ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

AUTOMOTIVE PER UNA MOBILITÀ INTELLIGENTE

Ciclo 36

Settore Concorsuale: 09/G1 - AUTOMATICA

Settore Scientifico Disciplinare: ING-INF/04 - AUTOMATICA

DATA-DRIVEN MASS ESTIMATION OF HEAVY-DUTY VEHICLES

Presentata da: Abdurrahman Isbitirici

Coordinatore Dottorato

Davide Moro

Supervisore

Paolo Falcone

Co-supervisore

Paolo Pavan

Esame finale anno 2024

FOREWORD

This Ph.D. is supported by the Ministry of National Education of the Republic of Türkiye. I would like to thank my supervisor Assoc. Prof. Paolo Falcone and Prof. Laura Giarré for their support during the Ph.D. I would like to thank IPG Automotive for allowing me to use the TruckMaker software. I also would like to thank Prof. Roberto Zanasi, Mrs. Christina Murari and the University of Modena and Reggio Emilia for providing computational resources. Lastly, I would like to thank Mehmet Emin Çetin, İsmail Bilgen and Hasan Kıvrak for all discussions during my Ph.D.

September 2024

Abdurrahman İŞBİTİRİCİ
(Ph.D. Student)

TABLE OF CONTENTS

Page

ABBREVIATIONS

ACC : Adaptive Cruise Control
ADAM : Adaptive Moment Estimation
ADAS : Advanced Driver Assistance System
ANN : Artificial Neural Network
ARX : Auto Regressive Exogenous
AV : Autonomous Vehicle
BEV : Battery Electric Vehicle
Bi : Bi-directional
CAN : Control Area Network
CNN : Convolution Neural Network
CR : Cyclic Resetting
CRF : Constant-Rate Forgetting
DNN : Deep Neural Network
ECU : Engine Control Unit
EKF : Extended Kalman Filter
ER : Exponential Resetting
EV : Electric Vehicle
FCEV : Fuel Cell Electric Vehicle
FFNN : Feed Forward Neural Network
GBP : Gaussian Belief Propagation
GPS : Global Positioning System
GPU : Graphics Processing Unit
GRU : Gated Recurrent Unit
HDV : Heavy-duty Vehicle
ICE : Internal Combustion Engine
KF : Kalman Filter
kNF : k-Nearest Factors
LKS : Lane Keeping System
LSTM : Long-Short Term Memory
MAE : Mean Absolute Error
ML : Machine Learning
MLP : MultiLayer Perceptron
MSE : Mean Square Error
NN : Neural Network
PE : Persistency of Excitation
ReLU : Rectified Linear Unit
RLS : Recursive Least Squares
RMSE : Root Mean Square Error
RMSprop : Root Mean Square Propagation
RNN : Recurrent Neural Network

SGD : Stochastic Gradient Descent
TCN : Temporal Convolutional Network
VDF : Variable-Direction Forgetting
VRDF : Variable-Rate Direction Forgetting
VRF : Variable-Rate Forgetting
WLTC : Worldwide harmonized Light vehicles Test Cycle

SYMBOLS

A_f	: Frontal area of the truck
b	: Bias
c	: Cell state
c_d	: Aerodynamic drag coefficient
f	: Forget gate
F_{aero}	: Aerodynamic drag force
F_{brake}	: Braking force
F_{grade}	: Force due to road gradient
F_{roll}	: Rolling resistance force
$F_{traction}$	: Traction force
g	: Gravitational acceleration
g_{final}	: Final drive ratio
g_r	: Gear ratio
h	: Hidden state
i	: Input gate
J	: Cost function
K	: Estimator gain
l	: The look-ahead distance
m	: Mass of the truck
N	: Number of samples
N_i	: Network index
o	: Output gate
P	: Covariance matrix
R_i	: Range index
r_{wheel}	: Radius of the wheel
T_{engine}	: Engine torque
T_j	: Test data index
v	: Truck speed
v_{wind}	: Wind speed
w	: Weight
W_{*x}	: Input Weights
W_{*h}	: Recurrent Weights
u	: Input of a system
x	: Input of a neural network
y	: Output of a system

ω : Engine speed
ρ : Air density
γ : Road gradient
μ : Rolling resistance coefficient
ϕ : Regressor
ε : Noise
λ : Forgetting factor
σ : Activation function

LIST OF TABLES

Page

LIST OF FIGURES

Page

DATA-DRIVEN MASS ESTIMATION OF HEAVY-DUTY VEHICLES

ABSTRACT

This study investigates mixed model- and learning-based approaches to the mass estimation problem in heavy-duty vehicles. The mass of a vehicle plays a pivotal role in various systems, including powertrain control and Vehicle Dynamics Control (VDC) systems. Effective mass estimation allows for more precise adjustments to engine power, braking systems, and suspension settings, leading to improved vehicle handling, fuel efficiency, and overall safety. Although direct mass measurement using sensors is a viable option, the substantial costs and complexities associated with sensor maintenance, integration, and calibration drive the search for alternative solutions.

The objective of this thesis is to introduce a methodology to mass estimation problem in heavy-duty vehicles, which harnesses the advantages of model-based and learning-based estimation methods. The proposed methodology builds upon the integration of Long-Short Term Memory (LSTM), which is a type of recurrent neural network, that supervises a Recursive Least Squares (RLS) vehicle mass estimator. The RLS estimator relies on a longitudinal vehicle dynamical model. The supervisory LSTM network is offline trained to recognize when the vehicle is operated such that the RLS estimator leads to an estimate with the desired accuracy while online enables the mass estimate update by the RLS estimator based on signals that include vehicle speed, longitudinal acceleration, engine torque, and engine speed.

The LSTM network is trained and tested in this thesis work using datasets artificially generated by a widely used simulation environment called TruckMaker. The simulation results demonstrate that the proposed methodology effectively forecasts the reliability of the RLS mass estimator, showcasing the potential of LSTM networks in enhancing the accuracy and trustworthiness of mass estimation of heavy-duty vehicles.

This thesis also presents a benchmark learning-based approach to mass estimation in heavy vehicles, that uses an LSTM network. In this case, a two-layer LSTM network is designed that utilizes vehicle speed, longitudinal acceleration, engine speed and engine torque to estimate the vehicle mass.

While learning-based methods, in principle, adapt to diverse conditions and capture complex system behaviors, model-based approaches, despite challenges in the data collection process, provide physical interpretability and ease of use for, e.g., control design. The discussion underscores the trade-offs between these approaches, recognizing their distinct strengths and limitations for mass estimation of heavy vehicles.

The limitations of the model-based strategy of this study are as follows: the clutch is always assumed to be engaged and only the highest gear is used, so data is collected at specified vehicle speed intervals. Braking scenarios are excluded as they require significantly larger amounts of data sets. Only straight-driving and flat-road cases are

considered. The wind effect is not included. Only straight driving limitation is used for the learning-based strategy although initially the same limitation of model-based strategy is used.

Keywords: mass estimation, long-short term memory, recursive least squares

1. INTRODUCTION

1.1 Motivation

There are ongoing studies such as [1] and [2] to reduce greenhouse gases and atmospheric contamination resulting from the emissions of heavy-duty (HDV) vehicles because road transport is responsible for approximately 20% of the total greenhouse gas emissions in the EU [3]. Although demand for battery electric vehicles (BEV) and fuel cell vehicles (FCEV) is increasing in the automotive sector due to their reduced emissions, trucks with internal combustion engines (ICE) are still popular because of existing infrastructure, affordability, range of driving, and the availability of spare parts and maintenance services. Also, faster refueling times and higher payload capabilities make diesel-engine HDVs preferable. However, they can only use approximately 23% of the fuel energy according to the study in [1]. Therefore, increasing energy efficiency is essential both for fuel economy and for reducing emissions.

The accurate knowledge of a heavy vehicle's mass is crucial to improve fuel efficiency and emission reduction. For example, automatic gear shifting algorithms rely on accurate vehicle mass data [4] to optimize the powertrain operation. This information further serves to offer truck drivers tailored suggestions, enabling them to adjust their driving styles based on the load of the truck and road profile, thereby enhancing the overall operational efficiency. Moreover, accurate mass estimation proves instrumental in enhancing vehicle safety, such as estimating braking distances depending on the current mass [5].

Mass estimation is also essential for buses since it contributes to several aspects, including the estimation of the passenger numbers. This estimation, in turn, serves resource planning, bus schedules, and adjustments to air conditioning systems relying on the weight of the bus, predicting the total passenger count [6]. Additionally, mass information has an essential role in determining the order of vehicles within a platoon, with considerations for factors such as braking distance. For instance, a truck with

more load may lead a fleet due to its larger braking distance, minimizing the risk of rear-end crashes while tailgating lighter trucks closely. On the contrary, heavier trucks may be strategically positioned at the rearmost position of the platoon when ascending uphill to avoid impeding the speed of other trucks. Beyond these considerations, mass information, in conjunction with inertia, holds promise for enhancing active safety systems and preventing rollovers [7].

1.2 Literature Survey

The mass of heavy-duty trucks and trailers is crucial for a variety of applications, such as load management, fuel efficiency optimization, and safety enhancement. The mass can be measured directly using sensors or estimated using various algorithmic approaches. This section provides an overview of the existing technologies and methodologies for vehicle mass measurement and estimation, highlighting the strengths and limitations of each approach.

1.2.1 Sensors

It is worth noting the existence of reliable load sensors [8] with a lifespan matching vehicle lifetime, affordability, and straightforward calibration procedures. Additionally, some weighing sensors are designed depending on an inclinometer to be used in uneven areas [9]. On the other hand, the availability of such sensors in truck and trailer combinations depends on the manufacturer of trailers. Consequently, load sensors may not always be available on every trailer towed by a truck. In cases where these sensors are absent, opting for the estimation of vehicle mass rather than relying on sensor measurements is definitely preferable. Even in scenarios where a load sensor is present, employing a mass estimation algorithm proves beneficial for detecting potential sensor failures, malfunctions, or the need for recalibration.

1.2.2 State-of-the-Art Mass Measurement and Estimation Technologies

One widely used algorithm to estimate the mass of a vehicle is Recursive Least Squares (RLS). In [10], an RLS mass estimation algorithm updates the mass estimate based on specific rules, including criteria such as longitudinal acceleration exceeding 0.1 m/s^2, engine torque between 40% and 90% of the maximum engine torque, lateral

acceleration not exceeding 5 m/s^2, and the gradient of the engine torque between -1000 and 1000 Nm/s. The estimate is not updated when the brake pedal is used and predefined gears are not selected. Although these kinds of criteria are specified based on measurements to estimate mass more accurately, defining such rules is time-consuming and might be impractical. In another work [11], RLS is preferred to predict both mass and road grade if road grade cannot be measured. The estimation process begins with estimating the road grade and ends with mass estimation by applying a low-pass filter, which is a second-order Butterworth filter, to the road slope, engine torque, vehicle speed, and acceleration. Not only mass estimation but also drag coefficient and rolling resistance are predicted based on longitudinal dynamics. In [12], a supervisory algorithm is used to find maneuvers with inertial dynamics. Then, an RLS mass estimator is used. Paper [13] introduces an RLS algorithm that has two separate forgetting factors for mass and road grade estimation that is proven to be better than using a single forgetting factor. The study in [14] demonstrates an estimation method where the mass and road grade are initially computed through adaptive least squares by assuming both are constants. Paper [15] presents an integrated mass estimator relying on an RLS and parameter adaptation using both roll and longitudinal vehicle dynamics.

Learning-based methods provide an alternative for heavy vehicle mass estimation. In [16], a neural network with 12 neurons in double hidden layers is proposed to estimate both road grade and mass. Rectified linear unit (ReLU) is used as an activation function. The training dataset comprises six different road scenarios, each with 10 km of data spanning eight various masses. Input signals for training the network include vehicle velocity, longitudinal acceleration, engine torque, their previous time step values, as well as previous time step mass and road grade estimations with uniform noise. [17] prefers a Deep Neural Network (DNN) with fully connected hidden layers to estimate trailer mass, employing the leaky ReLU activation function. The first hidden layer consists of 140 neurons. The neuron number is reduced gradually by 10 in the next layers up to the output layer. There is only one output which is the mass prediction. This DNN employs 1051 neurons in total. Additionally, the authors conducted a comparative analysis between Recurrent Neural Network (RNN) and DNN. Revealing errors with respect to various longitudinal velocities are around 9%

and 8.5% respectively. Paper [18] proposed a data-driven mass estimator relying on the bi-directional gated recurrent unit (Bi-GRU), claiming a mean absolute percentage error of less than 1%.

In [19], a Random Forest method is applied to classify the weight of e-scooter users. The choice of classification over regression is attributed to the limited availability of the dataset. In [20], fuzzy logic is used to integrate an RLS method with a machine-learning technique. The estimations obtained from both RLS and machine learning techniques are weighted according to criteria such as longitudinal acceleration, acceleration rate, and vehicle speed. Another weight is given to the prior mass estimation, serving as a fallback when both machine learning and RLS methods are deemed unreliable. Therefore, the fuzzy logic incorporates three weights based on defined rules by the authors, so that the weight assigned for dynamic model-based mass estimation is larger than other weights at higher speeds, as the algorithm converges more effectively compared to lower speeds. The machine learning component is designed with 1024 and 256 neurons in each hidden layer.

In [21], the estimation of heavy-duty vehicle mass employs Gaussian Belief Propagation with k-nearest factors (kNF-GBP). This methodology is designed to mitigate the impact of noisy data without the need for additional sensors. Various measurements, including vehicle velocity, acceleration, fuel consumption, engine load, speed, reference torque, and GPS location, are utilized. This approach not only provides the mean but also the variance of the mass estimate, offering a more comprehensive understanding of the uncertainty of the estimated mass.

1.2.3 Patents

The automotive industry has also contributed to advance methodologies for estimating vehicle mass. In [22], an RLS mass estimator is developed, leveraging longitudinal dynamics and a comprehensive set of measurements including wheel speeds, engine torque, gearbox data, brake pressures, and longitudinal acceleration. By checking whether the seat belt is fastened or not and monitoring the fuel level gauge, mass is predicted with a condition to reset the RLS algorithm upon the opening of any vehicle door. In [23], the estimation of the vehicle mass is performed using an RLS algorithm, which detects changes in the vertical position of the vehicle's

center of gravity. This variation is measured using lidar technology. Patent [24] proposed a three-step approach, involving an initial vehicle mass estimation stored in volatile memory upon engine initiation, subsequent estimations based on longitudinal dynamics during straight-line motion, and refined mass calculation during acceleration and deceleration events after the vehicle reaches the set speed. This sequential process aims at mitigating the errors arising from acceleration and speed measurement noise. Patent [25] presented a mass estimation method that depends on measurements of inertial sensors capturing vertical acceleration. The truck without any load serves as a reference for estimating the load using an RLS approach, particularly when both longitudinal and lateral accelerations are minimal. Last of all, patent [26] introduced a specialized mass estimation technique designed for vehicle platoons. This method involves assessing the mass difference between adjacent vehicles by considering longitudinal dynamics and enabling information exchange among trucks. Shared information includes torque, brake usage, and gaps between vehicles in the platoon.

1.3 Objectives

This PhD thesis work investigates the utilization of long short-term memory (LSTM) networks and LSTM networks combined with RLS techniques to estimate the mass of heavy vehicles with a target accuracy comparable to commercial load sensors. While such an objective can be achieved by only using LSTM networks at the cost of large training datasets, we also show that an LSTM-supervised RLS can lead to similar accuracy with smaller training datasets.

1.4 Contributions and Limitations

There are available load sensors and mass estimation approaches in the literature as mentioned in Section 1.2. However, the availability of load sensors in truck and trailer combinations may not always be possible. For instance, if the load sensors do not exist in one of the trailers in a multi-trailer combination then mass estimation is still necessary although other trailers have load sensors. Even if the sensors are available, their accuracy is affected during acceleration and braking since they are based on suspension dynamics. Despite the existence of model-based and learning-based

approaches in the literature, there is a need for their further development. RLS has already been used for mass estimation, however using only RLS is not suitable because the conditions to use RLS were not obtained from the data but defined manually and impractical for different truck models. Although FFNN and DNN-based mass estimation algorithms exist in the literature, they are not as efficient as RNNs while handling temporal dynamics since they do not have memory. Therefore, we propose LSTM-based and LSTM-supervised RLS-based mass estimation approaches. The main contributions of this thesis can be summarized below:

- A learning-based algorithm is employed as a supervisory mechanism for the mixed model-based mass estimation method in Chapter 4 in order to classify whether the mixed model-based mass estimator is reliable or not at the current time step.
- Online reliability is analyzed by implementing a trained network in Chapter 4 into Simulink, which is one of the most widely used model-based automotive software development tools.
- A virtual mass sensor is designed for heavy-duty vehicles based on the learning-based methods in Chapter 5. Mass estimation with an LSTM-based network is a more generic approach because only the features of the network have to be determined, LSTM uses the same weights and biases across different time intervals of the same signal and handles the temporal dynamics internally as opposed to FFNN. Therefore, the number of parameters to be tuned is decreased.
- Experimental validation is done by WLTC data where a fixed amount of training dataset is used for each network of various load intervals in Chapter 5.
- Another public bus dataset, which encompasses a range of wind speed, uphill and downhill scenarios, is similarly subjected to the learning-based technique. The effect of sequence length on the mass estimation accuracy is observed. Also, the comparison between LSTM and GRU methods is done in Chapter 5.

The limitations of the thesis are listed below.

- While data-driven approaches are useful for precisely estimating mass through the acquisition of real-world data, it is important to emphasize that these approaches

mostly depend on the availability and quality of data. For training, a large variety of high-quality data sets are required. Creating random data by using longitudinal dynamics is not realistic. On the other hand, obtaining real truck data with a real truck model is not usually possible. Therefore, it was decided to use high-fidelity simulation data in this thesis.

- Although data is normalized, the data-driven approach has to be retrained when a different powertrain or truck is used since the dynamics do not remain the same.
- The accuracy of both model-based and data-driven methods decreases in the presence of unmeasurable disturbances, such as various weather conditions whose measurements are not included for the data-driven method and whose model and parameters are not included for the model-based method. Generalizations cannot be made for unseen conditions.
- Another problem with data-driven methodologies is that they are less interpretable due to challenges in comprehending the reasoning behind the estimation.

1.5 Thesis Layout

The structure of the thesis is as follows. Chapter 2 gives information about the background of neural networks. Chapter 3 gives information about the background of recursive least squares and vehicle models. Chapter 4 describes mixed learning- and model-based mass estimation of heavy-duty vehicles. Chapter 5 describes the learning-based mass estimation of heavy-duty vehicles. Chapter 6 discusses the use of LSTM as a supervisor for mixed learning- and model-based mass estimation. Moreover, using LSTM-based mass estimation is compared with other methods and with using mass sensors. Chapter 7 serves as the conclusion of this study, summarizing the key findings and proposing directions for future research.

2. ARTIFICIAL NEURAL NETWORKS AND METHODOLOGY

Consider a system:

$$y = f(u), \tag{2.1}$$

where the system input vector u is mapped to the system output vector y. In section 2.1, how an artificial neural network (ANN) is used to formulate a mathematical model:

$$\hat{y} = \hat{f}(x), \tag{2.2}$$

of the system (2.1) by using samples of u and y is discussed. The input vector x may be equal to u or x may include only some signals in u and other additional signals based on the problem and available measurements.

2.1 Neural Networks

ANNs are mathematical models built upon nonlinear functions mapping a vector x of inputs into an output $\hat{y}$. In the most basic form of an ANN, there is a single unit called a *neuron* that serves as both the input and output node. This neuron is responsible for mapping input signals to output signals using an *activation function* as illustrated in Figure 2.1. The output is calculated as

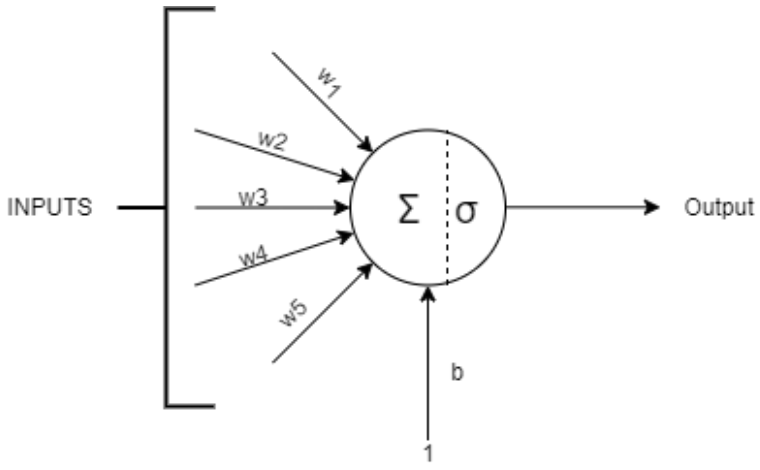

Figure 2.1 : Artificial Neuron

$$\hat{y} = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.3}$$

where the *features* are elements of the input vector $x = [x_1, \ldots, x_n]$, so the total number of network inputs is represented by n whereas the model output and activation function are denoted by $\hat{y}$ and σ respectively. The bias is denoted by b and the weight of the i_{th} input is denoted by w_i.

If there are multiple neurons in a network, then each output can be obtained by rewriting (2.3) as

$$\hat{y}_j = \sigma\left(\sum_{i=1}^{n} w_{ij} x_i + b_j\right), \tag{2.4}$$

where the corresponding neuron is denoted by j.

2.1.1 Feed Forward Neural Networks

A feedforward neural network (FFNN) or Multilayer Perceptron (MLP) is a network that uses linked layers or neurons to map inputs to outputs, as seen in Figure 2.2. An FFNN is referred to as a Deep Neural Network (DNN) when it has two or more hidden layers. Hidden layers are the layers between the input layer and the output layer. If there are hidden layers $l = 1, 2, ..., L$ in a network, then (2.4) can be rewritten as

$$h_{j_l}^{[l]} = \sigma_l\left(\sum_{i=1}^{n_{l-1}} w_{ij_l}^{[l]} h_i^{[l-1]} + b_{j_l}^{[l]}\right), \tag{2.5}$$

where the corresponding neuron, the number of neurons in a layer, the input and output of the corresponding layer, the corresponding activation function of the layer and the corresponding bias of the neuron are denoted by j_l, n_l, $h^{[l-1]}$, $h^{[l]}$, σ_l and $b_{j_l}^{[l]}$, respectively. w_{ij_l} represents the corresponding weight where j_l = 1, 2, ... n_l. Therefore, the input of the neural networks x is denoted as $h^{[0]}$ and the output of the neural networks $\hat{y}$ is denoted as $h^{[L+1]}$. The weights and biases for the corresponding layer are denoted as $\mathbf{W^{[l]}} = [w_{11}^{[l]}, \ldots, w_{1n_l}^{[l]}, w_{21}^{[l]}, \ldots, w_{2n_l}^{[l]}, w_{n_{l-1}1}^{[l]}, \ldots, w_{n_{l-1}n_l}^{[l]}]$ and $\mathbf{b^{[l]}}$ $= [b_1^{[l]}, \ldots, b_{n_l}^{[l]}]$. The network parameters can be lumped in $\mathbf{W} = [\mathbf{W^{[1]}}, \mathbf{W^{[2]}}, \mathbf{W^{[L+1]}}]$ and $\mathbf{b} = [\mathbf{b^{[1]}}, \mathbf{b^{[2]}}, \mathbf{b^{[L+1]}}]$.

Different activation functions σ could be used in an ANN. Activation functions, which are usually used in NNs, are written in Table 2.1 [27]. Some of them, including the hyperbolic tangent (tanh), the sigmoid, the rectified linear unit (ReLU), and the leaky

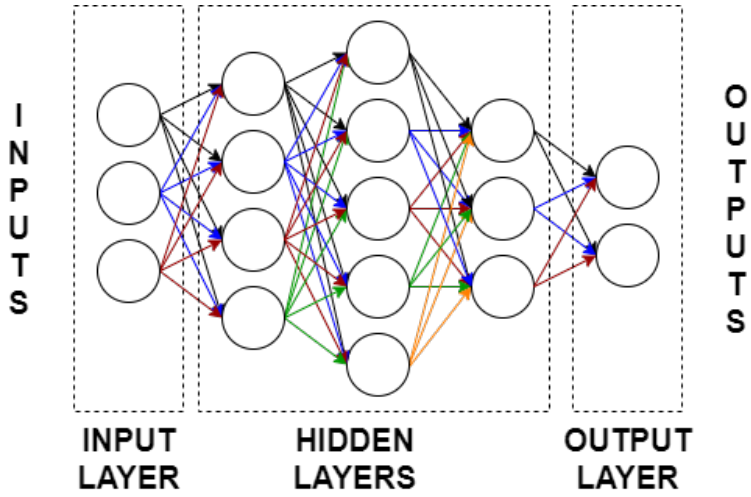

Figure 2.2 : Deep Neural Network

Table 2.1 : Activation Functions in Neural Networks

Activation Function	Min Value	Max Value	Equation
Sigmoid	0	1	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	-1	1	$f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
ReLU	0	∞	$f(x) = \max(0,x)$
Leaky ReLU	$-\infty$	∞	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$
Softmax	0	1	$f(x)_c = \frac{e^{x_c}}{\sum_{j=1}^{C} e^{x_j}}$

ReLU, are shown in Figure 2.3. The selection of activation functions relies on several factors, including the effectiveness of the training methods used to adjust the network's weights and biases based on the input and output data. For instance, the Rectified Linear Unit (ReLU) is not commonly used as an activation function in LSTM because it can suffer from the vanishing gradient problem, which is explained in section 2.1.3.1

After deciding the number of layers, neurons, and activation functions for a NN, the weights and biases of the NN are obtained as the result of the following training problem. Such a problem is for

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}), \tag{2.6a}$$

where

$$J(\boldsymbol{\theta}) = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\hat{y}-y)^2}, \tag{2.6b}$$

N is the total number of output samples and $\boldsymbol{\theta} = [\mathbf{W}, \mathbf{b}]$. The cost (2.6b) is calculated based on the deviation of the observed y values and the predicted outputs $\hat{y}$. The

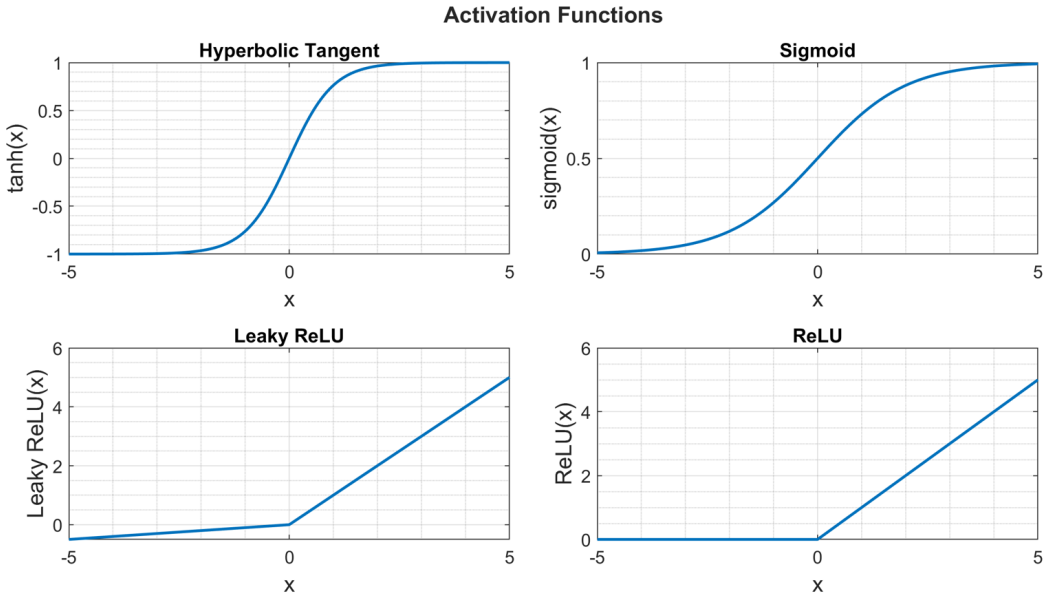

Figure 2.3 : Activation Functions

backpropagation technique [28] can be used to find a solution to (2.6a). The weights and biases in backpropagation are adjusted iteratively by using the gradient descent method, as indicated in (2.7) [29]

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha \nabla J(\boldsymbol{\theta}^{(i)}). \tag{2.7}$$

According to (2.7), the new values of $\boldsymbol{\theta}$ are obtained by subtracting the product of the learning rate, which is denoted by symbol α, and the *gradient* of the cost function ∇J with respect to $\boldsymbol{\theta}$ at the current iteration from the current values of $\boldsymbol{\theta}$.

When the learning rate is high, the updated weights and bias may deviate significantly from the ideal solution and diverge. Conversely, a low learning rate might lead to a slow convergence of the learning algorithm [30]. The iterative approach for updating parameters in (2.7) necessitates the initialization of the learning rate, weights and biases. Proper initialization is essential since it may lead to a learning process converging to the local minimum of the cost function. Regrettably, there is no established initialization procedure to start the iteration in (2.7) that avoids converging local minima.

2.1.2 Convolutional Neural Networks

FFNNs can struggle with the curse of dimensionality during the training process. One alternative to FFNNs might be CNNs. CNNs use a set of filters to slide over the

data to extract local features. Although CNNs are mostly preferred for spatial tasks related to computer vision and image recognition such that input data can be pixel values in 3 dimensions such as 64x64x3 [31], they can also be used to capture temporal patterns such as for time-series analysis [32]. CNNs and RNNs may also be used in combination [33]. One advantage of CNN over DNN is that CNN uses the same weights and biases across different time intervals of the same signal.

Temporal convolutional networks (TCNs) are 1-D convolutional neural networks with causal convolutions where future input data is not used to predict current output [34]. All the layers have the same length due to zero padding which enlarges the input size artificially in order to have the same dimensions of input and output [35]. However, causal convolution is not feasible to be used for longer sequences. Instead of applying the filter to the contiguous time steps of the sequential data, the filter may skip some time steps which is called dilation. The dilation factor specifies how many time steps are skipped by the filter. Dilated causal convolution might be used for longer sequence tasks with a larger filter size. Each element of a filter has a weight that is obtained as a result of training. If there are hidden layers $l = 1, 2, ..., L$ in a network, then (2.5) may be rewritten for a TCN as

$$\boldsymbol{h}_t^{[l]} = \sigma_l \left(\sum_{i=1}^{f_s^{[l]}} \boldsymbol{w}_i^{[l]} \odot \boldsymbol{h}_{t-(i+1)d^{[l]}}^{[l-1]} + \boldsymbol{b}^{[l]} \right), \tag{2.8}$$

where the inputs of the network $\boldsymbol{x}$ is denoted as $\boldsymbol{h}^{[0]}$ and outputs of the network $\hat{\boldsymbol{y}}$ is denoted as $\boldsymbol{h}^{[L+1]}$. $\boldsymbol{f}_s^{[l]}$ and $d^{[l]}$ represent filter length and dilation number for the l_{th} layer.

A TCN is shown in Figure 2.4 with a filter length of 3 on both layers, one dilation factor between the input and hidden layer, and two dilation factors between the hidden and output layers. The outputs and hidden units of the TCN are shown in (2.9)

$$\begin{aligned} \hat{\boldsymbol{y}}_T &= \sigma_2(\boldsymbol{w}_1^{[2]} \odot \boldsymbol{h}_T + \boldsymbol{w}_2^{[2]} \odot \boldsymbol{h}_{T-2} + \boldsymbol{w}_3^{[2]} \odot \boldsymbol{h}_{T-4} + \boldsymbol{b}^{[2]}), \\ \boldsymbol{h}_T &= \sigma_1(\boldsymbol{w}_1^{[1]} \odot \boldsymbol{h}_T + \boldsymbol{w}_2^{[1]} \odot \boldsymbol{h}_{T-1} + \boldsymbol{w}_3^{[1]} \odot \boldsymbol{h}_{T-2} + \boldsymbol{b}^{[1]}). \end{aligned} \tag{2.9}$$

2.1.3 Recurrent Neural Networks

RNNs are capable of modeling systems with memory, such as dynamical systems. RNNs are created by incorporating cyclic connections into FFNNs. In other words,

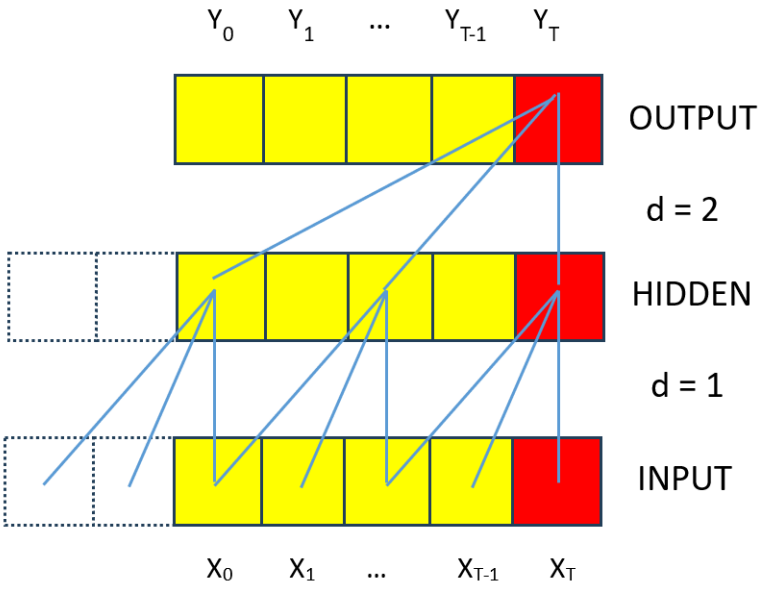

Figure 2.4 : Temporal Convolutional Network

the network takes past values of the predicted output signal as inputs, such as

$$\hat{y}_t = g_t(\hat{y}_{t-1}, \hat{y}_{t-2}, \ldots, x_t, \ldots). \tag{2.10}$$

Also, another recursive formula may be used as below [36, 37] :

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h), \tag{2.11a}$$

$$\hat{y}_t = \sigma(W_{yh}h_t + b_y). \tag{2.11b}$$

In (2.11a), the current hidden state h_t is computed based on the current input x_t, the previous hidden state h_{t-1}, bias b_h, and weights W_{hx} and W_{hh}. The activation function σ is applied to the sum of these terms. σ is not explicitly mentioned here, but it may be a sigmoid function or tanh, which are both differentiable. Equation (2.11b) calculates the predicted output $\hat{y}$ using the current hidden state h_t, current input x_t, bias b_y, and weights W_{yx} and W_{yh}. W_{*x} can be called input weights whereas W_{*h} can be called recurrent weights. Although a DNN may be used for modeling time series, it is necessary to include prior values of the same feature as distinct inputs. This results in an increase in the number of parameters since each input requires its own weights. However, the number of parameters does not increase for an RNN because prior values of the same features share the same weights.

2.1.3.1 Gradient Exploding and Vanishing Problem

The weights W_{**} and biases b_* in (2.11) are obtained as the solution of optimizing (2.6) as elaborated in Section 2.1. The learning process is influenced by two well-known issues. The gradient of the loss function in (2.7) may either exponentially drop or exponentially increase. The *vanishing* and *exploding* gradients are the terms used to describe these events. To address the issue of a growing gradient, the gradient *clipping* technique can be used [38]. If the gradient is above a certain threshold, it is reduced in magnitude. Nevertheless, the issue of gradient vanishing presents a more intricate challenge to address. Long Short-Term Memory [39] or variants of it, such as Gated Recurrent Unit (GRU) [40] techniques, successfully mitigate the issue of the vanishing gradient. LSTM is referenced in section 2.1.3.2 since it is used for the mass estimate in section 5.1.

2.1.3.2 Long Short-Term Memory

LSTM was first introduced with only input and output gates in [39]. The work in [41] then developed modern LSTMs by adding an adaptive forget gate. This enhancement enables the LSTM cell to learn how to reset memory blocks when needed. Input, output and forget gates, together with their weights, biases and sigmoid functions, play a crucial role in selectively permitting input signals to either be stored, outputted, or forgotten in an LSTM network cell. The incorporation of these gates regulates the flow of the cell's state. Therefore, unlike classic RNNs, the gradient in LSTM depends on the forget gate weights. This dependence allows updating the forget gate weights during training to prevent the gradient vanishing issue. Nevertheless, this mechanism does not address the problem of gradient explosion, which is handled by gradient clipping. The resulting architecture of an LSTM cell is presented in Figure 2.5 [42].

Equation (2.12) calculates the value of forget gate f_t, which is obtained by applying the sigmoid function σ to the sum of three terms: the weighted inputs $W_{fx}x_t$, the weighted previous hidden state $W_{fh}h_{t-1}$, and the bias term b_f

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f). \tag{2.12}$$

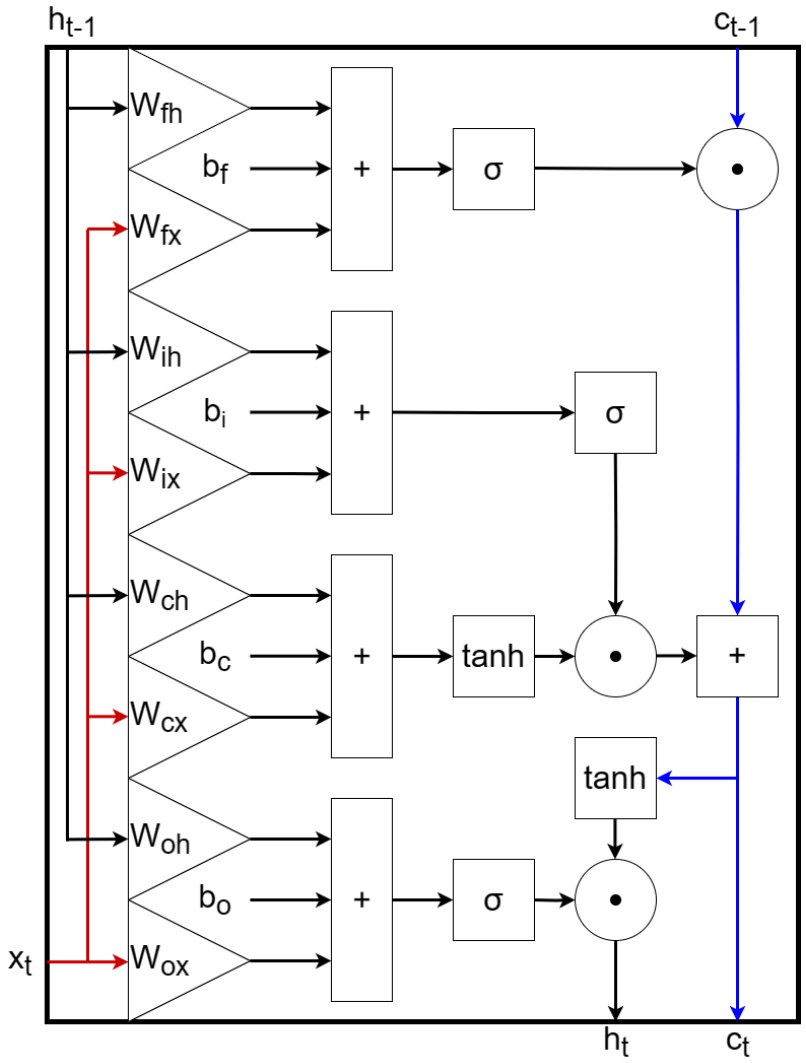

Figure 2.5 : LSTM Cell Structure

By using a similar structure as in (2.12), the values of the input (2.13) and output gates (2.14) are obtained

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i), \tag{2.13}$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o). \tag{2.14}$$

The terms W_{*x} and W_{*h} denote the input and recurrent weights, respectively. The current cell state c_t is calculated by taking the element-wise product of the forget gate f_t and the previous cell state c_{t-1} in order to retain important information from the past, and adding it to the element-wise product of the input gate i_t and the hyperbolic tangent of the sum of the weighted inputs, the weighted previous hidden state and the bias term in order to determine how much of the new information should be added to the cell state as

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c). \tag{2.15}$$

The element-wise multiplication is denoted as $\odot$. The output of the LSTM cell is hidden state h_t as

$$h_t = o_t \odot tanh(c_t). \tag{2.16}$$

In the literature, peephole connections refer to the direct connections from the cell state to the gates (input, forget, and output gates) in an LSTM network. These connections allow the gates to access the current cell state, enabling them to make more informed decisions about when to update or reset the cell state. This modification may improve the ability of the model to learn precise timings and dependencies. The values of input, output and forget gates for an LSTM with peephole connections can be written as below by modifying (2.12), (2.13) and (2.14) respectively

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f), \tag{2.17}$$

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i), \tag{2.18}$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o). \tag{2.19}$$

In [43], an LSTM with such additional features is named Vanilla LSTM, and the performance of eight variants is compared: without input gate, without forget gate, without output gate, without input activation function, without output activation function, coupled input and forget gate (CIFG) which is also known as GRU in the literature, without peepholes (NP) and full gate recurrence which have recurrent connections between all gates. Although none of the simplified variants visibly improve the performance, CIFG and NP do not significantly reduce the performance despite their reduced computational complexity. NP has already been shown in Figure 2.5 and CIFG (GRU) is shown in Figure 2.6.

GRU has two gates: an update gate denoted by z and a reset gate denoted by r as shown in (2.20) and (2.21), respectively

$$z_t = \sigma(W_{zx}x_t + W_{zh}h_{t-1} + b_z), \tag{2.20}$$

$$r_t = \sigma(W_{rx}x_t + W_{rh}h_{t-1} + b_r). \tag{2.21}$$

The output of a GRU cell is

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot tanh(W_{hx}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h). \tag{2.22}$$

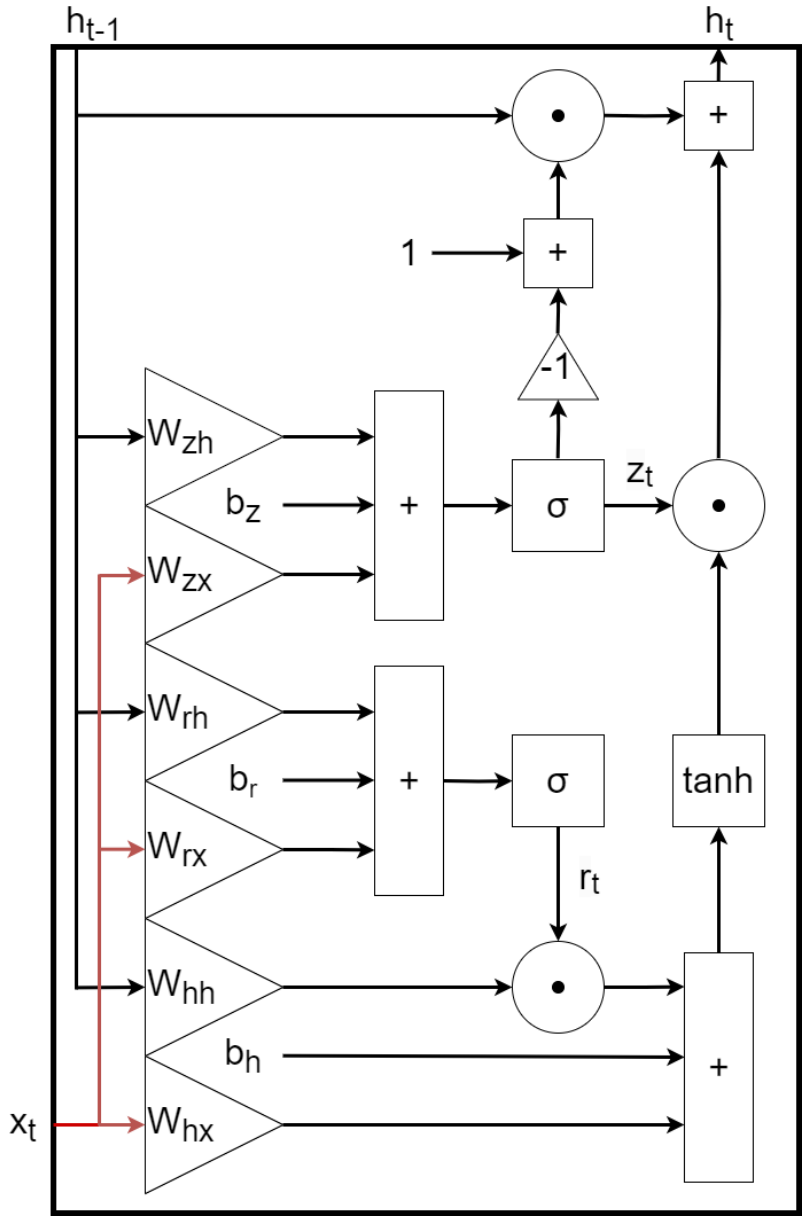

Figure 2.6 : GRU Cell Structure

The reason why GRU is named CIFG in [43] is because the update gate structure in (2.20) is the same as the input gate structure in (2.13) and $1 - z_t$ is used for forgetting instead of defining a forget gate as seen in (2.22). Therefore, the number of gates in each cell is reduced to two from three compared to NP variant of LSTM. As claimed in [42], GRU is simpler and less computationally demanding than LSTM thanks to a smaller number of parameters, whereas vanilla LSTM is more powerful and flexible since it has one more gate.

2.1.3.3 Applications in Sequence Modelling

LSTMs have found extensive applications in diverse fields, including natural language processing, speech recognition, image generation, video analysis, and time series prediction. In Figure 2.7, various sequence models are shown where input, hidden and output layers are shown with red, yellow and green colored square units, respectively.

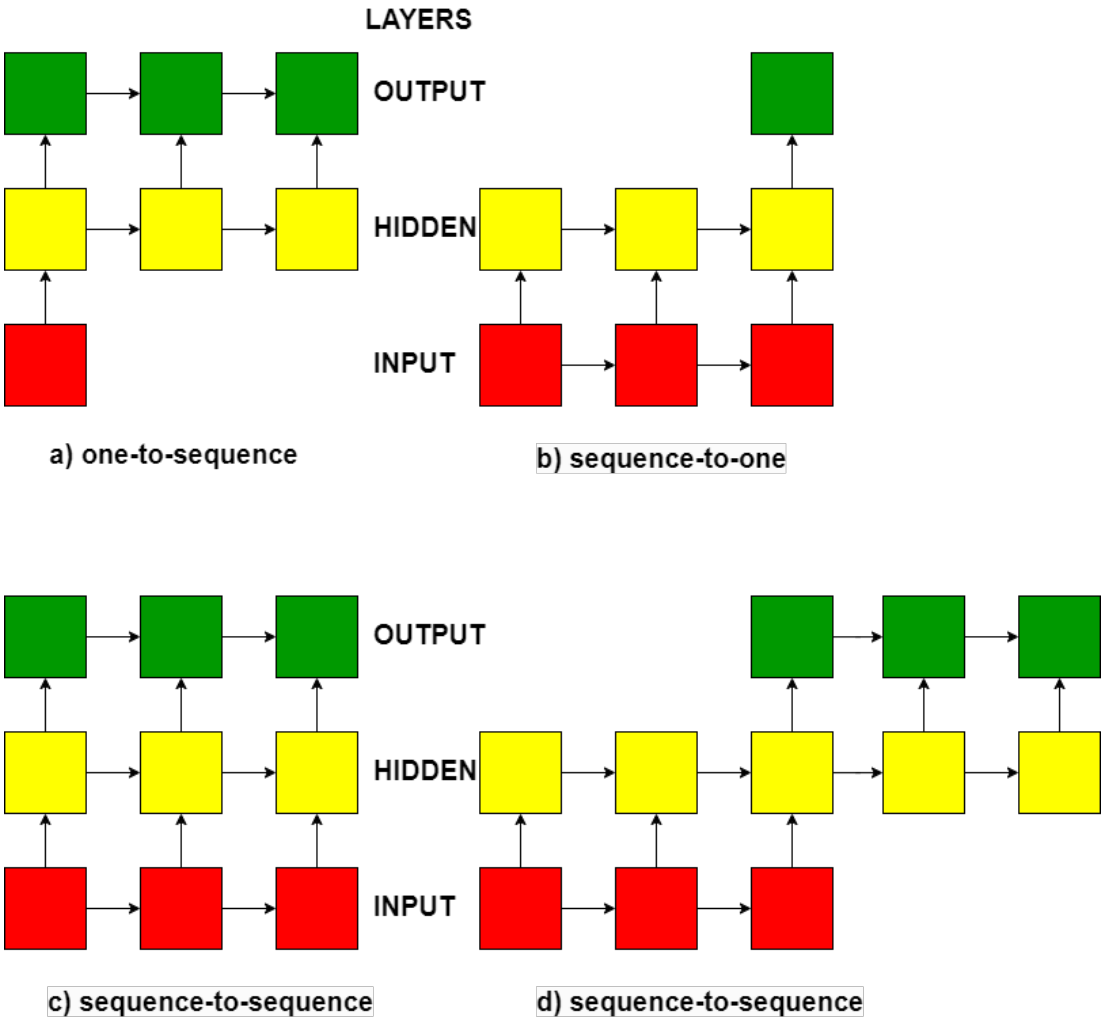

Figure 2.7 : Sequence Modeling

Figure 2.7a shows a one-to-sequence network where a single input results in a sequential output. An instance of a one-to-sequence network might be image captioning where an image is provided as an input and words describing the image are the outputs. In contrast, the sequence-to-one architecture shown in Figure 2.7b maps an entire sequence of inputs to a single output. Sentiment analysis can be included into this category since it determines whether a sentence is positive or negative. When

both inputs and outputs are sequential data, sequence-to-sequence networks are used. Figure 2.7c shows a sequence-to-sequence network where the output is computed at each time step, as seen in cases where input and output signals represent a dynamical system. Another type of sequence-to-sequence network is shown in Figure 2.7d where the output sequence is obtained after the entire input sequence is processed. As an example, a multi-layer LSTM is used to translate English sentences into French. The encoder-decoder structure is used, the sentence in English is encoded and then it is decoded into a corresponding French sentence [44].

Beyond these applications, LSTMs are widely adopted in black box modeling, parameter estimation and control of dynamical systems. One notable contribution is the development of an online LSTM-based learning technique for nonlinear regression, employing a particle filter approach [45]. The predictive capabilities of a modified LSTM with different activation functions are harnessed in order to anticipate the output of nonlinear systems [46]. In the maritime domain, LSTM and GRU have been employed for predicting acceleration in the vertical direction of a huge ship model, as discussed in [47]. The versatility of LSTMs extends to anomaly detection, as evidenced by the exploration of various LSTM networks for this purpose in [48]. The study presented in [49] introduces convex LSTMs for identifying dynamical systems. Furthermore, a novel algorithm for multivariate brake pressure estimation, relying on LSTM, is proposed in [50]. Notably, [48] offers a comprehensive survey of cutting-edge anomaly detection methods, with a focus on LSTMs.

2.2 Network Architecture Design and Training Procedure

Next, we recall the training problem, how to choose cost functions, output units, activation functions, and performance metrics based on the network architecture decided based on the problem. Then, how forward and backward passes and the number of parameters are computed are explained. Various optimization, overfitting prevention and parameter initialization methods proposed in the literature are explained. In addition, training procedure and testing are explained.

2.2.1 Loss Functions and Output Units

Mean squared error (MSE) or mean absolute error, as shown in (2.23) and (2.24) respectively, are commonly used for regression problems to minimize the error in the loss function by using the difference between the actual or measured output y and the predicted output $\hat{y}$ where N is the number of output samples

$$J_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{2.23}$$

$$J_{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{2.24}$$

For classification problems, the choice of the cost function depends on the number of classes. The binary cross-entropy loss function shown in (2.25) is a cost function that is used for binary classification problems to compare the measured output with the predicted output

$$J_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i). \tag{2.25}$$

On the other hand, the categorical cross-entropy loss function, as shown in (2.26), is used as the cost function if the problem is a multi-class classification problem where the number of outputs is equal to the number of the classes. The predicted outputs represent the probability distribution across the defined classes C

$$J_{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}). \tag{2.26}$$

where y_{ij} is the true label which is 1 when the output belongs to the j_{th} class and 0 if not. This loss function measures the difference between these predicted probabilities and the true class. If the true label y is represented by one-hot encoded vector, then (2.26) can be shown as in (2.27)

$$J_{CCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i). \tag{2.27}$$

One-hot encoding is used for binary vector representation of categorical data. Consider data with three classes: C_1, C_2, C_3. By using one-hot encoding, classes are represented such as $C_1 = [1,0,0]$, $C_2 = [0,1,0]$ and $C_3 = [0,0,1]$.

2.2.2 Activation Functions

The sigmoid activation function, also called the logistic activation function, is used in the output layer for binary classification which gives an output between 0 and 1. If the output of the network is less than 0.5, the prediction is accepted as 0; otherwise, it is accepted as 1.

The softmax activation function is used at the output layer of the multi-class classification problem as shown in Table 2.1 where C represents the number of classes and c represents the class which is computed at that time. The summation of probabilities of all classes is equal to 1. The class which has the highest probability is the predicted class.

ReLU is commonly used in the output layer for regression problems, and preferred in the hidden layers of FFNNs due to its simplicity and computational efficiency. ReLU introduces non-linearity by outputting zero for any negative input and passing positive inputs unchanged. ReLU contributes to sparsity in neural networks due to leading to many inactive neurons when the input is negative. Sparsity increases computational efficiency since less memory is used and training is faster. Also, since ReLU only deals with neurons with positive inputs, it may result in better interpretability. However, in some cases, if the neurons always output zero for all inputs then weights connected to that neurons can not be updated properly. This issue disrupts the learning process. For these cases, leaky ReLU can be preferred since leaky ReLU provides a small slope for negative inputs that ensures gradients flowing through all neurons.

For RNN, LSTM is commonly used as a hidden cell, and tanh or sigmoid function is preferred as an activation function.

2.2.3 Performance Metrics

While loss functions are used to minimize the error between actual and predicted values during the training process, performance metrics are used to understand how the trained model performs on new data that is not used during training.

Mean squared error and mean absolute error, which are explained in Section 2.2.1, can also be used as performance metrics of a network for regression problems. Also, root mean squared error, RMSE, shown in (2.28) can be used to evaluate the performance

of the network

$$J_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}. \tag{2.28}$$

For classification problems, performance metrics such as accuracy and precision can be obtained by using the confusion matrix shown in Table 2.2 [51]. If one of the classes

Table 2.2 : Confusion Matrix

		Predicted	
		Positive	**Negative**
Actual	**Positive**	True Positive	False Negative
	Negative	False Positive	True Negative

is defined as positive and the other class is defined as negative for binary classification, or if a chosen class is defined as positive and all other classes are defined as negative for multi-class classification then the terms in the confusion matrix can be explained as follows.

- **True Positive** is the case when both actual and predicted samples are positive.
- **False Negative** is the case when the actual value is positive but the predicted value is negative.
- **False Positive** is the case when the actual value is negative but the predicted value is positive.
- **True Negative** is the case when both actual and predicted samples are negative.

Accuracy is the number of correctly predicted classes out of total samples, as shown in (2.29)

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FalseNegative}. \tag{2.29}$$

If the classes are imbalanced, only considering accuracy might give false outcomes [52]. Precision is the number of correctly predicted positives out of predicted positives, as shown in (2.30)

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}. \tag{2.30}$$

In order to avoid false positives, precision can be used as the metric.

2.2.4 Forward Pass and Backpropagation

Processing the input data through the network is called *forward pass*. *Backpropagation*, also named backward pass, is an algorithm used to train neural networks by computing the gradients of the loss function concerning each weight throughout the backward passes. The gradients represent how much the loss would change if the weights are adjusted. In order to compute these gradients, the backpropagation algorithm relies on the chain rule of calculus, which systematically breaks down the gradient computation across the layers of the network. Weights and biases are updated in each iteration by using these gradients. One iteration is completed by completing one forward pass and backpropagation respectively.

Although obtaining numerical gradient is simpler than analytic gradient, such as forward difference approximation as shown in (2.31), the process is error-prone

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{2.31}$$

Therefore, the numerical gradient should only be used for gradient check. In practice, it is suggested to use central difference approximation as shown in (2.32) for the calculation of the numerical gradient at point x instead of the forward difference approximation to mitigate errors [53]

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{2.32}$$

However, this approximation increases the computational complexity.

Although symbolic differentiation can produce analytical solutions in some cases, it is not runtime efficient since the calculation of the derivatives may get exponentially expensive. Automatic differentiation is an approach that divides the function into a sequence of primitive operations to compute derivates [54]. It differs from both numerical and symbolic differentiation. For backpropagation, reverse mode automatic differentiation is used.

To give a simple example, consider a simple neural network that is used for binary classification with two inputs and only one hidden layer with only one neuron which has a linear activation function such that $f_h(x) = x$. Since the output is 0 or 1, a sigmoid activation function is used. Therefore $f(x) = 1/(1+e^{-(w_1x_1+w_2x_2)})$. The computational graph of this equation is shown in Figure 2.8 where blue values represent the forward

pass as shown in (2.33) and red values represent the backpropagation as shown in (2.34) whereas nodes represent primitive operations:

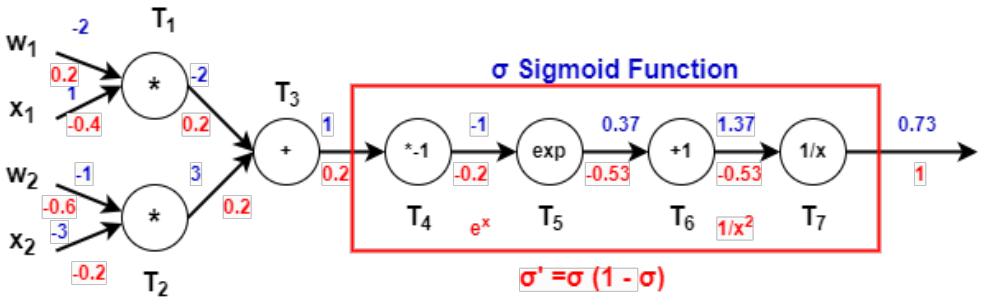

Figure 2.8 : Forward and Backward Pass

$$
\begin{aligned}
T_1 = w_1 x_1 = -2 * 1 = -2, \\
T_2 = w_2 x_2 = -1 * -3 = 3, \\
T_3 = T_1 + T_2 = 1, \\
T_4 = -T_3 = -1, \\
T_5 = e^{T_4} = e^{-1} = 0.37, \\
T_6 = T_5 + 1 = 1.37, \\
T_7 = 1/T_6 = 0.73.
\end{aligned} \tag{2.33}
$$

$$
\begin{aligned}
\frac{\partial T_7}{\partial T_7} = 1, \frac{\partial T_7}{\partial T_6} = \frac{-1}{T_6^2} = -0.53, \\
\frac{\partial T_6}{\partial T_5} = 1, \frac{\partial T_7}{\partial T_5} = \frac{\partial T_7}{\partial T_6}\frac{\partial T_6}{\partial T_5} = -0.53, \\
\frac{\partial T_5}{\partial T_4} = e^{T_4} = e^{-1} = 0.37, \frac{\partial T_7}{\partial T_4} = \frac{\partial T_7}{\partial T_5}\frac{\partial T_5}{\partial T_4} = -0.2, \\
\frac{\partial T_4}{\partial T_3} = -1, \frac{\partial T_7}{\partial T_3} = \frac{\partial T_7}{\partial T_4}\frac{\partial T_4}{\partial T_3} = 0.2, \\
\frac{\partial T_3}{\partial T_2} = \frac{\partial T_3}{\partial T_1} = 1, \frac{\partial T_7}{\partial T_2} = \frac{\partial T_7}{\partial T_3}\frac{\partial T_3}{\partial T_2} = 0.2, \\
\frac{\partial T_2}{\partial w_2} = x_2 = -3, \frac{\partial T_7}{\partial w_2} = \frac{\partial T_7}{\partial T_2}\frac{\partial T_2}{\partial w_2} = -0.6, \\
\frac{\partial T_2}{\partial x_2} = w_2 = -1, \frac{\partial T_7}{\partial x_2} = \frac{\partial T_7}{\partial T_2}\frac{\partial T_2}{\partial x_2} = -0.2, \\
\frac{\partial T_1}{\partial w_1} = x_1 = 1, \frac{\partial T_7}{\partial w_1} = \frac{\partial T_7}{\partial T_1}\frac{\partial T_1}{\partial w_1} = 0.2, \\
\frac{\partial T_1}{\partial x_1} = w_1 = -2, \frac{\partial T_7}{\partial x_1} = \frac{\partial T_7}{\partial T_1}\frac{\partial T_1}{\partial x_1} = -0.4.
\end{aligned} \tag{2.34}
$$

Since the derivative of the sigmoid function is [55]

$$\sigma' = \sigma(1 - \sigma). \tag{2.35}$$

Reverse mode automatic differentiation in (2.34) is equal to the derivative of the sigmoid function in (2.35)

$$\frac{\partial T_7}{\partial T_3} = 0.73 * (1 - 0.73) = 0.2. \tag{2.36}$$

2.2.5 Optimization Methods

One of the commonly preferred optimization algorithms is gradient descent which is already shown in Section 2.1.1. There are two more variants of this algorithm: stochastic gradient descent and mini-batch gradient descent. Stochastic gradient descent does not use the entire training dataset but only uses one training example for parameter update at each iteration [56]. Stochastic gradient descent is computationally less complex and can handle online learning since it requires much lower memory. However, gradient estimates are noisy and lead to high variance in updates. Therefore, the stochastic gradient descent is less stable and converges more slowly. For mini-batch gradient descent, the training data is divided into smaller datasets called mini-batches. Each mini-batch is used to compute the gradient and update the weights. This variant can be used when using the full batch is computationally expensive for a huge dataset since it has better convergence than stochastic gradient descent.

When the gradient moves in one direction, a momentum term λ can be used to speed up the convergence of stochastic gradient descent. Therefore, the gradient term $-\alpha\nabla J(\boldsymbol{\theta}^{(i)})$ in (2.7) can be replaced with $\mathbf{v}^{(i+1)}$ as in (2.37):

$$\begin{aligned} \mathbf{v}^{(i+1)} &= \lambda \mathbf{v}^{(i)} - \alpha\nabla J(\boldsymbol{\theta}^{(i)}), \\ \boldsymbol{\theta}^{(i+1)} &= \boldsymbol{\theta}^{(i)} + \mathbf{v}^{(i+1)}. \end{aligned} \tag{2.37}$$

There are variants of this method such as Nesterov momentum as shown in (2.38) [57]

$$\mathbf{v}^{(i+1)} = \lambda \mathbf{v}^{(i)} + \alpha\nabla J(\boldsymbol{\theta}^{(i)} - \lambda \mathbf{v}^{(i)}). \tag{2.38}$$

There are also other methods such as AdaGrad [58] which is an adaptive gradient algorithm that is efficient with sparse gradients. Root mean square propagation, RMSProp [59], which adds elementwise square of gradient terms to momentum terms, and this summation, denoted as s, is added to weights and biases for parameter update after square root is taken, as shown in (2.39) [60]

$$\begin{aligned} \mathbf{s}^{(i+1)} &= \lambda \mathbf{s}^{(i)} - (1-\lambda)(\nabla J(\boldsymbol{\theta}^{(i)}) \odot \nabla J(\boldsymbol{\theta}^{(i)})), \\ \boldsymbol{\theta}^{(i+1)} &= \boldsymbol{\theta}^{(i)} - \alpha\nabla J(\boldsymbol{\theta}^{(i)}) / \sqrt{\mathbf{s}^{(i+1)}}. \end{aligned} \tag{2.39}$$

RMSprop is well-suited for training deep neural networks where significant variation in gradients can occur across layers. Adaptive moment estimation (Adam) is an optimization method that was invented to merge the benefits of AdaGrad and RMSprop [61] as shown in (2.40) [62] where λ_1 and λ_2 are decay rates for the moment estimates:

$$\begin{aligned}
\mathbf{v}^{(i+1)} &= \lambda_1 \mathbf{v}^{(i)} - (1-\lambda_1)\nabla J(\boldsymbol{\theta}^{(i)}), \\
\mathbf{s}^{(i+1)} &= \lambda_2 \mathbf{s}^{(i)} - (1-\lambda_2)(\nabla J(\boldsymbol{\theta}^{(i)}) \odot \nabla J(\boldsymbol{\theta}^{(i)})), \\
\hat{\mathbf{v}}^{(i+1)} &= \mathbf{v}^{(i+1)} / (1-\lambda_1^{i+1}), \\
\hat{\mathbf{s}}^{(i+1)} &= \mathbf{s}^{(i+1)} / (1-\lambda_2^{i+1}),, \\
\boldsymbol{\theta}^{(i+1)} &= \boldsymbol{\theta}^{(i)} - \alpha \hat{\mathbf{v}}^{(i+1)} / \sqrt{\hat{\mathbf{s}}^{(i+1)}}.
\end{aligned} \tag{2.40}$$

2.2.6 Overfitting Prevention Techniques

Regularization techniques are used to prevent overfitting by adding constraints or penalties to the cost function. Different regularization and other methods exist which are concisely described next.

- **L1 regularization** is obtained by adding the sum of absolute values of the weights multiplied by a chosen parameter $\lambda \sum_{i=1}^{N} |w_i|$ to the cost function as a regularization term. This regularization tends to drive some weights to exactly zero, effectively removing the features, which are multiplied by those weights, from the model. Therefore, overfitting is unlikely because there are fewer parameters that could capture noise in the training data. This regularization is also called Lasso regression where Lasso means least absolute shrinkage and selection operator [63].

- **L2 regularization** is the case when the sum of square values of weights is multiplied by a chosen parameter $\lambda \sum_{i=1}^{N} w_i^2$ and added to the cost function as a penalty term.

- **Dropout** is a way to remove neurons randomly and temporarily from the network with a predefined probability at each iteration to make the network more robust to overfitting [64]. Overfitting is prevented since the network is less dependent on specific neurons, so it does not memorize noise in the training data. Therefore, the network generalizes well to new, unseen data. It is important to emphasize that dropout only happens throughout the training process.

- **Early stopping** or validation patience terms are used to stop training when the performance of the validation data set does not improve during a predefined number of epochs.

2.2.7 Parameter Initialization

As mentioned in Section 2.2.4, the loss function is obtained as a result of a forward pass where the current values of the weights and biases are used. Then weights and biases are updated by (2.7). However, at the beginning of the training process, these parameters need to be initialized. Parameter initialization is essential for efficient training of neural networks [65]. Therefore, while zero initialization is commonly used for biases, except for the forget gates of LSTM where one initialization may be preferred, various techniques are used for weights initialization [66].

- **Glorot Initialization**, also called Xavier initialization, is a technique that involves sampling the weights to be learnt from a uniform distribution between $\pm\sqrt{6/(n_i+n_o)}$, where n_i and n_o represent the numbers of inputs and outputs, respectively [67].

- **Orthogonal initialization** is another method where a randomly generated matrix is decomposed into an orthogonal matrix Q and an upper triangular matrix R using QR decomposition [68]. This orthogonal matrix is mostly preferred to initialize the recurrent weights of LSTM.

- **He initialization** is a method that is efficient when ReLU or PReLU activation functions are used. Initialization of the weights is done from the normal distribution with zero mean and $2/n_i$ variance [69].

2.2.8 Training Procedure and Testing

Deciding the architecture of a NN is an iterative process, including the following stages:

1. *Planning.* During this initial stage, the network structure is planned that involves determining the number of hidden layers and the number of units (neurons, kernels, or cells) in each layer, depending on the network architecture, and the activation

functions. The chosen network design leads to the acquisition of a collection of weights and biases, denoted as $\mathbf{W}$ and $\mathbf{b}$, which need to be learnt.

2. *Training and Validation.* The weights and biases $\mathbf{W}$, $\mathbf{b}$ are *learned* by solving the optimization problem (2.6), with the objective of minimizing the difference between the model output $\hat{y}$ and the system output measurements y, in the sense of the cost function (2.6b). During this stage, it is necessary to choose a set of *hyperparameters* [70] in order to optimize the learning process for both speed and accuracy. The hyperparameters discussed in this section include dropout probability, epoch number, weight initialization, batch size, and early stopping.

   An NN training is influenced by the range of values of the input signals, often known as the *features*. The lack of balance in the size of the characteristics may disrupt the training process, resulting in certain features being either fully or partly irrelevant. To prevent such a problem, features need to be *normalized*.

   The feasibility of training a neural network on a whole data set may be hindered by factors such as hardware availability, the size of the data set, and the complexity of the network. If such limitations exist, the dataset may be divided into smaller data sets which are referred to as mini-batches. During each iteration of the training process, the weights and biases of a mini-batch are modified.

   The epoch number represents how many times the learning algorithm is applied to the entire training dataset. The number of iterations in one epoch is obtained by dividing the number of training datasets by the batch size. A maximum epoch number is established to limit the length of the training process. The selected maximum epoch number should be sufficiently high to train the network effectively. As stated in section 2.1, if the learning rate is low, a higher number of epochs is required to achieve convergence. The loss function is plotted during epochs based on learning rates [71] as shown in Figure 2.9.

   The learning rate is contingent upon the selected optimization technique for training the neural network. The learning rate may either remain constant or be reduced after a certain period. Moreover, if the cost J in (2.6b) fails to decrease after a certain number of iterations, training may be halted by *early stopping*, regardless of the maximum epoch number.

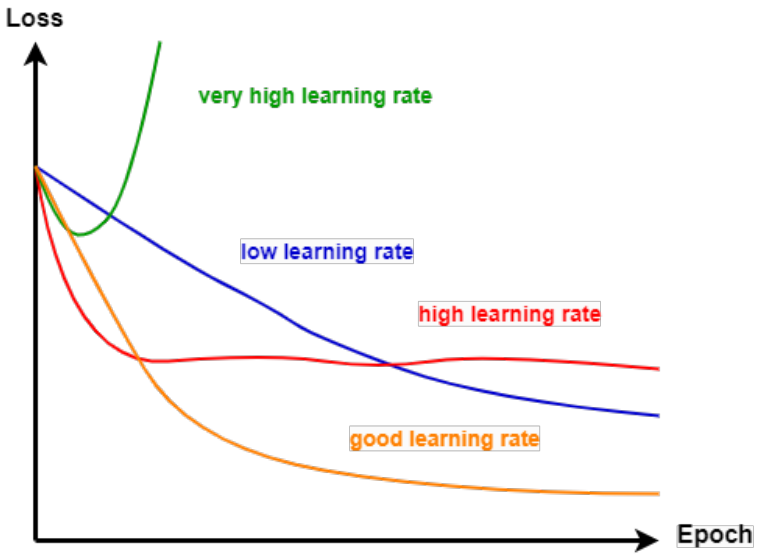

Figure 2.9 : Learning rate effect on the loss function

If hidden layers or neurons are not enough, the network cannot effectively comprehend the connection between inputs and outputs, so *underfitting* occurs. Conversely, if a network is created with an excessive number of weights and biases, the surplus parameters during the training phase may end up learning the noise, which inevitably impacts the measurements of the output y. This overfitting may be identified by assessing the performance of the learnt model $\hat{y} = \hat{f}(x, \boldsymbol{\theta}^*)$ on the validation dataset. Overfitting occurs when the discrepancy between the predicted value $\hat{y}$ and the actual value y is much larger when evaluated on the validation dataset compared to the training dataset. To address overfitting, it is recommended to consider using a network architecture with a reduced number of parameters during the planning phase. Additionally, to mitigate the issue of overfitting, one of the regularization techniques explained in Section 2.2.6 can be used.

3. *Testing.* During the testing phase, the acquired model is evaluated using a new dataset, which is distinct from the data used for training and validation. The reason why a test dataset is needed in addition to a validation dataset is that although the validation dataset is not used for training, it indirectly affects the training since it is used simultaneously during training and determines which set of parameters should be used in the acquired model. For the test dataset, if the error is deemed unsatisfactory, the hyperparameters can be tuned again. If the error still remains high, then a new network topology is chosen by returning to the planning phase.

2.2.9 Number of Weights and Biases

Each neuron has a bias term and receives input from the former layer, which can be seen in Figure 2.1. Therefore, if the numbers of neurons in the current layer and the previous layer are n_l and n_{l-1} respectively, then the number of weights for a fully connected layer is $n_l \times n_{l-1}$. When the number of biases, which is equal to the number of neurons in the current layer n_l is added, the total number of parameters for a fully connected layer is $n_l \times (n_{l-1} + 1)$. Fully connected or dense layers are not only used for FFNNs but also usually used in CNN and RNN architectures before the output layer based on the problem. If the number of inputs and the number of neurons of the fully connected layer is high, computational complexity increases since the number of parameters to be computed increases for the training process.

Consider a convolution layer that has k kernels. Each kernel has a bias term. If the kernel size is $s_1 \times s_2 \times s_3$, then a convolution layer has k biases and $k \times s_1 \times s_2 \times s_3$ weights, so total parameters are $k \times (s_1 \times s_2 \times s_3 + 1)$.

For RNNs, the recurrent unit has recurrent weight in addition to input weights. The total number of parameters depends on the RNN type. For LSTM, each cell has 4 biases due to input, output, forget gates, and cell state. If there are n_{l-1} inputs from the previous layer and n_l cells in an LSTM layer, the number of input weights is $4 \times n_l \times n_{l-1}$ and the number of recurrent weights is $4 \times n_l^2$. Therefore, the total number of parameters for an LSTM layer is $4 \times n_l \times (n_{l-1} + n_l + 1)$ [72]. Since GRU has one less gate than LSTM, the total number of parameters for a GRU layer is $3 \times n_l \times (n_{l-1} + n_l + 1)$.

Dropout or pooling layers do not have a direct effect on the number of parameters. However, by using pooling, since the dimension is reduced, the number of parameters for a fully connected layer decreases. Meanwhile, normalization layers have parameters. For instance, when layer normalization is used, since there is one weight and bias for each unit, the total number of parameters to be determined is $2 \times n_l$.

3. RECURSIVE LEAST SQUARES AND VEHICLE DYNAMICS

Next, we recall the fundamentals of least-squares estimation, along with its recursive version, and vehicle dynamics used in this thesis work.

3.1 Background on Recursive Least Squares

Consider a continuous-time, linear model

$$y(t) = \phi(t)\theta, \tag{3.1}$$

where $\theta \in \mathbb{R}^{n\times 1}$ is a vector of unknown parameters to be estimated, $y(t) \in \mathbb{R}^{p\times 1}$ is the system outputs vector and $\phi(t) \in \mathbb{R}^{p\times n}$ is the regressor at time t. If data is collected with a fixed sampling time T_s, (3.1) can be rewritten as

$$y_k = \phi_k\theta. \tag{3.2}$$

ϕ_k includes past measurements such as past inputs and outputs samples of a linear discrete-time dynamical system if an Auto-Regressive exogenous (ARX) model is considered such as

$$y_k = a_1y_{k-1} + a_2y_{k-2} + \ldots + a_ly_{k-l} + b_1u_{k-1} + b_2u_{k-2} + \ldots + b_mu_{k-m}, \tag{3.3}$$

where u_k is the input vector, $n = l + m$ and $\theta = [a_1\ a_2 \ldots a_l\ b_1\ b_2 \ldots b_m]'$ [73]. If N measurements are collected as: $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^{Np\times 1}$, $\Phi = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_N \end{pmatrix} \in \mathbb{R}^{Np\times n}$, the linear regression can be rewritten as

$$Y = \Phi\theta. \tag{3.4}$$

Unless Φ is a square matrix, the unknown parameters can be obtained by using the outputs and the pseudo-inverse of the regressor matrix by the equation

$$\theta = (\Phi^T\Phi)^{-1}(\Phi^TY). \tag{3.5}$$

However, the sensor measurements are perturbed by noise $\varepsilon = \begin{pmatrix} \varepsilon(1) \\ \vdots \\ \varepsilon(N) \end{pmatrix}$, so (3.4) is modified to $Y = \Phi\theta + \varepsilon$. After data is collected, least squares estimation can be used as a batch approach to estimate the parameters vector is found as

$$\hat{\theta} = arg\min_{\theta} \frac{1}{2}\|\varepsilon\|_2^2 == arg\min_{\theta} J(\theta). \tag{3.6}$$

In order to ensure a unique global minimizer, a regularization term is added to the least squares, so the least squares cost function can be written as

$$J(\hat{\theta}) = \frac{1}{2}[(Y - \Phi\hat{\theta})^T(Y - \Phi\hat{\theta}) + (\hat{\theta} - \theta_0)^T R(\hat{\theta} - \theta_0)], \tag{3.7}$$

where θ_0 is the initial value of the unknown parameters and $R \in \mathbb{R}^{n\times n}$ is positive definite. By taking the gradient of (3.7)

$$\nabla_{\hat{\theta}} J(\hat{\theta}) = -\Phi^T(Y - \Phi\hat{\theta}) + R(\hat{\theta} - \theta_0). \tag{3.8}$$

By setting the gradient in (3.8) to zero, optimal $\hat{\theta}$ is obtained as

$$\hat{\theta} = (\Phi^T\Phi + R)^{-1}(\Phi^T Y + R\theta_0). \tag{3.9}$$

If $\Phi^T\Phi + R$ is invertible, then there is a unique solution. However, the computational complexity is significant since the equation includes the inverse of a matrix. Also, the memory requirement increases store Φ when the number of measurements N increases.

Recursive least squares is an algorithm that, as the name implies, implementation is done recursively instead of using batch approach to reduce the computational burden and memory requirements. Therefore, instead of waiting for the collection of the whole data, data is processed sequentially [74]

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)[y(t) - \phi(t)\hat{\theta}(t-1)], \tag{3.10}$$

$$K(t) = P(t)\phi^T(t), P^{-1}(t) = P^{-1}(t-1) + \phi^T(t)\phi(t). \tag{3.11}$$

where K is the gain of the RLS algorithm and P_k is the error covariance matrix of the estimation error at the k_{th} step. Equations (3.10) and (3.11) are written in discrete time as

$$\hat{\theta}_k = \hat{\theta}_{k-1} + K_k[y_k - \phi_k\hat{\theta}_{k-1}], \tag{3.12}$$

$$K_k = P_k\phi_k^T, P_k^{-1} = P_{k-1}^{-1} + \phi_k^T\phi_k. \tag{3.13}$$

3.1.1 Persistent Excitation

Persistent excitation refers to a condition where the input signals to the system are sufficiently varied and informative, enabling accurate parameter estimation over time. Persistency can be defined as

$$\sum_{k=j}^{N+j} \phi_k^T \phi_k \geq \kappa I. \tag{3.14}$$

where κ is a positive constant, $j \geq 0$ and identity matrix $I \in \mathbb{R}^{n \times n}$. Persistent excitation is an important topic that guarantees the convergence of RLS.

Loss of persistency occurs when there are not enough changes in the inputs to satisfy the condition in (3.14). In this case, the information gathered from the inputs can not result in correct identification. Therefore, the data which is gathered during transient operation is valuable. However, using a forgetting factor which is explained in Section 3.1.3 diminishes or removes this data due to the exponentially decreasing weights used for older data.

3.1.2 Matrix Inversion Lemma

Matrix inversion lemma is an alternative method to find the inverse of a matrix [75]:

$$(A+BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1}+DA^{-1}B)^{-1}DA^{-1}, \tag{3.15}$$

where $A \in \mathbb{R}^{n_a \times n_a}$, $B \in \mathbb{R}^{n_a \times n_c}$, $C \in \mathbb{R}^{n_c \times n_c}$ and $D \in \mathbb{R}^{n_c \times n_a}$. If the matrix inversion lemma is used in RLS solution with the propagation of the covariance matrix in (3.13),

$$P_k = P_{k-1} - P_{k-1}\phi_k^T(I+\phi_k P_{k-1}\phi_k^T)^{-1}\phi_k P_{k-1}. \tag{3.16}$$

This equation decreases the computational burden when $n >> p$ [76]. Sherman-Morrison-Woodbury formula is a special form of the matrix inversion lemma where $n_c = 1$. The formula can be rewritten as

$$\left(A+uv^T\right)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}, \tag{3.17}$$

where $A \in \mathbb{R}^{n \times n}$ matrix and u and $v \in \mathbb{R}^{n \times 1}$ are column vectors. In order to reduce the computational complexity, the Sherman-Morrison-Woodbury formula is used in RLS solution in cases where there is a single output. Equation (3.16) can be rewritten as

$$P_k = P_{k-1} - \frac{P_{k-1}\phi_k^T\phi_k P_{k-1}}{1+\phi_k P_{k-1}\phi_k^T}, \tag{3.18}$$

so, the gain is rewritten as

$$K_k = P_k \phi_k^T = \frac{P_{k-1}\phi_k}{1 + \phi_k P_{k-1} \phi_k^T}. \tag{3.19}$$

3.1.3 Various RLS Approaches

Paper [77] provides a guide on the real-time implementation of RLS. The standard RLS algorithm is used with a growing window of data. Equation (3.7) may also be shown as

$$J_k(\hat{\theta}) = \frac{1}{2}[\sum_{i=0}^{k}(y_i - \phi_i\hat{\theta})^T(y_i - \phi_i\hat{\theta}) + (\hat{\theta} - \theta_0)^T R(\hat{\theta} - \theta_0)]. \tag{3.20}$$

A solution with a forgetting factor has been introduced in the literature when the parameters are time-varying in order to increase the speed of convergence of the RLS algorithm. According to this solution, the past data are retained, but their impact is diminished by using the forgetting factor λ that is a numerical value in the interval (0,1]. The forgetting factor is used in a geometric manner at each time step. The forgetting factor is incorporated into (3.20) by weighting past data exponentially over time as

$$J_k(\hat{\theta}) = \frac{1}{2}[\sum_{i=0}^{k}\lambda^{k-i}(y_i - \phi_i\hat{\theta})^T(y_i - \phi_i\hat{\theta}) + \lambda^{k+1}(\hat{\theta} - \theta_0)^T R(\hat{\theta} - \theta_0)]. \tag{3.21}$$

While using (3.12) as it is, (3.18) and (3.19) are updated respectively as

$$P_k = \frac{1}{\lambda}(P_{k-1} - \frac{P_{k-1}\phi_k^T\phi_k P_{k-1}}{\lambda + \phi_k P_{k-1}\phi_k^T}). \tag{3.22}$$

$$K_k = P_k\phi_k^T = \frac{P_{k-1}\phi_k^T}{\lambda + \phi_k P_{k-1}\phi_k^T}, \tag{3.23}$$

Equation (3.22) is obtained by taking the inverse of the error covariance matrix with matrix inversion lemma,

$$P_k^{-1} = \lambda P_{k-1}^{-1} + \phi_{k-1}^T\phi_{k-1}. \tag{3.24}$$

When using a forgetting factor in RLS, the algorithm becomes more sensitive to sudden changes in input values. This increased sensitivity is advantageous for adapting to new data quickly, but the number of steps to achieve convergence may become larger, or convergence may not occur due to the loss of persistency. If a lower forgetting factor is chosen, the recent data is weighted much more than the older data. Although the

sensitivity to sudden changes in the input signal increases, using a lower forgetting factor may lead to instability since the older data is quickly forgotten and the effective data points usage decreases. Therefore, a larger forgetting factor may be preferred despite slower convergence and lower adaptability to sudden changes [78]. Since the performance of RLS is often extremely sensitive to the chosen forgetting factor, this factor is usually obtained as a result of a trial and error process.

As opposed to the growing window approach, another alternative is to use finite window RLS. In this approach, a fixed window width is chosen. RLS estimation is done by adding the current data sample and removing the oldest data sample at each iteration [79]. Equation (3.20) is updated as

$$J_k(\hat{\theta}) = \frac{1}{2}[\sum_{i=k-l_w}^{k} (y_i - \phi_i\hat{\theta})^T(y_i - \phi_i\hat{\theta}) + (\hat{\theta} - \theta_0)^T R(\hat{\theta} - \theta_0)], \tag{3.25}$$

for sliding window case where l_w is the window length which is a non-negative integer and $l_w \leq k$. Paper [80] proposed a sliding-window RLS algorithm incorporating various regularization terms R into the cost function as shown in (3.7) and claimed that lower R causes faster convergence. Also, a larger window length results in better convergence. In [81], variable-rate forgetting is introduced as an alternative to constant rate-forgetting in (3.22). This approach is designed to ensure that the parameter estimates converge to their true values, even in scenarios where the noise in the system is not correlated with the regressor, as long as the system is under persistent excitation. Paper [82] claims that although the parameter estimates converge, they may not converge to their true values if excitation is not persistent. Also, error covariance P_k diverges and causes numerical instability. Variable-direction forgetting is proposed to prevent divergence of covariance by substituting the forgetting matrix Λ which is data-dependent for the constant forgetting factor in (3.22):

$$P_k^{-1} = \Lambda P_{k-1}^{-1}\Lambda + \phi_{k-1}^T\phi_{k-1}. \tag{3.26}$$

Necessary conditions for the global asymptotic stability of RLS are examined in [83] since persistence excitation is a sufficient but not a necessary condition. Lastly, [84] introduces two extensions of RLS: exponential resetting and cyclic resetting where exponential resetting is obtained by modifying (3.24) since the information matrix converges to zero when there is no excitation for exponential forgetting as

$$P_k^{-1} = \lambda P_{k-1}^{-1} + (1-\lambda)R_\infty + \phi_{k-1}^T\phi_{k-1}, \tag{3.27}$$

where R_∞ is positive definite. Cyclic resetting may also be used as another extension of RLS to ensure that the covariance matrix remains within certain limits during the estimation process because unbounded covariance may lead to numerical instability. Cyclic resetting is less computationally expensive than exponential resetting when $n >> p$ since cyclic resetting has the ability to use matrix inversion lemma. For cyclic resetting, orthogonal diagonalization of positive definite R_∞ is written as

$$R_\infty = Q_\infty D_\infty Q_\infty, \tag{3.28}$$

where orthogonal matrix Q_∞ and diagonal matrix D_∞ are both n x n-dimensional as shown:

$$Q_\infty = [Q_{\infty,0} \dots Q_{\infty,n-1}], \mathbf{D}_\infty = \begin{bmatrix} d_{\infty,0} & 0 & \cdots & 0 \\ 0 & d_{\infty,1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{\infty,n-1} \end{bmatrix} \tag{3.29}$$

Equation (3.27) is updated for cyclic resetting as

$$P_k^{-1} = \lambda P_{k-1}^{-1} + \frac{1-\lambda^n}{\lambda^{n-mod(k,n)-1}} R_{\infty,mod(k,n)} + \phi_{k-1}^T \phi_{k-1}, \tag{3.30}$$

where $R_{\infty,mod(k,n)}$ is defined as

$$R_{\infty,i} = d_{\infty,i} q_{\infty,i} q_{\infty,i}^T, \tag{3.31}$$

for all $i = 0, ..., n-1$.

3.1.4 Challenges and Limitations

One of the challenges of RLS is that its computational complexity increases when the number of parameters to be estimated increases since the algorithm involves matrix inversions and multiplications. Also, memory requirements may become significant as simulation time increases since some versions of RLS store the past data and estimated parameters in memory. Therefore, using these versions can be resource-intensive with respect to both time and memory. These problems can be partially solved by matrix inversion lemma by reducing the size of matrices that need to be inverted and recursion by focusing on the latest parameters. Also, a finite horizon might be preferred instead of an infinite time horizon to reduce the complexity.

If the model does not accurately capture the underlying system dynamics, the RLS algorithm may fail to converge to the correct parameters. Additionally, RLS is highly

sensitive to noise, which can significantly distort the parameter estimates. To mitigate the impact of noise, additional preprocessing or filtering steps may be required. Using a growing window of data can help reduce noise effects, but this approach introduces other challenges. Specifically, an infinite horizon window increases computational complexity and memory demands, and it also limits the ability of RLS to adapt to sudden changes in the system. To address this, a variant of RLS incorporates a forgetting factor into the cost function. While this adaptation allows RLS to respond more effectively to sudden changes by prioritizing recent data over older data, it also introduces the risk of divergence in the parameter estimates when the data are not persistently exciting, especially in the presence of sensor noise.

RLS is dependent on initial conditions, so convergence may take time due to the initial conditions. Also, when convergence occurs, the question arises whether RLS converges to a value that is in a suitable range, which is predefined based on the problem, or not. To sum up, convergence and consistency are challenging topics while using RLS.

RLS works well in case of persistency of excitation and RLS converges to true parameters with a low error if the model represents the dynamics. However, if the data is not persistently excited, RLS does not perform so well. Therefore, the correctness of the model and the quality of the data used with the model are quite significant for RLS performance.

3.2 Vehicle Dynamics

Before presenting the RLS-based mass estimator in Section 4.1, vehicle modeling should be reviewed. Suspension dynamics and longitudinal dynamics are commonly used for modeling purposes in mass estimation problems.

3.2.1 Suspension Dynamics

Vertical dynamics describe the motion of a vehicle in the vertical direction caused by external forces. The quarter-car suspension model is a simplified model representing one corner of the vehicle, including a mass (representing the vehicle body), a spring, and a damper. The suspension model shown in Figure 3.1 could be used to estimate the load of a trailer.

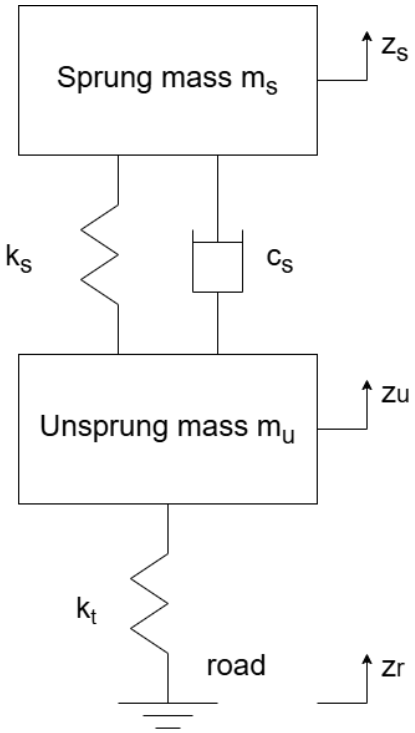

Figure 3.1 : Suspension Vehicle Dynamics

The mass of the parts that are supported by the suspension system is called sprung mass m_s while the mass of the components such as wheels and brakes is called unsprung mass m_u. The quarter-car suspension model equations are

$$m_s\ddot{z}_s + c_s(\dot{z}_s - \dot{z}_u) + k_s(z_s - z_u) = F_{\text{ext}}, \tag{3.32}$$

$$m_u\ddot{z}_u + c_s(\dot{z}_u - \dot{z}_s) + k_s(z_u - z_s) + k_t(z_u - z_r) = 0, \tag{3.33}$$

where the stiffness of the spring between the sprung and unsprung masses is shown by k_s whereas the stiffness of the tire (modeled as a spring) is k_t. Devices that dissipate kinetic energy and control the motion of the springs, providing a smoother ride and improved handling, are dampers or shock absorbers. The damping coefficient of the shock absorbers is represented by c_s. The vertical displacement of sprung mass, unsprung mass, and the road are represented by z_s, z_u and z_r respectively.

3.2.2 Longitudinal Dynamics

Next in this thesis, a mass estimator is designed based on the longitudinal dynamics recalled in this section.

The longitudinal vehicle motion can be described in terms of its longitudinal velocity that solves the differential equation derived from the second law of Newton in (3.34), whose terms of the right hand side are illustrated in Figure 3.2:

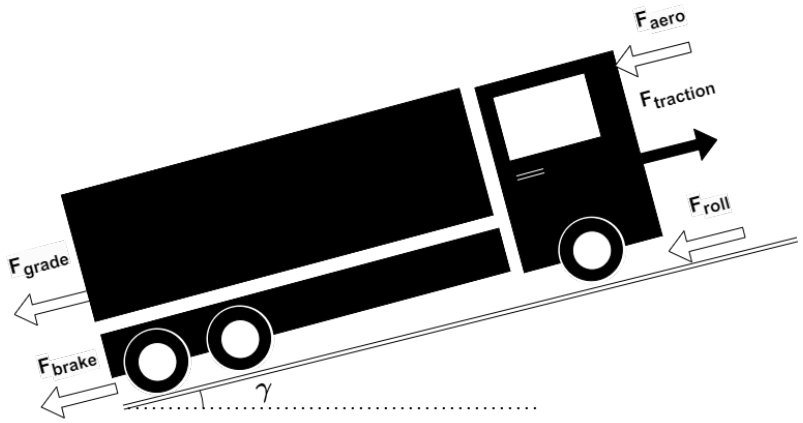

Figure 3.2 : Longitudinal Vehicle Dynamics

$$m\dot{v} = F_{traction} - F_{brake} - F_{aero} - F_{roll} - F_{grade}. \tag{3.34}$$

In (3.34), F_{aero} is the aerodynamic drag force, which is modeled as in (3.35):

$$\begin{aligned} F_{aero} &= 0.5\rho c_d A_f (v + v_{wind})^2, \\ &= f_{aero}(\rho, c_d, A_f, v, v_{wind}), \end{aligned} \tag{3.35}$$

where the symbol ρ represents the air density, the coefficient c_d is the aerodynamic drag, while A_f represents the frontal area of the truck. v is the velocity of the truck, whereas the wind speed component along the vehicle longitudinal direction is denoted as v_{wind} [85]. F_{brake} is the brake force which is not included in this study, as highlighted in [86]. Its inclusion in the longitudinal dynamics would overcomplicate the design of the mass estimator and introduce additional model uncertainties. The rolling resistance force, denoted as F_{roll}, is calculated as in (3.36):

$$\begin{aligned} F_{roll} &= \mu mg cos(\gamma), \\ &= f_{roll}(\mu, m, \gamma), \end{aligned} \tag{3.36}$$

where γ represents the slope of the road, μ denotes the coefficient of rolling resistance, m is the mass of the vehicle, and g is the acceleration due to gravity. The variable F_{grade} denotes the gravitational force, as in (3.37):

$$\begin{aligned} F_{grade} &= mg sin(\gamma), \\ &= f_{grade}(m, \gamma). \end{aligned} \tag{3.37}$$

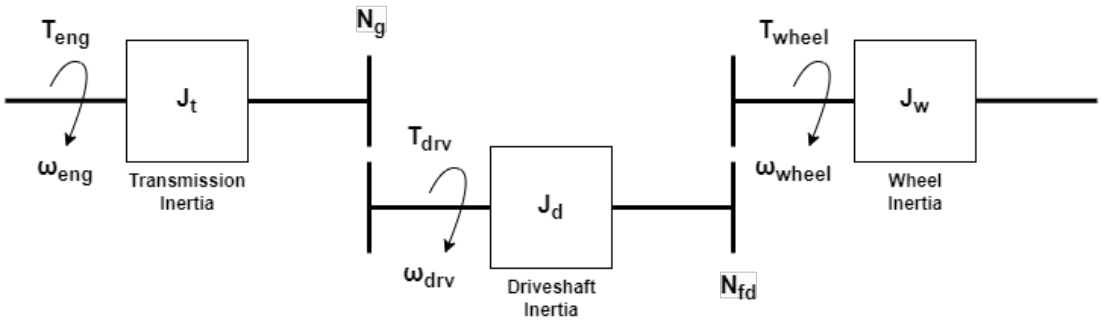

Figure 3.3 : Powertrain Inertia

In Figure 3.3, the vehicle powertrain is exemplified with its inertia highlighted. J_t is the equivalent inertia between the engine and gearbox, J_d is the equivalent inertia between the gearbox and differential, and J_w is the equivalent inertia between the differential and wheel. T_{eng} is the torque produced by the engine. The variable ω_{eng} is the rotational speed of the engine's crankshaft. N_g is the gear ratio whereas N_{fd} is used to denote the final drive ratio, and r_{wheel} is wheel radius. If stiffness and damping are ignored, N_g and N_{fd} can be written as:

$$N_g = \frac{\omega_{eng}}{\omega_{drv}}, N_{fd} = \frac{\omega_{drv}}{\omega_{wheel}}. \tag{3.38}$$

Inertial losses are calculated in (3.39):

$$\begin{aligned} J_t\dot{\omega}_{eng} &= T_{eng} - \frac{T_{drv}}{N_g}, \\ J_d\dot{\omega}_{drv} &= T_{drv} - \frac{T_{wheel}}{N_{fd}}, \\ J_w\dot{\omega}_{wheel} &= T_{wheel} - F_{traction}r_{wheel}. \end{aligned} \tag{3.39}$$

The equations in (3.39) can be combined resulting in

$$N_{fd}(N_g(T_{eng} - J_t\dot{\omega}_{eng}) - J_d\dot{\omega}_{drv}\dot{\omega}_{drv}) - J_w\dot{\omega}_{wheel} = F_{traction}r_{wheel}, \tag{3.40}$$

here $F_{traction}$ is the traction or propulsive force calculated in (3.41) from (3.40):

$$\begin{aligned} F_{traction} &= \frac{N_gN_{fd}(T_{eng} - J_{eq}\dot{\omega}_{eng})}{r_{wheel}}, \\ &= f_{traction}(T_{eng}, J_{eq}, \omega_{eng}, r_{wheel}, N_g, N_{fd}), \end{aligned} \tag{3.41}$$

where J_{eq} is calculated as (3.42):

$$J_{eq} = J_t + \frac{J_d}{N_g^2} + \frac{J_w}{N_g^2N_{fd}^2}. \tag{3.42}$$

From Equation (3.34), the mass of the truck m can be calculated as in (3.43):

$$\begin{aligned} m &= \frac{F_{traction} - F_{brake} - F_{aero}}{\dot{v} + \mu g cos(\gamma) + g sin(\gamma)}, \\ &= f(u(t), \xi). \end{aligned} \tag{3.43}$$

The input vector of the model (3.43) is $u(t) = (T_{eng},\ \omega_{eng},\ \gamma,\ v,\ \dot{v})$ which includes the data from the onboard sensors. The physical parameters of the truck model are represented by $\xi = (A_f,\ c_d,\ g,\ J_{eq},\ N_{fd},\ N_g,\ r_{wheel},\ \mu,\ \rho)$. The longitudinal dynamics model will be used in RLS by analyzing the relationship between the acceleration of the truck, engine torque, drag and rolling resistance forces to estimate the mass of the truck.

4. MIXED MODEL-BASED MASS ESTIMATION OF HEAVY-DUTY VEHICLES

4.1 Recursive Least Square (RLS) Mass Estimator

In this section, it is shown how the mass estimation algorithm can be built upon an RLS estimation algorithm as explained in Section 3.1 and an LSTM recalled in Section 2.1.3.2. This section explains how the vehicle longitudinal model in Section 3.2 is used to design the RLS mass estimator, while in Section 4.2 we show how an NN-based supervisor is designed to enable the mass update with the RLS algorithm.

From (3.43), the following model can be derived to obtain RLS inputs and output with respect to (3.2):

$$\begin{aligned} \phi &= F_{traction} - F_{aero}, \\ y &= \dot{v} + \mu g cos(\gamma) + g sin(\gamma), \\ \theta &= \frac{1}{m}. \end{aligned} \tag{4.1}$$

The regressor ϕ can be built based on the models (3.35) - (3.41) and the set of measurements made available by the onboard sensing and communication devices. Meanwhile, y is obtained based on (3.34) by collecting mass-related terms on one side of the equation.

- The *traction force* $F_{traction}$ can be calculated as in (3.41), where the engine torque T_{eng} is typically available from the Engine Control Unit along with the engine speed ω_{eng}. The current gear g_r is also available from the vehicle CAN bus. The equivalent inertia J_{eq}, the final gear ratio g_{final}, and the nominal wheel radius r_{wheel} are available from the vehicle design.

- The *aerodynamic drag force* F_{aero} can be calculated as in (3.35), where air density ρ can be estimated by using look-up tables and maps based on onboard sensors such as temperature, pressure, and mass air flow sensors. Aerodynamic drag

coefficient c_d is obtained by wind tunnel testing or determined by computational fluid dynamics (CFD) simulations. The frontal area of the truck A_f is available from the HDV design. Velocity v can be obtained by onboard sensors such as wheel speed sensors or GPS systems. Although wind speed and direction are measured by the wind vane and anemometer on weather stations and could be made available to the vehicle through I2V communication, they are not known at road level.

- Since the *rolling resistance force* F_{roll} and *road grade force* F_{grade} are directly dependent on the mass of the truck, they are not included in the regressor. Instead, their effect on the acceleration is added to y. The friction coefficient μ can be obtained by testing or can be determined by using established data such as from tire manufacturers or research studies. Inclinometers or tilt sensors or GPS-based elevation data can be used to obtain road gradient angles θ.

After the regressor ϕ and the output y are computed, a sliding window RLS estimator can be assembled as explained in Section 3.1.3. By trial and error, an RLS estimator is designed with a 2 min finite window length and 0.01 s sampling time.

4.2 Long-Short Term Memory (LSTM) Supervisor

In this section, we explain how an LSTM-based supervisor can be designed and trained that enables an RLS mass estimation with a desired accuracy. The idea proposed here is to train the supervisor to recognize the vehicle operation so that the RLS achieves the desired accuracy.

Determining the reliability of an RLS estimator is a binary classification problem. That is, the supervisor must classify the RLS input leading to the desired estimation accuracy. Since the data is time-based and the estimation is done in each time interval, this problem is a sequence-to-sequence classification problem.

In order to check whether the RLS mass estimator is accurate enough or not, a two-layered LSTM network is designed. Vehicle speed, engine speed, engine torque, longitudinal acceleration, and estimated mass are the inputs of the LSTM as shown in Figure 4.1. Labeled outputs are binary: 0 if mass estimation is not as accurate as desired and 1 if it is. The five inputs are normalized with min-max normalization. The input sequence length is chosen as 120 seconds with a 0.01 sampling time, which is the

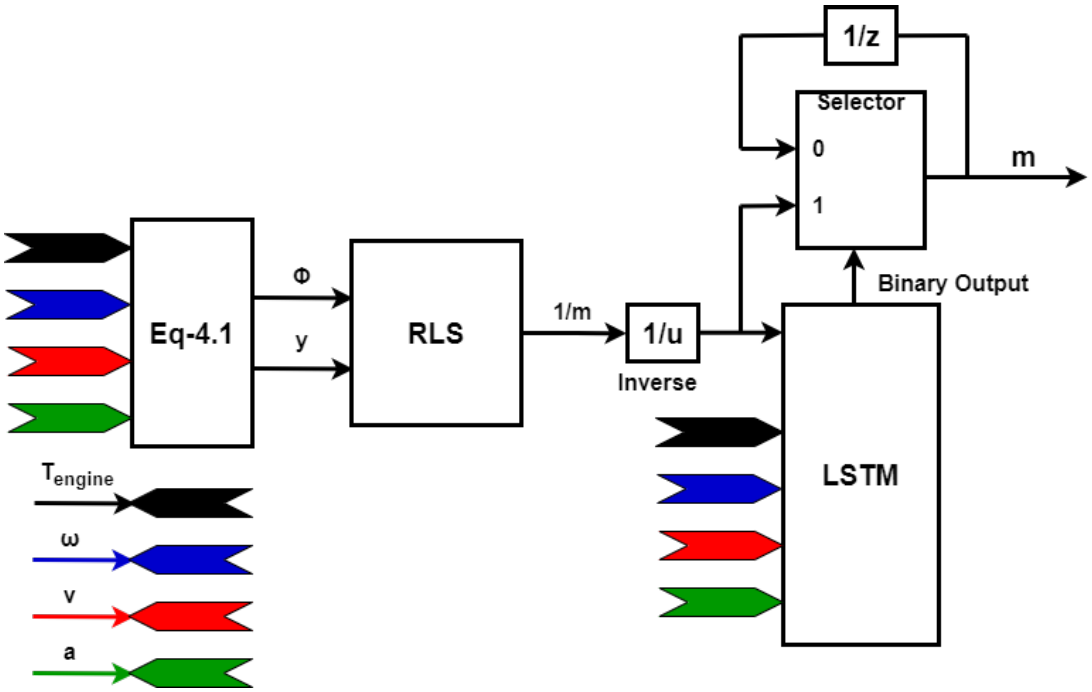

Figure 4.1 : Mixed-Model and Learning-Based Mass Estimator

finite horizon window of RLS. Therefore, each data sequence consists of a 5x12000 array. As shown in (2.12)-(2.16), the values of gates, the hidden and cell states are calculated at every time instant.

8 LSTM cells are used in both hidden layers. Outputs of the first layer are normalized by layer normalization. Layer normalization is a technique to decrease the training time by computing the mean and standard deviation of the hidden states in a layer and scaling and shifting the hidden states based on this computation [87]. Then, dropout with 0.2 probability is used to avoid overfitting, as seen in Figure 4.2. After the second LSTM layer, a dense layer that connects all hidden cells in the second layer with the output at each time step is used to classify the accuracy of the RLS mass estimator. The sigmoid activation function can be used in the output layer to determine the reliability of the RLS mass estimator.

Since the sequence length is 2 min, a 400 min dataset, built as explained in Section 4.3, is split into 200 instances. 80% of the data is allocated for training, while 10% of the data is allocated for validation, and 10% of the data is allocated for testing. Since imbalanced data decreases the accuracy of the network, training and validation have similar amounts of zeros and ones in the output.

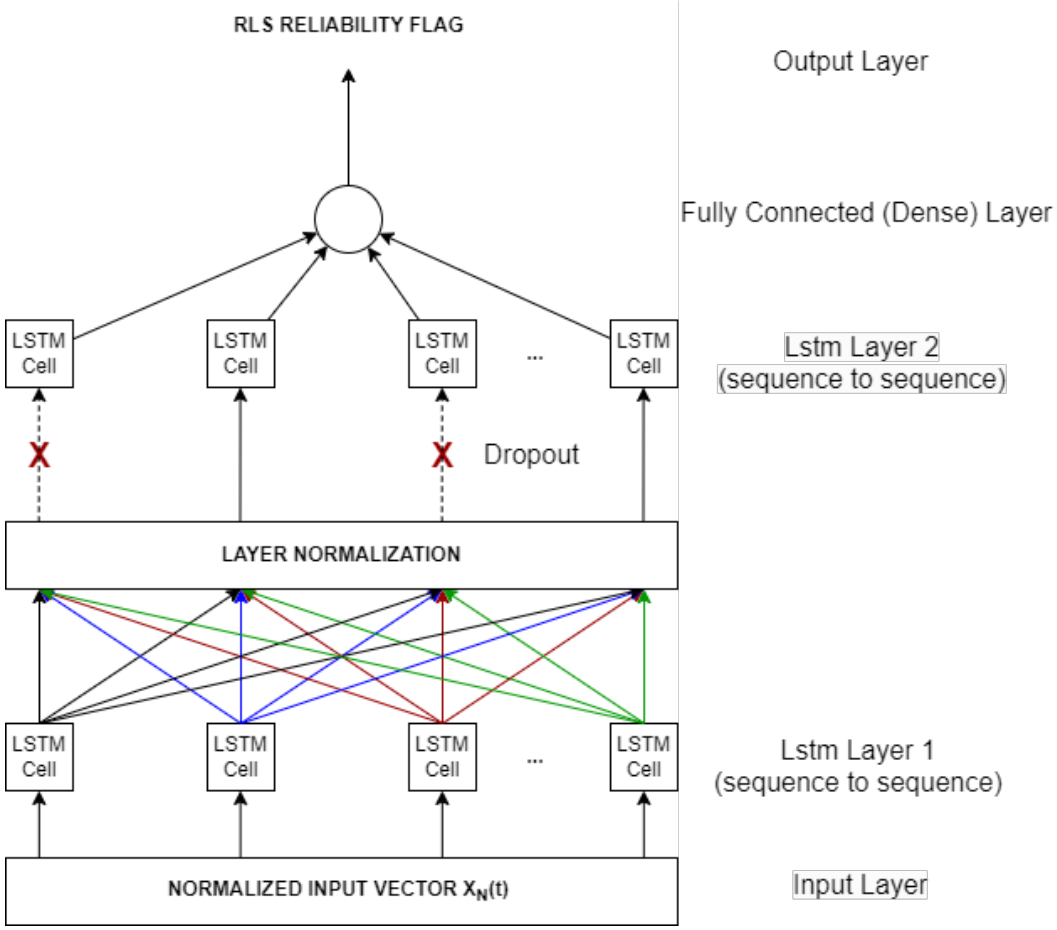

Figure 4.2 : RLS Supervisor

The epoch number is chosen as 500 in order to adequately train the network. ADAM optimizer is used and the initial learning rate is chosen as 0.002. The learning rate decays 1% every 100 epochs. The batch size is chosen as 160 in order to train the entire training dataset at once, so there are 500 iterations.

4.3 Synthetic Data Generation

The TruckMaker simulation software is used to generate synthetic data. Since excitation is necessary in order for RLS to estimate the vehicle mass accurately, the accelerator pedal is pressed and released frequently. A truck model with a fixed load is chosen to execute the designed maneuvers. Data about engine torque, engine speed, longitudinal vehicle speed and acceleration, and accelerator pedal position percentage (APP %) is collected as shown in Figure 4.3 and the available parameters in the longitudinal dynamics model in Section 3.2.2 are taken from Truckmaker 11.0.

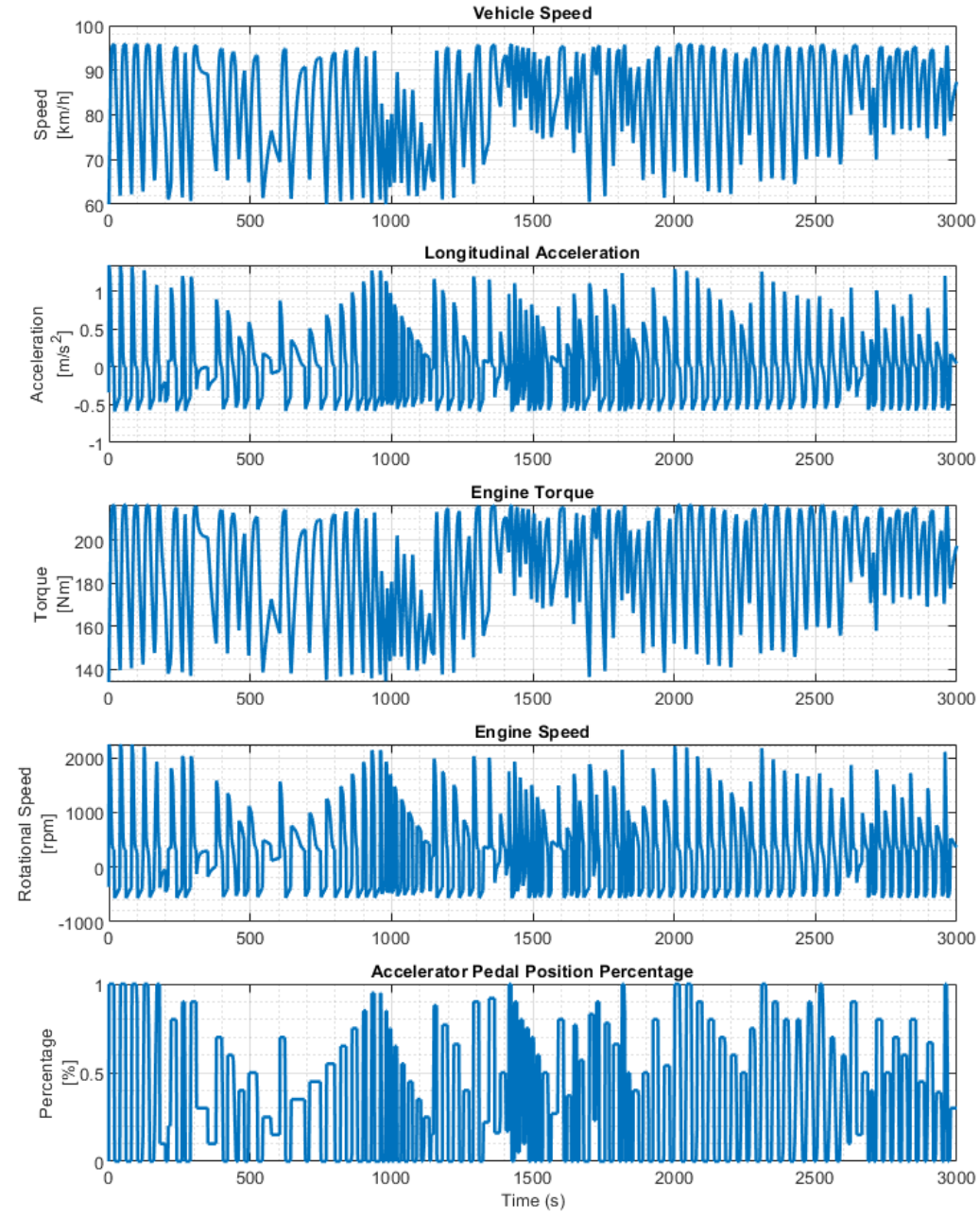

Figure 4.3 : Synthetic Data

If real truck data were to be used, the data collection process would be as follows: The engine torque is not directly measured but estimated while the engine speed is directly measured from a sensor on the crankshaft. The vehicle speed and acceleration can be obtained by fusioning GPS and IMU with wheel speed sensors and accelerometers. Since not all trucks have inclinometers, road gradient might be estimated from altitude changes of a specific location due to GPS. However, GPS signals might be lost in tunnels or might give false information in a tunnel inside a mountain. Digital elevation models based on navigation could be used. APP can be measured by a potentiometer

whereas a temperature sensor measures temperature. Some trucks have mass airflow (MAF) sensors and manifold absolute pressure (MAP) sensors. Since the change in altitude affects the air density, air density can be estimated from a combination of ECU signals and GPS altitude data. In this study, air density is assumed to be constant.

It should be highlighted that the LSTM supervisor gets more complex if higher availability is needed. In this case, more data is necessary, and the training is longer and computationally expensive. In order to balance the availability of the vehicle mass estimate with the desired accuracy and the complexity of the LSTM-based supervisor and its training process, some restrictions are introduced. Data is collected based on the following restrictions.

- *Straight driving*. Data is generated while driving on a straight road. This allows using only longitudinal vehicle dynamics in the design of the RLS mass estimator.
- *Flat roads.* Downhill or uphill scenarios are not included, such that road grade information is not necessary. Hence, such restrictions could be removed if road grade is available, for example, from map data and GPS.
- *Engaged clutch*. Data is generated while the clutch is fully engaged. This restriction removes from the data logs the transients due to gear shifting that would require an LSTM supervisor with higher complexity. Otherwise, the RLS mass estimator would be available for larger intervals, but estimation accuracy would diminish.
- *Fixed gear.* For a similar reason as in the previous bullet, data is collected with a fixed gear. In this case, the highest gear is chosen, thus enabling a mass update once the vehicle drives on the highway.
- *Limited speed*. For the same reason explained in the previous bullet, data is logged in the interval [60,95] kph.
- *No braking*. RLS mass estimator is not used during braking since braking dynamics are much more complex and less accurate than traction dynamics. Indeed, retarder significantly complicates the braking system modeling.
- *No wind.* Wind effect is not included in order to have a more accurate model. For simulation purposes, it could be added but in reality, it is not known how to get wind information on road level.

A 5-ton load is added to the truck, which has 7 tons of unladen mass. 8 simulation datasets are collected with 3000 s simulation time with a 100 Hz sampling frequency. Therefore, a total of 400 min of data are collected.

4.4 Accuracy Analysis

The LSTM-based accuracy classification algorithm for the RLS mass estimator, designed as explained in Section 4.2, is implemented and trained using MatLab R2022a with the Deep Learning toolbox, using both training and validation datasets. A NVIDIA A100 Tensor Core GPU with 40 GB memory is used to execute the training task. The target accuracy of the RLS mass estimator is 97%, so the estimation error should not be larger than 3%.

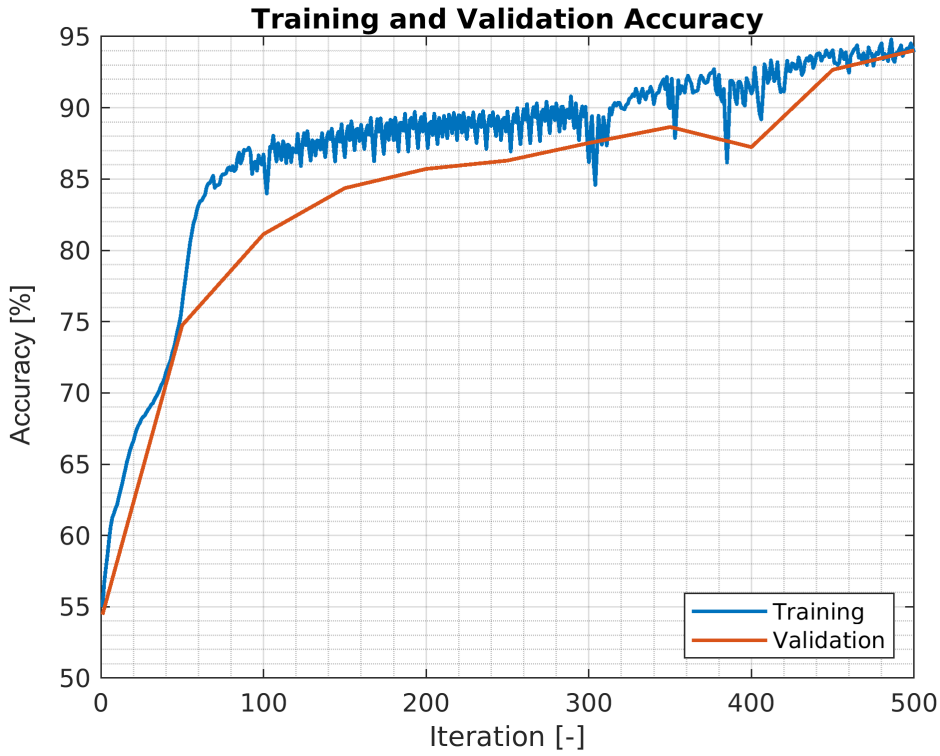

Figure 4.4 : Training and Validation Accuracy

160 training dataset and 20 validation dataset instances with each 2 min sequence are used for training. The accuracy resulting from the training and validation process is shown in Figure 4.4. The figure shows that there is no underfitting or overfitting since both training and validation accuracy are more than 90%, so the learning process is successful.

After the weights and biases are obtained which leads to the best validation accuracy, the accuracy of the LSTM-supervised RLS is measured on the test dataset. Results from 5 out of 20 tests are shown in Figure 4.5.

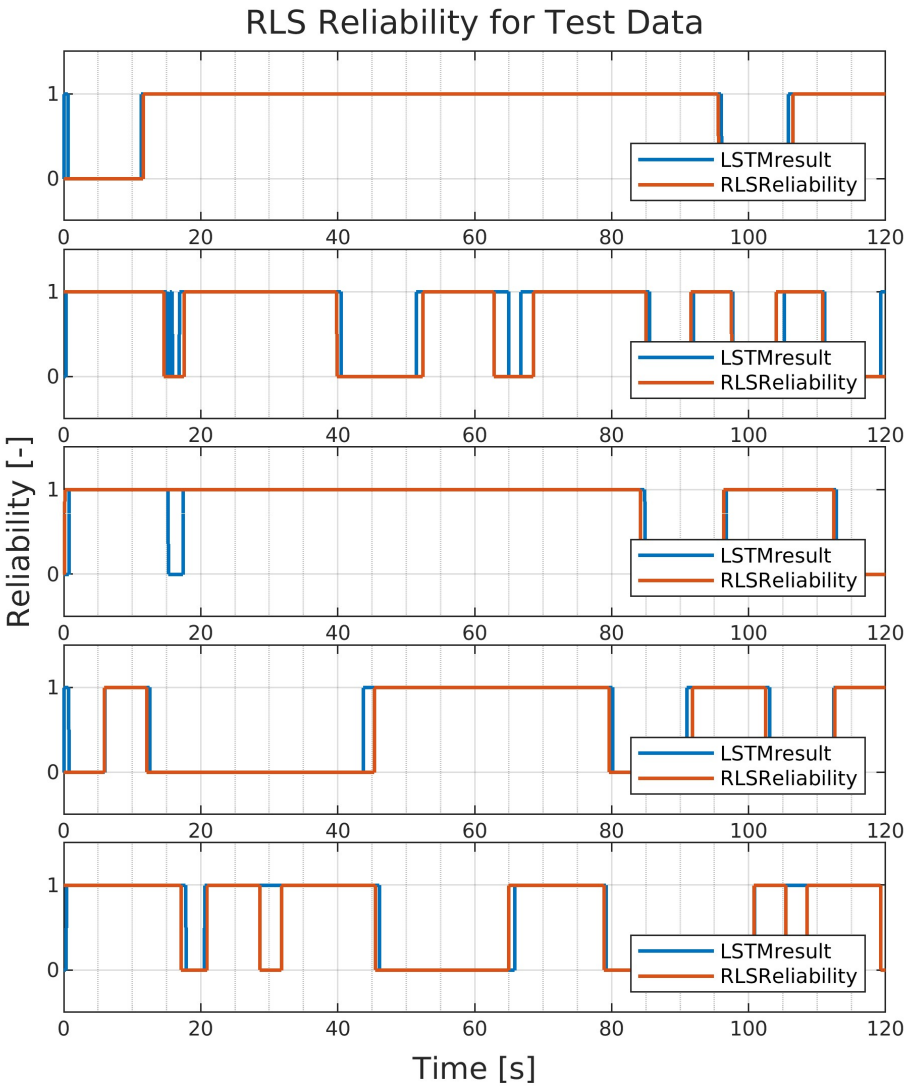

Figure 4.5 : RLS Reliability on 5 out of 20 Synthetic Test Data

The accuracy of the LSTM on the entire test dataset is 93.35%. In other words, the RLS is accurate in 93.35% of the test time. However, accuracy may not be the only evaluation criterion in this case. Indeed, while in case of a false negative, the estimator can use the previous mass estimate. However, LSTM should not predict false positives. Otherwise, the mass estimate is updated inaccurately. Therefore, precision metric as written in Section 2.2.3 might be preferred for this problem. The precision of the entire dataset is calculated as 92.04%.

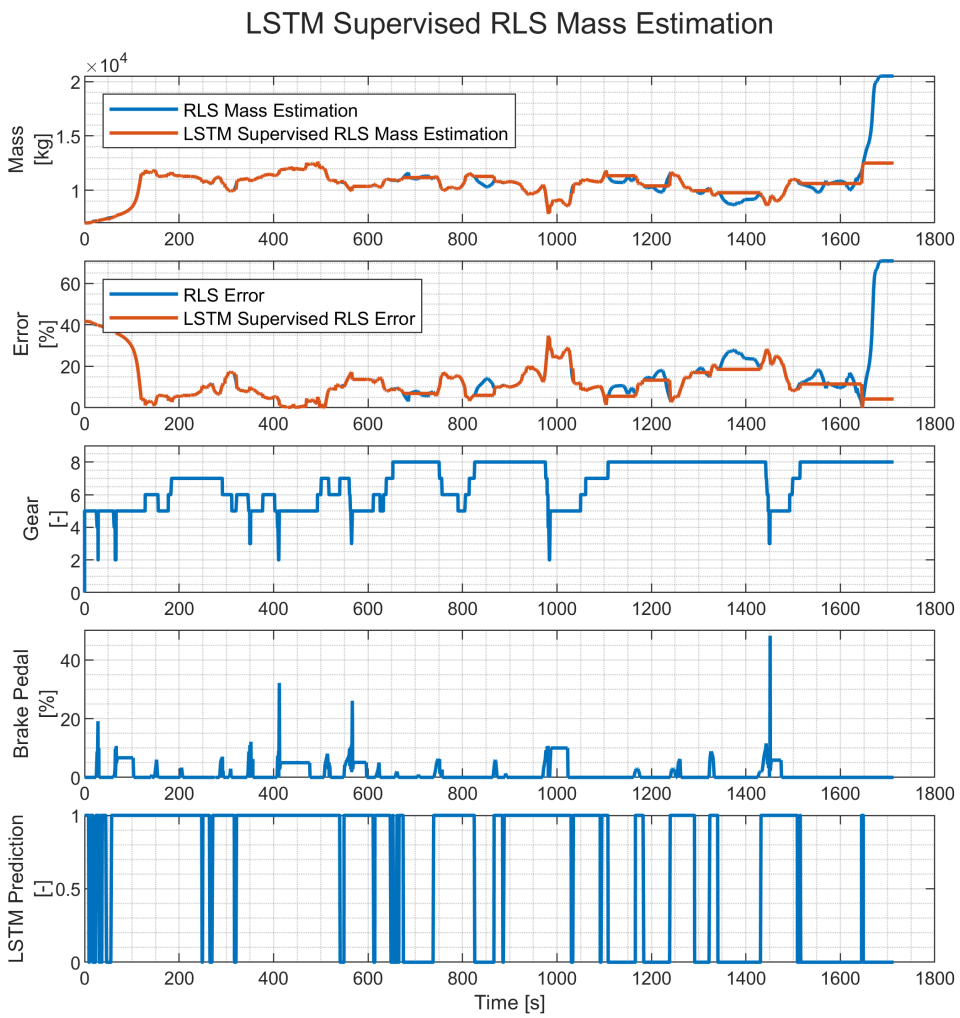

Figure 4.6 : RLS Estimate with WLTC Data

In Figure 4.6, the WLTC class 2 driving cycle is shown, which is used for testing purpose with a truck whose mass is 12 tons. The estimation error of the RLS with LSTM supervision is as good as 5%. Braking increases the error because braking is not included in the training data. Also, larger errors are attained at gears different from the highest gear engaged since training data is collected only for the highest gear. Normally, the supervisor is disabled during braking or for different gears. However, it is activated to show the importance of the quality of the training data. As seen in the figure, LSTM supervision has no contribution to decreasing the mass error when the gear is not engaged to the highest gear. However, LSTM-supervised RLS performs better at the highest gear. The reason for false positives might be because of braking in 120 s intervals.

5. LEARNING-BASED MASS ESTIMATION OF HEAVY-DUTY VEHICLES

In this chapter, a learning-based mass estimator is designed for both trucks and city buses. The accuracy of the truck mass estimator is analyzed with respect to the mass variations. Such an analysis is meant to verify the estimator capability of operating in wide ranges of payload masses. The mass estimator is designed for city buses with wider availability (i.e., fewer restrictions than in Section 5.2 are required for buses). Indeed, the most significant mass variation in buses is due to passengers. Hence, the mass estimator should be able to quickly converge to the correct mass estimate. The bus data includes the wind effect, the road gradients, and the engagement of all gears.

5.1 LSTM-Based Mass Estimation

The mass estimation of heavy-duty vehicles is a regression problem in which the mass is calculated as a function of, among others, engine torque and speed, vehicle speed and acceleration, and other physical quantities, as described in (3.34).

This research utilizes an LSTM network to approximate such a function. The choice of an LSTM network is due to its superior training capabilities in comparison to ordinary RNNs. The LSTM network used in this manuscript is shown in Figure 5.1. This consists of two hidden layers, with each layer containing 16 LSTM cells. Every cell iteratively handles its hidden state and cell state signals $h_{t-i},\ c_{t-i},\ i = 0.01,\ 0.02,\ \ldots,\ 10$. Dashed squares in Figure 5.1 show recurrent connections from previous time steps, and the rightmost solid squares depict the LSTM cells. The LSTM cell processes the input vector $\mathbf{x} = [v, \dot{v}, T_{eng}, \omega_{eng}, APP\%]$: including vehicle speed, longitudinal acceleration, engine torque, engine speed, and accelerator pedal position as shown with vertical black and blue arrows. The previous outputs $h_{t-i},\ c_{t-i}$, are highlighted with horizontal black arrows. The hidden states $h_t^{[1]}$ from each cell in the first layer serve as the inputs for all cells in the second layer $x_t^{[2]}$ as shown with black arrows between LSTM layers in the rightmost part of the figure. The superscripts indicate the hidden layer number. The two levels are of

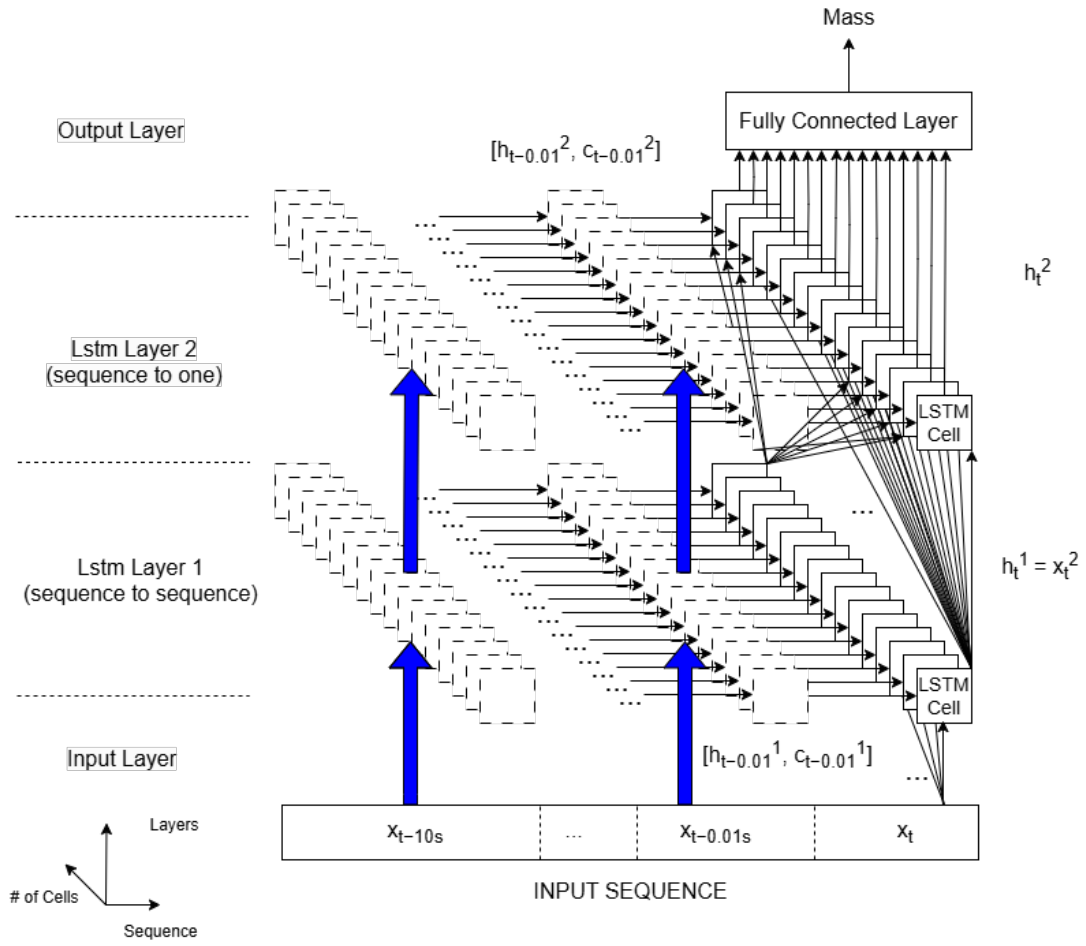

Figure 5.1 : LSTM-based Mass Estimator

sequence-to-sequence and *sequence-to-one* types, respectively. That is, the first layer outputs a sequence from a sequence, while the second layer produces a single output, which is the mass of the truck from the sequence obtained from the first LSTM layer. The hidden state and cell state information is calculated at every time step in the second layer, according to the equations in Equations (2.15) and (2.16). The mass is estimated by the use of a fully connected layer, which establishes a connection between the output mass and all the hidden states $h_t^{[2]}$ of the 16 LSTM cells in the second layer at the final time step. The network has more than 3000 weights and biases. LSTM networks often need a lower number of parameters for training compared to DNNs. Indeed, in deep neural networks (DNNs), the prior values of the features serve as supplementary inputs that need extra weights and biases. For instance, the DNN presented in [17] for mass estimation consists of almost 90000 parameters.

The trial-and-error approach is guided by the assessment of the network using the validation dataset, as described in Section 2.2.8. It is important to note that using simpler networks, such as those with just one layer or two layers with fewer LSTM cells per layer, results in *underfitting*. This underfitting leads to a poorer accuracy of the estimate during the training phase compared to the desired accuracy target. Conversely, the use of more complex LSTM networks has resulted in overfitting. Specifically, a portion of the network's structure is used to fit the measurement noise. Hence, overfitting results in inaccurate mass estimation as predicting a random quantity (the noise) over fresh data just increases the estimation error.

Designing an LSTM network consists of defining a set of *hyperparameters* that affect the training process, hence the performance of the network. The hyperparameters and their corresponding values, for the network designed in this thesis work, are shown in Table 5.1.

Table 5.1 : Hyperparameters

Hyperparameters	**Content**
Hidden layer number	2
Neuron number in the hidden layers	16, 16
LSTM layers output mode	sequence, one
Dropout	50%
Optimizer	Adam [61]
Computational resource	Gpu
Epoch number	5000
Initial learning rate	0.0025
Sequence length	longest or shortest
Output network	best-validation-loss
Gradient threshold	1

Since simulation time is fixed for training and validation datasets, the sequence length option can be either the longest or the shortest. If the sequence lengths were not equal, sequence padding would be necessary for the longest option, whereas sequence truncation would be needed for the shortest option. The drawback of sequence padding is that it increases memory usage and decreases the accuracy of the network. On the other hand, truncation based on the shortest length may cause valuable information loss.

This research does not impose any restrictions on validation patience or early stopping, and the training procedure continues uninterrupted until its completion. The set of

calculated weights and biases that minimize the error on the validation dataset is preserved, while all other sets of weights and biases from other iterations of the training dataset are erased after the training process is completed.

5.2 Data Collection

In order to generate the synthetic data, a highly accurate simulation tool called TruckMaker is used. After a truck model is chosen, various maneuvers are executed by adjusting the gear number, brake pedal, clutch, and accelerator pedal positions. Also, the initial gear number and initial vehicle speed are chosen not to start simulations from the standstill position. The data features extracted in this research are engine torque, engine speed, longitudinal vehicle acceleration, vehicle speed, and accelerator pedal position.

The training and validation data sets are constructed by extracting data samples from the created synthetic data based on the following criteria.

- *Engaged clutch.* Data samples are chosen only while the clutch is engaged. The purpose of this criteria is to exclude from the training dataset any samples that include transients, related to clutch engagement/disengagement, that require additional parameters for a satisfactory fit but are not easily available and also not necessary for accurately estimating the vehicle mass unless a significant mass change happens during this small time interval (very unlikely).

- *Fixed gear.* The training dataset consists of data samples gathered from the same gear. Including the whole gear range in the training would require extra parameters, without enhancing the accuracy of the mass estimate, but rather its availability. Therefore, the highest gear is selected.

- *Restricted speed.* Training and validation data sets are collected for vehicle speeds within the range of 60 to 95 kph. Similar to the previous criteria, this constraint is implemented to prevent the need for extra factors to account for speed-related phenomena, such as wind resistance, over a broad range of vehicle speeds while maintaining the same level of accuracy in estimation.

- *No braking.* Braking maneuvers are not part of the training dataset because the relation between actual braking force and brake pedal position can be complex to model and require additional parameters in the network. For example, the braking force generated by a brake pedal input might vary dramatically as a consequence of the heating of the brake pads. Also, information related to retarders is necessary.

- *Straight driving.* Measurements are gathered during linear driving, focusing only on motion in the direction of travel.

- *Flat roads.* There is no inclination to avoid complicating the network by introducing road grade as an additional feature.

- *Without wind.* The wind influence is disregarded due to the absence of identified wind data at the road level in the real world. Also, the simulator does not have this conversion, the wind velocity is neglected, so only the speed of the vehicle affects the drag force.

The training data for the LSTM-based mass estimator consists of samples gathered from a chosen truck with masses ranging from 6.8 to 7.8 tons. The increments in curb mass are 125 kg, so there are 9 various masses in this interval. Once a 1-ton load is added to the unladen HDV, the process is repeated with a load ranging from 0 to 2 tons, increasing by 250 kg each time. This is followed by a burden ranging from 0 to 3 tons, increasing by 375 kg each time. Finally, a load ranging from 0 to 4 tons is applied, increasing by 500 kg each time. With such mass ranges, we have investigated the capability of a fixed-complexity network of estimating the vehicle mass with a target accuracy, over increasing mass ranges.

All maneuvers last 10 seconds. For each simulation generating a maneuver, one of the four different velocities is chosen as the beginning velocity, where the minimum initial velocity is 60 kph since only gear eight is used for simulations. Initial velocity is increased by 10 kph until it reaches 90 kph. Two starting accelerator pedal position percentages, not pressed (%0) or fully pressed (%100), are selected for each initial speed. A total of 55 maneuvers were created for these eight combinations of the initial vehicle speed and accelerator pedal position. There are 11 final pedal position percentages chosen, ranging from 0% to 100%, with increments of 10%. Five different time intervals are defined as 0.5, 1, 2, 5, 10 seconds, which are the required times to

attain these percentages. Figure 5.2 shows, as an example, one of the 55 maneuvers, characterized by a 60 kph initial velocity, 0% initial accelerator pedal percentage, and 90% end accelerator pedal percentage, all occurring within a period of 0.5 seconds. These combinations provide a total of 3960 distinct simulations corresponding to 11 h of driving time in total for each load interval, such as 0–1 ton for a selected HDV. The training data set comprises 3465 simulations, while the validation data set is built on the remaining simulations.

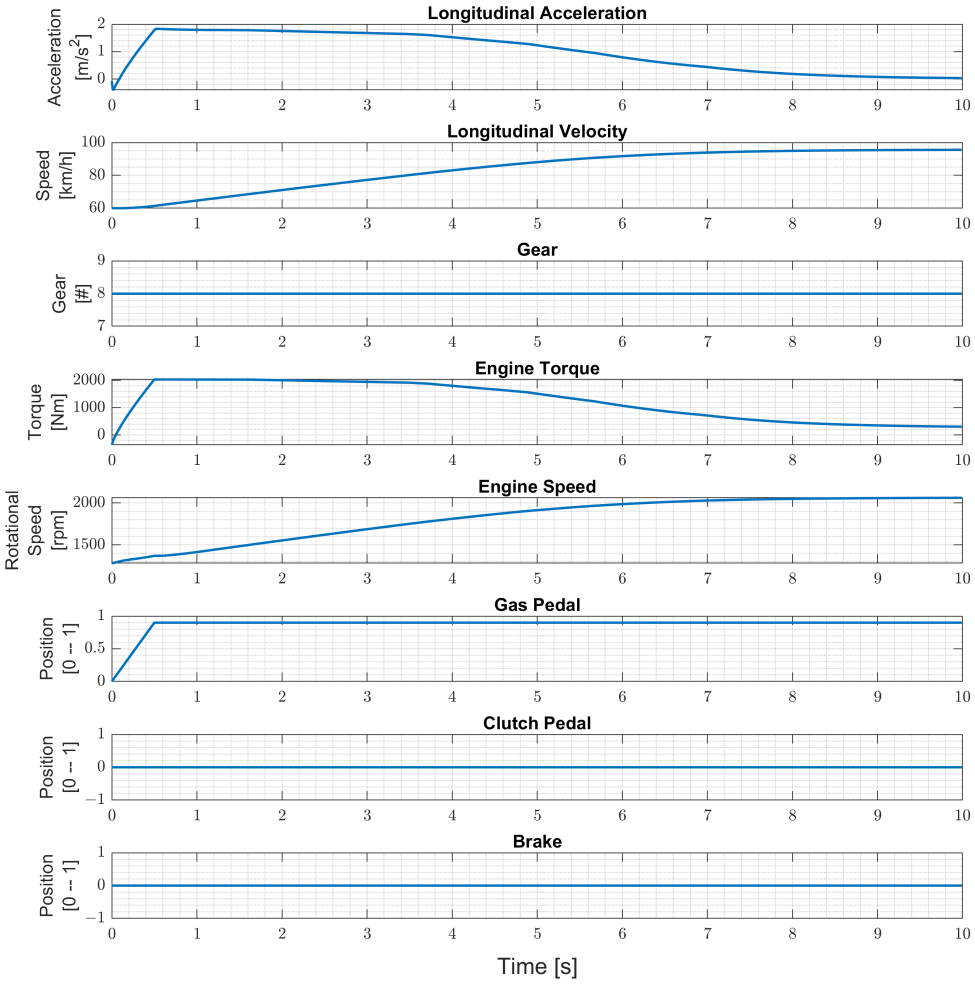

Figure 5.2 : Synthetic Data for Training

The test dataset is also created by using TruckMaker. Worldwide harmonized Light Vehicles Test Cycle (WLTC) class 2, except for speeds that exceed the maximum velocity of the HDV is used as the speed profile. WLTC is preferred over the New European Driving Cycle (NEDC) because of its dynamic nature and longer overall distance and cycle duration compared to the NEDC. Since data is collected based on

longitudinal dynamics, a straight road profile is chosen. Since the training dataset does not have various road gradients, the test dataset is also collected on flat roads. Different from the training dataset, the test dataset is collected by an increment of 200 kg of load starting from the unladen HDV until the maximum load reaches 4 tons. The same mass range of training data is used for test data. Specifically, while the WLTC driving cycle is designed for passenger vehicles, WLTC is modified in this thesis by excluding any cycle segments that exceed the maximum velocity of trucks and combining the remaining segments. The acquired model is only evaluated in the topmost gear. Consequently, the time sequence of features between 1520 seconds and 1580 seconds is used as the test dataset, with intervals of 10 seconds. Consequently, the first three tests only include acceleration. Test four consists of both acceleration and deceleration segments, test five may be considered a constant speed test; and test six is a slow deceleration test, as seen in Figure 5.3.

In summary, the training and validation data sets are gathered by considering different accelerated pedal positions and rates of excitation, while the test dataset is obtained using the realistic driving cycle WLTC.

The training data set, as explained in this work, necessitates the execution of a series of acceleration maneuvers. If the data logs of a testing vehicle are accessible, those maneuvers may be extracted from the data logs. Alternatively, simulation models may be used, as shown in this research. An alternative approach involves merging the data logs obtained from the actual vehicle with simulation data, which presents a feasible opportunity for lowering the expenses associated with training. However, it needs to be highlighted that merging these two approaches can only be successful if the simulation model is accurate and gives realistic results.

5.3 Simulation Results and Discussion

The learning-based mass estimation algorithm is trained using the data sets explained in Section 5.2. The training was conducted, as explained in Section 5.1 using the MatLab R2022a Deep Learning toolbox with an NVIDIA A100 Tensor Core 80 GB GPU.

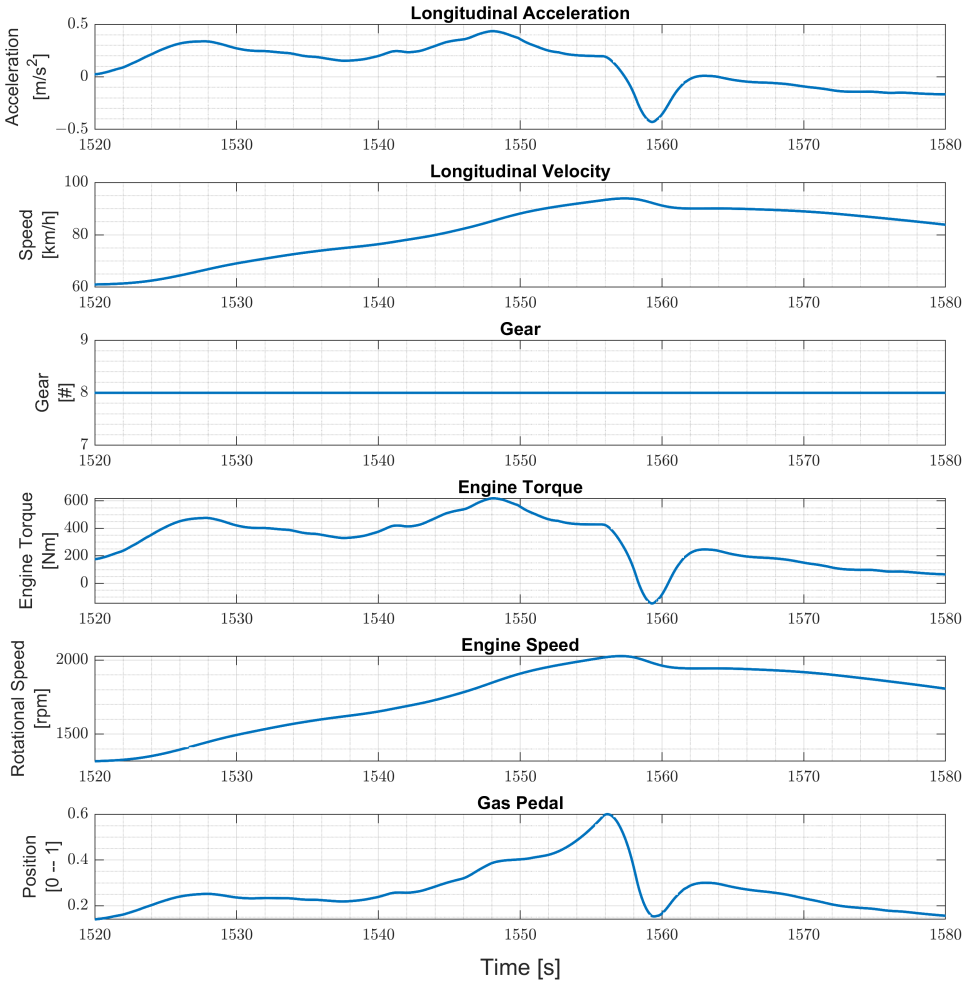

Figure 5.3 : Test Dataset Relying on WLTC

Specifically, four networks denoted as N_i, where i belongs to the set $\{1,2,3,4\}$, are trained using four distinct training data sets. Every data set comprises 11 hours of driving data, acquired by adjusting the vehicle mass within the ranges $R_i = [6.8, 6.8+i]$. These training data sets were specifically collected for a selected truck. Each network denoted as N_i, is trained to estimate the mass of the vehicle within a certain range of masses, denoted as R_i. The four networks are subsequently assessed using a test dataset from the same truck, and the outcomes are shown in Figures 5.4–5.7 correspondingly. In order to more accurately assess the effectiveness of the four networks, the test data set has been divided into 10-second time intervals, denoted as T_j, where j ranges from 1 to 6 because between 1520 s and 1580 s time intervals of WLTC data is used. The estimated mass and the error in the mass of each network, N_i, are presented for each time interval, T_j, using vehicle masses

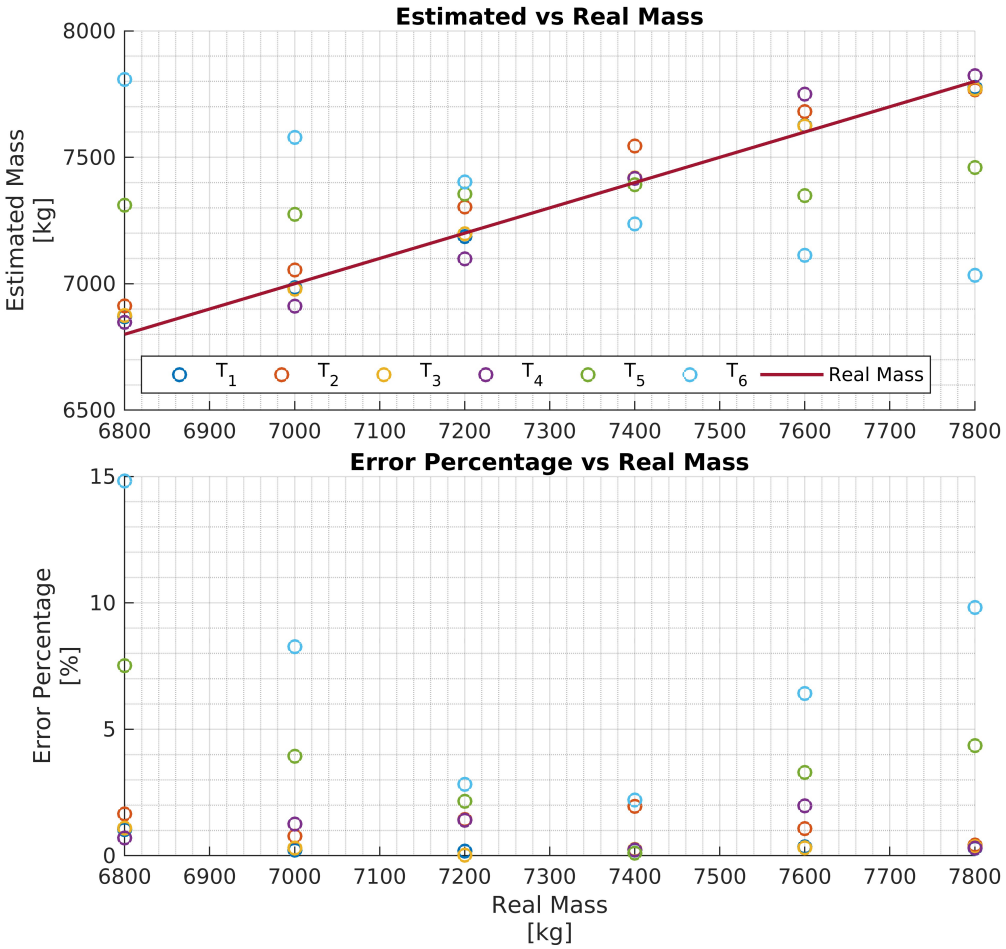

Figure 5.4 : N_1 Performance on Test Dataset in R_1

within the specified range, R_i. For instance, Figure 5.4 depicts the assessment of network N_1 that was trained using the vehicle mass in the range $R_1 = [6.8,\ 7.8]$ tons. The six distinct colors represent the mistakes in estimating mass throughout the time intervals $T_j,\ j \in \{1,2,\ldots,6\}$. The error between actual and estimated mass is noticeably reduced in the time intervals T_1, T_2, and T_3 of the test dataset, which correspond to acceleration maneuvers. Conversely, the estimation error is high in T_5, and T_6 when the accelerator pedal, as shown at the bottom subplot in Figure 5.3, is released and the engine brake occurs.

While it is possible to collect more training datasets or increase the sequence length of current datasets to add brake maneuvers, doing so may lead to a fall in the accuracy of the estimate during the acceleration phase, which is not necessary. It is possible to extract a straightforward logic to activate the network and get an accurate mass estimate by analyzing the distribution of the estimation error regarding the longitudinal

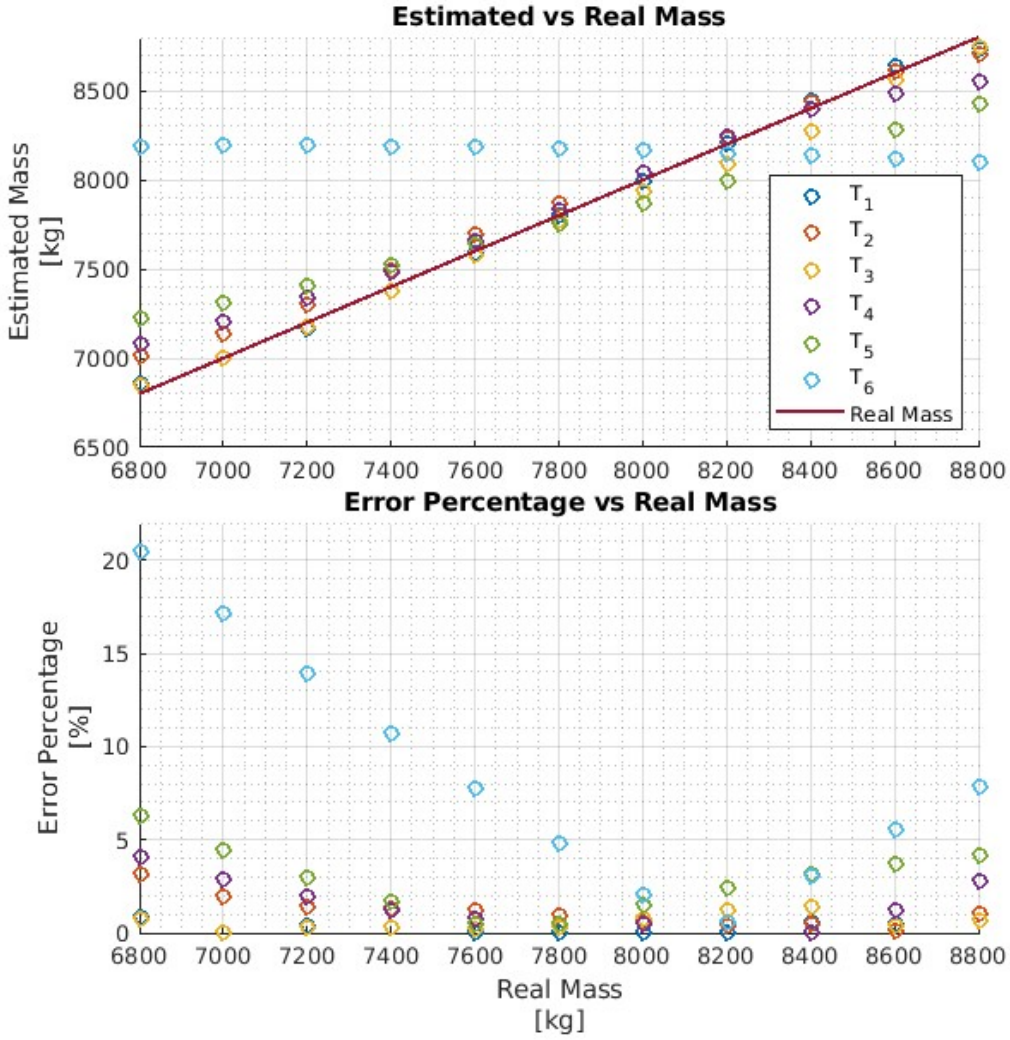

Figure 5.5 : N_2 Performance on Test Dataset in R_2

acceleration and/or accelerator pedal signals. Figures 5.5–5.7 provide the assessment of networks N_2, N_3, and N_4 using the same driving cycle, which results similar to Figure 5.3. It is important to note that these networks, in comparison to network N_1, are trained using broader mass ranges of 2, 3, and 4 tons, respectively, instead of 1 ton.

As anticipated, mass estimation errors in the networks increase with the mass range. However, while all networks lead to an estimate error below 5% in T_1, T_2, T_3, and also in T_4 since acceleration occurs most of the time in this interval, the estimation error noticeably rises in the remaining portion of the test data set. The networks properly estimate the mass during acceleration maneuvers, just as they do for vehicles with broader ranges of mass. Accurate estimation of mass is not achieved at constant speed due to the absence of any stimulation. One possible explanation for the significant estimate error during deceleration might be because the training data is generated using

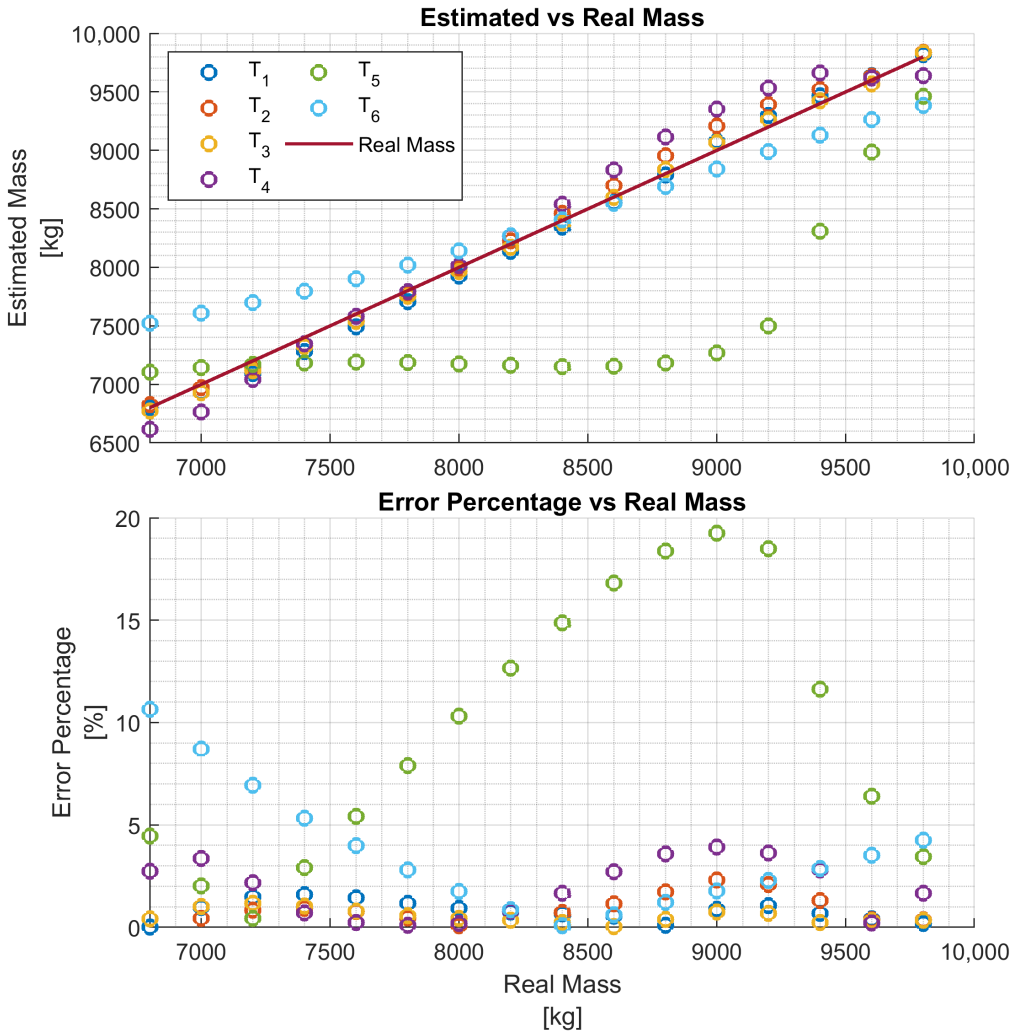

Figure 5.6 : N_3 Performance on Test Dataset in R_3

various percentages of accelerator pedal position, and the HDV begins to accelerate when the pedal is pressed at approximately 13% when it is unladen. Thus, the available data is sufficiently comprehensive to capture acceleration dynamics, while there is insufficient training data to learn about deceleration logic.

5.4 Data Generation For Various Environmental Factors

Synthetic data is generated by choosing a city bus with five gears in TruckMaker 11.0. Engine torque, engine speed, longitudinal vehicle acceleration, vehicle speed, brake pedal position, road elevation, and wind speed are the data features. Data is generated based on straight driving, so measurements are gathered when there is no steering.

The training data for the LSTM-based mass estimator consists of samples gathered from a chosen bus with masses ranging from 14 to 18 tons. The increments in curb

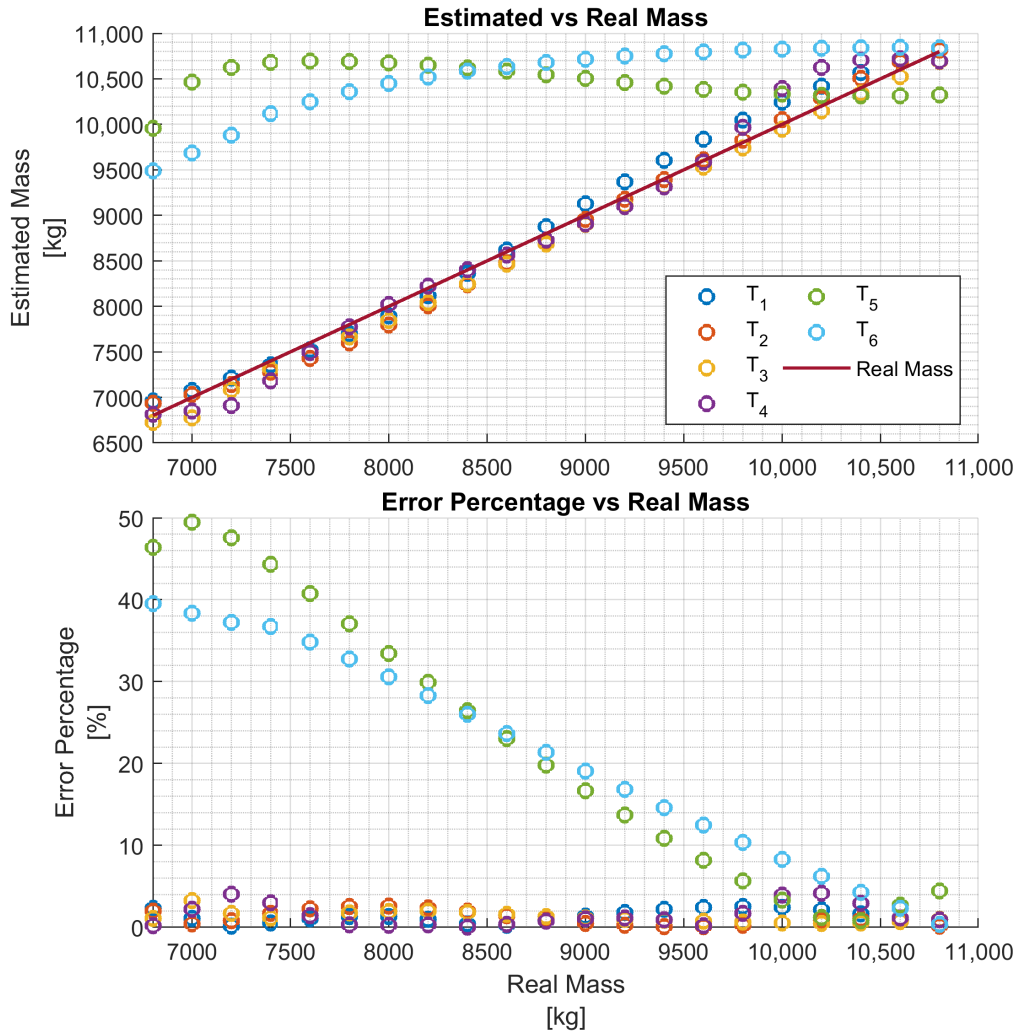

Figure 5.7 : N_4 Performance on Test Dataset in R_4

mass are 500 kg, so there are 9 various masses in this interval. 5 different wind speeds and 9 different road gradient scenarios are generated for each mass, so a total of 405 scenarios are generated. For training and test data, the driving cycle starts with acceleration from standstill to a predefined speed and then stopping. Three different predefined speeds are used for a driving cycle as shown in Figure 5.8. All driving cycles have different lengths. Approximately 11 h of driving time at 20 Hz is used to collect the data. The driving cycle for training and test data is different. Also, the test data includes all downhill, uphill, and flat roads for one driving cycle as shown in Figure 5.9. Eight different masses are chosen with an increment of 0.5 ton starting from 14.25 tons. Three road gradients and three wind speeds are chosen differently than the values in the training data as shown in Table 5.2.

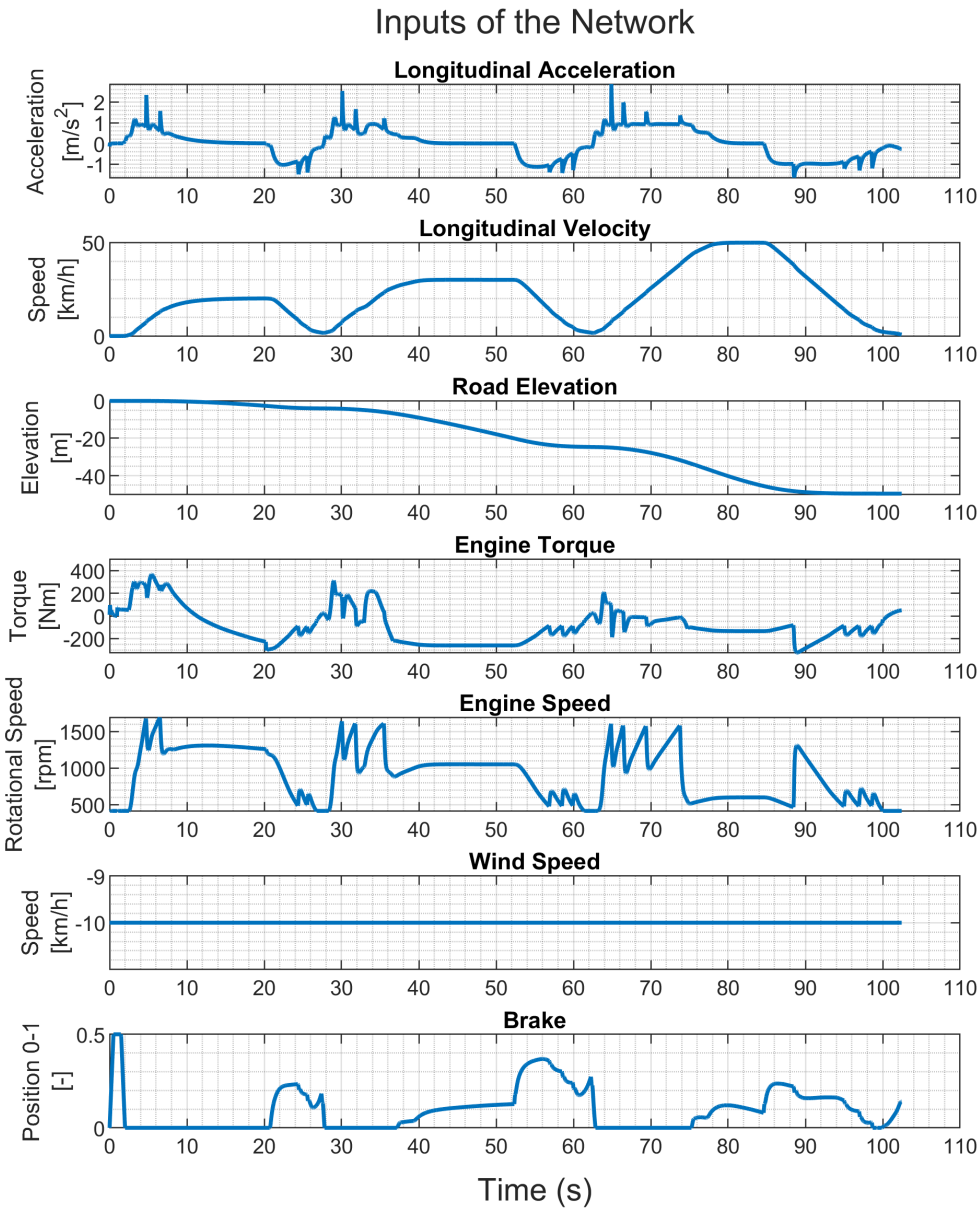

Figure 5.8 : Training Dataset

5.5 Training, Validation and Testing

A double-layer LSTM network with 15 neurons in each layer is used after a trial-and-error approach. In the beginning, eight data features are used. Then, the acceleration pedal position is removed since engine torque is enough to capture the dynamics. Also, since engine speed is known, the gear number is not given to the network as an additional input. Therefore, the road elevation, wind speed and brake pedal positions are added and the accelerator pedal position is removed, which is different from the features for the truck data shown in Section 5.1.

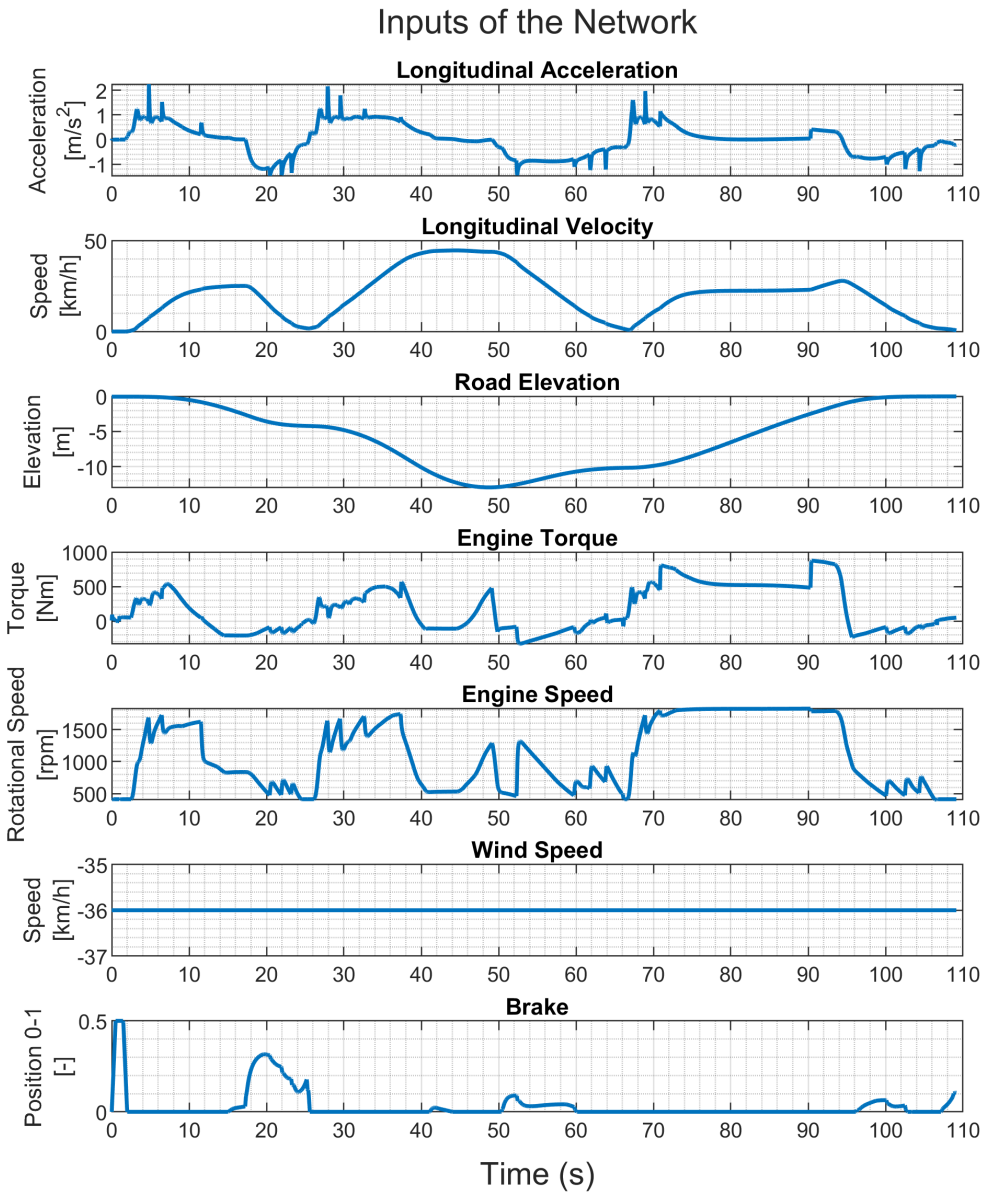

Figure 5.9 : Test Dataset

Since sequence lengths are different, either sequence padding or sequence truncation might be used. However, sequence padding decreases the accuracy of the network while increasing memory usage. Although some useful information might be lost by sequence truncation, the truncation effect is diminished by splitting data into equal smaller sequence lengths, such as 5 s, 10 s and 20 s.

One-ninth of the training data is allocated for validation. The minibatch size is adjusted to equal the amount of remaining data, so the epoch number is equal to the iteration number. The network is trained for 20000 epochs. The initial learning rate is multiplied

Table 5.2 : Data generation for various environmental factors

Data	Wind (kph)	Road Gradient%	Mass(ton)	Min-Max Speed (kph)
Training-Validation	0	0	14, 14.5	0-20
	10	±2	15, 15.5	
	20	±4	16, 16.5	0-30
	30	±6	17, 17.5	
	40	±8	18	0-50
Testing	12	0	14.25, 14.75	0-25
	24		15.25, 15.75	0-45
	36	±5	16.25, 16.75	0-35
			17.25, 17.75	

by a predetermined factor and reduced after each multiple of a predefined epoch number has passed.

5.6 Comparison of LSTM and GRU Cells

Table 5.3 : Validation Error Comparison for Different Number of LSTM Cells and Sequence Lengths

Sequence Length [s]	Cell	Number in Each Layer	Training Error [%]	Validation Error[%]	Initial Learning Rate	Drop Factor	Drop Iteration Period
20	LSTM	18	2	3	0.004	0.95	500
20	LSTM	17	3	3	0.005	0.95	500
20	LSTM	16	3	3	0.005	0.99	100
20	LSTM	16	3	3	0.005	0.95	500
20	LSTM	16	3	3	0.004	0.95	500
20	LSTM	**15**	3	**2**	0.004	0.95	500
20	LSTM	14	4	7	0.004	0.95	500
10	LSTM	16	5	5	0.005	0.99	100
10	LSTM	16	5	5	0.005	0.95	500
10	LSTM	16	5	5	0.004	0.95	500
5	LSTM	16	8	8	0.004	1	500
5	LSTM	16	12	10	0.002	1	500
5	LSTM	16	8	8	0.004	0.95	500
5	LSTM	16	7	8	0.004	0.9	1000
5	LSTM	16	8	8	0.005	0.9	1000
5	LSTM	10	13	14	0.005	0.99	100
5	LSTM	10	13	14	0.005	0.99	1000

As shown in Table 5.3, a double-layer network with more than 15 cells in a layer does not improve the accuracy of the mass estimation whereas lower cell numbers decrease the accuracy. In the table, it is also shown that if the sequence length is chosen larger, the mass estimation error decreases from 8% to less than 3% for the validation data if the ADAM optimizer is chosen.

The effect of the unavoidable discrepancy of the estimation at the different windows can be handled by choosing higher sequence lengths during the training. In case, Rmsprop optimization is chosen, the validation error increases to 5% when the same hyperparameters are used. If the stochastic gradient descent method is chosen, the validation error increases to 23%. However, validation error decreases to 5% when the initial learning rate is changed from 0.004 to 0.04.

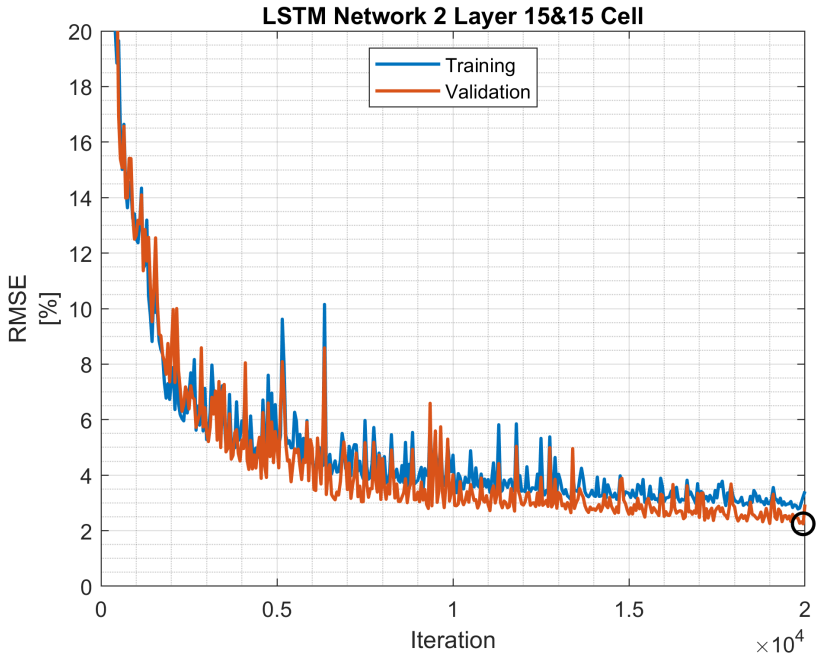

Figure 5.10 : Training and Validation Error for the LSTM network

Figure 5.10 shows the training and validation error during the training. There is no underfitting or overfitting since the training error is low and the validation error is not so different from the training error. Mass estimation on the validation data is shown in Figure 5.11. If there were overfitting, the network would memorize the training data, so the validation error would be much higher than the training error as seen in Figure 5.12. Evaluation of this mass estimator with the test data is far away from expectations, as seen in Figure 5.13 although there is no memorizing based on the training and validation data.

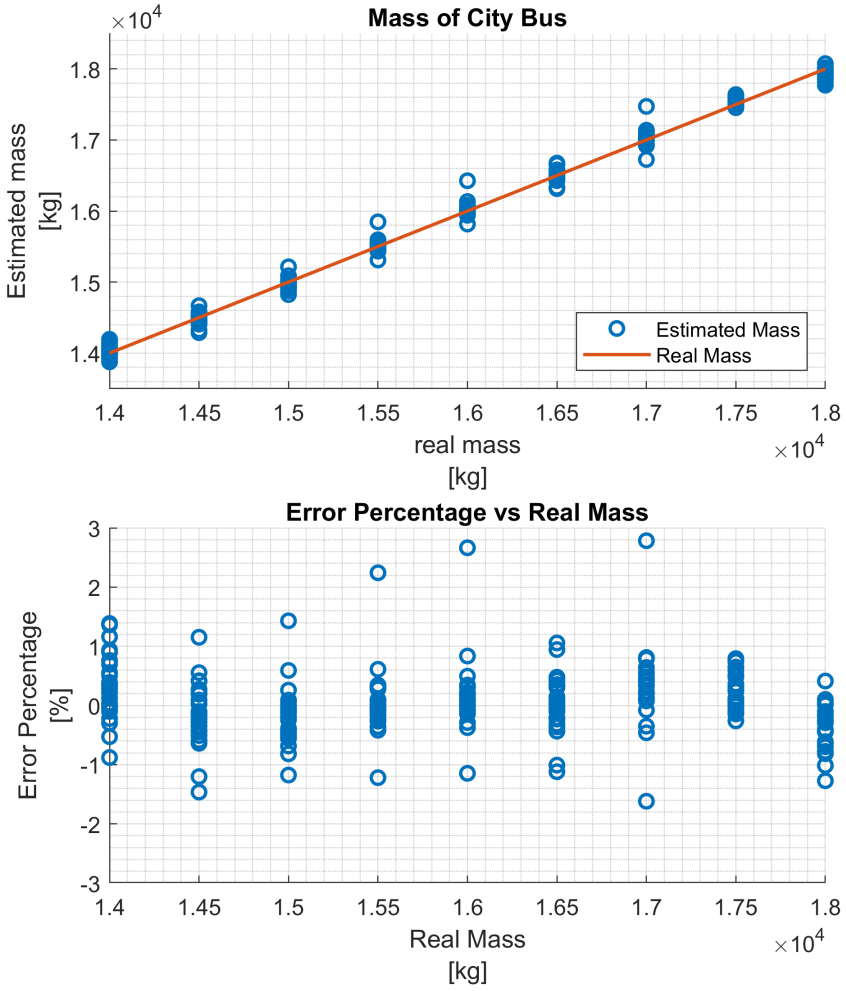

Figure 5.11 : Mass Estimation on Validation Data

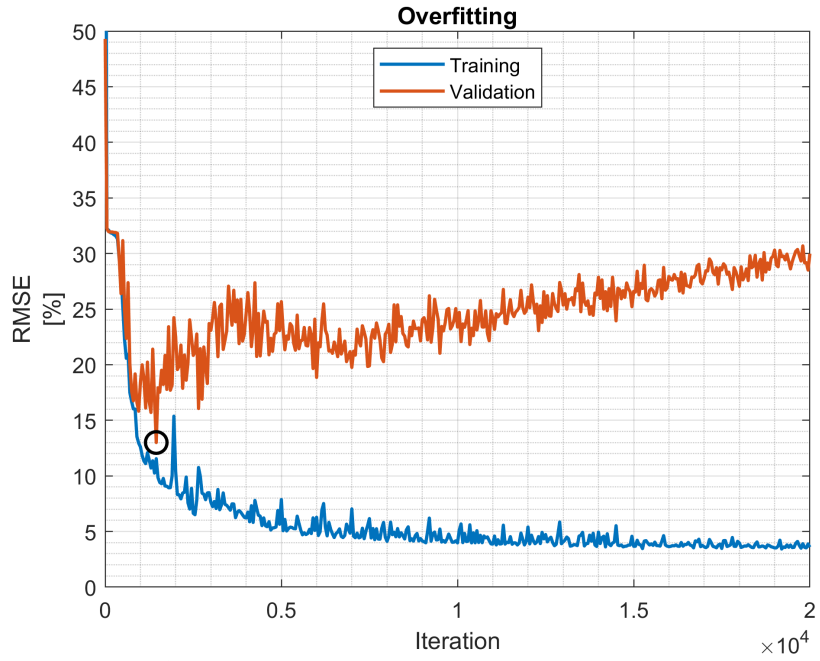

Figure 5.12 : Ovefitting in the LSTM network

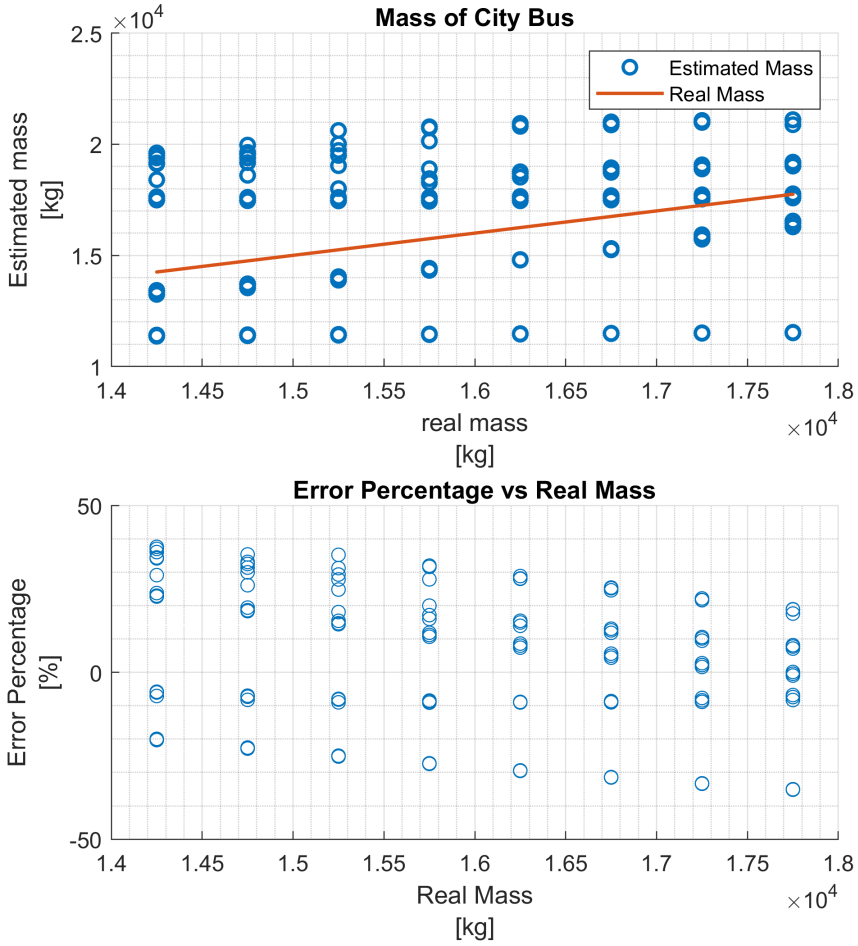

Figure 5.13 : Evaluation of the LSTM-based mass estimation

Training data is not enough to capture the dynamics of the test data because the training and validation data come from a different distribution than the test data. In other words, the test data contains patterns which were not included in the training and validation set. Therefore, the model performance can not be generalized to unseen data.

Table 5.4 represents the training and validation errors for various GRU cells. The error percentage of the GRU-based network is higher although the number of cells is increased compared to the LSTM-based network. Even though the number of parameters for a GRU cell is fewer than an LSTM cell, the total number of parameters is larger for GRU-based mass estimation since more GRU cells are used to decrease the validation error. While a double-layer LSTM-based mass estimator with 15 LSTMs in each layer has 3286 parameters, a double-layer GRU-based mass estimator with 20 GRUs in each layer has 4170 parameters.

Table 5.4 : Validation Error Comparison for Different Number of GRU Cells

Sequence Length [s]	Cell	Number in Each Layer	Training Error [%]	Validation Error[%]	Initial Learning Rate	Drop Factor	Drop Iteration Period
20	GRU	24	4	4	0.004	0.95	500
20	GRU	**20**	3	**4**	0.004	0.95	500
20	GRU	19	12	13	0.004	0.95	500
20	GRU	18	7	8	0.004	0.95	500
20	GRU	16	10	9	0.004	0.95	500
20	GRU	15	5	6	0.004	0.95	500
20	GRU	14	14	23	0.004	0.95	500
20	GRU	12	7	19	0.004	0.95	500
5	GRU	16	20	28	0.004	1	500
5	GRU	16	22	29	0.004	0.9	1000
5	GRU	12	17	29	0.004	0.9	1000
5	GRU	8	28	28	0.004	0.99	100

6. COMPARISON AND DISCUSSION OF MASS ESTIMATION METHODS

In the literature, various studies have employed the RLS method for vehicle mass estimation. Traditionally, the activation rules for RLS are defined manually, which can either excessively limit RLS usage or activate RLS inappropriately. Also, this manual process is time-consuming and impractical when different trucks or trailers are used. A significant contribution of this study is the use of Long Short-Term Memory (LSTM) networks to automate the activation and deactivation of RLS, thereby improving the flexibility and efficiency of the mass estimation process.

Machine learning approaches such as FFNN and DNN have also been applied to vehicle mass estimation in the literature. These approaches, however, lack memory, so requiring input values from each time step to account for the fast dynamics. In contrast, LSTM networks manage these dynamics internally, allowing for a more generic approach. This reduces the number of parameters that need to be tuned, which is another key contribution of this study.

Table 6.1 provides a comparative analysis of the RLS and LSTM methods for mass estimation. It is important to note that using LSTM to supervise RLS also increases computational complexity, training time, memory, and data requirements.

The datasets were collected exclusively by using the highest gear, focusing on various accelerator pedal positions for a chosen initial speed in Chapter 4 and Chapter 5. Additional data could also be collected, and a deeper network might be used by adding another layer with more LSTMs. Although this would provide generalisation since there is only one single network, it will increase the complexity and data collection costs. On the other hand, eight separate networks can be used with the same structure for each gear. So, each network might perform better for the specified gear. Since mixed model-based estimation is used when there is no gearshifting, the same gear is also used for learning-based mass estimation. The highest gear is chosen because more velocity intervals and lower excitation cases are analyzed. Also, the effect of the aerodynamic drag force is higher in the highest gear due to higher speeds. Another

Table 6.1 : Comparison of Recursive Least Squares (RLS) and Long Short-Term Memory (LSTM) based on various criteria

Criteria	RLS	LSTM
Algorithm Complexity	It requires less computational power.	Computationally expensive.
Sequence Length	Convergence takes time so simulation time needs to be longer	Sequence length is smaller than RLS method
Training Time	Tuning is easier.	Extensive training time is necessary to adjust the number of layers or neurons.
Memory Requirements	Low memory is enough; it is more suitable for real-time applications with limited resources.	High memory is needed to store weights and biases.
Data Requirements	Less data is necessary.	Large datasets are required for training for generalization.
Accuracy	Depends on the model accuracy.	Depends on the features and data quality.
Adaptability	Accurate vehicle model with calibrated parameters are necessary to be known.	It is necessary to retrain the network when the vehicle changes.
Robustness	It is negatively affected if the weather is windy.	It is negatively affected if the weather is windy.
Explainability	Easier to debug since there is a model.	Harder to debug due to the unknown dynamics.

reason is that torque losses are smaller compared to lower gears. Therefore, the main assumption is that the dock station is near the highway and the time to reach the highest possible gear is neglected. The lowest gear might be a better option for urban driving. However, the launch gear is not a fixed gear for a truck; it depends on the load of the truck.

The mass of the truck is assumed to be constant during the simulations. Since the simulation times are 10 s for the LSTM approach and 120 s for the RLS approach, the fuel consumption during these intervals is negligible compared to the mass estimation error. Mass change during the motion is uncommon unless an object falls from or enters into the truck. Thus, fixed loads are used during the simulations. One advantage of using fixed load is the usage of a sequence-to-one

model instead of a sequence-to-sequence model because sequence-to-one models require fewer computational resources and lower memory usage. Data preparation is easier compared to sequence-to-sequence models. The reason why sequential data is used instead of labeled data is because the latency and fast dynamics affect the estimation even if the mass does not change over time. Otherwise, FFNN or DNN would be preferred. On the other hand, the sequence-to-sequence model is used for the LSTM-supervised RLS method although the fixed load is used because in this case, the output of the network is not the mass but the error percentage, and the aim is to find the intervals when the RLS estimates the mass with less than a predefined error percentage.

The mass of the truck does not have to be computed continuously during the journey. It is more important to have more accurate estimates than the interval when mass estimation is activated. Therefore, estimation is done when the curvature is negligible. As a result, lateral dynamics are not included and the number of inputs in the network did not increase. In the same way, braking maneuvers are not included for the trucks since braking signals are not as reliable as acceleration signals. Also, fewer inputs are necessary for estimation during thrust than during braking since retarder is also used during braking.

While the suggested virtual load sensor is comparable in accuracy to leaf spring sensors under some assumptions and does not need any physical installation, its *training cost* may be much more than the calibration expenses of a hardware load sensor. For example, the LSTM network created for the suggested virtual load sensor necessitates a substantial amount of driving data, up to 11 hours. Each distinct powertrain type requires the collection of a corresponding dataset for the virtual sensors to operate well. Additionally, the network must be retrained, although the network topology stays unchanged. Moreover, the accuracy of the measurements may be influenced by external interferences, such as wind, particularly when the HDV is traveling at high speeds, such as on highways. Nevertheless, this problem is present in all mass estimating methods, since it is impossible to distinguish between the impact of imprecise powertrain models and external disruptions.

Inclinometer-based weight sensors are more expensive and they are preferred for uneven terrain conditions. Air suspension load sensors have better accuracy, but they

are also more expensive. Attaching sensors to the suspension system in production vehicles is challenging due to cost and geometric constraints [88].

For trucks, the mass change of the vehicle before and after loading the cargo will cause the suspension of the vehicle to be compressed. If there is no effective mass estimation method, the larger suspension deformation will directly affect the accuracy of the estimation of the vehicle acceleration, road gradient, and other parameters, thus affecting the performance of the vehicle safety system. Therefore, the research on the mass estimation of the truck has great significance [89]. The real load sensors are usually designed based on suspension dynamics whereas the virtual load sensor in this study is designed based on longitudinal dynamics. For the sensor-equipped HDVs, mass estimation may be used for fault detection and diagnosis if the difference between mass estimation and load sensor value is more than a threshold. The accuracy of the sensors is essential for both RLS and LSTM methods for accurate mass estimation. Measurements from the simulator are directly used in this study.

The reliability of the estimate lowers when there is intensive driving that leads to considerable pitch angles. Because the pitch angle also impacts the signal of the acceleration sensor. The pitch angle of the truck is altered due to unequal suspension deflection between the axles. Examples of situations that might cause load transfer and unequal forces on the suspension include sudden acceleration or deceleration, as well as driving on roads with slopes [88].

It is theoretically viable to obtain a target estimate error of less than 5% during a brake maneuver. However, this would need a training data set that contains samples acquired specifically during braking. Therefore, in order to get the desired level of accuracy in both acceleration and braking movements, it is inevitable that the size of the training data set would need to be enlarged, resulting in a more costly training phase.

To effectively handle tasks such as docking, loading, and unloading, it is sufficient to promptly determine the mass of an HDV as it starts its journey. This allows the onboard systems, including the automatic gear shift control system, to have access to an accurate estimate of the HDV's mass.

7. CONCLUSION AND FUTURE WORK

This research introduces a virtual mass sensor for heavy-duty vehicles by employing an LSTM network. This LSTM network, consisting of approximately 3000 parameters, achieves an error margin of less than ±5% across a broad load range of up to 4 tons under predefined conditions. Trained with 11 hours of driving data obtained from the TruckMaker simulation software for each ton interval, the LSTM-based mass estimator displays comparable accuracy to leaf spring load sensors during accelerating maneuvers.

To evaluate the scalability of the LSTM-based mass estimator across different load mass ranges, the experiments were conducted with the same amount of training dataset each containing 11 hours of data. The findings show that the accuracy remains competitive with the real sensors within the zero to four-ton load range during accelerating maneuvers. Therefore LSTM-based approach may be preferred relative to the associated training costs.

The study also details the methodology for conducting simulations to generate training and validation datasets. Although TruckMaker was utilized due to its highly accurate models, it is important to emphasize the necessity of replicating the training and testing processes with real-truck experimental data to validate these findings further.

To assess the impact of environmental conditions, an additional dataset was collected for a bus equipped with five gears, which included various wind speeds and road gradients. This dataset encompassed both acceleration and braking phases across all gears. Although the mass estimator could not estimate the mass due to the inadequate and low quality of the data, it was observed that the LSTM network outperforms the Gated Recurrent Unit (GRU) in terms of training error. Also, although the GRU has fewer training parameters, the computational complexity of the LSTM-based mass estimator is lower since more GRU is necessary compared to LSTM due to obtaining similar accuracy. Additionally, the study shows that increasing the sequence length enhances the accuracy of mass estimation.

The accuracy of the suggested virtual load sensor should be evaluated in comparison to the existing commercial sensors. However, key limitations of the LSTM-based mass estimator for real-world vehicle deployment should be highlighted. One major challenge is the need for re-training when different powertrains are involved. Additionally, the accuracy of the estimator diminishes in the presence of unmeasurable disturbances, such as adverse weather or icy road conditions.

Furthermore, this study proposes the effectiveness of the LSTM network as a supervisory mechanism within a model-based approach, specifically in augmenting the RLS method for mass estimation in heavy vehicles. Integrating a two-layer LSTM network as a reliability flag identifies the trustworthiness of the RLS mass estimator, with a pre-set error threshold of 3%. Our findings indicate an accuracy exceeding 90%, which enables precise forecasting of the reliability of the RLS mass estimator. Future research should extend the study to encompass various gears, masses, and terrains particularly as more accurate models become available.

REFERENCES

[1] **Cunanan, C.**, **Tran, M.K.**, **Lee, Y.**, **Kwok, S.**, **Leung, V. and Fowler, M.** (2021). A review of heavy-duty vehicle powertrain technologies: Diesel engine vehicles, battery electric vehicles, and hydrogen fuel cell electric vehicles, *Clean Technologies*, *3*(2), 474–489.

[2] **Liu, F.**, **Mauzerall, D.L.**, **Zhao, F. and Hao, H.** (2021). Deployment of fuel cell vehicles in China: Greenhouse gas emission reductions from converting the heavy-duty truck fleet from diesel and natural gas to hydrogen, *International journal of hydrogen energy*, *46*(34), 17982–17997.

[3] **Breuer, J.L.**, **Samsun, R.C.**, **Stolten, D. and Peters, R.** (2021). How to reduce the greenhouse gas emissions and air pollution caused by light and heavy duty vehicles with battery-electric, fuel cell-electric and catenary trucks, *Environment international*, *152*, 106474.

[4] **Xu, C.**, **Geyer, S. and Fathy, H.K.** (2019). Formulation and comparison of two real-time predictive gear shift algorithms for connected/automated heavy-duty vehicles, *IEEE Transactions on Vehicular Technology*, *68*(8), 7498–7510.

[5] **Chen, Y.L.**, **Shen, K.Y. and Wang, S.C.** (2013). Forward collision warning system considering both time-to-collision and safety braking distance, *2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, pp.972–977.

[6] **Ritter, A.** (2021). Optimal Control of Battery-Assisted Trolley Buses, *(Ph.D. thesis)*, ETH Zurich.

[7] **Kober, W. and Hirschberg, W.** (2006). On-board payload identification for commercial vehicles, *2006 IEEE International Conference on Mechatronics*, IEEE, pp.144–149.

[8] **Technoton**, (2023), Axle Load Sensors, `https://jv-technoton.com/products/axle-load-sensors/`, accessed on November 26, 2023.

[9] **VPG**, (2023), VPG Onboard Weighing, `https://www.vpgonboard.com/products/truckweigh`, accessed on December 12, 2023.

[10] **Paulsson, E. and Åsman, L.**, (2016), Vehicle Mass and Road Grade Estimation using Recursive Least Squares, *(Master thesis)*, Lund University.

[11] **Bae, H.S.**, **Ryu, J. and Gerdes, J.C.** (2001). Road grade and vehicle parameter estimation for longitudinal control using GPS, *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, pp.25–29.

[12] **Fathy, H.K.**, **Kang, D. and Stein, J.L.** (2008). Online vehicle mass estimation using recursive least squares and supervisory data extraction, *2008 American control conference*, IEEE, pp.1842–1848.

[13] **Vahidi, A.**, **Stefanopoulou, A. and Peng, H.** (2005). Recursive least squares with forgetting for online estimation of vehicle mass and road grade: theory and experiments, *Vehicle System Dynamics*, *43*(1), 31–55.

[14] **McIntyre, M.L.**, **Ghotikar, T.J.**, **Vahidi, A.**, **Song, X. and Dawson, D.M.** (2009). A two-stage Lyapunov-based estimator for estimation of vehicle mass and road grade, *IEEE Transactions on vehicular Technology*, *58*(7), 3177–3185.

[15] **Kim, D.**, **Choi, S.B. and Oh, J.** (2012). Integrated vehicle mass estimation using longitudinal and roll dynamics, *2012 12th International Conference on Control, Automation and Systems*, IEEE, pp.862–867.

[16] **Torabi, S.**, **Wahde, M. and Hartono, P.** (2019). Road grade and vehicle mass estimation for heavy-duty vehicles using feedforward neural networks, *2019 4th international conference on intelligent transportation engineering (ICITE)*, IEEE, pp.316–321.

[17] **Korayem, A.H.**, **Khajepour, A. and Fidan, B.** (2020). Trailer mass estimation using system model-based and machine learning approaches, *IEEE Transactions on Vehicular Technology*, *69*(11), 12536–12546.

[18] **Zhang, H.**, **Yang, Z.**, **Shen, J.**, **Long, Z. and Xiong, H.** (2023). Dynamic mass estimation framework for autonomous vehicle system via bidirectional gated recurrent unit, *IET Control Theory & Applications*.

[19] **Leoni, J.**, **Strada, S.**, **Tanelli, M. and Savaresi, S.M.** (2023). Real Time Passenger Mass Estimation for e-scooters, *2023 American Control Conference (ACC)*, IEEE, pp.1741–1746.

[20] **Yu, Z.**, **Hou, X.**, **Leng, B. and Huang, Y.** (2022). Mass estimation method for intelligent vehicles based on fusion of machine learning and vehicle dynamic model, *Autonomous Intelligent Systems*, *2*(1), 4.

[21] **Eagon, M.**, **Fakhimi, S.**, **Pernsteiner, A. and Northrop, W.F.** (2022). Mass Detection for Heavy-Duty Vehicles using Gaussian Belief Propagation, *2022 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, pp.1655–1661.

[22] **Mittal, A. and Fairgrieve, A.**, (2018), Vehicle Mass Estimation, US Patent App. 15/553,874.

[23] **Rezaeian, A. and Li, D.**, (2023), Vehicle center of gravity height detection and vehicle mass detection using light detection and ranging point cloud data, US Patent 11,702,085.

[24] **Huang, X.**, (2020), Method for real-time mass estimation of a vehicle system, US Patent 10,612,961.

[25] **Jundt, O.**, **Juhasz, G.**, **Weis, R. and Skrabak, A.**, (2022), System and method for identifying a change in load of a commercial vehicle, US Patent App. 17/310,886.

[26] **Switkes, J.P.**, **Erlien, S.M. and Schuh, A.B.**, (2019), Applications for using mass estimations for vehicles, US Patent 10,216,195.

[27] **Cho, G.** (2022). Artificial Neural Network Methods for Lithium-Ion Battery Behavior Predictions, *(Ph.D. thesis)*.

[28] **Schmidhuber, J.** (2015). Deep learning in neural networks: An overview, *Neural networks*, *61*, 85–117.

[29] **LeCun, Y.**, **Bengio, Y. and Hinton, G.** (2015). Deep learning, *Nature*, *521*(7553), 436–444.

[30] **Karpathy, A.**, (2016), Lecture-3: Linear Classification 2 - Optimization, CS231N Deep Learning for Computer Vision, Stanford University, `https://www.youtube.com/watch?v=qlLChbHhbg4&list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC`, accessed on May 16, 2024.

[31] **O'Shea, K. and Nash, R.** (2015). An introduction to convolutional neural networks, *arXiv preprint arXiv:1511.08458*.

[32] **Lim, B. and Zohren, S.** (2021). Time-series forecasting with deep learning: a survey, *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200209.

[33] **Qin, T.**, **Yao, Z.**, **Fan, H.**, **Xia, R. and Chen, T.** (2023). Vehicle Road Grade Prediction Based on CNN-LSTM, *IFAC-PapersOnLine*, *56*(2), 3066–3071.

[34] **Borghesi, A. and Milano, M.**, (2022), ML and Time Series, 78778 Intelligent Systems Lecture Notes, University of Bologna.

[35] **Bai, S.**, **Kolter, J.Z. and Koltun, V.** (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, *arXiv preprint arXiv:1803.01271*.

[36] **Goodfellow, I.**, **Bengio, Y. and Courville, A.** (2016). *Deep learning*, MIT press.

[37] **Rodriguez, P.**, **Wiles, J. and Elman, J.L.** (1999). A recurrent neural network that learns to count, *Connection Science*, *11*(1), 5–40.

[38] **Pascanu, R.**, **Mikolov, T. and Bengio, Y.** (2013). On the difficulty of training recurrent neural networks, *International conference on machine learning*, Pmlr, pp.1310–1318.

[39] **Hochreiter, S. and Schmidhuber, J.** (1997). Long short-term memory, *Neural computation*, *9*(8), 1735–1780.

[40] **Chung, J.**, **Gulcehre, C.**, **Cho, K. and Bengio, Y.** (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling, *arXiv preprint arXiv:1412.3555*.

[41] **Gers, F.A.**, **Schmidhuber, J. and Cummins, F.** (2000). Learning to forget: Continual prediction with LSTM, *Neural computation*, *12*(10), 2451–2471.

[42] **Ng, A.**, (2021), Sequence Models, `https://www.youtube.com/watch?v=S7oA5C43Rbc`, accessed on May 16, 2024.

[43] **Greff, K.**, **Srivastava, R.K.**, **Koutník, J.**, **Steunebrink, B.R. and Schmidhuber, J.** (2016). LSTM: A search space odyssey, *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

[44] **Sutskever, I.**, **Vinyals, O. and Le, Q.V.** (2014). Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, *27*.

[45] **Ergen, T. and Kozat, S.S.** (2017). Efficient online learning algorithms based on LSTM neural networks, *IEEE transactions on neural networks and learning systems*, *29*(8), 3772–3783.

[46] **Feng, L.** (2023). Predicting Output Responses of Nonlinear Dynamical Systems With Parametrized Inputs Using LSTM, *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, *8*, 97–107.

[47] **Su, Y.**, **Lin, J.**, **Zhao, D.**, **Guo, C.**, **Wang, C. and Guo, H.** (2020). Real-time prediction of large-scale ship model vertical acceleration based on recurrent neural network, *Journal of Marine Science and Engineering*, *8*(10), 777.

[48] **Lindemann, B.**, **Maschler, B.**, **Sahlab, N. and Weyrich, M.** (2021). A survey on anomaly detection for technical systems using LSTM networks, *Computers in Industry*, *131*, 103498.

[49] **Wang, Y.** (2017). A new concept using LSTM Neural Networks for dynamic system identification, *2017 American control conference (ACC)*, IEEE, pp.5324–5329.

[50] **Xing, Y. and Lv, C.** (2019). Dynamic state estimation for the advanced brake system of electric vehicles by using deep recurrent neural networks, *IEEE Transactions on Industrial Electronics*, *67*(11), 9536–9547.

[51] **Fawcett, T.** (2006). An introduction to ROC analysis, *Pattern recognition letters*, *27*(8), 861–874.

[52] **Scikit**, (2024), Metrics and scoring: quantifying the quality of predictions, `https://scikit-learn.org/stable/modules/model_evaluation.html`, accessed on January 19, 2024.

[53] **Baydin, A.G.**, **Pearlmutter, B.A.**, **Radul, A.A. and Siskind, J.M.** (2018). Automatic differentiation in machine learning: a survey, *Journal of Machine Learning Research*, *18*, 1–43.

[54] **Grosse, R.**, (2018), Automatic Differentiation, CSC 321 Intro to Neural Networks and Machine Learning, University of Toronto, https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf, accessed on May 16, 2024.

[55] **Karpathy, A.**, (2016), Lecture 4; Backpropagation, Neural Networks 1, CS231n Winter 2016, Stanford University, https://www.youtube.com/watch?v=i94OvYb6noo&list=PLZHmVqFZ9EzuAJWUMkLlFxJIX9MCJ8cSn&index=4, accessed on June 03, 2024.

[56] **Ruder, S.** (2016). An overview of gradient descent optimization algorithms, *arXiv preprint arXiv:1609.04747*.

[57] **Sutskever, I.**, **Martens, J.**, **Dahl, G. and Hinton, G.** (2013). On the importance of initialization and momentum in deep learning, *International conference on machine learning*, PMLR, pp.1139–1147.

[58] **Duchi, J.**, **Hazan, E. and Singer, Y.** (2011). Adaptive subgradient methods for online learning and stochastic optimization., *Journal of machine learning research*, *12*(7).

[59] **Tieleman, T.**, **Hinton, G.** *et al.* (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

[60] **Ng, A.**, (2017), RMSProp - Deep Learning by DeepLearningAI, https://www.youtube.com/watch?v=_e-LFe_igno&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=21, accessed on May 16, 2024.

[61] **Kingma, D.P. and Ba, J.** (2014). Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.

[62] **Ng, A.**, (2017), Adam optimization Algorithm, https://www.youtube.com/watch?v=JXQT_vxqwIs&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=22, accessed on May 16, 2024.

[63] **Tibshirani, R.** (1996). Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *58*(1), 267–288.

[64] **Srivastava, N.**, **Hinton, G.**, **Krizhevsky, A.**, **Sutskever, I. and Salakhutdinov, R.** (2014). Dropout: a simple way to prevent neural networks from overfitting, *The journal of machine learning research*, *15*(1), 1929–1958.

[65] **Narkhede, M.V.**, **Bartakke, P.P. and Sutaone, M.S.** (2022). A review on weight initialization strategies for neural networks, *Artificial intelligence review*, *55*(1), 291–322.

[66] **MathWorks**, (2023), Initialize Learnable Parameters for Custom Training Loop, https://se.mathworks.com/help/deeplearning/ug/initialize-learnable-parameters-for-custom-training-loop.html?searchHighlight=Glorot&s_tid=srchtitle_support_results_1_Glorot, accessed on May 16, 2024.

[67] **Glorot, X. and Bengio, Y.** (2010). Understanding the difficulty of training deep feedforward neural networks, *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, pp.249–256.

[68] **Hu, W.**, **Xiao, L. and Pennington, J.** (2020). Provable benefit of orthogonal initialization in optimizing deep linear networks, *arXiv preprint arXiv:2001.05992*.

[69] **He, K.**, **Zhang, X.**, **Ren, S. and Sun, J.** (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE international conference on computer vision*, pp.1026–1034.

[70] **Chowdhury, K.**, (2023), 10 Hyperparameters to Keep an Eye on for your LSTM Model—and other Tips, https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4, accessed on November 28, 2023.

[71] **Li, F.F.**, **Karpathy, A. and Johnson, J.**, (2016), Neural Networks-3, CS231n Winter 2016, Stanford University, https://cs231n.github.io/neural-networks-3/, accessed on January 20, 2023.

[72] **Karakaya, M.**, (2022), LSTM: Understanding the Number of Parameters, https://www.kaggle.com/code/kmkarakaya/lstm-understanding-the-number-of-parameters, accessed on May 16, 2024.

[73] **Bittanti, S.** (2019). *Model identification and data analysis*, John Wiley & Sons.

[74] **Mahadi, M.**, **Ballal, T.**, **Moinuddin, M. and Al-Saggaf, U.M.** (2022). A recursive least-squares with a time-varying regularization parameter, *Applied Sciences*, *12*(4), 2077.

[75] **Bernstein, D.** (2018). *Scalar, vector, and matrix mathematics: theory, facts, and formulas-revised and expanded edition*, Princeton university press.

[76] **Lai, B.**, **Islam, S.A.U. and Bernstein, D.S.** (2021). Regularization-induced bias and consistency in recursive least squares, *2021 American Control Conference (ACC)*, IEEE, pp.3987–3992.

[77] **Islam, S.A.U. and Bernstein, D.S.** (2019). Recursive least squares for real-time implementation [lecture notes], *IEEE Control Systems Magazine*, *39*(3), 82–85.

[78] **Bruce, A.** (2022). Learning to Forget Advanced Online Recursive Identification for Estimation and Adaptive Control, *(Ph.D. thesis)*.

[79] **Zhang, Q.** (2000). Some implementation aspects of sliding window least squares algorithms, *IFAC Proceedings Volumes*, *33*(15), 763–768.

[80] **Hoagg, J.B.**, **Ali, A.A.**, **Mossberg, M. and Bernstein, D.S.** (2011). Sliding window recursive quadratic optimization with variable regularization, *Proceedings of the 2011 American Control Conference*, IEEE, pp.3275–3280.

[81] **Bruce, A.L.**, **Goel, A. and Bernstein, D.S.** (2020). Convergence and consistency of recursive least squares with variable-rate forgetting, *Automatica*, *119*, 109052.

[82] **Goel, A.**, **Bruce, A.L. and Bernstein, D.S.** (2020). Recursive least squares with variable-direction forgetting: Compensating for the loss of persistency [lecture notes], *IEEE Control Systems Magazine*, *40*(4), 80–102.

[83] **Bruce, A.L.**, **Goel, A. and Bernstein, D.S.** (2021). Necessary and sufficient regressor conditions for the global asymptotic stability of recursive least squares, *Systems & Control Letters*, *157*, 105005.

[84] **Lai, B. and Bernstein, D.S.** (2022). Exponential Resetting and Cyclic Resetting Recursive Least Squares, *IEEE Control Systems Letters*, *7*, 985–990.

[85] **Rajamani, R.** (2011). *Vehicle dynamics and control*, Springer Science & Business Media.

[86] **Torabi, S.** (2020). Fuel-efficient driving strategies, *(Ph.D. thesis)*, Chalmers University of Technology.

[87] **Ba, J.L.**, **Kiros, J.R. and Hinton, G.E.** (2016). Layer normalization, *arXiv preprint arXiv:1607.06450*.

[88] **Ahn, Y.**, **Han, S.**, **Kim, G. and Huh, K.** (2024). Slope and Mass Estimation for Semi-Trailer Truck Considering Pitch Angle Variations, *IEEE Transactions on Intelligent Vehicles*.

[89] **Wang, B.**, **Wang, H.**, **Wu, L.**, **Cai, L.**, **Pi, D. and Wang, E.** (2020). Truck mass estimation method based on the on-board sensor, *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, *234*(10-11), 2429–2443.

CURRICULUM VITAE

Name Surname: Abdurrahman İşbitirici

Place and Date of Birth: İzmir - 20/07/1988

E-Mail: abdurrahmanisbitirici@gmail.com

EDUCATION:

- **M.Sc.:** 2015, Istanbul Technical University, Faculty of Engineering, Department of Mechanical Engineering, System Dynamics & Control Programme
- **B.Sc.:** 2012, Yeditepe University, Faculy of Engineering, Electrical & Electronics Engineering Department
- **B.Sc.:** 2011, Yeditepe University, Faculy of Engineering, Mechanical Engineering Department