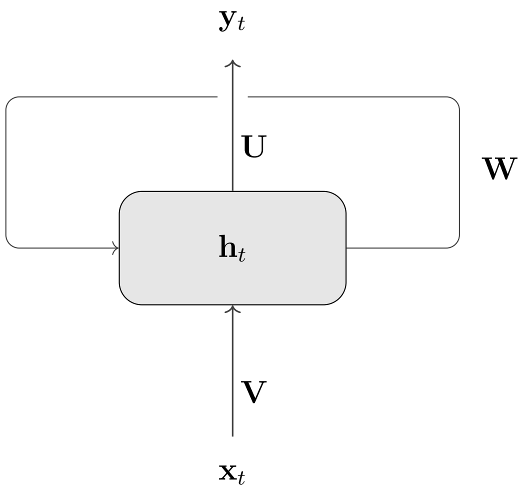

Figure 2.8: A single Recurrent Neural Network cell with input $\mathbf{x}_t$, output $\mathbf{y}_t$, hidden state $\mathbf{h}_t$, and parameter matrices $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{U}$ denoted.

Equation 2.16 describes how the RNN cell computes its hidden state, $\mathbf{h}_t$. $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{U}$ are parameter matrices with weights and biases which are trained using *Backpropagation Through Time* (BPTT), analogously to the feedforward network presented in Section 2.4.2 (Updating the parameters of a neural network), but adhering to the temporal aspect of the network. The output $\mathbf{y}_t$, in Equation 2.17, is the activation of the weighted hidden state of the RNN cell, analogously to the output layer of an MLP. Note that the activation function for the hidden state g_1 and the one which yields the output g_2 must not be the same.

$$\mathbf{h}_t = g_1\left(\mathbf{V}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}\right) \tag{2.16}$$

$$\mathbf{y}_t = g_2\left(\mathbf{U}\mathbf{h}_t\right) \tag{2.17}$$

The vanilla RNN and BPTT does however have a shortcoming: As the timespan increases, long lived dependencies from the computed gradients tend to either vanish or explode [22]. This behaviour occurs because the gradients of the hidden state are propagated by multiplication, causing the gradients to exponentially go towards either infinity (explode) or zero (vanish) [23]. Hence, conventional RNNs are deemed impractical for predictions spanning over a large time period [24].

[11]In a recurrent neural network the neurons are usually referred to as cells. We will adhere to that convention here.

2.5.2 Long short-term memory

Long Short-Term Memory (LSTM) is a technique proposed by Hochreiter and Schmidhuber as a response to the issues with conventional RNNs' inability to handle dependencies over large time spans [24]. A neural network utilizing LSTM cells is also recurrent, but the network circumvents the issue with exploding/vanishing gradients by using a *constant error carrousel* (CEC). The CEC keeps the so called *error flow* constant through the cell. Hochreiter and Schmidhuber achieved this by linearly incorporating the state of the cell in the next step in time, instead of multiplying it as in BPTT.

LSTM cells come in multiple shapes and forms, hence it is difficult to give a general description of the technique. Figure 2.9 depicts the internals of an LSTM cell as it was first presented by Hochreiter and Schmidhuber. In the cell below, the memory content $\mathbf{s}_c$ is protected by a multiplicative *input gate* which is trained to not allow for irrelevant inputs to pertubate the content. Likewise, a similar *output gate* protects other units from pertubation, should the cell store irrelevant content.

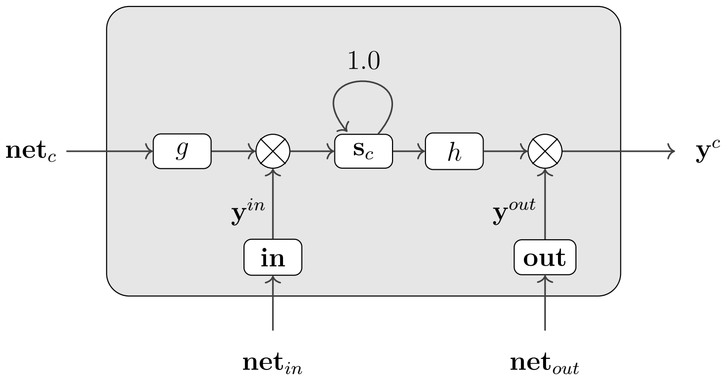

Figure 2.9: The architecture of an LSTM cell as described by Hochreiter and Schmidhuber [24].

The inputs to the LSTM cell are up to the user to specify. Hochreiter and Schmidhuber said in their original article that: "input units, gate units, memory cells, or even conventional hidden units [. . .] may convey useful information about the current state of the net" [24]. Hence, the inputs to the LSTM cell are denoted simply as **net**, with a subscript to distinguish between them.

The *internal state* of the cell is calculated in accordance to Equation 2.18, which in turn gives rise to the output of the cell, presented in Equation 2.19.

$$\mathbf{s}_c(t) = \mathbf{s}_c(t-1) + \mathbf{y}^{in}(t)g(\mathbf{net}_c(t)) \tag{2.18}$$

$$\mathbf{y}^c(t) = \mathbf{y}^{out}(t)h(\mathbf{s}_c(t)) \tag{2.19}$$

Even though there are multiple implementations of the LSTM cell, they all incorporate the CEC.

2.5.3 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is another technique for avoiding the issue with exploding and vanishing gradients, as was discussed in Section 2.5.1 (Recurrent neural networks). The GRU is less complex than the LSTM unit, yet has been shown to achieve similar or better results in some tasks [25].

GRU was presented in 2014 by Cho *et al.* and is named after its gating mechanism [26]. The gating mechanism was proven particularly successful at learning the structure of sentences on the fly. Figure 2.10 depicts the internals of a GRU as was described in the original article by Cho *et al.* [26]. Note, in particular, the gates z and r which are commonly referred to as the update and reset gate respectively.

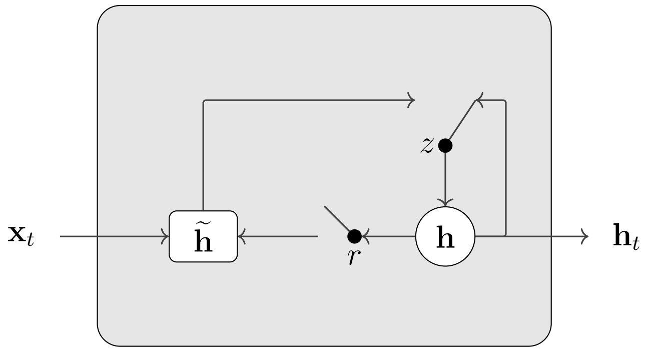

Figure 2.10: The internals of a GRU, as described in the original article by Cho *et al.* [26].

The activation at time t of the GRU $\mathbf{h}_t$ is a linear interpolation between the activation $\mathbf{h}_{t-1}$ from the previous iteration and the candidate activation $\tilde{\mathbf{h}}_t$, as given in Equation 2.20[12].

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h_{t-1}} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{2.20}$$

The update gate $\mathbf{z}_t$ dictates to which extent the GRU updates its activation and content respectively. $\mathbf{z}_t$ is computed in accordance to Equation 2.21.

$$\mathbf{z}_t = \sigma\left(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}\right) \tag{2.21}$$

$\mathbf{h}_t$ also contains a candidate activation $\tilde{\mathbf{h}}_t$ which is computed as seen in Equation 2.22.

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}\mathbf{x}_t + \mathbf{r}_t \odot (\mathbf{U}\mathbf{h}_{t-1})\right) \tag{2.22}$$

[12]The $\odot$ operator is the Hadamard product which, given two matrices of the same dimensions, produces a matrix where each element (i, j) in the resulting matrix is the product of the elements with the same indices from the original two matrices.

$\mathbf{r}_t$ is the reset gate of the GRU. When $\mathbf{r}_t \approx \mathbf{0}$, the GRU "forgets" its previously computed state and treats $\mathbf{W}\mathbf{x}_t$ as "its first input". $\mathbf{r}_t$ is computed in accordance to Equation 2.23.

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}\right) \tag{2.23}$$

2.6 Different models for predicting sequences

A neural network configuration is more than its cell units, the cell units need to be structured as a *model*. This section provides a thorough description of two models which are suitable for performing time-series predictions.

2.6.1 One to many recurrent neural network

The first model is a *one to many*-model, visualised in Figure 2.11. At each step in time the model takes a single value as input and produces a sequence as output. If the input to the model at time t is the velocity of a vehicle v_t, the output of the model is a vector of length N where N denotes "the number of future speed samples to predict". If $N = 5$ the model input will be $\mathbf{x} = [\, v_t \,]$ and the model output will be $\mathbf{y}^{pred} = [\, v_{t+1}^{pred} \; v_{t+2}^{pred} \; v_{t+3}^{pred} \; v_{t+4}^{pred} \; v_{t+5}^{pred} \,]$.

The motivation behind the model is that relevant information about the history of samples could be learned by the time dependent cell units.

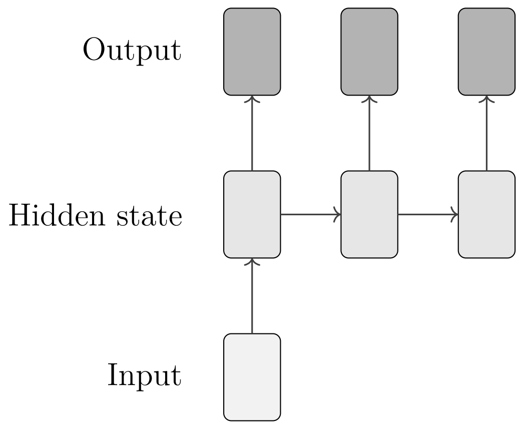

Figure 2.11: The principle layout behind a one to many-model.

2.6.2 Many to many recurrent neural network

The one to many-model in Section 2.6.1 attempts to predict some number of upcoming samples using only one sample as input. However, previous samples (v_{t-1},

$v_{t-2}, \ldots, v_0$) could possibly convey some relevant information that is *not* already captured in the hidden state of the network. In other words, it might be beneficial to provide the network with a sequence as input. The many to many-model takes a fixed number of inputs M and produces a fixed number of outputs N.

If $M = N = 5$, the input to the model at time t will be a vector $\mathbf{x} = [\, v_{t-4} \;\; v_{t-3} \;\; v_{t-2} \;\; v_{t-1} \;\; v_t \,]$ and the output of the model will be a vector $\mathbf{y}^{pred} = [\, v_{t+1}^{pred} \;\; v_{t+2}^{pred} \;\; v_{t+3}^{pred} \;\; v_{t+4}^{pred} \;\; v_{t+5}^{pred} \,]$.

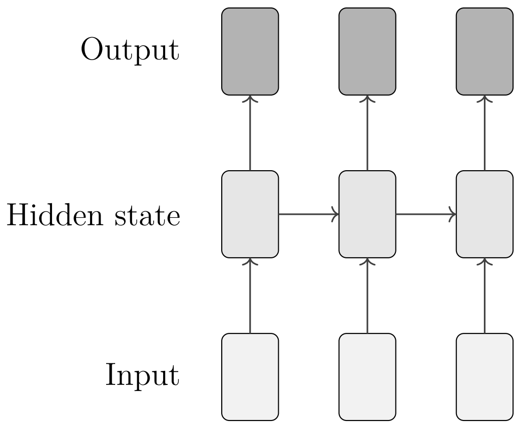

Figure 2.12: The principle layout behind a many to many-model.

3 Methodology

The methodology of choice for this thesis is *Design Science Research* (DSR). DSR is similar to the familiar methodology *empirical research*, but where the latter attempts to describe, explain, and predict the world, DSR adds the creation of *artefacts* in order to *change* the world for the better [27].

In conjunction with creating artefacts, DSR also creates knowledge about the artefacts and their environment. Hence, DSR contributes both to solving practical problems as well as increasing the knowledge about the problem area.

3.1 Activities in design science research

The method framework for design science research includes five main activities, visualised in Figure 3.1 [27].

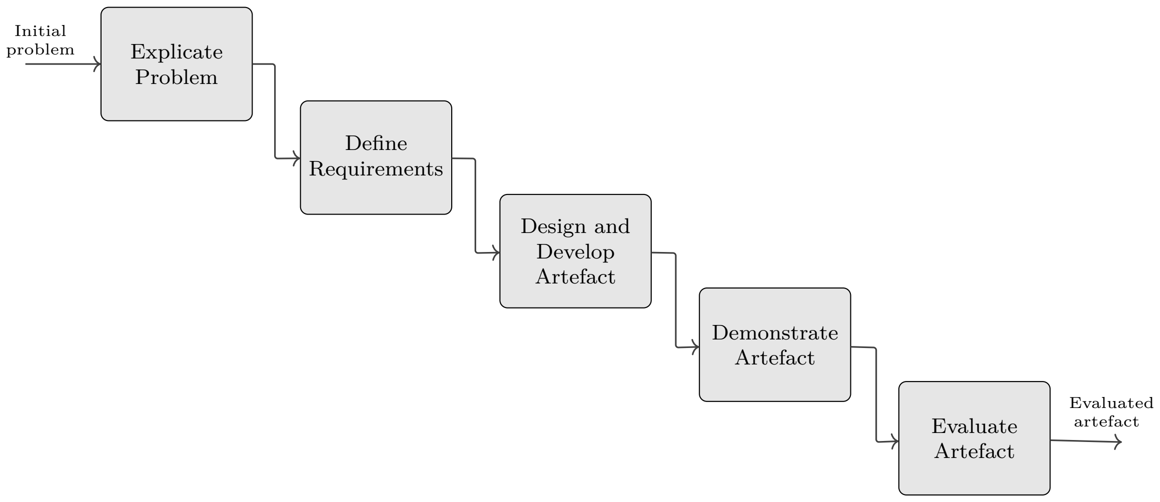

Figure 3.1: The five main activities in design science research [27].

Explicate Problem is concerned with investigating and analysing a given problem. The problem at hand should be significant within some field and during this activity the problem is scrutinised and justified.

Define Requirements outlines a solution — to the now well defined problem — in the form of an artefact and elicits requirements.

Design and Develop Artefact creates an artefact which fulfills the requirements.

Demonstrate Artefact takes the artefact and demonstrates its functionality. The demonstration should show that the artefact is capable of solving an instance of the explicated problem.

Evaluate Artefact determines how well the artefact fulfills the requirements and to what extent it is able to solve the initial problem.

Contrary to the diagram in Figure 3.1, a design science research project is not sequential but very iterative. Hence, the arrows in the diagram should not be interpreted as a temporal ordering, but a means of visualising relationships between activities; denoting what is the expected input and output of each activity.

3.2 Operationalization of research methodology

The following sections relates the thesis to each of the activities presented in Section 3.1.

3.2.1 Explicate problem

It is clear that energy efficiency is of utmost importance to Volvo Cars, and the working hypothesis is that accurate predictions about future speed-profiles can be used to improve the range of Volvo Cars' vehicles. Accurate predictions can be used to select control parameters for the vehicles' powertrain and in turn reduce the energy consumption/increase the range by/of the vehicles.

Decreasing energy consumption and increasing range for vehicles are both beneficial for society as a whole. As an example could EVs/PHEVs — charged with electricity from less pollutive energy sources — reduce the need for fossil fuel.

The research necessary to answer the problem at hand is well suited to answer questions about time-series prediction in general. However, the data used throughout the thesis constitutes of logs from the 250 C30 Electric vehicles that Volvo Cars delivered during the second half of 2011 [28].

3.2.2 Define requirements

As there are two research questions to answer the section is diveded into two parts, one for each research question.

3.2.2.1 Requirements for research question 1

The first research question concerns a screening process. This entails implementing and evaluating two different recurrent neural network models and three different neural network cell units, yielding a total of six configurations, as presented in Table 3.1. The cell units are presented in Sections 2.5.1 (Recurrent neural networks), 2.5.2 (Long short-term memory), and 2.5.3 (Gated Recurrent Unit), while the models are presented in Sections 2.6.1 (One to many recurrent neural network) and 2.6.2 (Many to many recurrent neural network).

Configuration	**Cell Unit**	**Model**
`A`	RNN	One to many
`B`	LSTM	One to many
`C`	GRU	One to many
`D`	RNN	Many to many
`E`	LSTM	Many to many
`F`	GRU	Many to many

Table 3.1: The different neural network configurations to be evaluated as part of the screening process, to answer research question 1.

Said configurations are to be evaluated using minimally preprocessed data from approximately 100 routes, split into a larger *training set* and a smaller *test set*, simple network shapes, and default hyper parameters[1].

Each model will have nine layers (1 input layer, 7 (hidden) recurrent layers and one fully connected `TimeDistributed`[2] layer as output) and each recurrent layer will be 60 cell units wide. All configurations will use the default Adam optimizer [29] and the mean squared error as loss function. The Adam optimizer will use Keras default settings [30]. The configurations will be trained for 250 epochs without any regularisation.

The configurations will be evaluated in accordance to their ability to correctly predict the future 60 speed samples of a vehicle. The metric to be used is the summed *root mean squared error* (RMSE) for all test routes for each predicted time step. The best configuration is the one whose summed RMSE over all time steps is the lowest. The evaluation process and the results for the six configurations are presented in Section 5.1.

[1]What constitutes *default* hyper parameters is clarified in Section 4.1.1.

[2]More information about the different layers is provided in Section 4.1.1.

3.2.2.2 Requirements for research question 2

The second research question focuses on using the cell unit and neural network model which answers research question 1 as a basis for achieving the highest possible performance. The best possible prediction should be achieved by adding additional data, preprocessing said data, shaping the network, etc., all in accordance to Section 2.4.4 (Tuning the performance of a neural network). However, adding new data might make it difficult to compare the result between the research questions. Hence, the evaluation process for the second research question will be split into two: one with, and one without, the new data.

The additional data available, and the explicit preprocessing of said data, is outlined in Section 4.2 but Table 3.2 presents the different parameters (and their respective levels) that can be tuned to answer research question 2.

Parameter	**Abbreviation**	**Level 1**	**Level 2**
Network depth	ND	Shallow (8)	Deep (32)
Depth of Discharge	DoD	Not Present	Present
Early stopping	ES	Off	On
Differencing	D	Off	On
Aligning	A	Off	On
Smoothening	S	Off	On

Table 3.2: Table of the different parameters and their respective levels to be evaluated when developing Artefact 2.1 and 2.2.

Testing all combinations of parameters (a 2-factorial experiment design [31]) yields a total of $2^6 = 64$ configurations to be evaluated.

In order to decrease the number of runs required, a *fractional factorial experiment design* will be adopted [31]. By assuming high-order interactions between parameters to be negligible, configurations where three or four out of the five last parameters in Table 3.2 were set to Level 2 are to be dismissed. The motivation for this is that the potential influence from either of those parameters could possibly be distinguishable from the less confounded configurations. Ultimately, the number of configurations to train are 34 per dataset.

The evaluation will be conducted in the same way as in the screening process: Each configuration will be evaluated in accordance to its ability to correctly predict the future 60 speed samples of a vehicle's speed profile in accordance to the RMSE for all test routes, for each predicted sample.

3.2.3 Design and develop artefact

The design and development processes for the respective artefacts differs little between the two research questions. The development phase concerns implementing the configurations defined by the respective requirements for Artefact 1, Artefact 2.1 and Artefact 2.2. Said configurations could be implemented from scratch using a general purpose programming language but that would entail spending a lot of time writing code to solve a problem already solved. Luckily there are several open source software libraries available which allows for a more top level approach to designing neural networks.

3.2.3.1 TensorFlow and Keras

The Google developed TensorFlow[3] is an open source software library that has quickly become one of the most popular frameworks for developing deep learning models [33]. TensorFlow performs numerical computations using data flow graphs where each node in the graph represents some mathematical operations and the edges between the nodes carries tensors[4] on which the mathematical operations are applied.

As powerful as TensorFlow is, it is not trivial to master. Better suited for quick models and proof-of-concepts is Keras, a high-level neural networks API that runs on top of TensorFlow [35]. Keras offers the power of TensorFlow at a fraction of the complexity, making it well suited for this particular purpose.

3.2.4 Demonstrate and Evaluate artefact

As previously discussed, the end goal of the thesis concerns finding the best suited configuration for performing speed-profile time-series predictions. Hence, each neural network configuration will be evaluated on its ability to yield as accurate predictions as possible. The evaluation process of a neural network configuration includes a demonstration by definition and details about the evaluation process is provided in Chapter 5.

[3]Kovalev *et al.* found Keras with *Theano* as backend to be the best suited framework for implementing deep neural networks, in late 2016 [32]. However, as the development of Theano has ceased, the runner-up backend TensorFlow was chosen instead.

[4]Tensors are generalisations of scalars and vectors used to describe linear relations between scalars, vectors and other tensors [34].

4

Implementation

This chapter describes the the development process for the various neural network configurations, defined in Section 3.2.2 (Define requirements). The implementation steps are presented in such a way that it should be possible for the reader to recreate the results presented in Chapter 5.

4.1 Keras cell units

As was mentioned in Sections 2.5.1 (Recurrent neural networks), 2.5.2 (Long short-term memory), and 2.5.3 (Gated Recurrent Unit), the different cell units have different strengths and weaknesses. Therefore, models with all three cell units were implemented and evaluated to give a better idea of which is better suited for the task at hand. Keras has integrated support for all of the above [35].

- The `SimpleRNN` class offers a fully-connected RNN where the output is fed back as input, just as was described in Section 2.5.1 (Recurrent neural networks).
- The `LSTM` class implements an LSTM as described by Hochreiter and Schmidhuber in Section 2.5.2 (Long short-term memory).
- The `GRU` class implements a GRU as described by Cho *et al.*, and in Section 2.5.3 (Gated Recurrent Unit).

Unless otherwise stated the implemented models uses the default arguments as set by Keras. The parameters for the cell units and the optimizer are given explicitly in Appendix B.

4.1.1 Keras cell unit parameters

Due to the high level of abstraction in Keras, there are a lot of arguments that can be provided to the various cell units in a model. Most of these are self explanatory[1]

[1]For instance, `units`, denoting the number of units in a layer.

but others are none obivous. Below follows brief explanations of the arguments that were used when developing the configurations used to conduct the experiments presented in Chapter 5.

Stateful is a boolean parameter which determines whether the last state of each cell in a model should be carried over from one batch to the next. When `stateful = True` is used, it gives the developer better control over the hidden state in the cell units as (s)he must explicitly program the model to reset the hidden states for the cells.

Return sequences is a boolean parameter which determines whether to output the last output in an output sequence, or the entire sequence. It is necessary to set `return_sequences = True` when stacking cell units, as this provides the next layer with one output for each input time step. Additionally, it is necessary to return sequences when the goal is to predict a sequence of outputs with a `Dense` output layer wrapped in a `TimeDistributed` layer.

Return states is a boolean parameter which determines whether the specified layer should return only the output of the layer or also include the hidden (and potential cell) state of the layer.

Time distributed is a *layer wrapper* which is often used in combination with the `Dense` layer. By adding, for instance, a layer `TimeDistributed(Dense(1))` as output layer to a model, the resulting output will be a vector of some length equal to the number of values in the sequence returned by the previous layer.

4.2 Preprocessing of data

Artefact 1 allowed for "no unnecessary preprocessing" of the data. Hence, the only preprocessing done was reformatting the data such that it was better suited for supervised learning, as described in Section 2.2.1.1 (Time-series prediction as a supervised learning problem), and normalizing the data to the interval [-1, 1] in accordance to Section 2.4.4.1 (Normalizing data). As part of developing Artefact 2.1 and 2.2, a large part of the implementation phase was concerned with feature selection and preprocessing of the data.

Section 4.2.1 and 4.2.2 are relevant for the development and evaluation of all artefacts, as it concerns the Volvo C30s' logging systems and the log data as a whole. The succeeding sections concerns only Artefact 2.1 and 2.2, as they deal with additional feature extraction and preprocessing.

4.2.1 The raw log data

Firstly, the log files produced by the Volvo C30s' logging systems are not divided into *driving cycles* or some fixed interval of time. Hence, before any meaningful feature extraction could be performed, log files had to be merged, sorted and split into files covering one calendar day each, in order to better resemble driving cycles. The feature Power Mode, found in the log data, denotes the *ignition key position*[2] and was successfully used to filter out irrelevant data[3]. After this initial filtering the feature was discarded as the Power Mode would have the same value for all extracted data points.

Furthermore, the Volvo C30 log data is very verbose with respect to the number of features logged in each vehicle. Each log contained information about 63 features. Out of these features, only one (Vehicle speed) was used when developing Artefact 1. Despite the numerous features available, the developers of the Volvo C30 logging system were convinced that little additional information could be retrieved from the remaining features. Hence, only one additional feature (Depth of Discharge) was extracted from the raw log data. Table 4.1 lists the final set of features along with their units.

Feature	**Unit**
Vehicle speed	km/h
Depth of Discharge (DoD)	%

Table 4.1: The features and corresponding units of interest in the Volvo C30 Electric log data.

Finally, the logging system in the Volvo C30 vehicles is event triggered. This means that, instead of uniformly sampling the data[4], the system appends a row to the log file only if some of the 63 features changes at least some predefined amount. In addition to this, the logging system does not store all features every time an event is triggered. Instead, only the feature which triggered the event is stored. An event based logging system might use less storage but if the amount a feature is required to change in order to trigger an event is too large the data will be very sparse.

This is of high relevance for the thesis because the speed of a vehicle is only logged if the measured speed changes by more than 5 km/h. This results in the raw time-series extracted from the Vehicle speed data from the log files to look little like an actual speed-profile for a vehicle. Figure 4.1 shows the result of plotting the logged vehicle speed versus time, for an arbitrary route.

[2]In total there are 9 positions including: (0) Key out, (2) Key Approved and (7) Running. The only samples stored were those were Power Mode had value 7.

[3]Irrelevant data included samples logged as a car was charging or other "non driving related" data.

[4]I.e. storing every feature every second.

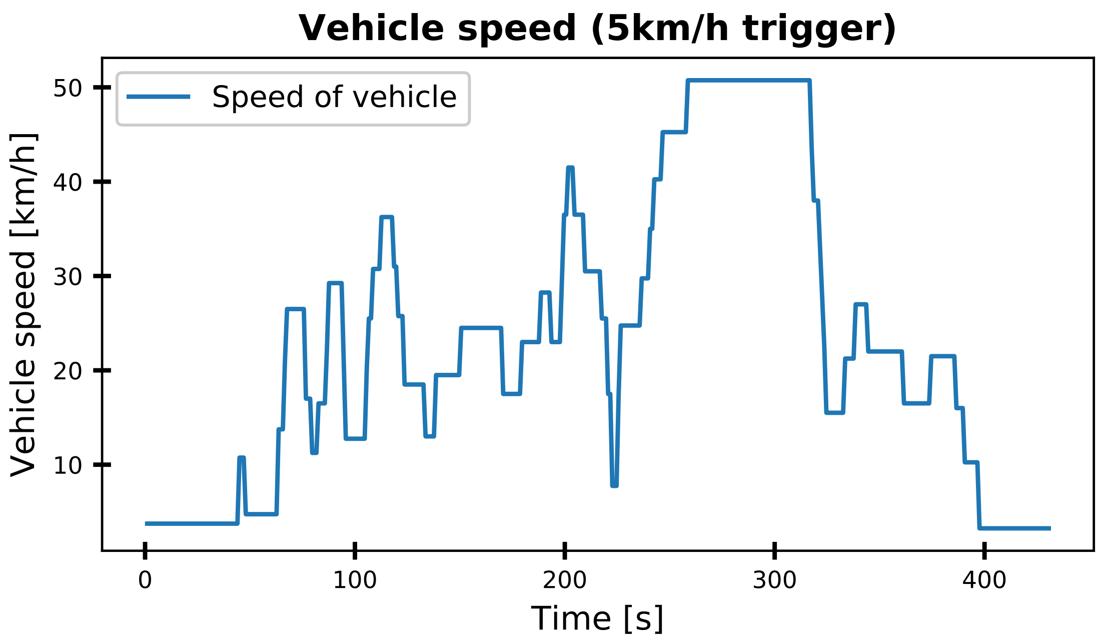

Figure 4.1: The result of plotting the sampled vehicle speed versus the time elapsed in seconds.

4.2.2 Removing and padding short routes

Despite removing the log data corresponding to “not driving” (i.e. Power Mode was not equal to 7), as was mentioned in Section 4.2.1, the logs still contained data which did not resemble relevant drive cycles. After inspection it became clear that several “drive cycles” contained little or no actual driving, but the vehicle was turned on and remained idle for a long period of time. By using the geographical data available in the log data, it was possible to remove the log data which did not correspond to a relevant geographical displacement. The filtering could possibly have been done looking only at the vehicle speed (if a majority of the speed was equal to 0, the drive cycle could have been discarded) but the geographical data was simultaneously used for aligning drive cycles (see Section 4.2.3). That method required the geographical data.

In order to allow for efficient data manipulation, it was also necessary to *pad* the routes. This entailed adding trailing zeros to the routes which were shorter than the longest route in the dataset.

4.2.3 Aligning the log data

The logging system used in the Volvo C30 vehicles is quite unreliable. When looking at the data, this is particularly evident during the beginning of driving cycles. During system boot up, the vehicles require some undefined amount of time before connecting to the server side of the system. This time vary between every startup, leading to irregularities in the log data with respect to the acquired speed-profiles. More specifically, if a driver drives the same geographical route twice — but the vehicle takes t_1 and t_2 seconds respectively to establish a connection with the server — the resulting log data will look similar, but with an offset of $t_1 - t_2$ seconds. Additionally, the event triggered logging system does not guarantee that the number of samples stored for a certain distance is the same for two drive cycles.

It seems feasible that better aligned data could make the temporal association between routes more robust. Since the neural network configurations used depends heavily on the temporal aspects of the data, avoiding issues such as the connection problem mentioned above might be very important.

Figure 4.2 shows the result of plotting the vehicle speed-profiles from two different drive cycles, for the same geographical route[5], superimposed. Due to traffic conditions and other aspects affecting the driving, the speed profiles are not exactly the same. Additionally, the speed-profile of `Drive cycle 2` is slightly shorter then that of `Drive cycle 1`.

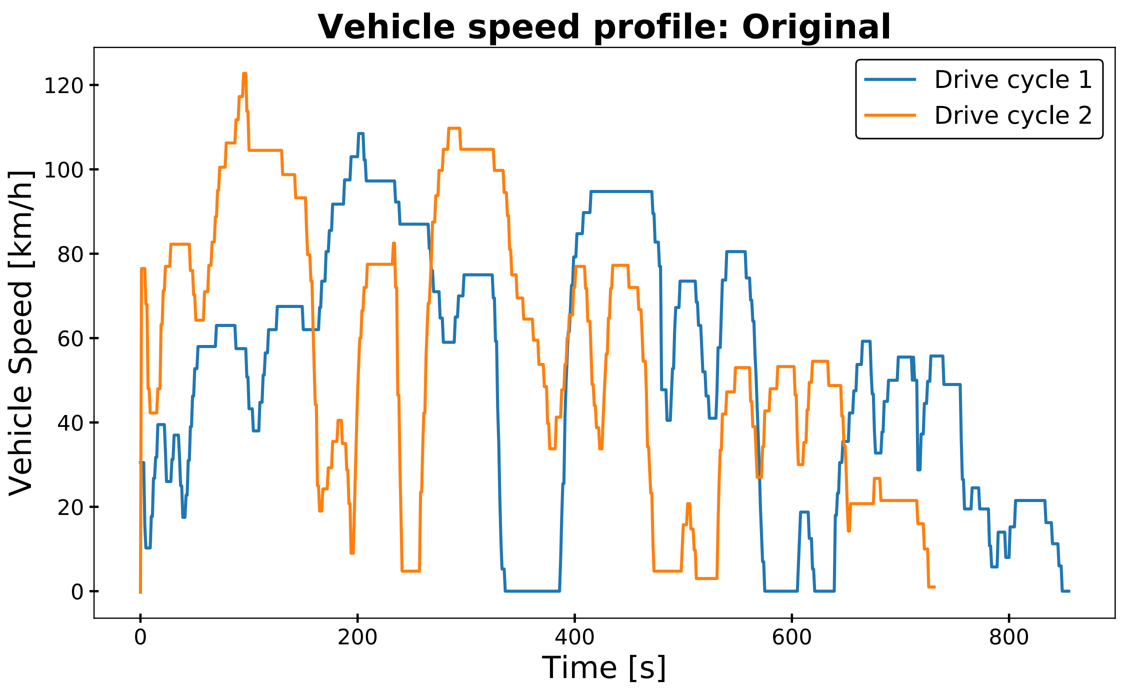

Figure 4.2: The speed-profiles of two drive cycles, for the geographically same route, superimposed.

Alignment was achieved by letting the shortest drive cycle for each geographically distinct route act as a base route. The longer drive cycles could then be truncated

[5]It is possible to conclude which speed-profiles correspond to which routes by using the location data in the log files. Note that the location data is solely used as part of the data alignment and not as part of any neural network configurations. It would *not* be possible to utilise this in an implemented system, but would either be dependant on a local model or a more robust communication protocol to ensure no data is lost due to connection issues.

to fit the shortest drive cycle such that the summed squared difference between each coordinate for the drive cycles was minimised. Figure 4.3 shows the result after the speed-profiles depicted in Figure 4.2 had been truncated and aligned.

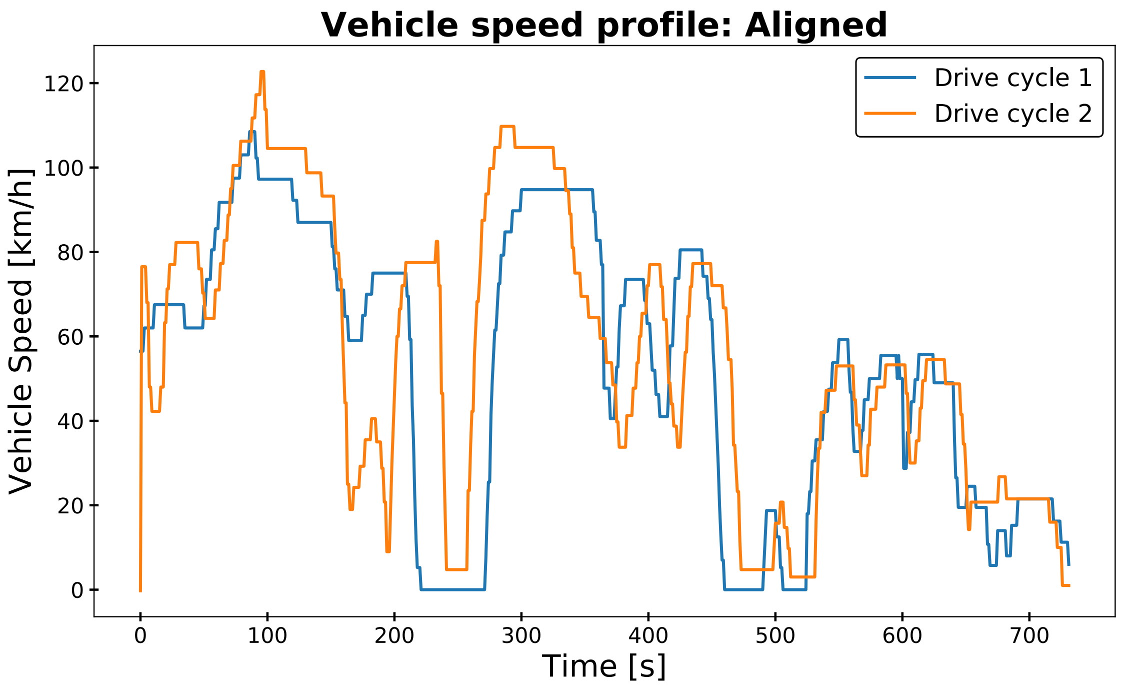

Figure 4.3: The result after the speed-profiles depicted in Figure 4.2 had been truncated to conform with the drive cycle with fewest data points.

As can be seen by inspecting the time axis and the data for `Drive cycle 2`, said cycle seems to be the shortest one for the corresponding route, as it is unchanged in its truncated representation. The speed-profile for `Drive cycle 1`, however, has been truncated both at the beginning and the end in order to acquire the closest *location wise* fit.

4.2.4 Smoothening of the vehicle speed-profiles

In order to acquire data that better resembled the speed-profile seen in Figure 2.1 in Section 2.1 (Time-Series), the speed-profiles from the Volvo C30 log data were filtered by applying a lowpass Butterworth filter of order eight, with a cutoff of 0.125 times the Nyquist rate, once forward and once backwards on the data [36]. The result is presented in Figure 4.4

However, smoothening the data is not without cause for concern. The filter makes the data smoother, but adds information to the data that was not previously there. For instance, if one compares the plateaux that occurs at time 260–310 in the original data to the smoothened result, the speed at the beginning and end of the plateaux has increased. The increase is not large, but due to the possible error of 4 km/h that inherently exists in the data this added information must be taken into consideration.

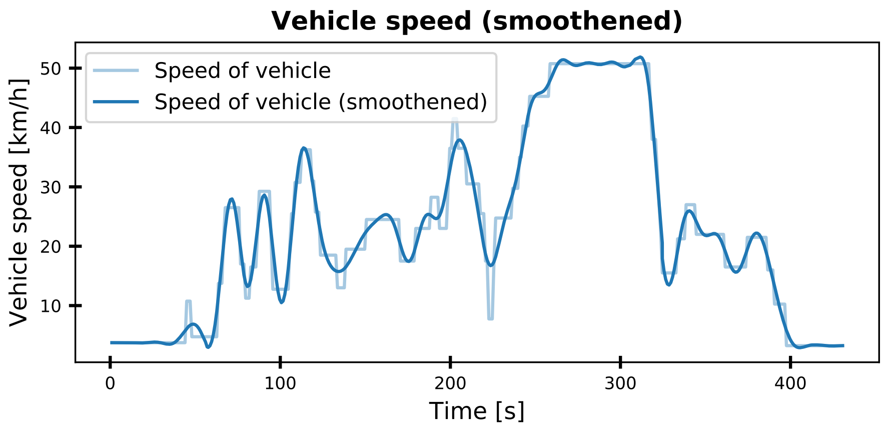

Figure 4.4: The result after smoothening the vehicle speed-profile. The smoothened speed-profile is plotted on top of the original speed-profile.

Additionally, the low speed sampled at time 220 in the original data is smoothened out to approximately 18 km/h instead of the original 8km/h.

4.2.5 Differencing

A sample of the data was tested for stationary characteristics in accordance to Section 2.1.1 (Stationary and non-stationary processes). The conclusion reached was that *all* samples got a lower ADF-statistic after differencing, but that the p-value for *some* routes was sufficiently low *before* differencing. However, the number of routes whose characteristics was non-stationary was large enough to justify adding a differenced representation of the data. Figure 4.5 shows an arbitrary time-series from the log data as well as its differenced representation.

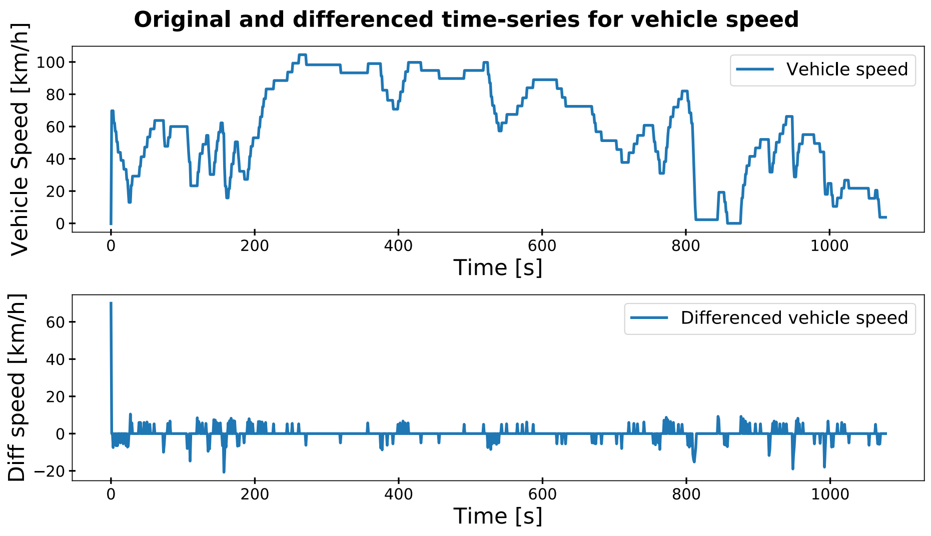

Figure 4.5: Original (top plot) and differenced (bottom plot) representation of an arbitrary vehicle speed time-series from the Volvo C30 Electric log data.

5 Evaluation

Due to the successive ordering of the research questions, the evaluation process — as was the development process — is split into two distinct parts. Initially the configurations outlined in Section 3.2.2.1 (Requirements for research question 1) were evaluated. This resulted in Artefact 1 and Section 5.1 describes the evaluation process for said artefact.

After Artefact 1 had been evaluated, it was used as a foundation to develop and evaluate Artefact 2.1 and 2.2, in accordance to Section 3.2.2.2 (Requirements for research question 2). The evaluation processes for Artefact 2.1 and 2.2 are presented in Section 5.2.

The metric for determining the performance of each configuration (as was discussed in Section 3.2.2, Define requirements) is obtained in accordance to Algorithm 1.

Algorithm 1: Pseudo code for how to compute the performance metric.

```
1 for time-series 1 to T do
2     for sample (second) 1 to S do
3         obtain prediction vector from neural network configuration;
4         compute RMSE vector, from prediction vector and actual data;
5     end
6 end
7 compute average RMSE for each sample, from RMSE vectors;
8 compute summed RMSE, from sample-wise average RMSE;
```

The prediction vector and RMSE vector on rows 4 and 5 have shape $(1, 60)$. When all time-series have been processed, the result is a vector of shape $(T \times S, 60)$. From this vector, the average RMSE for each sample is computed, resulting in a vector of shape $(1, 60)$ on row 8. The summed RMSE on row 9 is a scalar which is obtained by summing the elements in the aforementioned vector.

Each evaluation phase was run multiple times, in order to increase the robustness of the experiment. Every run started with selecting speed-profiles for the training set and test set at random, from the overall set of available speed-profiles.

The hardware used to train the configurations differs between the first two artefacts

(Artefact 1 and Artefact 2.1) and the third (Artefact 2.2)[1]. Due to this, the times denoted for the different configurations should not be compared directly, if they were not trained on the same system.

5.1 Evaluating Artefact 1

The six configurations defined in Table 3.1 were evaluated in accordance to the requirements defined in Section 3.2.2.1 (Requirements for research question 1). As was stated in that section: the only things that differed between the configurations were the cell units and model types. Each configuration was evaluated five times, using 108 speed-profiles, and the 108 speed-profiles were processed in accordance to the process described in Section 4.2.2 (Removing and padding short routes).

A *trivial baseline prediction* (Configuration `T`) was added to provide a point of reference which did not depend on any complex algorithms or neural network configurations. The trivial predictor takes as input a speed sample and repeats the sample N times. If $N = 5$, and the trivial predictor receives as input v_t, the configuration will output $\mathbf{y}^{pred} = [v_t \;\; v_t \;\; v_t \;\; v_t \;\; v_t]$. Figure 5.1 shows the result of plotting the trivial predictions on top of an arbitrary speed-profile. It is worth noting that the padding mentioned in Section 4.2.2 (Removing and padding short routes) favors the trivial predictor as any horizontal line will correspond exactly to the output of the trivial predictor.

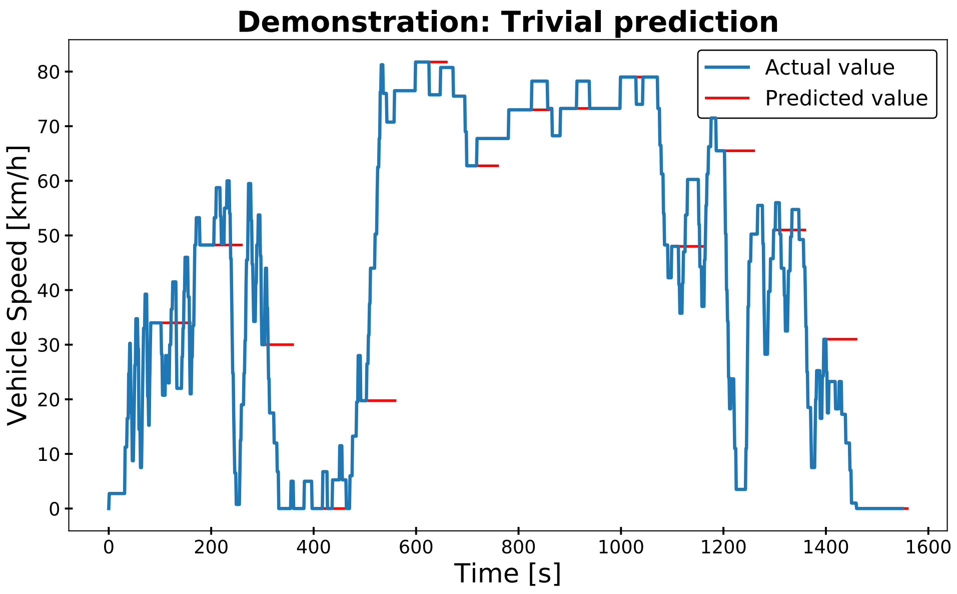

Figure 5.1: An example of the result acquired when applying the trivial prediction on every 100th sample of an arbitrary vehicle speed time-series.

[1]Artefact 1 and 2.1 were trained and evaluated on a system with an Intel Atom C3558 CPU and 16 GiB of RAM. Artefact 2.2 was trained and evaluated on a system with an NVIDIA Quadro M4000 GPU, an Intel Xeon E5-2620 CPU and 64 GiB of RAM.

Table 5.1 shows the result from the evaluation. Due to the multiple runs of each configuration, as was eluded to earlier, the RMSE column in Table 5.1 denotes the *average* of the five summed RMSE values acquired during evaluation. Table 5.1 also shows the average training time for each configuration. A verbose table, constituting of all runs for all configurations, can be found in the Appendix (Table C.1).

Configuration	**Cell Unit**	**Model**	**RMSE**	**Time**
T	*N/A*	*N/A*	*1078*	*N/A*
A	RNN	One to many	979	1:24
B	LSTM	One to many	1561	5:35
C	GRU	One to many	852	4:33
D	RNN	Many to many	833	1:24
E	LSTM	Many to many	1354	5:28
F	GRU	Many to many	849	4:37

Table 5.1: Table showing the average summed RMSE for each configuration as well as the training time for each configuration in hours and minutes.

Figure 5.2 shows a plot where the configuration with lowest overall RMSE (the many to many-model with RNN cell units, Configuration D) is used to plot its prediction — for every 100 data points — on top of the actual vehicle speed time-series, for an arbitrary series from the test data. Note that the visualisation shows only every 100th prediction, but the RMSE evaluation was done using all predictions.

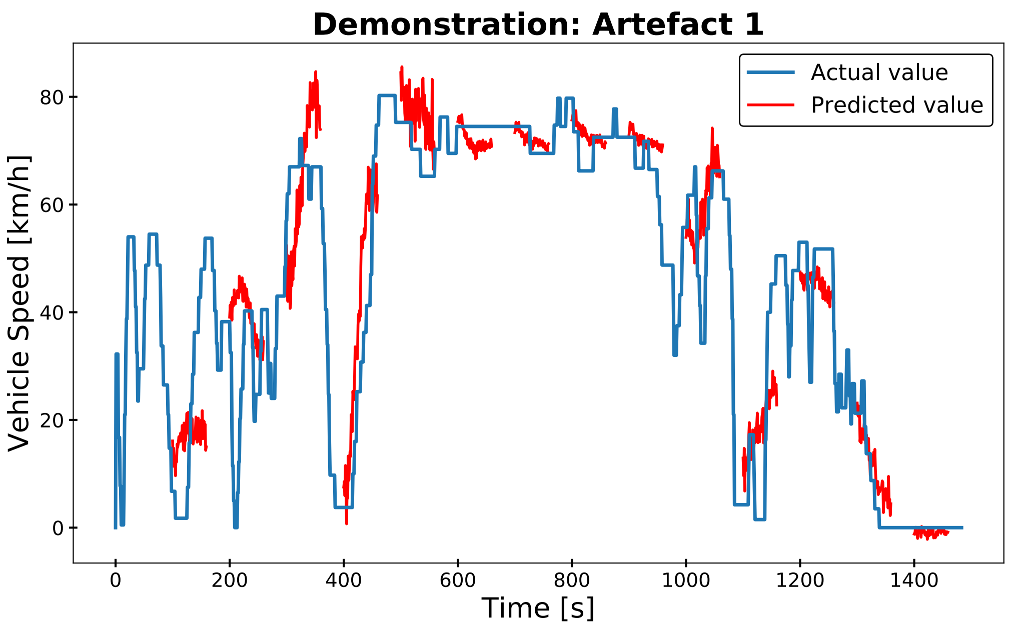

Figure 5.2: The result when the best configuration from Table 5.1 is used to predict the 60 future speed samples for every 100 points of actual data.

5.1.1 Analysis of the result from evaluating Artefact 1

Looking at the result in Table 5.1, the many to many-model with RNN cell units is the best suited configuration for predicting the upcoming 60 speed samples of a speed-profile. The average RMSE per sample for the best configuration is ≈ 14 km/h and the summed RMSE of the model's predictions is 16 km/h lower than that of the second best configuration (the many to many-model with GRU cell units). 16 km/h over 60 samples, however, is not a very large difference, as the average difference per sample is ≈ 0.26 km/h. With an inherent margin of error of 4 km/h, due to the event triggered sampling of the vehicle speed as was discussed in Section 4.2.1 (The raw log data), this margin is very small.

In addition to the average RMSE per sample we can also analyse the average RMSE per *time-step*. Figure 5.3 shows the average RMSE for each time-step and configuration[2]. The plotted lines are labeled in accordance to Table 5.1.

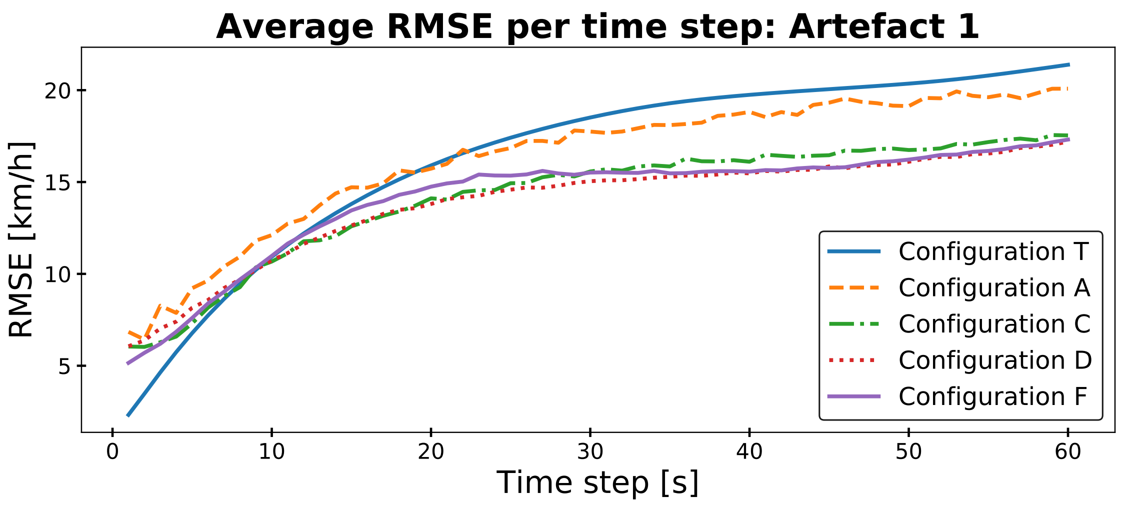

Figure 5.3: Plot showing the average RMSE for each time-step, for the five best configurations.

An interesting observation is found when comparing the many to many-models with RNN and GRU cell units (configurations `D` and `F` respectively). Initially, the GRU configuration has the lower average RMSE but around $t + 8$ the RNN configuration intersects the former and the average RMSE of the many to many-model with GRU cell units becomes higher than even that of its one to many-model counterpart, for about 15 seconds. Around $t + 40$ the average RMSE of the many to many-models with RNN and GRU cell units converges. Furthermore, the trivial predictor has the lowest average RMSE for the first ≈ 10 samples, leading to the conclusion that for the immediate future, the best prediction is simply to predict the current velocity.

[2]The configurations using LSTM cell units (configurations `B` and `E`) performed significantly worse than the other configurations, why including them in the same plot made it difficult to visualise the difference between the remaining configurations. Hence, they are omitted from all plots.

Figure 5.4 shows the difference between each configuration and the highest performing configuration (the many to many-model with RNN cell units, Configuration D). This representation clearly shows how the two GRU configurations (Configurations C and F) predicts slightly better than the overall highest performing configuration for the initial couple of samples, but slightly worse over a longer period of time. Additionally, the average RMSE for the trivial predictor (Configuration T) for the initial ≈ 5 time-steps is significantly lower than that of all other configurations. The predictions of the trivial predictor are not worse than those of the best configuration until the $t + 10$th sample.

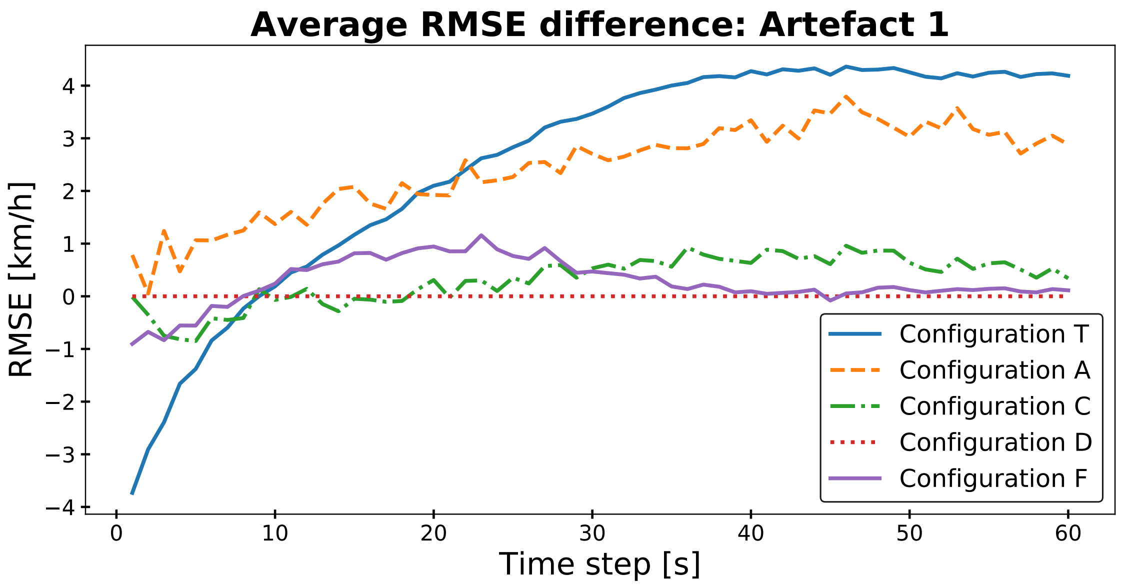

Figure 5.4: Plot showing the difference in average RMSE between the five best configurations, using the highest performing configuration as a baseline. Refer to Table 5.1 for the configurations abbreviation list.

5.2 Evaluating Artefact 2

After concluding the evaluation process for Artefact 1, the many to many-model with RNN cell units (Configuration D) was deemed the best suited configuration to answer research question 1. Hence, this configuration was used as the basis during the development of Artefact 2. Artefact 2 was evaluated in accordance to the requirements specified in Section 3.2.2.2 (Requirements for research question 2).

Artefact 2 introduces both new parameters to tune and additional data (an additional data feature and more speed-profiles). Hence, the evaluation process for the second artefact is split into two: one where only the new parameters and feature are evaluated (Artefact 2.1), and one where the new parameters, feature, *and* the additional number of routes are evaluated (Artefact 2.2).

The best configurations for each evaluation process will be shown in the sections below. In accordance to Table 3.2, the - and + in the result tables denotes a parameter being set to “Level 1” (non-present/inactive) and “Level 2” (present/active) respectively.

5.2.1 Artefact 2.1

Artefact 2.1 evaluates the 34 configurations presented in Section 3.2.2.2 (Requirements for research question 2), with the same number of routes as was used for the evaluation of Artefact 1 (108 routes). Each configuration was evaluated five times and the result presented is the average acquired over the five runs. Table 5.2 shows the configuration, parameters used, and the result for the six best configurations. The complete results can be found in the Appendix (Table C.2).

Configuration	ND[1]	DoD[2]	ES[3]	D[4]	A[5]	S[6]	RMSE	Time
0	-	-	-	-	-	-	663	1:35
1	-	-	-	-	-	+	675	1:34
9	-	+	-	-	-	+	680	1:35
5	-	+	-	-	-	-	717	1:34
6	-	-	-	-	+	+	756	2:12
12	-	+	-	-	+	-	780	2:13

1: Network depth, 2: Depth of discharge, 3: Early stopping, 4: Differencing, 5: Aligning, 6: Smoothening

Table 5.2: Table showing the six configurations with lowest average RMSE after the evaluation process for Artefact 2.1.

Figure 5.5 shows a plot where the configuration with lowest overall RMSE (the configuration with no additional preprocessing or tuning, Configuration 0) is used to plot its prediction — for every 100 data points — on top of the actual vehicle speed time-series, for an arbitrary series from the test data.

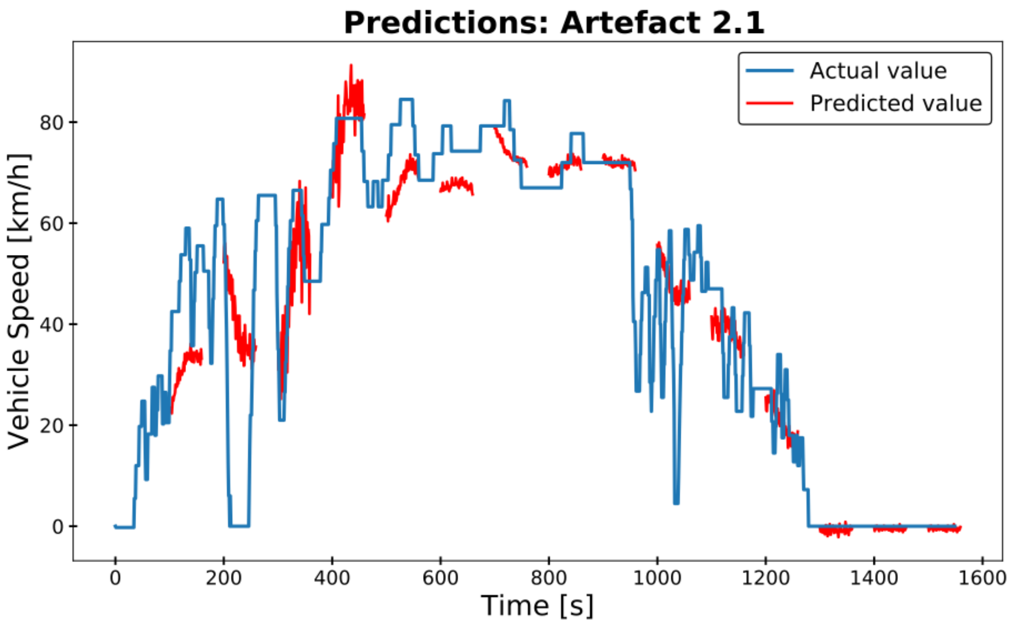

Figure 5.5: The result when the best configuration from Table 5.2 is used to predict the 60 future speed samples for every 100 points of actual data.

5.2.2 Analysis of the result from evaluating Artefact 2.1

The best configuration for Artefact 2.1 (Configuration `0`) makes use of no additional preprocessing or tuning and is therefor very similar to the best performing configuration of Artefact 1 (Configuration `D`). However, the configurations evaluated as part of Artefact 2.1 has one more hidden layer than those evaluated as part of Artefact 1, in accordance to Table 3.2. The average RMSE per sample for the predictions of Configuration `0` is $\approx$ 11 km/h, 3 km/h lower than that of Configuration `D`.

The second and third best configurations (Configuration `1` and Configuration `9` respectively) both make use of smoothening but the third best configuration also makes use of the additional data feature “Depth of discharge”. The second and third best configurations have an average RMSE of only 0.20 km/h and 0.28 km/h per sample higher than the best performing configuration, respectively. It is worth noting that three out of the six configurations in Table 5.2 make use of the smoothening parameter and equally many use the “Depth of discharge” parameter, but that no configuration among the top six uses the deep neural network model, early stopping, nor differencing.

As for Artefact 1, it is interesting to depict the difference in RMSE per time-step. Figures 5.6 shows the average RMSE per time-step for the configurations in Table 5.2. The plot shows a similar behaviour as in Figure 5.3: the best performing configuration (the configuration without any additional preprocessing or tuning, Configuration `0`) does not yield predictions with lowest RMSE for the initial (10) time-steps, but performs relatively better as the time increases.

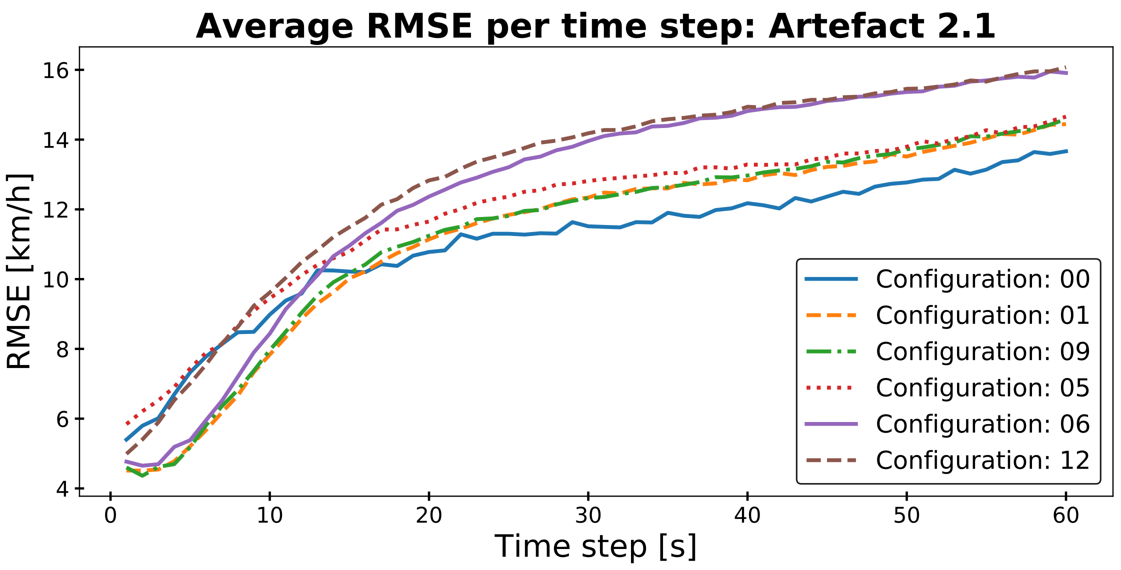

Figure 5.6: Plot showing the average RMSE per time step, for the top six configurations evaluated as part of Artefact 2.1. Consult Table 5.2 for the configuration abbreviations.

Figure 5.7 shows the difference between the best performing configuration (Configuration `0`) and the other configurations in Table 5.2 more clearly.

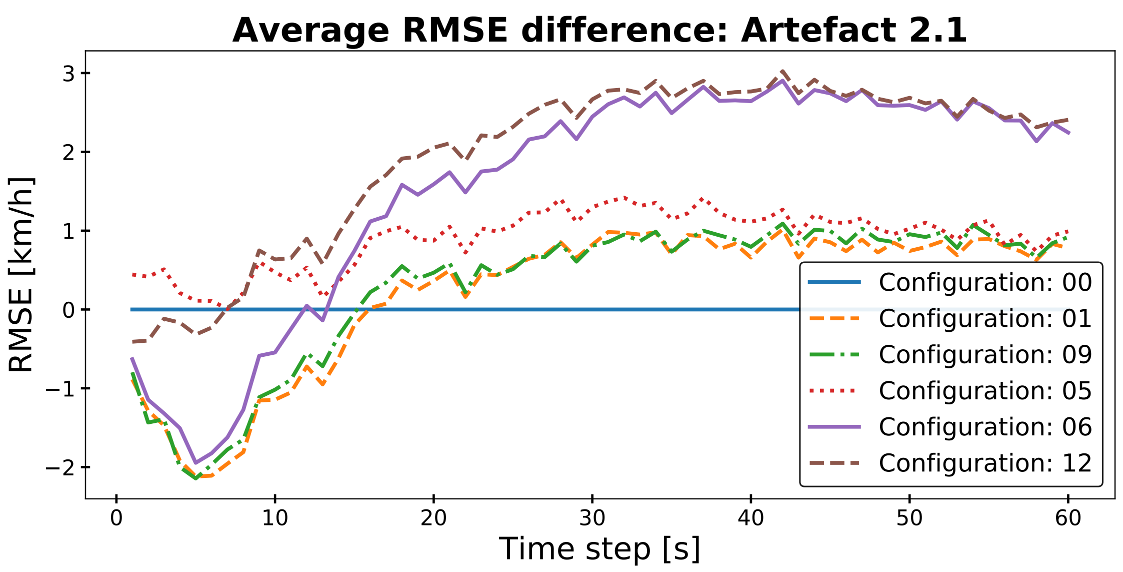

Figure 5.7: Plot showing the difference in average RMSE between the top six configurations evaluated as part of Artefact 2.1, using the best performing configuration (Configuration `0`) as a baseline. Consult Table 5.2 for the configuration abbreviations.

The best configurations seem to be the "least complex" configurations. It is noteworthy that no configuration with more than two parameters active (set to Level 2) makes the top six list. The best configuration which utilises early stopping (parameter `ES`) is the 7th best configuration (Configuration `8`), and the best configuration which uses differencing (parameter `D`) is the 8th best configuration (Configuration `7`). The best configuration with more than two parameters active is Configuration `24`, as the 14th best configuration. It uses smoothening, differencing, and it is also the best configuration which uses the deep neural network model (parameter `ND` set to Level 2).

5.2.3 Artefact 2.2

The final evaluation process concerns the configurations from Artefact 2.1 evaluated with 832 routes, instead of 108. Each configuration was trained and evaluated three times and the result presented is the average of the result acquired over all runs. Table 5.3 shows the result for the six configurations with lowest average RMSE, as well as the result for the trivial predictor `T` defined in Section 5.1. A verbose table, constituting of all iterations for all configurations, can be found in the Appendix (Table C.3).

Figure 5.8 shows a plot where the configuration with lowest overall RMSE (Configuration `6`) is used to plot its prediction — for every 100 data points — on top of the actual vehicle speed time-series, for an arbitrary series from the test data. Configuration `6` makes use of smoothening and aligning of data. Note that the visualisation shows only every 100th prediction, but the RMSE evaluation was done using all predictions.

Configuration	ND[1]	DoD[2]	ES[3]	D[4]	A[5]	S[6]	RMSE	Time
6	-	-	-	-	+	+	520	4:45
1	-	-	-	-	-	+	524	4:35
9	-	+	-	-	-	+	534	4:44
2	-	-	-	-	+	-	539	4:46
12	-	+	-	-	+	-	549	4:39
0	-	-	-	-	-	-	573	4:28
T	NA	NA	NA	NA	NA	NA	600	NA

1: Network depth, 2: Depth of discharge, 3: Early stopping, 4: Differencing, 5: Aligning, 6: Smoothening

Table 5.3: Table showing the six configurations with lowest average RMSE, and the trivial predictor T, after the evaluation process for Artefact 2.2.

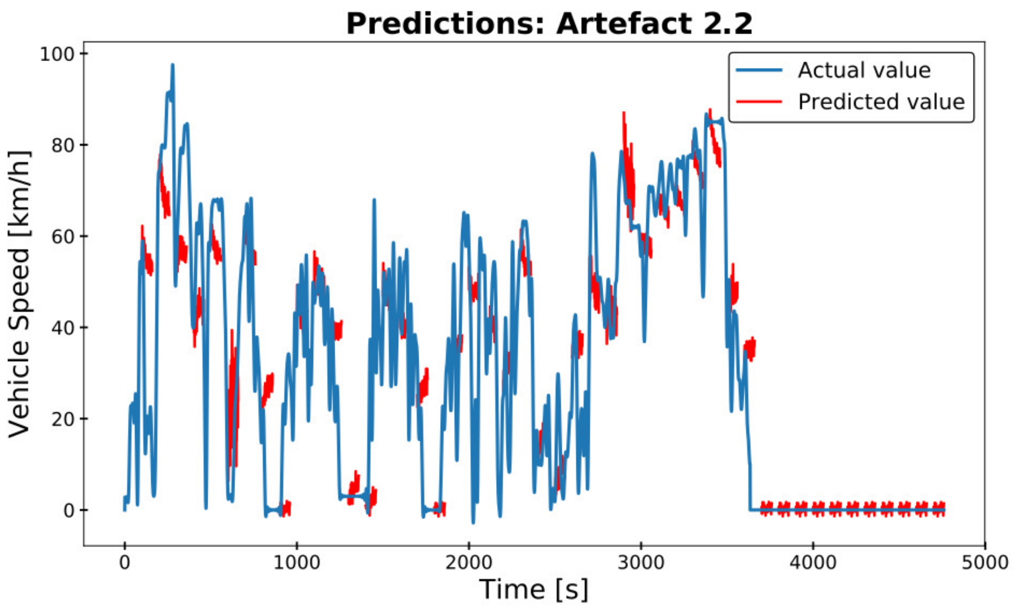

Figure 5.8: The result when the best configuration from Table 5.3 is used to predict the 60 future speed samples for every 100 points of actual data.

5.2.4 Analysis of the result from evaluating Artefact 2.2

An initial observation with respect to Artefact 2.2 concerns not the configurations but the data used. Looking at the time-axis in Figure 5.8 we see that the number of samples for the longest speed-profile has increased, with respect to the initial 108 routes used when evaluating Artefact 1 and Artefact 2.1. This is due to some outliers in the data set from which the speed-profiles, for evaluating the configurations, are drawn. One really long speed-profile will cause very large padding of the normal length speed-profiles. In turn, this results in long tails of zeroes for all but the longest speed-profiles and a configuration which learns to correctly classify 0 km/h will have a big advantage. It is possible that the result is skewed because of this.

The result of the trivial predictor `T` makes the padding issue more apparent. The RMSE of `T` was 1078 when more similar length speed-profiles were considered, but the RMSE has dropped to 600 due to the long tails of zeroes in the larger evaluation set. Considering the performance metric for the best performing configuration (Configuration 6), the average RMSE per sample is ≈ 9 km/h, compared to ≈ 14 km/h for Artefact 1 and ≈ 11 km/h for Artefact 2.1.

Analogously to both Artefact 1 and Artefact 2.1, Figures 5.9 and 5.10 show the average RMSE per time-step and the difference between the best configuration and the remaining six configurations presented in Table 5.3, respectively. The plots shows that the difference between the top six configurations is small. Figure 5.10 clearly shows how the difference between Configuration `6` and the remaining top 5 configurations is within ±1 km/h per time-step.

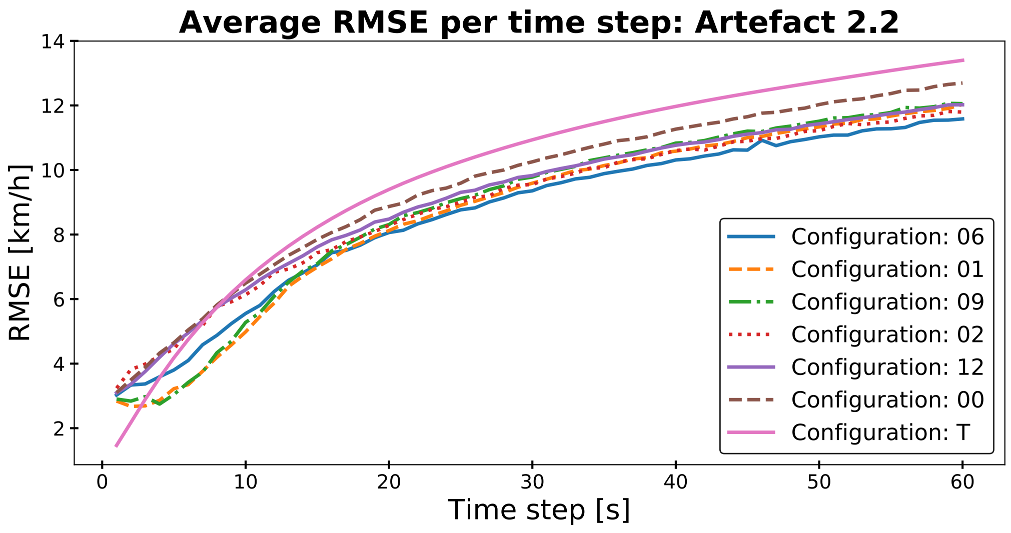

Figure 5.9: Plot showing the average RMSE per time step for the configurations in Table 5.3.

As was the case for Artefact 2.1, the result clearly shows that a deeper network is not beneficial. Consulting Table C.3, we see that the best configuration which utilises the deep network model (Configuration `18`) has an average summed RMSE of 645. This makes it the 11th best configuration. Just before Configuration `18` — as the 10th best configuration — we find the first configuration which uses differencing of the data (Configuration `3`) with an average summed RMSE of 613. Lastly, the most successful configuration which uses early stopping is Configuration 4 with an average summed RMSE of 608, making it the 8th best configuration.

Focusing on the top five configurations, it is interesting to compare them to the 6th best configuration (the configuration without any additional preprocessing or tuning, Configuration `0`). It seems feasible that smoothening (parameter `S`) and aligning (parameter `A`) are beneficial, as all top five configurations has at least one of these parameters active. On the contrary, "Depth of discharge" (parameter `DoD`) seems to have little or no effect on the performance. It might be the case that parameter `DoD` is only useful when used together with either `S` or `A`, but Table 5.3

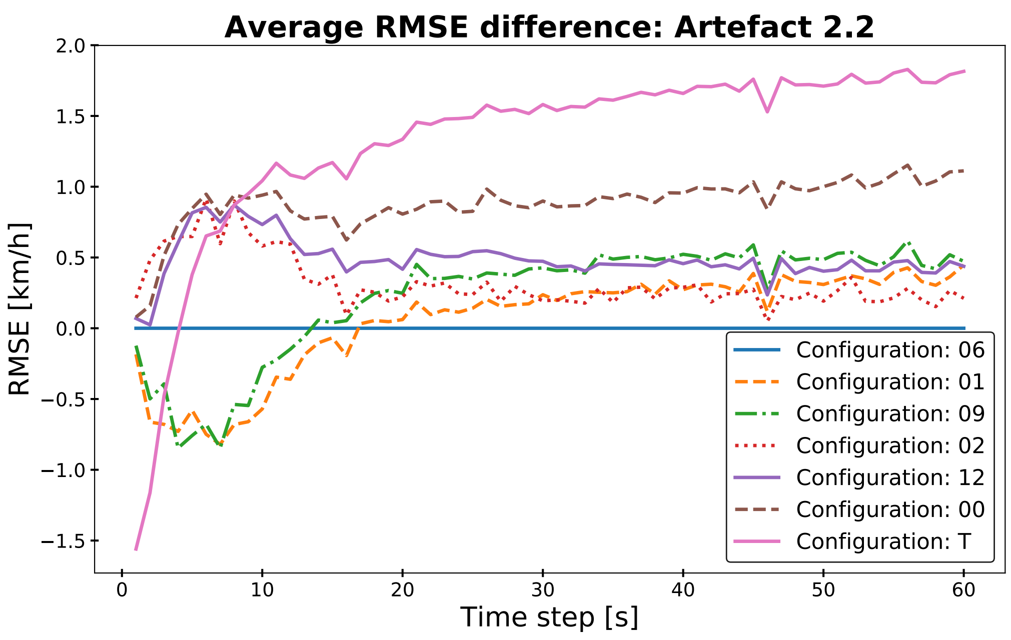

Figure 5.10: Plot showing the difference in average RMSE between the configurations in Table 5.3, using the best performing configuration (Configuration `6`) as a baseline.

shows that the configurations where `DoD` is active is always worse than the case where the parameter is removed. This is extra clear when considering Configuration `7` which uses *only* parameter `DoD` and is the 7th best configuration (i.e. one below Configuration `0` which uses no additional preprocessing or tuning).

5.2.5 Comparison of Artefact 2.1 and Artefact 2.2

The difference in the evaluation process between the two artefacts is the data. As was discussed in Section 5.2.4, the issue with the padding becomes clear when considering the large improvement of the trivial predictor's performance. However, considering the top 6 configurations for both evaluation processes, the result is quite clear with respect to which parameters yields the highest performance.

Configurations: 0, 1, 6, 9, and 12 are among the top 6 configurations for both evaluations, and therefore present in both Table 5.2 and Table 5.3. The two differences between the two tables are: (1) Configuration 5 makes the top 6 list when evaluating Artefact 2.1 while Configuration 2 makes the top 6 when evaluating Artefact 2.2, and (2) the mutual arrangement of the configurations differs between the evaluation processes. Both processes does however show that the less complex configurations yield higher performance. Additionally, configurations with parameter `DoD` (Depth of Discharge), is commonly found *below* the configuration with same parameters but with `DoD` set to Level 1.

Finally, the parameters which are set to Level 2 for the configurations in Tables 5.2 and 5.3 are `DoD`, `A` (Aligning), and `S` (Smoothening). The last two of these preprocessing steps were both added in order to make the data more similar to what it would look like if one were to implement a speed-profile prediction algorithm in a car, where vehicle speed could be sampled at will and connectivity issues are easier to avoid. Hence, it seems likely that more frequently sampled data (as was seen in the time-series example in Figure 2.1) and speed-profiles without truncated beginnings and ends (as was discussed in Section 4.2.3) could benefit the result.

5.3 Comparison of the result to previous studies

As was eluded to earlier, the best configuration from research question 1 performs better than a trivial predictor. The RMSE of Artefact 1 is 0.77 times that of the trivial predictor. In turn, the RMSE of Artefact 2.1 is 0.80 times that of Artefact 1 and the RMSE of Artefact 2.2 is 0.62 times that of Artefact 1. Whether this justifies implementing a recurrent neural network configuration as the basis for a time-series decision making process depends on the specifics of the concerned project. As was presented in Section 5.2.4 (Analysis of the result from evaluating Artefact 2.2), the average RMSE per sample for the best performing configuration is $\approx$ 9 km/h. It might be the case that this is an unacceptable margin of error for a vehicle speed-profile estimator.

However, the results presented for Artefacts 2.1 and 2.2 (Sections 5.2.1 and 5.2.3) shows that all visualised configurations has an RMSE $<$ 10 km/h for the predictions corresponding to the first 10 samples[3]. The results from Fotouhi *et al.* shows that their MLP yielded predictions with an RMSE $<$ 10 km/h only for the first six predicted samples [5]. Hence, Fotouhi *et al.* might benefit from implementing a recurrent neural network as the basis for their decision making algorithm and attempt to predict either as many or more samples as their current algorithm. It should, however, be said that the trivial predictor yield an RMSE $<$ 10 km/h for the predictions corresponding to the first seven samples when using 108 routes and the first 23 samples when using 832 routes, making it better than the algorithm used by Fotouhi *et al.* The low RMSE for the trivial predictor can likely be attributed to the padding of the data, which is likely the largest drawback of the experiments.

5.4 Future work

The following sections gives a brief introduction to some suggested future work.

[3]Artefact 2.2 has an RMSE $<$ 10km/h for $\approx t + 40$ but, as was discussed in the corresponding sector, this might be due to the padded data

5.4.1 Adding another model

Both models used in the thesis (Section 2.6, Different models for predicting sequences) take inputs — and produce outputs — of fixed length. In turn — as was discussed in Section 4.2.2 (Removing and padding short routes) — this requires the speed-profiles to be padded. The padding results in several short comings which could possibly be alleviated if a model capable of handling variable length inputs/outputs is used instead. The biggest drawback is probably the uncertainty with respect to the predictions, where it is difficult to attribute a good result to good predictions in general or to a configuration being capable of predicting the zeros that constitutes the padding of the speed-profiles.

The sequence to sequence-model [26], [37] is capable of taking a variable length input and generate a variable length prediction. This is achievable by splitting the model into two parts: an encoder and a decoder. The sequence to sequence-model has been successfully used within the field of machine translation [37], [38]. When used for translation, the encoder encodes a sentence in language L_1 and the decoder uses the encoded *state* to translate the sentence to language L_2.

Figure 5.11 presents the principle behind the model. On the left hand side of the figure the model receives A, B and <START> as inputs. A and B is some sequence which we want the model to *encode* and <START> is some token which denotes that the decoder should start decoding. The encoding is done by some recurrent neural network which discards the output of the network, but keeps the hidden state (and cell state in the case where the model is made up of LSTM or GRU cell units). When the input sequence has been encoded, the hidden state is transferred to the decoder which generates a prediction one sample at the time. The decoder generates some value X, which in turn is provided as input to the decoder in the next step. Eventually, the decoder generates an <END> token, ending the generated sequence.

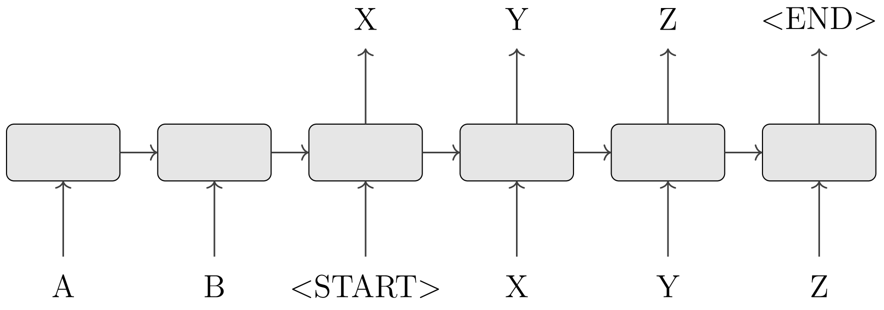

Figure 5.11: The principle layout behind a sequence to sequence-model.

An attempt to implement a configuration which made use of the sequence to sequence-model was made. The configuration was given, as input, part of a speed-profile $\mathbf{x} = [\, v_0 \;\; v_1 \;\; \dots \;\; v_{t-1} \,]$ to encode. During the decode phase v_t was given as input at iteration t whereafter a prediction v_{t+1} was retrieved. v_{t+1}, in turn, was given as input to the decoder which yielded a prediction v_{t+2}, etc. The decoder produced predictions until either an <END> token was predicted or the number of predictions reached some predetermined limit. The result for one round of predictions was a vector $\mathbf{y}^{pred} = [\, v_{t+1}^{pred} \;\; v_{t+2}^{pred} \;\; \ldots \;\; v_{t+i}^{pred} \,]$.

Unfortunately, the initial results from the model were very poor and initial training indicated that training a sequence to sequence-configuration would take significantly longer time than training the configurations with the previously mentioned models. Due to the poor initial results[4] and the indicated training time, it was decided to omit the sequence to sequence-model from the project, but the possibilities of using variable length inputs and outputs is very appealing to the task at hand.

5.4.2 Predicting distance remaining

In addition to the future speed-profile of the vehicle, the remaining distance for the current drive cycle is also of interest. A successful implementation of the previously mentioned sequence to sequence-model could yield the wanted result, as a predicted <END> token indicates that the configuration predicts the end of a route. The cumulative sum of *velocity* × *time*-products could then be used to compute the remaining distance for the drive cycle.

It is possible to use the presented configurations in an attempt to predict the remaining distance. With known sample time delta we can compute the travelled distance from the speed-profiles available. Reversing this data results in a "distance remaining" feature which can be used to train the neural network using supervised learning analogously to the vehicle speed. This approach was considered at the beginning of the thesis, and the necessary preprocessing was performed, but as the required training time became apparent the idea had to be left untested as it did not directly contribute to the aim of the thesis.

[4]It could not be excluded that the poor results were the result of an implementation error. The sequence to sequence-model is more complex than the models used throughout the thesis. Debugging the model would therefore likely be more difficult than the previous two models and this would in turn require additional time from the project.

6

Conclusion

The aim of this thesis is to investigate to what extent it is possible to use different recurrent neural networks for predicting the future speed-profile of a vehicle. In order to fulfill the objective, the thesis answers two research questions; both a broader screening process and a more detailed study of a particular neural network configuration.

6.1 Answers to research questions

The research questions, defined in Section 1.4 (Research questions), are restated and answered below.

Research question 1	Which, out of six, neural network configuration yields the best predictions, with respect to lowest summed root mean squared error, after minimal preprocessing of data and minimal tuning?

The many to many-model (Section 2.6.2, Many to many recurrent neural network) with RNN cell units (Section 2.5.1, Recurrent neural networks) achieved the lowest summed RMSE after minimal preprocessing of data and minimal tuning. The RMSE of Artefact 1 is 0.77 times that of the trivial predictor defined in Section 5.1 (Evaluating Artefact 1).

Research question 2	How should the data be preprocessed, and how should the configuration — based on the best configuration according to research question 1 — be tuned, in order for its predictions to achieve as low summed root mean squared error as possible?

When using the same amount of data as when answering research question 1 (108 routes), the lowest summed RMSE was achieved when no additional preprocessing or tuning was performed. When using more data (832 routes), the lowest summed RMSE was achieved when the data was smoothened (Section 4.2.4, Smoothening of the vehicle speed-profiles) and aligned (Section 4.2.3, Aligning the log data). The RMSE of Artefact 2.1 and 2.2 is 0.80 and 0.62 times that of Artefact 1 respectively.

Bibliography

[1] *Volvo Cars to go all electric*, Visited: 2018-02-09, 2017. [Online]. Available: https://www.media.volvocars.com/global/en-gb/media/pressreleases/210058/volvo-cars-to-go-all-electric.

[2] *The future is Electric*, Visited: 2018-05-14, 2018. [Online]. Available: https://group.volvocars.com/company/innovation/electrification.

[3] M. Conti, R. Kotter, and G. Putrus, "Energy Efficiency in Electric and Plug-in Hybrid Electric Vehicles and Its Impact on Total Cost of Ownership", in *Electric Vehicle Business Models: Global Perspectives*, D. Beeton and G. Meyer, Eds. Springer International Publishing, 2015, pp. 147–165, ISBN: 978-3-319-12244-1. DOI: 10.1007/978-3-319-12244-1_9. [Online]. Available: https://doi.org/10.1007/978-3-319-12244-1_9.

[4] J. Froehlich and J. Krumm, "Route prediction from trip observations", Tech. Rep., 2008.

[5] A. Fotouhi, M. Montazeri, and M. Jannatipour, "Vehicle's velocity time series prediction using neural network", *International Journal of Automotive ...*, vol. 1, no. 1, pp. 21–28, 2011. [Online]. Available: http://www.iust.ac.ir/ijae/browse.php?a%7B%5C_%7Did=8.

[6] J. Park, D. Li, Y. L. Murphey, J. Kristinsson, R. McGee, M. Kuang, and T. Phillips, "Real time vehicle speed prediction using a Neural Network Traffic Model", *The 2011 International Joint Conference on Neural Networks*, pp. 2991–2996, 2011, ISSN: 2161-4393. DOI: 10.1109/IJCNN.2011.6033614.

[7] S.-i. Jeon, S.-t. Jo, Y.-i. Park, and J.-m. Lee, "Multi-Mode Driving Control of a Parallel Hybrid Electric Vehicle Using Driving Pattern Recognition", *Journal of Dynamic Systems, Measurement, and Control*, vol. 124, no. 1, p. 141, 2002, ISSN: 00220434. DOI: 10.1115/1.1434264. [Online]. Available: http://dynamicsystems.asmedigitalcollection.asme.org/article.aspx?articleid=1409389.

[8] A. Konar and D. Bhattacharya, *Time-Series Prediction and Applications.* Springer, 2017, vol. 127, p. 255, ISBN: 978-3-319-54597-4. DOI: 10.1007/978-3-319-54597-4. [Online]. Available: http://link.springer.com/10.1007/978-3-319-54597-4.

[9] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. John Wiley & Sons, Inc., 2016, p. 1257, ISBN: 9781118674918.

[10] R. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts: Melbourne, Australia, 2013, Accessed on 2018-01-30. [Online]. Available: `http://otexts.org/fpp/`.

[11] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959, ISSN: 0018-8646. DOI: `10.1147/rd.33.0210`.

[12] D. Barber, *Bayesian Reasoning and Machine Learning.* New York, NY, USA: Cambridge University Press, 2012, p. 682, ISBN: 9780521518147. [Online]. Available: `http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/181115.pdf`.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Advances In Neural Information Processing Systems*, pp. 1–9, 2012, ISSN: 10495258. DOI: `http://dx.doi.org/10.1016/j.protcy.2014.09.007`. arXiv: `1102.0183`.

[14] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Idea Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943, ISSN: 0007-4985. DOI: `10.1007/BF02478259`. arXiv: `arXiv:1011.1669v3`. [Online]. Available: `http://www.cse.chalmers.se/%7B~%7Dcoquand/AUTOMATA/mcp.pdf`.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*, 9. New York, NY, USA: Springer, 2006, vol. 53, pp. 1689–1699, ISBN: 978-0-387-31073-2. DOI: `10.1117/1.2819119`. arXiv: `arXiv:1011.1669v3`.

[16] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient BackProp", in *Neural Networks: Tricks of the Trade*, Springer, 2003, pp. 9–48, ISBN: 3-540-65311-2.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, ISSN: 00280836. DOI: `10.1038/323533a0`. arXiv: `arXiv:1011.1669v3`.

[18] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network", *Proceedings Of The International Joint Conference On Neural Networks*, vol. 1, pp. 593–605, 1989, ISSN: 08936080. DOI: `10.1109/IJCNN.1989.118638`. [Online]. Available: `http://ieeexplore.ieee.org/xpl/freeabs%7B%5C_%7Dall.jsp?%7B%5C&%7Darnumber=118638`.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning.* MIT press Cambridge, 2016, vol. 1.

[20] C. M. Bishop, "Regularization and Complexity Control in Feed-forward Networks", *Proceedings International Conference on Artificial Neural Networks ICANN'95*, vol. 1, pp. 141–148, 1995. [Online]. Available: `http://neural-server.aston.ac.uk/`.

[21] P. J. Werbos, "Backpropagation Through Time: What It Does and How to Do It", *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990, ISSN: 15582256. DOI: `10.1109/5.58337`.

[22] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. [Online]. Available: `http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf`.

[23] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks", 2012, ISBN: 08997667 (ISSN). DOI: 10.1109/72.279181. arXiv: 1211.5063. [Online]. Available: http://arxiv.org/abs/1211.5063.

[24] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural computation*, pp. 1735–1780, 1997, ISSN: 0899-7667. DOI: 10.1.1.56.7752. arXiv: 1206.2944.

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", pp. 1–9, 2014. arXiv: 1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555.

[26] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", 2014-06. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[27] P. Johannesson and E. Perjons, *An Introduction to Design Science.* 2014, p. 197, ISBN: 978-3-319-10631-1. DOI: 10.1007/978-3-319-10632-8. [Online]. Available: http://link.springer.com/10.1007/978-3-319-10632-8.

[28] *Volvo C30 Electric*, Visited: 2018-02-07, 2011. [Online]. Available: https://www.media.volvocars.com/us/en-us/media/pressreleases/37345.

[29] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", pp. 1–15, 2014, ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[30] *Adam optimizer in keras*, Visited: 2018-05-28. [Online]. Available: https://keras.io/optimizers/#adam.

[31] D. C. Montgomery, *Design and Analysis of Experiments*, 8th ed. John Wiley & Sons, Inc., 2012, pp. 183–448, ISBN: 978-1-118-14692-7.

[32] V. Kovalev, A. Kalinovsky, and S. Kovalev, *Deep Learning with Theano, Torch, Caffe, TensorFlow, and Deeplearning4J: Which One Is the Best in Speed and Accuracy?* 2016-10.

[33] *TensorFlow*, Visited: 2018-02-08. [Online]. Available: https://www.tensorflow.org/.

[34] *What is a tensor?*, Visited: 2018-02-08. [Online]. Available: https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php.

[35] *Keras*, Visited: 2018-02-08. [Online]. Available: https://keras.io/.

[36] T. S. Community, *Scipy.signal.filtfilt*, Visited: 2018-02-07. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html.

[37] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks", 2014-09. arXiv: 1409.3215. [Online]. Available: http://arxiv.org/abs/1409.3215.

[38] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning", 2017-05. arXiv: 1705.03122. [Online]. Available: http://arxiv.org/abs/1705.03122.

A

Appendix 1

Derivation of Equation 2.8

$$\begin{aligned}
\delta_j^L &= \frac{\partial L}{\partial z_j^L} \\
&= \sum_{k=0}^{n} \frac{\partial L}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \\
&\overset{1}{=} \frac{\partial L}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\
&= \frac{\partial L}{\partial a_j^L} \sigma' \left(z_j^L \right)
\end{aligned} \tag{A.1}$$

Derivation of Equation 2.9

$$\begin{aligned}
\delta_j^l &= \frac{\partial L}{\partial z_j^l} \\
&= \sum_{k=0}^{n} \frac{\partial L}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\
&= \sum_{k=0}^{n} \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \\
&= \sum_{k=0}^{n} w_{kj}^{l+1} \delta_k^{l+1} \sigma' \left(z_j^l \right)
\end{aligned} \tag{A.2}$$

[1]The activation a_k^L of neuron k is nonzero only when $k = j$ for its corresponding weighted input z_j^L of neuron j. Hence the partial derivative $\frac{\partial a_k^L}{\partial z_j^L}$ is zero when $k \neq j$.

B

Appendix 2

SimpleRNN arguments

Argument	Value
units	60
activation	tanh
use_bias	True
kernel_initializer	glorot_uniform
recurrent_initializer	orthogonal
bias_initializer	zeros
kernel_regularizer	None
recurrent_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
recurrent_constraint	None
bias_constraint	None
dropout	0.0
recurrent_dropout	0.0
return_sequences	True
return_state	False
go_backwards	False
stateful	True
unroll	False

Table B.1: The parameters used for the `SimpleRNN` cell unit.

LSTM arguments

Argument	Value
units	60
activation	tanh
recurrent_activation	hard_sigmoid
use_bias	True
kernel_initializer	glorot_uniform
recurrent_initializer	orthogonal
bias_initializer	zeros
unit_forget_bias	True
kernel_regularizer	None
recurrent_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
recurrent_constraint	None
bias_constraint	None
dropout	0.0
recurrent_dropout	0.0
implementation	1
return_sequences	True
return_state	False
go_backwards	False
stateful	True
unroll	False

Table B.2: The parameters used for the `LSTM` cell unit.

GRU arguments

Argument	Value
units	60
activation	tanh
recurrent_activation	hard_sigmoid
use_bias	True
kernel_initializer	glorot_uniform
recurrent_initializer	orthogonal
bias_initializer	zeros
kernel_regularizer	None
recurrent_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
recurrent_constraint	None
bias_constraint	None
dropout	0.0
recurrent_dropout	0.0
implementation	1
return_sequences	True
return_state	False
go_backwards	False
stateful	True
unroll	False
reset_after	False

Table B.3: The parameters used for the `GRU` cell unit.

Adam optimizer

Argument	Value
lr	0.001
beta_1	0.9
beta_2	0.999
epsilon	None
decay	0.0
amsgrad	False

Table B.4: The parameters used for the Adam optimizer.

C

Appendix 3

Result table for evaluation of Artefact 1

Conf	**RMSE 1**	**RMSE 2**	**RMSE 3**	**RMSE 4**	**RMSE 5**	**Avg**
A	944	1016	1008	1089	837	979
B	935	992	1919	1949	2012	1561
C	691	845	943	971	810	852
D	748	837	749	852	980	833
E	991	793	2085	856	2047	1354
F	903	913	878	743	808	849

Table C.1: The result from evaluating the configurations in Table 3.1 in accordance to their summed root mean squared error. The last column denotes the average root mean squared error over the five runs and is the value used to ultimately decide which configuration is the best.

Result table for evaluation of Artefact 2.1

Conf	ND	DoD	ES	D	A	S	R1	R2	R3	R4	R5	Avg	Time
0	-	-	-	-	-	-	700	702	697	672	544	663	01:35
1	-	-	-	-	-	+	735	678	598	620	745	675	01:34
2	-	-	-	-	+	-	855	736	782	828	814	803	01:57
3	-	-	-	+	-	-	902	833	831	937	834	867	01:34
4	-	-	+	-	-	-	785	876	848	819	822	830	00:31
5	-	+	-	-	-	-	774	712	736	640	722	717	01:34
6	-	-	-	-	+	+	791	729	831	736	693	756	02:12
7	-	-	-	+	-	+	826	712	894	719	797	790	01:33
8	-	-	+	-	-	+	749	662	891	821	781	781	00:23
9	-	+	-	-	-	+	711	598	783	694	614	680	01:35
10	-	-	-	+	+	-	850	1184	1030	908	1025	999	02:16
11	-	-	+	-	+	-	834	730	828	996	784	835	00:40
12	-	+	-	-	+	-	691	755	823	845	789	780	02:13
13	-	-	+	+	-	-	822	917	912	874	892	883	01:02
14	-	+	-	+	-	-	794	846	777	954	838	842	01:34
15	-	+	+	-	-	-	868	804	787	873	813	829	00:25
16	-	+	+	+	+	+	1507	1156	1014	1258	1010	1189	01:01
17	+	-	-	-	-	-	2562	1058	836	1013	966	1287	06:06
18	+	-	-	-	-	+	837	997	1972	1087	1096	1198	06:13
19	+	-	-	-	+	-	1322	1164	941	1946	1181	1311	08:38
20	+	-	-	+	-	-	1010	870	893	982	827	916	06:01
21	+	-	+	-	-	-	1610	1418	1448	1658	1107	1448	00:44
22	+	+	-	-	-	-	1644	1302	1858	1467	827	1420	06:06
23	+	-	-	-	+	+	1229	1022	1159	2168	1011	1318	08:34
24	+	-	-	+	-	+	733	910	1041	802	814	860	06:00
25	+	-	+	-	-	+	1774	1607	1377	1113	1142	1403	00:43
26	+	+	-	-	-	+	1209	935	1281	1002	1221	1130	06:02
27	+	-	-	+	+	-	868	1053	1064	1116	1106	1042	08:32
28	+	-	+	-	+	-	1549	1369	1309	1204	1582	1403	00:55
29	+	+	-	-	+	-	1282	1328	1259	964	1142	1195	08:16
30	+	-	+	+	-	-	914	926	972	938	922	935	01:05
31	+	+	-	+	-	-	1014	944	891	907	1045	960	06:04
32	+	+	+	-	-	-	1423	1395	1445	1397	1428	1418	00:44
33	+	+	+	+	+	+	1622	1511	1798	1313	1611	1571	01:50

Table C.2: The verbose result from evaluating the candidate configurations for Artefact 2.1. The Avg-column denotes the average root mean squared error over the five runs and is the value used to ultimately decide which configuration is the best.

Result table for evaluation of Artefact 2.2

Conf	ND	DoD	ES	D	A	S	R1	R2	R3	Avg	Time
0	-	-	-	-	-	-	600	578	542	573	04:28
1	-	-	-	-	-	+	526	535	509	524	04:36
2	-	-	-	-	+	-	577	522	519	539	04:46
3	-	-	-	+	-	-	635	571	633	613	04:39
4	-	-	+	-	-	-	621	568	634	608	01:12
5	-	+	-	-	-	-	585	602	572	586	04:27
6	-	-	-	-	+	+	565	497	498	520	04:45
7	-	-	-	+	-	+	758	738	689	728	04:08
8	-	-	+	-	-	+	709	648	679	678	00:49
9	-	+	-	-	-	+	525	524	553	534	04:44
10	-	-	-	+	+	-	931	728	859	839	04:37
11	-	-	+	-	+	-	686	658	635	660	00:46
12	-	+	-	-	+	-	548	515	585	549	04:39
13	-	-	+	+	-	-	778	611	582	657	03:18
14	-	+	-	+	-	-	610	654	820	695	04:53
15	-	+	+	-	-	-	584	642	613	613	01:00
16	-	+	+	+	+	+	991	1201	1131	1108	01:57
17	+	-	-	-	-	-	648	682	703	678	20:01
18	+	-	-	-	-	+	649	582	703	645	18:34
19	+	-	-	-	+	-	799	1083	767	883	20:32
20	+	-	-	+	-	-	608	682	852	714	18:53
21	+	-	+	-	-	-	886	1241	1291	1140	02:26
22	+	+	-	-	-	-	736	710	757	734	19:05
23	+	-	-	-	+	+	649	1522	747	972	18:03
24	+	-	-	+	-	+	1035	1041	1258	1111	18:39
25	+	-	+	-	-	+	1164	762	1140	1022	03:21
26	+	+	-	-	-	+	1029	846	633	836	18:43
27	+	-	-	+	+	-	904	2752	953	1536	19:13
28	+	-	+	-	+	-	836	1121	672	876	03:38
29	+	+	-	-	+	-	612	753	579	648	18:41
30	+	-	+	+	-	-	773	679	738	730	03:28
31	+	+	-	+	-	-	1689	723	643	1019	20:04
32	+	+	+	-	-	-	1340	1221	1309	1290	02:14
33	+	+	+	+	+	+	1607	1661	1590	1619	04:45

Table C.3: The verbose result from evaluating the candidate configurations for Artefact 2.2. The Avg-column denotes the average root mean squared error over the five runs and is the value used to ultimately decide which configuration is the best.