DATA DRIVEN METHODOLOGIES FOR TRUCK PLATOONING

A Thesis

submitted by

KARTHIK S

ME18B149

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY
in MECHANICAL ENGINEERING
&
MASTER OF TECHNOLOGY
in DATA SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **DATA DRIVEN METHODOLOGIES FOR TRUCK PLATOONING**, submitted by **KARTHIK S**, to the Indian Institute of Technology, Madras, for the award of the degree of **BACHELOR & MASTER OF TECHNOLOGY**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Shankar Ram C S
Research Guide
Professor
Dept. of Engineering Design
IIT Madras

Dr. Devika K B
Co - Guide
Adjunct Professor
Dept. of Engineering Design
IIT Madras

Place: Chennai

Date: 05-May-2023

ACKNOWLEDGEMENTS

Throughout my time in the institute and this project, I have received a lot of support and assistance from multiple people.

First and foremost, I express my gratitude to **Prof. Shankar Ram C S** for accepting my collaboration request and for his support. His systematic approach and professionalism in handling things inspired me. His attention to detail has been instrumental in shaping my research skills.

I am also grateful to **Dr. Devika K B** and **Dr. Rohith** for their invaluable contributions to this project. Their expertise, discussion availability, and guidance have been vital to my progress over the past year.

Additionally, I would like to thank my friends at **IIT Madras** and **Raftar Formula Racing** for their invaluable support and contribution to my knowledge and understanding of research and life.

Finally, I would like to thank my grandparents, parents and brother Balaji for the support and encouragement they have provided me throughout my years of study and through the process of researching and writing this thesis.

Karthik S

ABSTRACT

KEYWORDS: Deep learning, Drive cycle, Electric trucks, Encoder-decoder model, LSTM, Platoon, State-of-charge.

Platooning is a potential option for extending the range of electric trucks, especially during long-haul operations. However, the speed that should be followed by the platoon such that the route would be completed with the available electrical energy is essential. This thesis presents a novel deep-learning approach to obtain the speed profile to be followed by an autonomous electric truck platoon considering various constraints, such as the available state of charge (SOC) in the batteries along with other vehicles and road conditions, while ensuring that the platoon is string stable. The model has been trained using various known highway drive cycles to ensure the framework is suitable for long-haul highway operations.

Encoder-decoder models were trained and hyperparameter tuning was performed for the same. Finally, the most suitable model based on the root mean squared error and mean absolute percentage error on the validation dataset has been chosen for the application. For testing the framework, drive cycle/speed prediction corresponding to different desired SOC profiles has been presented. A case study has also shown the relevance of the proposed framework in predicting the drive cycle on various routes and its impact on making critical policy decisions during the planning of electric truck platoons. This study would help to efficiently plan the feasible routes for electric trucks considering multiple constraints such as battery capacity, expected discharge rate, charging infrastructure availability, route length/travel time, and other operating conditions.

Similarly, a model was developed for the opposite problem of predicting energy consumption along a route. Since the follower vehicles are affected by the aerodynamic drag, it is necessary to quantify the advantage obtained by moving in a platoon compared to moving individually. Similar deep learning techniques as the previous case were used to construct and train this model. The same dataset as the route planning problem was also used to train and validate this model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology Madras
ML	Machine Learning
DL	Deep Learning
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
SOC	State-Of-Charge
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAPE	Mean Absolute Percentage Error

CHAPTER 1

Introduction

1.1 Truck Platooning

Truck platooning is the linking of two or more trucks in convoy using connectivity technology and automated driving support systems. These vehicles automatically maintain a set, close distance between each other when they are connected for certain parts of a journey, for instance, on motorways.

The truck at the head of the platoon acts as the leader, with the vehicles behind reacting and adapting to changes in its movement – requiring little to no action from drivers. In the first instance, drivers will always remain in control, so they can also decide to leave the platoon and drive independently. [1]

The concept of truck platooning dates back to 1970s, when the European ARAMIS project platooned 25 small transit vehicles running a foot apart at 50 mph on a French test track. [2]

By 1992, the first vehicle platooning experiments were successfully concluded, and the four-vehicle platoon capability was demonstrated for visitors on the I-15 HOV lanes in San Diego in 1994 [3]. There are currently pilot trials of platooning in the world in regions such as Europe, the United States, South Korea, Singapore, Japan and Australia.

The primary advantages of truck platooning are:

- Air drag reduction due to the effect of the front vehicle.
- Efficient driving, faster delivery and transportation of goods.

An important concept involving platoons is **string stability**. Intuitively, a platoon is said to be string stable if it has the property, i.e., the disturbances are not amplified when propagating along the vehicle string.

1.2 Problem formulation and objectives

A truck platoon is a potential option for significant efficiency improvement, especially during long-haul operation. Currently, only a control theory based solution exists for the analysis of platoons.

Machine learning and Deep learning as a field have gained massive advancements for solving problems and finding patterns in various industries where obtaining first principle equations between parameters have been difficult/nearly impossible/inefficient. Hence, it was decided to use this approach for the analysis of platoons as well.

This project aims to explore the problem from the above perspective and obtain data-driven solutions. Two different use cases have been discussed and deep learning based models have been prepared for the same.

The first model predicts the leader vehicle's speed based on the expected power/SOC input along a route. The second model predicts the energy consumption of the follower vehicle using the drive cycle as input.

1.2.1 Structure of the thesis

This thesis is broken down into four parts as follows:

1. Dataset creation
2. Overview of models used
3. Speed prediction of the leader vehicle
4. Energy prediction of the follower vehicle

In the first section Chap. 2, the existing controller-based solution for the platoon speed tracking problem is discussed, and all the data required for this project is collected and stored. In addition to that, preprocessing is done on the same to make it compatible with the models.

In the second section Chap. 3, a mathematical overview of the deep learning model used - Long Short Term Memory (LSTM) network along with the Encoder Decoder models are discussed.

Following that, the next two sections Chap. 4 and 5, describe the model-building process and tuning for the following use cases:

In Chap. 4, the expected speed profile (drive cycle) for the leader vehicle was predicted given the expected battery discharge profile along the route and other vehicle operating conditions such as the mass, time headway and friction coefficient.

In Chap. 5, the power consumed by the follower vehicles based on the drive cycle is predicted. The power cycle of the leader vehicle is easily obtained due to direct first principle relations. However, for the follower vehicles, the aerodynamic drag effects make the deep learning solution more efficient than the first principle solutions.

CHAPTER 2

Dataset creation

2.1 Literature review

2.1.1 A Dynamics-Based Adaptive String Stable Controller for Connected Heavy Road Vehicle Platoon Safety [4]

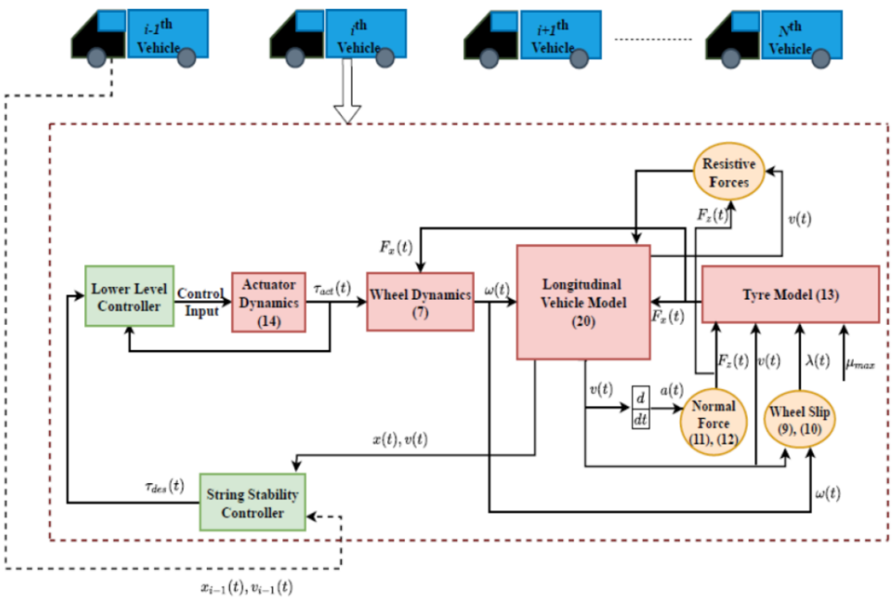


Figure 2.1: Overall design framework of the controller [4]

A string stable controller has been implemented which can simulate the movement of a truck platoon given the velocity profile of the leader vehicle and the initial operating conditions (time headway, friction coefficient and the mass of the truck).

Some of the assumptions of the controller were:

- The vehicles travel on a straight road.
- Only longitudinal dynamics of the vehicle is considered.
- The tyre model parameters and vehicle parameters are assumed to be known.
- There is equal distribution of load on the left and right wheels of a specific axle of the vehicle.

The factors considered for this controller were 1) Brake/PT actuator dynamics 2) Resistive Forces 3) Tyre model and Wheel dynamics 4) Aerodynamic drag effects, and 5) Communication delays. The overall layout of the controller is shown in Fig 2.1.

Extensive analysis and simulations were done for different operating conditions. Also, the importance of each parameter was checked by checking the controller with and without that particular parameter. Finally, after successful results, the framework was finalized.

2.2 Data collection and processing

The process of developing a machine/deep learning model requires a dataset of known inputs and outputs for training and validation.

An important assumption is that the controller model discussed in the precious section is a good measure of the true data. Since getting the data by actually running the trucks is challenging, this assumption was made. Moreover, since the controller uses first principles upto tyre modelling, it is a fairly accurate representation of the actual scenarios. Using the platoon controller model [4] and known drive cycles, the following were obtained for various routes and operating conditions:

- The energy profiles (and thereby Power and SOC) for different drive cycles for different operating conditions.
- The tracked speed of the follower vehicles for every combination of the route and the operating conditions.

In this work, four commonly used heavy vehicle highway drive cycles, the European Transient Cycle (ETC) [5], Millbrook cycle [6], the Highway Fuel Economy Test (HWFET or HFET) cycle [7], and the Heavy Heavy-Duty Diesel Truck (HHDDT) cycle [8], were used. Their details are tabulated in Table 2.1, and plots are presented in Fig. 2.2.

Table 2.1: Drive cycles used

Purpose	Name of Cycle	Distance (m)	Duration (s)	Avg. Speed (km/h)
Training	ETC - part 2 and 3	25620	1200	76.85
	Millbrook Heavy Duty	17903	780	82.6
	HWFET	16450	765	77.7
Validation	HHDDT	37336	2083	64.21

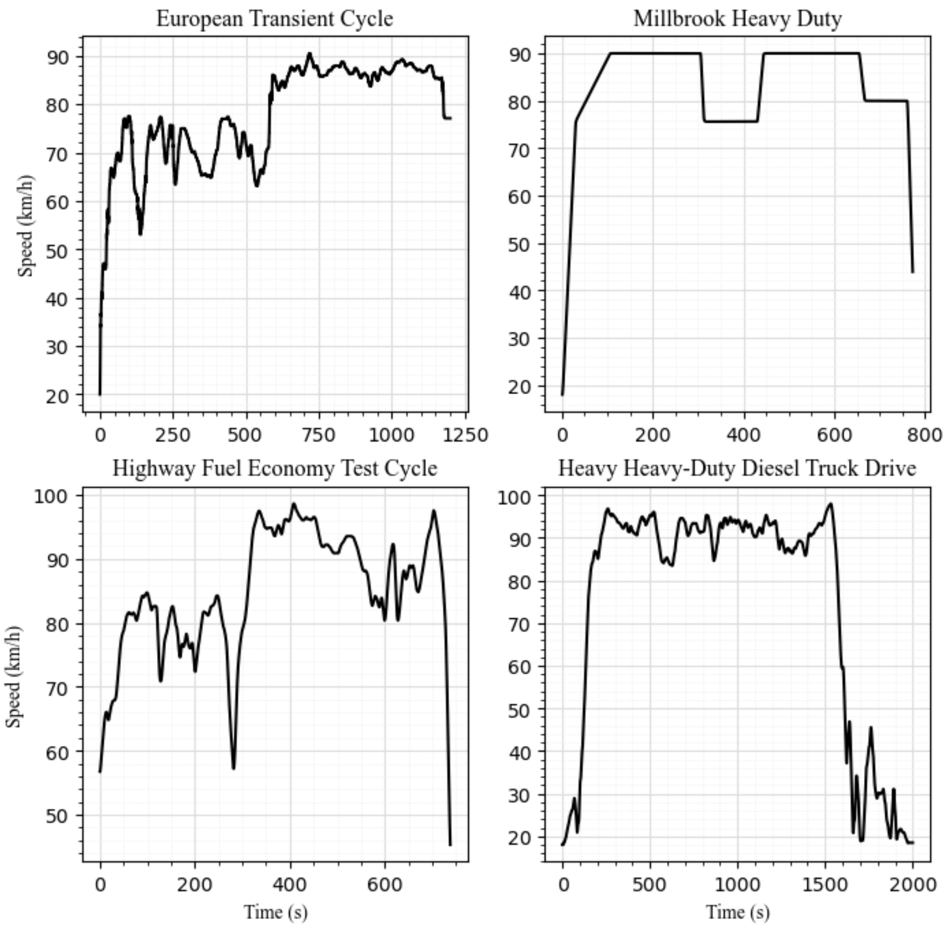


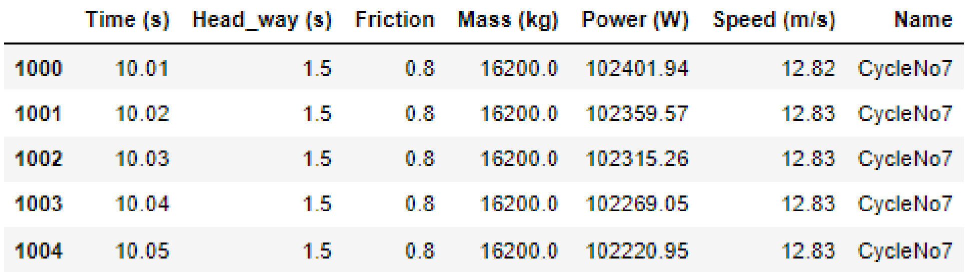
Figure 2.2: Drive cycles used

The European Transient Cycle (ETC) is used for emission certification of heavy-duty diesel engines in Europe starting in the year 2000. It was developed at Aachen, Germany, based on actual road cycle measurements of heavy duty vehicles [5].

Millbrook cycles [6] are an extensive list of test cycles developed by Millbrook Proving Ground, an English vehicle testing centre and one of the largest vehicle testing centres in Europe. The heavy-duty motorway cycle was taken as one of the drive cycles for training this model.

The Highway Fuel Economy Test Cycle (HWFET) cycle is a chassis dynamometer driving schedule developed by the US Environmental Protection Agency [7]. The driving cycle is developed to simulate a mixture of interstate highway and rural driving.

The Heavy Heavy-Duty Diesel Truck Drive (HHDDT) schedule [8] is a chassis dynamometer test developed by the California Air Resources Board with the cooperation of West Virginia University. The test consists of four speed-time modes: idle, creep, transient and (high-speed) cruise. Since the main advantage of platoons is in the freeways, the cruise mode was taken as the drive cycle to be tested with the model.



	Time (s)	Head_way (s)	Friction	Mass (kg)	Power (W)	Speed (m/s)	Name
1000	10.01	1.5	0.8	16200.0	102401.94	12.82	CycleNo7
1001	10.02	1.5	0.8	16200.0	102359.57	12.83	CycleNo7
1002	10.03	1.5	0.8	16200.0	102315.26	12.83	CycleNo7
1003	10.04	1.5	0.8	16200.0	102269.05	12.83	CycleNo7
1004	10.05	1.5	0.8	16200.0	102220.95	12.83	CycleNo7

Figure 2.3: Sample entries from the dataset

The energy profiles were generated for various road and vehicle conditions using the electric platoon controller model for all these drive cycles. Two different tyre-road friction coefficients of values $\mu = 0.3$, and $\mu = 0.8$, representing low and high friction conditions respectively were considered. Both fully laden (16200 kg) and partially laden (10450 kg representing 60% loading) operating conditions were taken into account. Similarly, different time headway magnitudes ($h = 1$ s and $h = 1.5$ s), the time gap between two successive vehicles in the platoon, were also considered while generating the profiles.

Using the above conditions, for each route (drive cycle), 8 energy profiles were obtained, one for each combination of the vehicle conditions, amounting to a total of 32 energy profiles for the 4 considered drive cycles. Of these, 24 profiles (from the 1st three drive cycles) were used for training the model and 8 (from the fourth drive cycle) for validation.

The energy profile is a cumulative quantity. Hence, to make it instantaneous, it was converted to a power profile. However, this profile can be converted to an energy or SOC profile whenever required.

With regard to the number of samples, since the controller had a sampling time of 0.01s, the total size of the training dataset came out to be 2,196,000 and the validation dataset is 1,666,400. A sample of how the entries will look is shown in Fig 2.3

Feature scaling is an essential pre-processing step for many machine learning algorithms [9]. If features are left unscaled, there is a high chance that some features dominate due to their sheer order of magnitude and create an unnecessary bias in the model. Since the range and scale of each variable are different in the considered data set, all the parameters were normalized to lie between 0 and 1.

2.3 Performance metrics

In order to train the model, an objective function has to be defined. Since both use cases are regression problems (speed/power is a continuous variable), the Mean Squared Error (MSE) is a good indicator of the model's performance and has been chosen as the objective function.

Consider the case of predicting speed. If v_i denotes the actual speed and $\hat{v}_i$ denotes the predicted speed at the i^{th} instant of the drive cycle, then the MSE is given by

$$L(\mathbf{v}, \hat{\mathbf{v}}) = \frac{1}{n}\sum_{i=1}^{n}(v_i - \hat{v}_i)^2. \tag{2.1}$$

Training is followed by validation. For validating a model, appropriate performance metrics and data must be defined. Since the objective function is an indicator of how close the predictions are to actual values, Root Mean Squared Error (RMSE) can be chosen as the performance metric. This is used instead of MSE to scale down the units to that of the output variable. Moreover, Mean Absolute Percentage Error (MAPE) was also used as a performance metric to compare the predicted output with the actual drive cycle as a percentage change. Mathematically, they are defined as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i - \hat{v}_i)^2}, \tag{2.2}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{v_i - \hat{v}_i}{v_i}\right|. \tag{2.3}$$

The above expressions can be similarly derived for the power prediction use case as well. If the performance on the training data is only evaluated, it will result in overfitting, i.e., the model may become too specialized to the training data and fail to generalize. To address this issue, the available dataset is typically split into training and validation sets. The model is trained on the training set, and its performance is evaluated on the validation set, which serves as a proxy for the unseen test data. This allows us to monitor the model's performance on new data and detect if it is overfitting to the training data. Therefore, validation data is not part of the training data to ensure that the model is evaluated on data it has not encountered during training.

CHAPTER 3

Long Short Term Memory (LSTM) Networks

3.1 Literature review

A Recurrent Neural Network (RNN) is a type of neural network that can process sequential data by retaining information from previous inputs. LSTM is a type of RNN architecture that is designed to overcome the problem of vanishing gradients in traditional RNNs [10]. The vanishing gradient problem occurs when the gradients of the weights in the network become extremely small as they are propagated back through time, making it difficult for the network to learn long-term dependencies. Figure 3.1 and Algorithm 1 present the layout and detailed steps involved in building an LSTM layer.

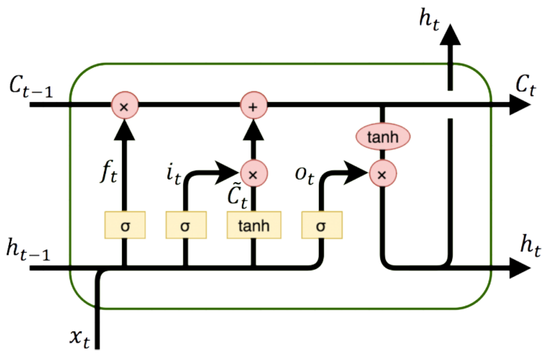


Figure 3.1: Structure of an LSTM layer [11]

In this algorithm, the forward pass of a Long Short-Term Memory (LSTM) network is presented, which is a type of recurrent neural network (RNN) architecture. Given an input sequence $\mathbf{x}_{1:n}$ and an initial hidden state $\mathbf{h}_0$, the LSTM computes a sequence of hidden states $\mathbf{h}_{1:n}$ and output values $\mathbf{y}_{1:n}$.

The LSTM maintains a cell state vector $\mathbf{C}_t$ that can store information over long periods of time. At each time step t, the LSTM computes four gates that control how information is processed and propagated through the network. The input gate $\mathbf{i}_t$ determines how much of the input at time t is added to the cell state, the forget gate $\mathbf{f}_t$ determines how much of the previous cell state should be retained, and the output gate

Algorithm 1 LSTM Network

```
1:  function LSTM($\mathbf{x}_{1:n}$, $\mathbf{h}_0$)
2:      Initialize LSTM weights $\boldsymbol{\theta} = \mathbf{W}_{xi}, \mathbf{W}_{hi}, \mathbf{b}_i, \mathbf{W}_{xf}, \mathbf{W}_{hf}, \mathbf{b}_f, \mathbf{W}_{xo}, \mathbf{W}_{ho}, \mathbf{b}_o,$
3:  $\mathbf{W}_{xg}, \mathbf{W}_{hg}, \mathbf{b}_g, \mathbf{W}_{hy}, \mathbf{b}_y$
4:      for $t = 1$ to $n$ do
5:          $\mathbf{i}_t \leftarrow \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$        ▷ Input gate
6:          $\mathbf{f}_t \leftarrow \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$        ▷ Forget gate
7:          $\mathbf{o}_t \leftarrow \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$        ▷ Output gate
8:          $\tilde{\mathbf{C}}_t \leftarrow \tanh(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)$        ▷ Cell gate
9:          $\mathbf{C}_t \leftarrow \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$        ▷ Cell state
10:         $\mathbf{h}_t \leftarrow \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$        ▷ Hidden state
11:         $\mathbf{y}_t \leftarrow \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y$        ▷ Output value
12:     end for
13:     return $\mathbf{y}_{1:n}$, $\mathbf{C}_n$, $\mathbf{h}_n$
14: end function
```

$\mathbf{o}_t$ determines how much of the current cell state should be output as the hidden state $\mathbf{h}_t$. The cell gate $\tilde{\mathbf{C}}_t$ determines how much of the input at time t should be added to the cell state.

Prior to the forward pass of the LSTM network, the model parameters need to be initialized. The parameters consist of weight matrices and bias vectors that are used to transform the input and hidden state at each time step. Specifically, the weight matrices include $\mathbf{W}_{xi}$, $\mathbf{W}_{hi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{xg}$, $\mathbf{W}_{hg}$, and $\mathbf{W}_{hy}$, while the bias vectors are $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, $\mathbf{b}_g$, and $\mathbf{b}_y$. Together, these parameters form a set denoted as $\boldsymbol{\theta}$. The initialization step sets the initial values of all parameters in $\boldsymbol{\theta}$.

At each time step t, the LSTM computes the input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, output gate $\mathbf{o}_t$, and cell gate $\tilde{\mathbf{C}}t$ using a sigmoid activation function σ and the input at time t and the previous hidden state $\mathbf{h}_{t-1}$. The cell state $\mathbf{C}_t$ is updated using the input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, and cell gate $\tilde{\mathbf{C}}_t$. The hidden state $\mathbf{h}_t$ is computed using the output gate $\mathbf{o}_t$ and the updated cell state $\mathbf{C}_t$. Finally, the output value $\mathbf{y}_t$ is computed as a linear function of the hidden state $\mathbf{h}_t$. The algorithm returns the sequence of output values $\mathbf{y}_{1:n}$, the final cell state $\mathbf{C}_n$, and the final hidden state $\mathbf{h}_n$.

3.2 Encoder Decoder models

Sequence-to-sequence (Seq2Seq) networks are a type of deep learning model that can process sequences of variable length and generate output sequences of variable length.

They are often used for tasks such as machine translation, speech recognition, and text summarization, and could be potentially adaptable for the current task i.e., speed planning for vehicle platooning. This section describes a type of Seq2Seq network, called the encoder decoder LSTM network, which has been appropriately adopted for thsi problem.

An encoder-decoder LSTM network is a specific type of encoder-decoder architecture that uses LSTM units to capture temporal dependencies in sequential data, such as natural language or time-series data. This architecture consists of an encoder LSTM network that transforms the input sequence into a fixed-length vector representation and a decoder LSTM network that generates the output sequence (Fig. 3.2).

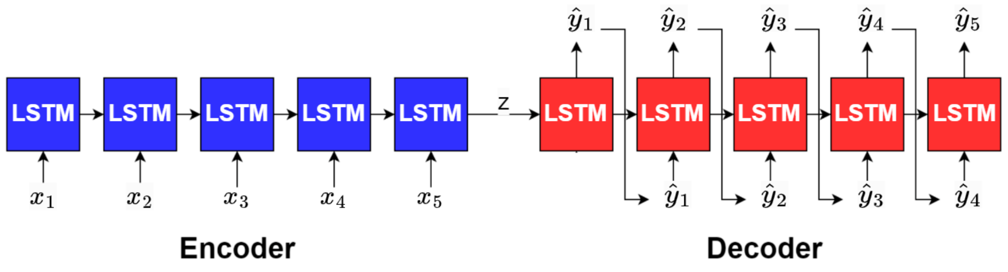


Figure 3.2: Encoder decoder LSTM model structure

One of the key advantages of these networks is their ability to handle variable-length input and output sequences, which is important for tasks such as machine translation where the input and output sequences can have different lengths in different languages. Additionally, they can capture long-term dependencies in the input sequence which allows them to generate more accurate and coherent output sequences. They are a powerful and flexible approach to sequence modeling that have been successfully applied to a wide range of natural language processing tasks.

CHAPTER 4

Use case 1: Speed prediction model for route planning

In this study, a sequential data model that is suitable for predicting the speed profile of a leader vehicle in the platoon system based on the instantaneous power input and the operating conditions is presented.

4.1 Linear regression based model

Any machine learning solution is started with the simplest possible model and then depending on the result, proceeded to more complex models. Hence, a linear regression model was implemented on the dataset. The RMSE came out to be **27.96 km/h** and the MAPE was **62.04 %**. The plot of predictions from this model over various operating conditions is shown in Fig 4.1.

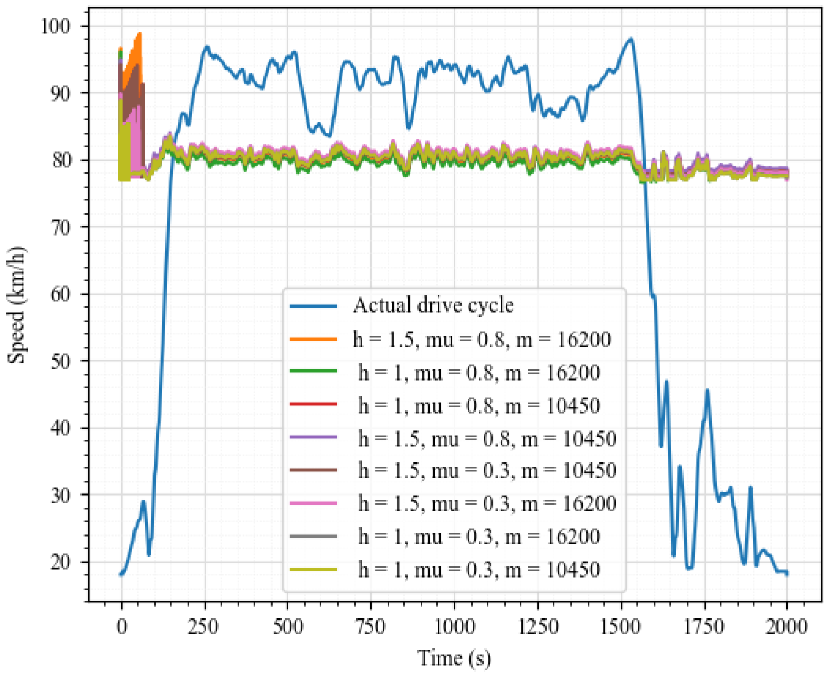


Figure 4.1: Linear Regression model - prediction plot

From the results, it is clear that the performance is unsatisfactory. This can be attributed to 2 reasons:

1. A linear model is not enough to express the relationship between the input and the output. Non-linear relations require more complex models to represent them.
2. An important assumption in a linear regression model is that the features are independent of time. The relation between the values at an instant in this case is dependent on the previous instances. This is not taken into account in this linear regression model.

The next section describes a model which solves the above two issues and performs better than the linear regression model.

4.2 Encoder decoder LSTM model

From the above linear regression results, it is clear that a more advanced model that captures the sequential nature of the input and the output is necessary. Hence, it was decided to move forward with the encoder decoder LSTM model as the use case aligns with that of a seq2seq model.

4.2.1 Model architecture

Let $P(t)$ be the power at any time instant t. Every drive cycle was divided into groups of an arbitrary number N. If the total instances were not divisible by the group size, zero padding was applied before the first group. For example, the validation drive cycle has 208300 instances. For $N = 500$, 200 instances of 0 were prefixed to this drive cycle so that it becomes divisible by 500.

Consider an arbitrary group j of N instances $\hat{\mathbf{p}}_j = [p(t), p(t+\Delta t), ..., p(t+(N-1)\Delta t)]$ where, Δt is the sampling time i.e., 0.01s. The initial few observations of each group will be missing significant information and dependency from previous instances as they are in the previous group. To accommodate this, every input sequence will be prefixed with an arbitrary number (M) of instances from the previous group (from the end). The final input sequence was then $\mathbf{p}_j = [p(t-M\Delta t), ..., p(t-\Delta t), p(t), p(t+\Delta t), ..., p(t+(N-1)\Delta t)]$ with length $(M + N)$. This sequence $\mathbf{p}_j$ is given as input to the encoder LSTM. This generates a hidden contextual representation of the power sequence. Mathematically, the encoder LSTM can be represented as follows:

$$\mathbf{h}_{et}, \mathbf{C}_{et} = \text{LSTM}_{\text{encoder}}(p(t), \mathbf{h}_{e(t-1)}, \mathbf{C}_{e(t-1)}) \quad \text{where } t \in [t - M\Delta t, t + (N-1)\Delta t], \tag{4.1}$$

where, $\mathbf{h}_{et}$ is the hidden state of the encoder LSTM at time t, $\mathbf{c}_{et}$ is the cell state of the encoder LSTM at time t, and $\text{LSTM}_{\text{encoder}}$ is the LSTM function with trainable parameters $\boldsymbol{\theta}_{\text{e}}$. The initial hidden state $\mathbf{h}_{e(t-(M-1)\Delta t)}$ and cell state $\mathbf{C}_{e(t-(M-1)\Delta t)}$ are both set to zero.

The final hidden state $\mathbf{h}_{e(t+(N-1)\Delta t)}$ represents the fixed-length vector representation of the input sequence, which contains the most important information necessary for generating the output. However, the operating conditions have not been incorporated yet. To accommodate the same, the output of the encoder LSTM (which encompasses the hidden representations) was combined with the operating conditions and passed into the decoder LSTM. The encoder LSTM's final hidden state is given as $\mathbf{h}_{e(t+(N-1)\Delta t)}$ and the cell state is given by $\mathbf{C}_{e(t+(N-1)\Delta t)}$. Next step is to include the operating conditions and pass it as initial decoder input along with the hidden representations. The initial decoder input can be given as $[h_{d0}, C_{d0}]$ where,

$$\mathbf{h}_{d0} = [\mathbf{h}_{e(t+(N-1)\Delta t)}, \mu_j, H_j, m_j], \tag{4.2}$$

$$\mathbf{C}_{d0} = [\mathbf{C}_{e(t+(N-1)\Delta t)}, \mu_j, H_j, m_j]. \tag{4.3}$$

Here, μ_j, H_j, m_j are the friction coefficient, time-headway and the mass respectively of that corresponding group.

An iteration of the decoder LSTM can be represented as follows:

$$\hat{y}_j(i), \mathbf{h}_{di}, \mathbf{C}_{di} = \text{LSTM}_{\text{decoder}}(\hat{y}_j(i-1), \mathbf{h}_{d(i-1)}, \mathbf{C}_{d(i-1)}), \tag{4.4}$$

where, $\hat{y}_j(i)$ is the i^{th} decoder output of this group j, $\hat{y}_j(i-1)$ is the output at the previous instant, $\mathbf{h}_{di}$ and $\mathbf{C}_{di}$ are the hidden state and cell state of the decoder LSTM, $\text{LSTM}_{\text{decoder}}$ is the LSTM function with trainable parameters $\boldsymbol{\theta}_{\text{d}}$.

The decoder LSTM generates an output of a size equal to the length of the hidden representations. Hence, it was passed into a fully connected linear layer for the final output to match the dimensions of the input group i.e., N, which essentially is the speed of the leader vehicle at the same instances of that particular input group (excluding the previous instances included).

If $\hat{v}(t)$ represents the predicted speed of the truck at an instant t, the final output $\hat{\mathbf{v}}_j = [\hat{v}(t), ..., \hat{v}(t + (N-1)\Delta t)]$, where, j is the index of that particular group (input is $\mathbf{p}_j$) is given by

$$\hat{\mathbf{v}}_j = \mathbf{W}_v \hat{\mathbf{y}}_j + \mathbf{b}_v, \tag{4.5}$$

where, $\hat{\mathbf{y}}_j$ is the vector representing the decoder output of this group. The complete model structure of the above discussed encoder decoder LSTM for platoon speed prediction is given in Algorithm 2.

Algorithm 2 Encoder-Decoder LSTM forward pass

```
1: Input: Power sequence for a group j ($\mathbf{p}_j$), Operating conditions of that group ($\mu_j, H_j, m_j$), LSTM encoder and decoder parameters ($\text{LSTM}_{\text{encoder}}(\boldsymbol{\theta_e}), \text{LSTM}_{\text{decoder}}(\boldsymbol{\theta_d})$), Dense layer parameters ($\mathbf{W_v}, \mathbf{b_v}$)
2: Output: Predicted speeds $\hat{\mathbf{v}}_j$
3: function EncDecLSTM_ForwardPass($\mathbf{p}_j, \mu_j, H_j, m_j, \boldsymbol{\theta_e}, \boldsymbol{\theta_d}, \mathbf{W_v}, \mathbf{b_v}$)
4:     $\mathbf{h}_{e(t-(M-1)\Delta t)} \leftarrow \mathbf{0}$        ▷ Initial hidden state of encoder LSTM
5:     $\mathbf{C}_{e(t-(M-1)\Delta t)} \leftarrow \mathbf{0}$        ▷ Initial cell state of encoder LSTM
6:     for $t = t - M\Delta t$ to $t + (N-1)\Delta t$ do
7:         $\mathbf{h}_{et}, \mathbf{C}_{et} = \text{LSTM}_{\text{encoder}}(p(t), \mathbf{h}_{e(t-1)}, \mathbf{C}_{e(t-1)})$        ▷ Encode input sequence, take only hidden and cell state
8:     end for
9:     $\mathbf{h}_{d0} = [\mathbf{h}_{e(t+(N-1)\Delta t)}, \mu_j, H_j, m_j]$        ▷ Including operating conditions to the hidden representation
10:    $\mathbf{C}_{d0} = [\mathbf{C}_{e(t+(N-1)\Delta t)}, \mu_j, H_j, m_j]$        ▷ Assigning initial decoder input as final encoder states
11:    $\hat{y}_j(0) \leftarrow 0$        ▷ Initial predicted target vector
12:    for $i = 1$ to $LSTM\_Units$ do
13:        $\hat{y}_j(i), \mathbf{h}_{di}, \mathbf{C}_{di} = \text{LSTM}_{\text{decoder}}(\hat{y}_j(i-1), \mathbf{h}_{d(i-1)}, \mathbf{C}_{d(i-1)})$        ▷ Decode hidden state of encoder LSTM
14:    end for
15:    $\hat{\mathbf{v}}_j = \mathbf{W}_v \hat{\mathbf{y}}_j + \mathbf{b}_v$        ▷ Prediction of speed using dense layer
16:    return $\hat{\mathbf{v}}_\mathbf{j}$
17:
18: end function
```

4.2.2 Training algorithm

The previous section described the structure of the encoder decoder LSTM model used in this study. This section describes the training procedure of the same.

The backpropagation algorithm for LSTM consists of two main steps as follows: Computing the gradients of the error with respect to the weights (which involves computing various other gradients and applying chain rule), and then using those gradients to update the weights of the network. Computing the gradients involves taking in all the parame-

Table 4.1: Hyperparmeter description

Hyperparameter name/symbol	**Description**
M	Number of instances from previous group prefixed to the current group as input
N	Number of instances taken per group for an input sequence
learning_ rate	Learning rate in the gradient update equation
LSTM_ Layers	Number of LSTM layers in encoder and the decoder
LSTM_ Units	Dimensionality of the LSTM output space i.e., hidden representation z
epochs	Number of iterations
batch_ size	Number of sequences processed before updating the weights

ters as well as the sequences as inputs and gives the gradients of the loss function with respect to all the learnable parameters using chain rule [12]. Once these gradients have been computed, the next step is to update the weights with these values. The complete training process is shown in Algorithm 3 and has been used to train the models. The concise list of all the hyperparameters involved with this model is shown in Table 4.1.

.

Algorithm 3 Encoder-Decoder LSTM training

```
1: for $i = 1$ to epochs do
2:     Initialize $\Delta\boldsymbol{\theta_e}, \Delta\boldsymbol{\theta_d}, \Delta\mathbf{W}_v, \Delta\mathbf{b}_v \leftarrow 0$    ▷ $\Delta\theta_{\mathbf{e}} = \frac{\partial L}{\partial \theta_{\mathbf{e}}}$
3:     Randomly sample batch_ size no. of sequences
4:     for every $(\mathbf{p}_j, \mathbf{v}_j)$ in the above sampled set do
5:         $\hat{\mathbf{v}}_j$ = EncDecLSTM_ForwardPass($\mathbf{p}_j, \mu_j, H_j, m_j, \boldsymbol{\theta_e}, \boldsymbol{\theta_d}, \mathbf{W}_v, \mathbf{b}_v$)
6:         $\Delta\boldsymbol{\theta_e}, \Delta\boldsymbol{\theta_d}, \Delta\mathbf{W}_v, \Delta\mathbf{b}_v \mathrel{+}= \left.\frac{\partial L}{\partial \boldsymbol{\theta_e}}, \frac{\partial L}{\partial \boldsymbol{\theta_d}}, \frac{\partial L}{\partial \mathbf{W}_v}, \frac{\partial L}{\partial \mathbf{b}_v}\right]_{\mathbf{p}_j, \mu_j, H_j, m_j, v_j, \hat{\mathbf{v}}_j}$    ▷ Computing gradients
7:     end for$[\boldsymbol{\theta_e}, \boldsymbol{\theta_d}, \mathbf{W}_v, \mathbf{b}_v] \leftarrow [\boldsymbol{\theta_e}, \boldsymbol{\theta_d}, \mathbf{W}_v, \mathbf{b}_v]$ - learning_ rate*$[\Delta\boldsymbol{\theta_e}, \Delta\boldsymbol{\theta_d}, \Delta\mathbf{W}_v, \Delta\mathbf{b}_v]$    ▷ Weight update
8: end for
9: return $\boldsymbol{\theta_e}, \boldsymbol{\theta_d}, \mathbf{W}_v, \mathbf{b}_v$
```

4.2.3 Baseline model

A baseline encoder-decoder LSTM model was initially created and used as a starting point for further improvements. The input sequence consisted of 100-time instances and was prefixed with the last 5 instances from the previous input sequence (M is 5 and N is 100). The implementation was carried out in Python®, utilizing the Keras [13] and

TensorFlow [14] libraries. The default learning rate of 0.001 from the TensorFlow package was chosen for this model.

It is important to note that the selection of hyperparameters is typically determined by experimentation and tuning on a validation set. An approach to improve the baseline model can be to start with a small value for a hyperparameter and gradually increase it until the performance on the validation set stopped improving or began to degrade. Based on this strategy, a single LSTM layer was selected for both the encoder and decoder, with each layer containing 64 units. The model was trained for 20 epochs with a batch size of 8.

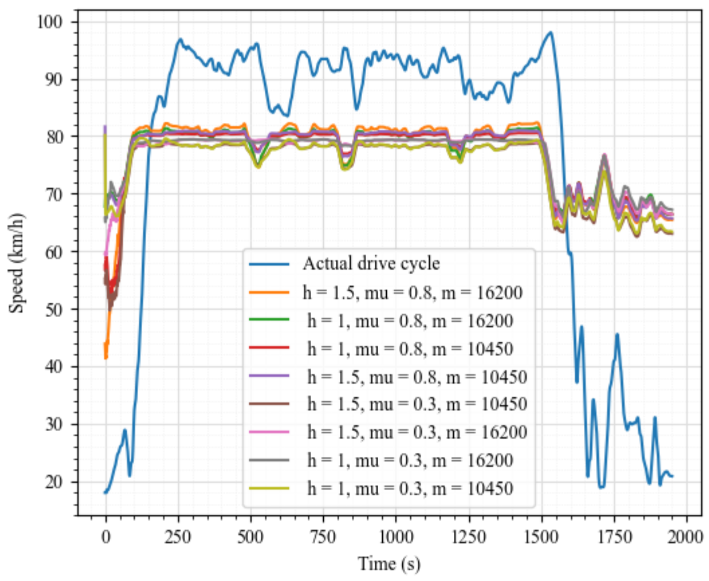


Figure 4.2: Baseline encoder decoder LSTM model - prediction plot

Since the aerodynamic drag is proportional to the square of the vehicle speed, the performance of a platoon is most effective and efficient only at higher speeds. Hence, it is reasonable to assess the performance of the platoon only during the high-speed portions. Considering this fact, the performance metrics were calculated only from 250 s to 1500 s in the validation dataset where the platoon is in high-speed operation. The RMSE was computed and found to be **23.02 km/h**. The MAPE was obtained to be **25.87%**. Note that the metrics were calculated averaging over all the operating conditions. The plots for every operating condition is shown in Fig 4.2. These values served as the benchmark for further improvements through hyperparameter tuning, which is discussed in the subsequent subsection.

4.2.4 Hyperparameter Tuning and Final Model

Hyperparameter tuning is a crucial step in deep learning that involves finding the optimal values of the hyperparameters that govern the behavior of the model. It helps in improving model performance, reducing overfitting, and ensuring faster convergence. It is important to experiment with different hyperparameters and evaluate their impact on the model's performance to find the optimal values that best suit the specific problem at hand. The range of values for each hyperparameter is listed as shown in the second column in Table 4.2.

Table 4.2: Hyperparameter tuning and the final model

Hyperparameter	**Range of values used for tuning**	**Parameter value in final model**
M	1, 5, 10, 50, 100, 500	5
N	10, 50, 100, 500, 1000, 5000	500
learning_ rate	0.00001, 0.001, 0.01, 0.1	0.001
LSTM_ Layers	1, 2, 3	1
LSTM_ Units	64, 128, 256, 512, 1024	256
epochs	10, 20, 50, 100	10
batch_ size	8, 16, 32, 64	8

From the baseline model with the values of M and N respectively being 5 and 100, different combinations of M and N were tested in both directions (higher and lower). The best model resulted when M is 5 and N is 500.

The learning rate is an important hyperparameter to tune because it affects the convergence speed and accuracy of the model. The learning rate trade-off refers to the balance between the convergence speed and the accuracy of the model during the training process. In general, a high learning rate leads to faster convergence but may cause the optimization algorithm to overshoot the optimal solution and lead to oscillations or even divergence. On the other hand, a low learning rate leads to a more accurate solution, but it may take a longer time to converge and can be sensitive to initialization. Hence, a range of values from 0.00001 to 0.1 was considered and the best model had a learning rate of 0.001.

In general, the number of units of the LSTM layer determines the number of memory cells in the layer, which enables the model to capture long-term dependencies in the input sequence. A larger value can potentially improve the model's capacity to learn complex patterns in the data but also requires more computational resources and may lead to overfitting if the training data is limited. Considering this, values from 64 to 1024 were chosen for tuning and the model with 256 units gave the best performance.

The number of LSTM layers was tuned in increments of 1, starting from a single layer. It was observed that as the number of layers increased, there was no improvement in performance. Hence, one layer each for the encoder and decoder was chosen.

Following a similar strategy for the other hyperparameters (epochs and batch_size), the model with the most optimum performance metric was chosen. The final model bears the hyperparameters shown in the last column of Table 4.2. The RMSE on validation data for this model was obtained to be **7.64 km/h**, which is the best among all the models trained. Concurring with this, the MAPE for this model was obtained to be **7.25%**, the lowest among the trained models. Hence, this was finalized as the best model. The plots for this model is shown in Fig 4.3. The metrics for indiviual operating conditions are described in Table 4.3.

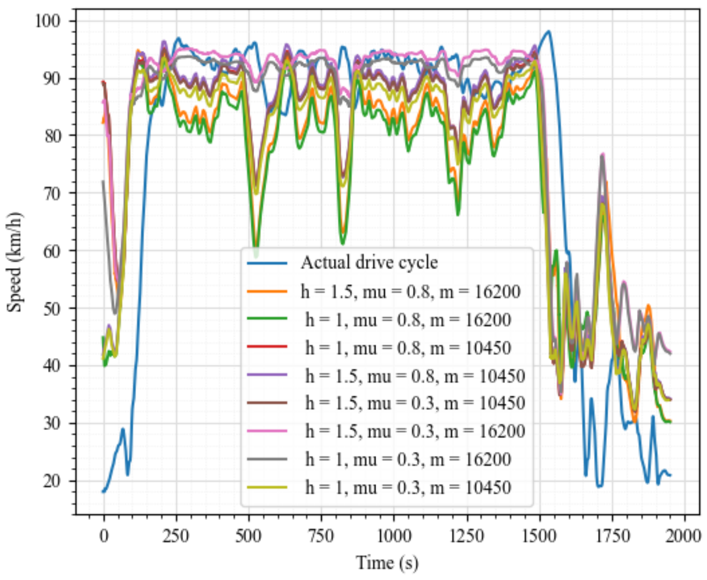


Figure 4.3: Hyperparameter tuned final model - prediction plot

Figure 4.4 shows the predictions of the baseline model and the final model on the validation data along with the actual drive cycle for $\mu = 0.3$, $h = 1$ s, $m = 16200$ kg. From the plot, it can be observed that hyperparameter tuning improved the performance of the model and the final model fits better than the baseline model, which was underfitting on the validation dataset.

Table 4.3: Speed prediction final model - Performance analysis

Road Condition	Time headway	Vehicle Mass	RMSE (km/h)	MAPE (%)
Dry ($\mu = 0.8$)	h=1	▶	6.94	5.96
		▷	13.03	14.33
	h=1.5	▶	7.01	5.96
		▷	11.67	12.21
Wet ($\mu = 0.3$)	h=1	▶	8.09	7.49
		▷	3.51	2.84
	h=1.5	▶	7.05	6.09
		▷	3.86	3.07

▷ - Fully laden, ▶ - Partially laden.

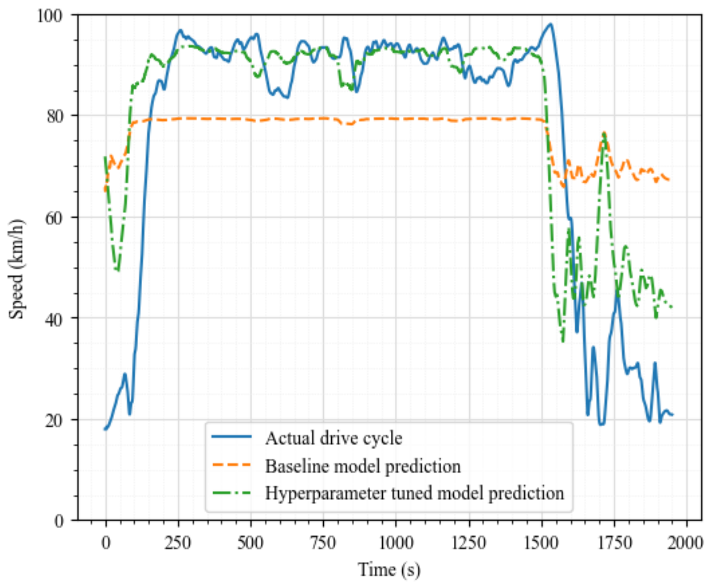
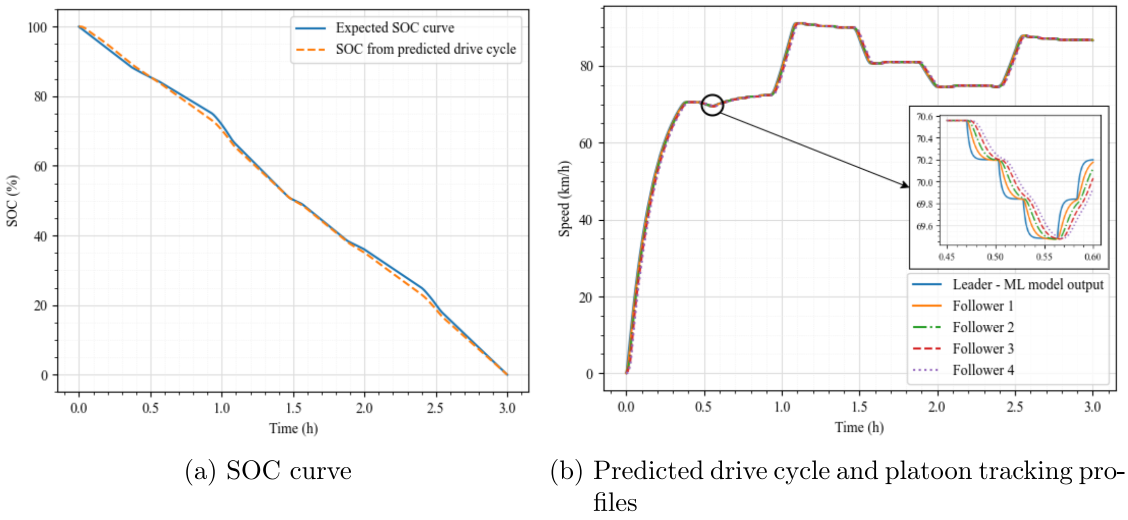


Figure 4.4: Comparison between baseline model and final model predictions for the case $\mu = 0.3, h = 1, m = 16200$

4.3 Results and Discussion

The proposed DL-based framework could predict the electric truck platoon's speed profiles/drive cycles to suit a particular driving condition, route and constraints based on a particular energy demand profile. Such a profile could be generated, considering multiple factors like prior route information, vehicle and battery capacity, and availability of charging infrastructures. For instance, an arbitrary battery usage profile as shown in Fig. 4.5(a) (presented in terms of SOC curve) was generated for a fully laden electric truck ($m = 16200$ kg) equipped with a battery pack capacity of 270 kWh operating on a dry level road. All the electric truck parameters as presented in [15] have been used to simulate the vehicles. The battery pack was assumed to be able to satisfy the power demanded by the electric motor irrespective of the SOC magnitude, and the vehicle was driven continuously till the whole battery is exhausted (which is ≈ 3 h). This should give an approximate range of 220 - 280 km based on various road conditions.



(a) SOC curve

(b) Predicted drive cycle and platoon tracking profiles

Figure 4.5: Prediction of drive cycle for an arbitrary SOC profile

Figure 4.5(b) presents the predicted speed profile corresponding to an individual truck's energy demand as given by Fig. 4.5(a). The platoon is then made to follow the predicted/generated speed profile, and the tracking plots are presented in Fig. 4.5(b) as predicted by the controller shown in Fig 2.1. The autonomous controller ensured stable platoon operation and the follower trucks were able to follow the predicted drive cycle without collisions. The platoon could travel ≈240 km for the presented full discharge cycle. The actual SOC profile corresponding to the leader truck based on the framework is also presented (red curve in Fig. 4.5(a)), which closely follows the demanded SOC curve. This results show the capability of the framework in predicting the drive cycles for the electric truck platoon such that the platoon is string stable and each truck in the platoon is able to track the available SOC profile to meet battery constraints.

4.4 Demonstrated case study: Electric truck platoon routing problem

The presented framework could potentially be used to formulate policy decisions such as route selection between two locations based on multiple factors like road type, route information, and constraints such as road elevation, speed limitations, and availability of charging infrastructures. Figure 4.6 presents such a routing problem where an electric truck platoon has been planned to travel between two warehouses ‘A’ and ‘B’ with options to travel 4 different routes, Route 1 to Route 4. Each route was assumed to be having different lengths, (with Route 1 being the longest and Route 3 being the shortest) and were equipped with charging stations at varying locations. Each vehicle in the platoon is assumed to be fully loaded with a battery pack capacity of 270 kWh with an approximate range of 220 - 280 km from a fully charged condition. Each charging station (as presented in Fig. 4.6) is assumed to be equipped with fast charging technology which could fully charge the batteries in $\approx$ 2 h. The speed profiles were predicted considering the time constraint, platoons taking any route from ‘A’ to ‘B’ should have similar travel time.

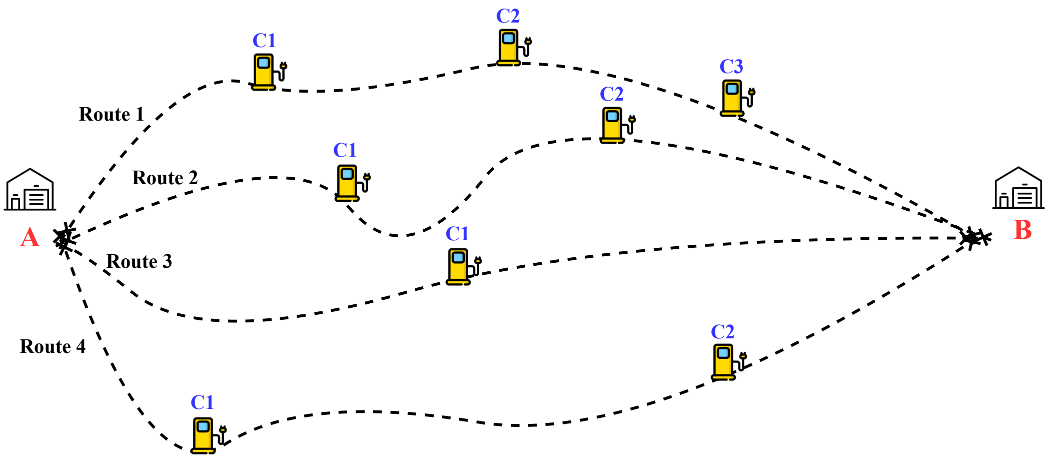


Figure 4.6: HGV platoon routing problem schematic (not to scale).

The framework predicts the speed corresponding to each route, and charging location, and time to reach the destination. For instance, the platoon has to travel faster compared to other routes while travelling route 1 to satisfy the time constraint because route 1 is the longest. Route 1 has three charging stations, where the batteries are fully charged and the platoon can resume its travel. The predicted speed profiles are presented in Fig. 4.7. The idling profiles during charging are not shown for brevity. Similarly, for Route 3, one can opt to drive at lower speeds. This is also necessary as the charger location is much farther, and one has to opt for lower speeds, lower acceleration/deceleration rates to minimise the energy consumption and extend range. The model could predict the speed profiles adopting to the route requirements and constraints, giving the fleet operator to choose a route of their convenience and capabilities. One should note that the presented routing problem is not absolute, but just a sample case to show the capability of the proposed approach. The framework has the flexibility to adopt its parameters incorporating wide range of operating conditions and constraints to give policy decisions for optimal electric truck routing.

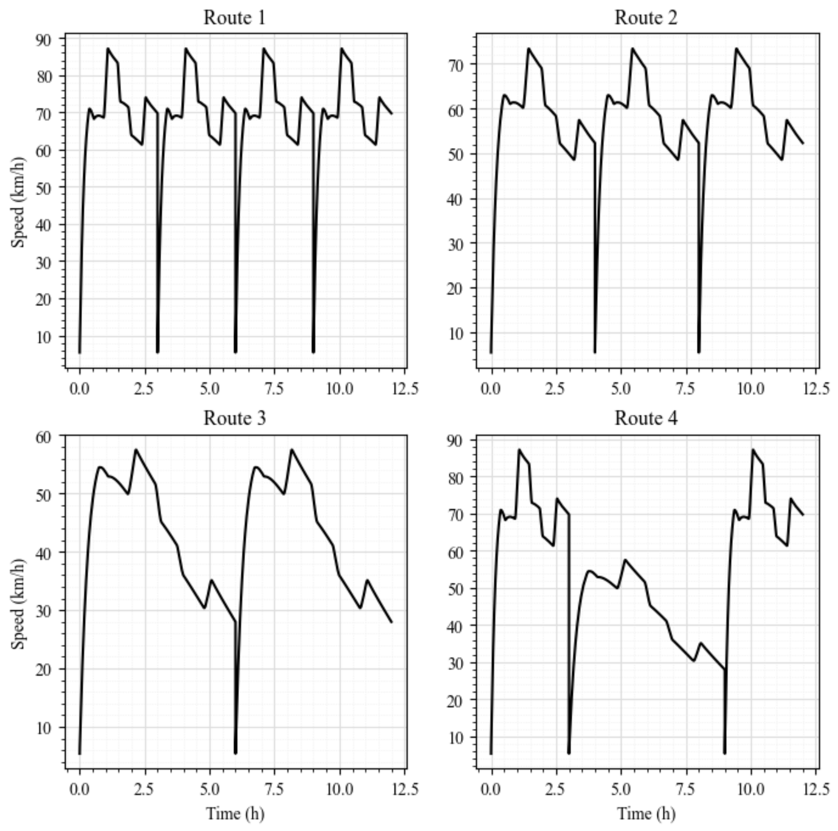


Figure 4.7: Predicted drive cycles for different routes

In addition to predicting the drive cycles, the presented framework could also give different descriptive parameters representing the drive cycle called kinematic parameters [6, 16]. The parameters for the considered case are computed as proposed by [16] and are presented in Table 4.4. These kinematic parameters could give further insights into the driving patterns to be followed and the capabilities and constraints of individual vehicles in the platoon. These parameters could also be used as a quantitative measure to represent different road and traffic conditions [16]. Sixteen critical parameters characterising each predicted drive cycles (for each route) were computed and presented. The predicted drive cycles have average speeds of 69.1, 57.5, 42.7, and 55.9 km/h for Routes 1 - 4, respectively, with Route 1 being faster and Route 3 being slower with maximum speeds of 87.1 km/h, and 57.5 km/h, respectively. The acceleration and deceleration values have direct impact on the energy consumption. Route 3, having to travel more distance to the next charging point should consume less power compared to other three routes. In addition to driving at lower speeds, it is required to have smaller acceleration and deceleration

Table 4.4: Kinematic parameters for the predicted drive cycles

Group	**Parameter**	**Units**	**Route 1**	**Route 2**	**Route 3**	**Route 4**
Distance related	Total distance	km	830	690.5	512.2	671.1
Time related	Total time	s	43200	43200	43200	43200
	Drive time spent accelerating	s	12592.8	11538.29	9939.024	11154.672
	Drive time spent decelerating	s	30607.09	31661.57	33260.76	31810.64
	% of time accelerating	%	29.15	26.71	23.00	25.82
	% of time decelerating	%	70.85	73.29	76.99	73.64
Speed related	Average speed	km/h	69.166	57.542	42.690	55.93
	Standard deviation of speed	km/h	11.758	9.869	10.323	16.84
	Maximum speed	km/h	87.137	73.405	57.485	87.137
Acceleration related	Average acceleration	m/s^2	0.000414	0.000301	0.000145	0.000819
	Average positive acceleration	m/s^2	0.00862	0.0057	0.00343	0.00642
	Average negative acceleration	m/s^2	-0.00296	-0.0017	-0.00084	-0.0013
Stop related	Number of stops	-	4	3	2	3
	Number of stops per km	/km	0.00482	0.004345	0.003904	0.00447
	Average stop duration	s	7200	7200	7200	7200
	Average distance between stops	km	207.5	230.2	256.1	223.7

rates compared to other routes. The average acceleration parameter in Table 4.4 corroborates this fact, where the value corresponding to Route 3 was approximately 2.86, 2.07, and 5.65 times smaller compared to that of Route 1, 2, and 3, respectively. Apart from using the kinematic parameters for drive cycle/driving profile characterisation, these parameters could give further insights into the route, charging location placement, required battery capacity to complete a particular route and further design strategies.

CHAPTER 5

Use case 2: Energy prediction of follower vehicles

The previous chapter described a model to predict the expected drive cycle of the leader vehicle from an expected SOC/power profile along a route. In this chapter, the opposite use case is discussed. The power cycle will be predicted given the drive cycle. Since the main motivation of this use case came from the fact that accounting for air drag would not be as efficient as a data driven model, it is imperative to do this analysis for the follower vehicles. Hence, the output power cycle will be that of the first follower vehicle.

From the previous chapter, it was clear that linear regression models aren't suitable for sequential data as it does not account for the dependencies between instances. Hence, for this use case, an encoder decoder LSTM model was used to begin the analysis.

5.1 Encoder decoder LSTM model

The model was constructed very similar to the previous use case, the only change being the interchange of input and output. Hence, the hyeprparemeters M and N in this case denote power at instances instead of speed.

5.1.1 Baseline model

A baseline encoder-decoder LSTM model was initially created and used as a starting point for further improvements. The input sequence consisted of 100-time instances and was prefixed with the last 5 instances from the previous input sequence (M is 5 and N is 100). The implementation was carried out in Python®, utilizing the Keras [13] and TensorFlow [14] libraries. The default learning rate of 0.001 from the TensorFlow package was chosen for this model.

The RMSE was computed and found to be **26.17 kW**. The MAPE was obtained to be **24.59%**. Note that the metrics were calculated averaging over all the operating

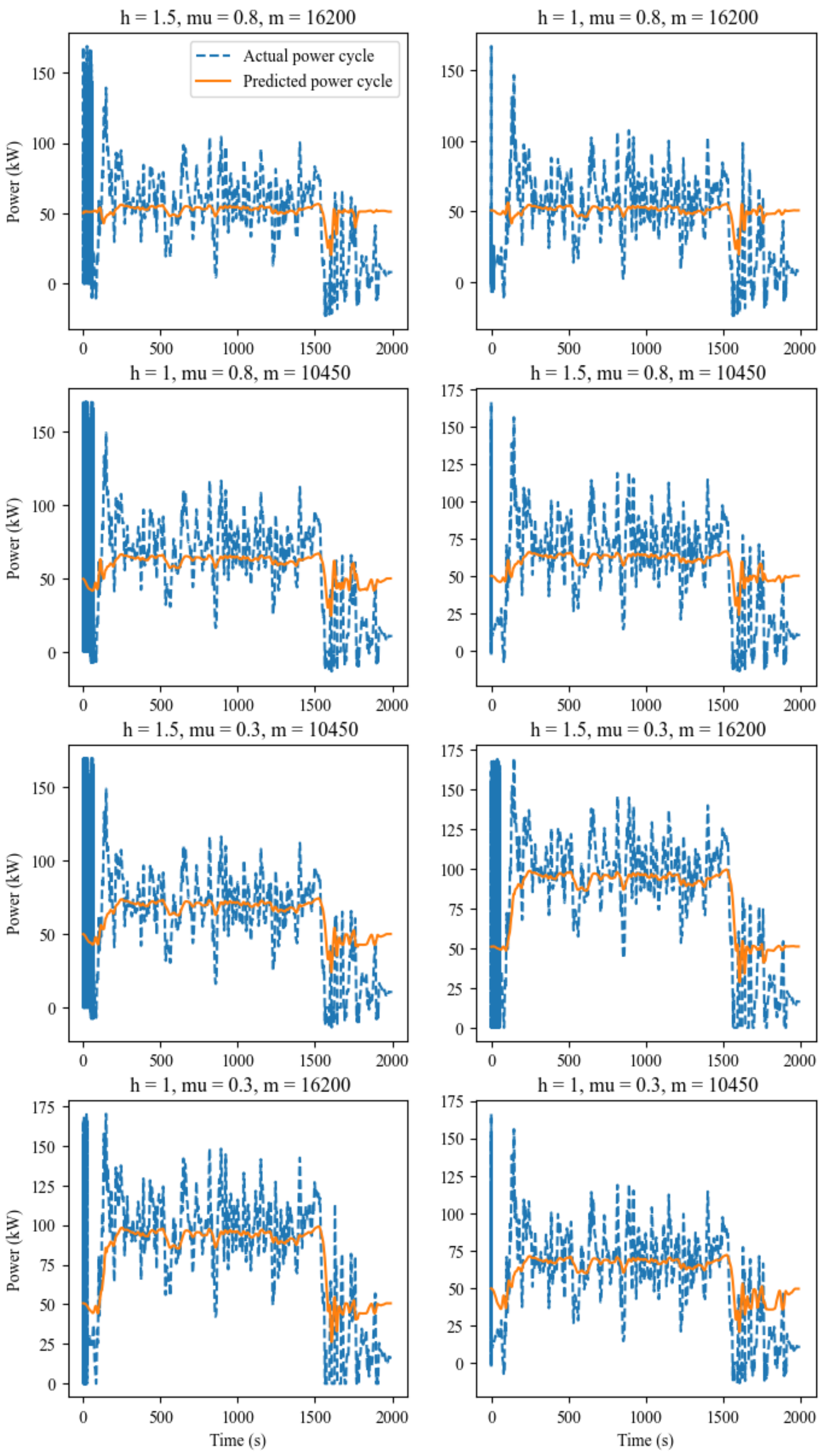


Figure 5.1: Baseline model plots - Energy prediction

conditions. The plots for every operating condition is shown in Fig 5.1. These values served as the benchmark for further improvements through hyperparameter tuning, which is discussed in the subsequent subsection.

5.1.2 Hyperparameter tuning and final model

Table 5.1: Hyperparameter tuning and the final model

Hyperparameter	**Range of values used for tuning**	**Parameter value in final model**
M	1, 5, 10, 50, 100, 500	50
N	10, 50, 100, 500, 1000, 5000	500
learning_ rate	0.00001, 0.0001, 0.001, 0.01, 0.1	0.0001
LSTM_ Layers	1, 2, 3	1
LSTM_ Units	64, 128, 256, 512, 1024	256
epochs	10, 20, 30, 50, 100	30
batch_ size	8, 16, 32, 64	8

The range of values for each hyperparameter is listed as shown in the second column in Table 5.1. Hyperparameter tuning was done similar to the previous chapter and the best model came out to have MSE of **13.33 kW** and MAPE of **6.95 %**. The detailed split of each case is shown in Table 5.2. The plots for this model are shown in Fig 5.2. Note that the best model hyperparameters in the previous use case and this case are different, thereby implying the importance of tuning.

Table 5.2: Energy prediction final model - Performance analysis

Road Condition	**Time headway**	**Vehicle Mass**	**RMSE (km/h)**	**MAPE (%)**
Wet ($\mu = 0.3$)	h=1	▶	10.69	5.63
		▷	18.82	8.82
	h=1.5	▶	14.20	7.71
		▷	14.47	6.50
Dry ($\mu = 0.8$)	h=1	▶	15.69	8.50
		▷	10.27	6
	h=1.5	▶	11.09	5.82
		▷	11.42	6.60

▷ - Fully laden, ▶ - Partially laden.

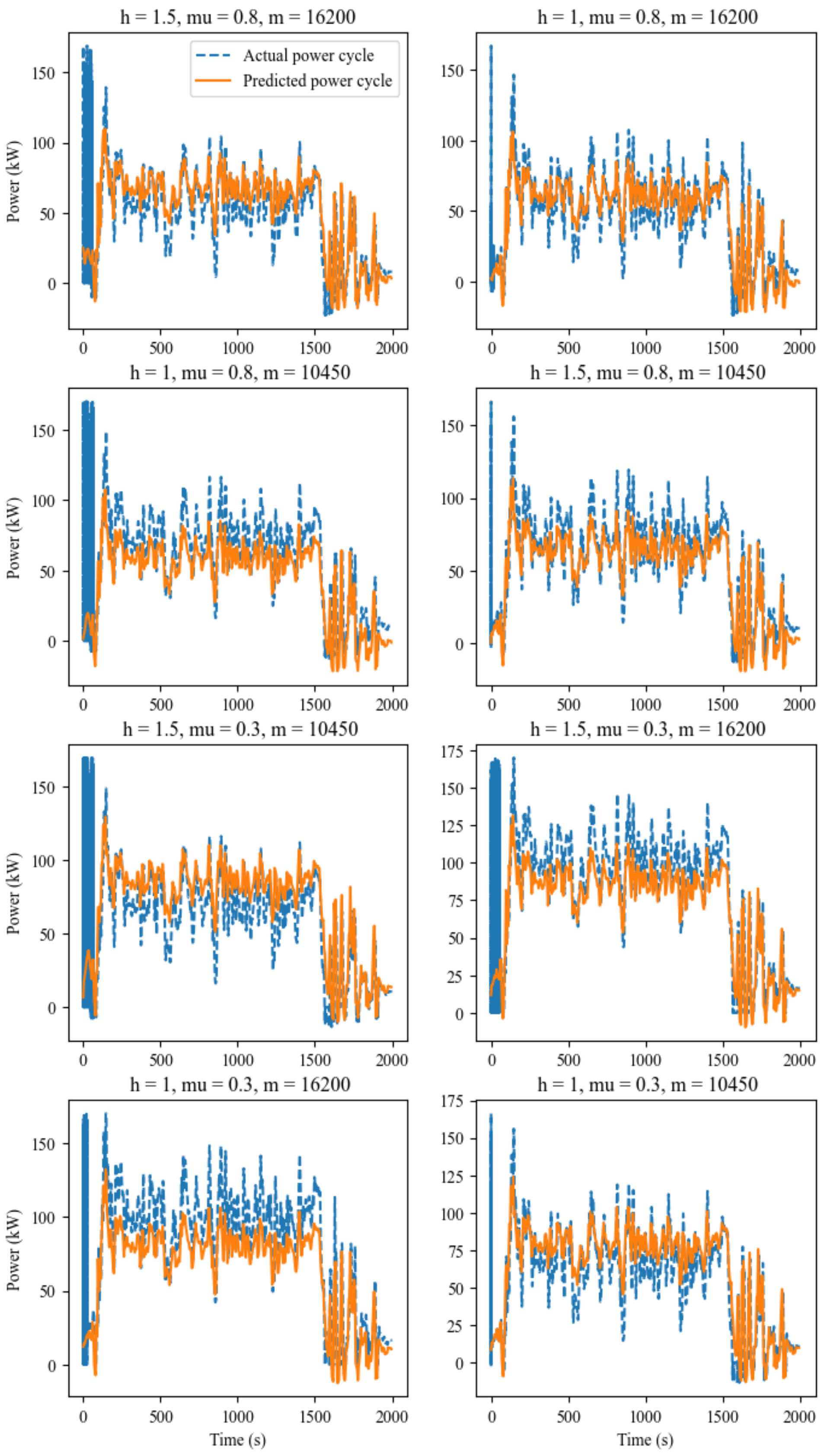


Figure 5.2: Final model plots - Energy prediction

CHAPTER 6

Conclusion and future scope

A systematic approach towards truck platoon route planning for long-haul freight movement has been addressed utilizing data driven techniques. A couple of use cases were listed for which data driven solutions can be devised. The first one is the prediction of drive cycle of the leader vehicle from expected SOC profile that can be obtained from battery data/historical data of other vehicles. A linear regression model was initially implemented. Based on the results, a more complex encoder decoder LSTM model was implemented which was then hyperparameter-tuned to obtain the final model for this application. An example case study for this use case which pertains to route planning using battery power and charging station constraints was also demonstrated.

The next use case that was addressed in this project is the prediction of power cycle for a route. This was done for the follower vehicle as the main motivation of this use case is to account for the efficiency of platooning. For this use case too, an encoder decoder LSTM model was implemented and tuned similar to the previous case and the best model among the ones was finalized.

The overall final outcome of this project is that Deep Learning models can indeed be trained and used to model platoons instead of a first principles model to improve computational efficiency wherever required.

Future work includes investigating other use cases such as prediction of minimum required time headway for stability and efficiency. Moreover, an appropriate model can be made to check the string stability of a platoon. Different model architectures can also be experimented for all the use cases.

REFERENCES

[1] Platooning roadmap, ACEA - European Automobile Manufacturers' Association.
URL `https://www.acea.auto/files/Platooning_roadmap.pdf`

[2] Truck platooning: History, benefits, future, trucks.cardekho.com/.

[3] M. Martínez-Díaz, C. Al-Haddad, F. Soriguera, C. Antoniou, Platooning of connected automated vehicles on freeways: a bird's eye view, Transportation Research Procedia 58 (2021) 479–486, xIV Conference on Transport Engineering, CIT2021. `doi:https://doi.org/10.1016/j.trpro.2021.11.064`.
URL `https://www.sciencedirect.com/science/article/pii/S2352146521008231`

[4] K. B. Devika, G. Rohith, V. R. S. Yellapantula, S. C. Subramanian, A dynamics-based adaptive string stable controller for connected heavy road vehicle platoon safety, IEEE Access 8 (2020) 209886–209903.

[5] DieselNet, Emission test cycles: European transient cycle.
URL `https://dieselnet.com/standards/cycles/etc.php`

[6] T. Barlow, S. Latham, I. S. McCrae, P. Boulter, A reference book of driving cycles for use in the measurement of road vehicle emissions, TRL Published Project Report (2009).

[7] United States Environmental Protection Agency, Dynamometer drive schedules: Highway fuel economy.
URL `https://dieselnet.com/standards/cycles/etc.php`

[8] DieselNet, Emission test cycles: Heavy heavy-duty diesel truck schedule.
URL `https://dieselnet.com/standards/cycles/hhddt.php`

[9] X. Wan, Influence of feature scaling on convergence of gradient iterative algorithm, in: Journal of physics: Conference series, Vol. 1213, IOP Publishing, 2019, p. 032021.

[10] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997) 1735–1780.

[11] W. Commons, File:lstm.png — wikimedia commons, the free media repository, [Online; accessed 8-March-2023] (2020).
URL `https://commons.wikimedia.org/w/index.php?title=File:LSTM.png&oldid=469457581`

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

[13] F. Chollet, et al., Keras, `https://keras.io` (2015).

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, et. al., TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`

[15] K. B. Devika, G. Rohith, S. C. Subramanian, String stable control of electric heavy vehicle platoon with varying battery pack locations, Journal of Vibration and Control 28 (5-6) (2022) 577–592.

[16] P. De Haan, M. Keller, Modelling fuel consumption and pollutant emissions based on real-world driving patterns: the hbefa approach, International journal of environment and pollution 22 (3) (2004) 240–258.