

UNIVERSITY OF NEVADA, RENO

# **Optimization and Anomaly Detection for Smart City-based Applications**

A dissertation submitted in partial fulfillment of the requirements for the  
degree of Doctor of Philosophy in Computer Science and Engineering

by

Raj Mani Shukla

Dr. Shamik Sengupta / Dissertation Advisor

May 2020

© 2020 Raj Mani Shukla  
ALL RIGHTS RESERVED

UNIVERSITY  
OF NEVADA  
RENO

**THE GRADUATE SCHOOL**

We recommend that the dissertation prepared  
under our supervision by

**RAJ MANI SHUKLA**

entitled

**Optimization and Anomaly Detection for Smart City-based  
Applications**

be accepted in partial fulfillment of the  
requirements for the degree of

**DOCTOR OF PHILOSOPHY**

Shamik Sengupta, Ph.D. – Advisor

Monica Nicolescu, Ph.D. – Committee Member

David Feil-Seifer, Ph.D. – Committee Member

Lei Yang, Ph.D. – Committee Member

Shahriar Badsha, Ph.D. – Committee Member

Hao Xu, Ph.D. – Graduate School Representative

David Zeh, Ph.D. – Dean, Graduate School

May 2020

# Abstract

Smart cities are envisioned to include million of sensors and devices tied together through the Internet. The sensors generate huge amount of data that can be potentially used to develop several applications. Connected vehicles can be considered as a significant realm of the smart city. Furthermore, among connected vehicles, Plug-in Electric Vehicles (PEVs) are becoming an integral component of smart city. Plug-in Electric Vehicles (PEVs) can play a major role in reducing carbon footprints in transportation sector. However, with the growing increase in the usage of PEVs, their charging infrastructure need to be comprehensively developed.

This thesis develops optimization strategies for parked and en route PEV charging. First, the thesis develops a charging strategy for parked vehicles. For parked vehicle charging, electricity demand must have less variation at different times in a day. This helps in avoiding any detrimental effect on power electronic equipment, reduces electricity cost, and simplifies the electric energy demand prediction from the smart grid. This thesis delineates an intelligent aggregator architecture to automate parked Plug-in Electric Vehicle (PEV) charging in large parking places to reduce dynamic load variation. The thesis

presents a set of novel rectangles placement-based algorithms to schedule the PEV charging based on their arrival, departure time at parking place, and their charging requirement.

En route PEVs have different requirements as compared to parked PEVs. To facilitate en route PEV charging, this research investigates an integrated COP architecture. The COP constitutes a Communication unit that provides network management solution for PEV charging, an Optimization unit for allocating charging station to PEVs, and a Prediction unit to determine traffic flow information in advance. The thesis shows how three units can interact with each other to provide efficient charging infrastructure to PEVs.

A related problem regarding application service development in smart city is the anomaly detection. The detection of any variation in environmental parameters (anomalies/outliers) is important problem for efficient functioning of automated application services. However, the false anomalies by malicious manipulation of sensor data may have adverse effect on such services. Thus, for connected communities, this research explores these two interrelated problems: 1) detection of anomalies and 2) classification between real anomalies from manipulated anomalies.

For solving anomaly detection problem, this thesis presents a scalable anomaly detector that uses Hierarchical clustering in conjunction with Long Short-Term Memory (LSTM) neural network. Hierarchical clustering provides scalability to the anomaly detector by finding correlated sensors. The LSTM neural network is coupled with the robust statistics, M-estimator, to accurately detect outliers in time-series data.

The thesis proceeds to discuss the distinction between anomalies occurring due to external environmental variations or intentional manipulation/faults in sensors. Towards this direction, our research has focused on using the idea that heterogeneous sensors may not directly affect a certain measured quantity. However, if measured values from individual sensors are combined then that may have correlation with a certain value. Specifically, this research uses air quality, wind-speed, and temperature to predict vehicular traffic. Based on the predicted quantity, we have employed rule-based statistical approach to detect malicious anomalies.

Dedicated to family and  
everyone who I shared this  
journey with.

# Acknowledgements

I would like to sincerely thank first and foremost my advisor, Dr. Shamik Sen-gupta, for the patience and support throughout this journey. I could not have imagined having a better advisor and mentor, who have guided me through meetings and discussions to help me reach this point. I would like to thank my committee: Dr. Monica Nicolescu, Dr. David Feil-Seifer, Dr. Lei Yang, Dr. Shahriar Badsha, and Dr. Hao Xu for their feedback and suggestions.

I would also like to acknowledge the important role played by the ones that surrounded me through these years. In particular, all my fellow lab mates, with whom I had mind opening discussions and was able to have other perspectives to my problems. The Graduate Student Association for providing invaluable programs to help students excel and which I was a consistent user of.

In addition, I would like to thank the Department of Computer Science and Engineering, University of Nevada, Reno for their financial support. I would like to acknowledge the role of NIT Kurukshetra and BIET Jhansi in my life for building a strong initial base standing on which I could achieve my doctoral degree. Last, but definitely not the least, I would especially like to thank my family for the constant emotional and moral support that they have provided and without whom I would not have reached so far.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Contents</b>	<b>vi</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Plug-in Electric Vehicle (PEV) charging	3
1.1.1 Electricity demand curve	3
1.1.2 Charging time requirement	4
1.2 Anomaly detection	5
1.2.1 Scalability issue	6
1.2.2 Distinguishing true anomaly with injected anomaly	7
1.3 Thesis organization	8
<b>2 Background studies</b>	<b>9</b>
2.1 Connected vehicles in smart city	9
2.2 PEV charging	12
2.2.1 Charging infrastructure development	12
2.2.2 Scheduling of PEV charging	13
2.3 Anomaly detection techniques	15
2.3.1 PCA-based anomaly detection techniques	16
2.3.2 Anomaly detection based on statical characteristics	19
2.3.3 Clustering-based anomaly detection techniques	22
<b>3 Parked Plug-in Electric Vehicle (PEV) charging</b>	<b>26</b>
3.1 System model	28
3.2 Methodology	30
3.2.1 Problem statement	30
3.2.2 PEV schedule using rectangle placement	33

	Main algorithm . . . . .	35
	Initial rectangle placement . . . . .	37
	Voltage level selection . . . . .	38
	Rectangle shift . . . . .	39
3.3	Results . . . . .	41
3.3.1	Simulation parameters . . . . .	42
3.3.2	Analysis of power variation . . . . .	43
3.3.3	Timing analysis . . . . .	44
3.4	Observations and remarks . . . . .	45
<b>4</b>	<b>En route Plug-in Electric Vehicle (PEV) charging</b>	<b>47</b>
4.1	System model for en route PEV charging . . . . .	49
4.1.1	Smart grid and charging station architecture . . . . .	51
4.1.2	COP architecture . . . . .	54
4.2	Methodology . . . . .	55
4.2.1	Prediction unit design . . . . .	56
	Correlation analysis . . . . .	56
	Neural network for deep learning . . . . .	59
	Input feature vector selection . . . . .	63
	Backpropagation learning . . . . .	64
4.2.2	Optimization unit design . . . . .	66
	Problem statement . . . . .	67
	Set cardinality-based search . . . . .	70
	Complexity analysis . . . . .	73
4.3	Results . . . . .	75
4.3.1	Performance metrics . . . . .	77
4.3.2	Prediction unit results . . . . .	77
4.3.3	Optimization unit results . . . . .	79
4.3.4	Timing analysis . . . . .	81
4.4	Observations and remarks . . . . .	83
<b>5</b>	<b>Anomaly detection</b>	<b>84</b>
5.1	Preliminaries . . . . .	87
5.1.1	Problem statements . . . . .	87
5.1.2	Hierarchical clustering . . . . .	88
5.1.3	LSTM neural network . . . . .	89
5.2	Anomaly detector architecture . . . . .	90
5.3	Methodology . . . . .	91
5.3.1	Clustering module . . . . .	93
	Principal Component Analysis . . . . .	93
	Hierarchical clustering . . . . .	95
	Dendrogram and number of clusters . . . . .	96

	Cluster center . . . . .	99
5.3.2	Anomaly detection . . . . .	99
	Segmentation . . . . .	100
	M-Estimator . . . . .	100
	LSTM neural network . . . . .	102
	LSTM neural network design . . . . .	105
	Anomaly detector . . . . .	106
5.4	Results . . . . .	108
5.4.1	Materials and methods . . . . .	108
5.4.2	Evaluation metrics . . . . .	109
5.4.3	Performance evaluation . . . . .	110
5.4.4	Performance analysis as parameter $\alpha$ is tuned . . . . .	113
5.4.5	Performance analysis as cluster size varies . . . . .	113
5.4.6	Comparison with other approaches . . . . .	115
5.4.7	Computation complexity . . . . .	120
5.5	Observations and remarks . . . . .	120
<b>6</b>	<b>Performance analysis of a traffic prediction application in the presence of malicious anomalies</b>	<b>121</b>
6.1	System model for performance analysis . . . . .	123
6.2	Prediction performance . . . . .	125
6.3	Observations and remarks . . . . .	126
<b>7</b>	<b>Malicious anomaly detection</b>	<b>127</b>
7.1	Attack model, problem statement, and solution motivation . . . . .	129
7.1.1	Attack model . . . . .	129
7.1.2	Problem statement . . . . .	131
7.1.3	Solution motivation . . . . .	131
7.1.4	Ensemble learning . . . . .	133
7.2	System architecture for malicious anomaly detection . . . . .	134
7.2.1	Data preprocessing . . . . .	135
7.2.2	Machine learning model . . . . .	138
7.2.3	Thresholding . . . . .	140
7.2.4	Counter . . . . .	140
7.3	Evaluation results . . . . .	141
7.3.1	Data . . . . .	141
7.3.2	Base learner parameters . . . . .	142
	MLP . . . . .	143
	LSTM . . . . .	143
	CNN . . . . .	144
	CNN_LSTM . . . . .	144
7.3.3	Evaluation metrics . . . . .	145

7.3.4	Prediction results . . . . .	146
7.3.5	Anomaly detection . . . . .	148
	Analysis as magnitude is varied . . . . .	148
	Analysis as parameter $\beta$ is varied . . . . .	149
	Analysis as parameter $K$ is varied . . . . .	150
	Analysis as percentage of anomalies vary . . . . .	151
7.4	Observations and remarks . . . . .	152
<b>8</b>	<b>Conclusions and future research direction</b>	<b>153</b>
	<b>Bibliography</b>	<b>155</b>

# List of Tables

3.1	Simulation parameters for parked PEV charging . . . . .	40
4.1	Simulation parameters for en route PEV charging . . . . .	76
4.2	Performance for training at one location and prediction at other . . . . .	78
4.3	Prediction unit timing analysis . . . . .	81
5.1	Analysis for positive anomalies . . . . .	110
5.2	Analysis for negative anomalies . . . . .	111
5.3	Analysis as percentage of anomalies vary . . . . .	112
7.1	Mean Average and Mean Squared error for CO as input variable . . . . .	146
7.2	Mean Average and Mean Squared error for BC as input variable . . . . .	147
7.3	Mean Average and Mean Squared error for NO <sub>2</sub> as input variable . . . . .	147
7.4	Mean Average and Mean Squared error for NO <sub>x</sub> as input variable . . . . .	148
7.5	Mean Average and Mean Squared error for PM <sub>2.5</sub> HR as input variable . . . . .	148
7.6	Performance measurement as anomaly magnitude is varied . . . . .	149
7.7	Performance measurement as parameter $\beta$ is varied . . . . .	150
7.8	Performance measurement as parameter $K$ is varied . . . . .	151

# List of Figures

1.1	Vehicular traffic variation . . . . .	6
3.1	System model for parked PEV charging . . . . .	29
3.2	Rectangle placement . . . . .	34
3.3	Total power . . . . .	35
3.4	Load profile . . . . .	41
3.5	Average power variation per time slot as number of PEVs are varied . . . . .	42
3.6	Average slope variation as number of PEVs are varied . . . . .	43
3.7	Load factor as number of PEVs are varied . . . . .	44
3.8	Execution time . . . . .	45
4.1	COP architecture . . . . .	49
4.2	System model for en route PEV charging . . . . .	50
4.3	Correlation analysis . . . . .	57
4.4	Flow matrix . . . . .	63
4.5	A comparison of actual and predicted traffic . . . . .	78
4.6	Prediction error . . . . .	78
4.7	RMSE values at different locations . . . . .	79
4.8	Average error at different locations . . . . .	79
4.9	RMSE with multi-step ahead prediction . . . . .	80
4.10	Average error with multi-step ahead prediction . . . . .	80
4.11	Average charging time as number of vehicles are varied with constant number of charging stations . . . . .	81
4.12	Average charging time as number of charging stations are varied with constant number of vehicles . . . . .	81
4.13	Execution time of the proposed algorithms . . . . .	82
5.1	Traffic count at different locations . . . . .	86
5.2	Architecture describing anomaly detection process . . . . .	90
5.3	Dendrogram explaining cluster formation . . . . .	97
5.4	Euclidean distance between clusters . . . . .	98
5.5	LSTM neural network . . . . .	102
5.6	Comparison of performance metrics as $\alpha$ varies . . . . .	112

5.7	Performance metrics as euclidian distance varies . . . . .	114
5.8	Number of models as euclidian distance is varied . . . . .	115
5.9	Precision comparison for traffic data . . . . .	116
5.10	Recall comparison for traffic data . . . . .	117
5.11	F-measure comparison for traffic data . . . . .	117
5.12	Precision comparison for pollutant data . . . . .	118
5.13	Recall comparison for pollutant data . . . . .	118
5.14	F-measure comparison for pollutant data . . . . .	119
6.1	Effect of attack as magnitude of attack is varied . . . . .	124
6.2	Effect of attack as percentage of values in input vector are varied	124
7.1	The model describing the attack surface . . . . .	130
7.2	Malicious anomaly detector architecture . . . . .	135
7.3	Metrics variation as percentage of anomalies vary . . . . .	151

# Chapter 1

## Introduction

The concept of smart cities is evolving to the new idea of Smart and Connected Communities (SCC) that incorporate smart homes, smart transportation systems, smart grids, health-care services, smart learning, all tied together through the Internet connection. The integration assists in various forms like crowd management in cities, reducing greenhouse gases, promoting new business models, making government agencies responsive and efficient, improving livability and quality of life. For instance, in a connected community, the appliances in smart homes can take the information from user's travel patterns and switch on (or off) the appliances like HVAC system [1]. The smart grid uses traffic information in the city and helps in demand-side management (DSM) of electric supply. Smart transportation assists local authorities in traffic management and helps people adopt safe driving practices in cities. Farmers can be provided information about weather, crop conditions, and market price which may help them decide the best time to harvest their crops and best place to sell them for maximum profit. The SCC foster sustainability and quality of life, enhances security and safety of people and environment, and promote a better

quality of life [2]. It also boosts profit for businesses, aids in reducing losses, and advocates better health-care and emergency services.

The smart cities has been enabled due to the presence of a millions of sensors and devices. These sensors seamlessly measure environmental parameters and transfers them to servers like edge or cloud. The efficient information transfer is made possible due to the Internet connectivity and the technologies like 5G communication and Software-defined Networks (SDN). Using the data generated from the sensors, several automated applications and services can be developed for the users. Often, the application development requires an optimization method that performs allocation or scheduling tasks. For example, in ride-sharing services, data obtained from users, vehicles, and traffic sensors is used to schedule then pick up, drop off times and vehicle route. Similarly, several other application services can be developed to assist users in their day to day activities.

Smart and connected vehicles are one of the significant realm of smart city. In the connected vehicle scenario, the sensors dispersed throughout the city measure the environmental parameters and the data is transferred to the centralized infrastructure. Smart transportation is one of the principle component of smart city that not only eases driver's experience but also ensures safety and sustainability in cities. Connected vehicles include Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure(V2I) enabling plethora applications like real-time navigation, improving freight efficiency, adjusting speed limit based on real world conditions, optimize fuel consumption and many more. In connected vehicles, use of Plug-in Electric Vehicles (PEVs) is growing at a very rapid rate. PEVs require their charging infrastructure and development of applications to assist

in scheduling their PEV charging. Anomaly detection is another problem related to application development in smart city. Anomaly is any deviation in measured data from sensors. This deviation can occur due to variation in environmental factors. The optimization algorithms making decision for a certain application rely on these anomalies to take proactive action. This helps in better and efficient service provisioning of the applications. However, both the PEV charging and anomaly detection poses certain challenges in the context of smart city that are described below.

## **1.1 Plug-in Electric Vehicle (PEV) charging**

en route PEV charging is one of the application that requires attention due to the popularity of electric vehicles as a transportation choice. PEVs provide green and cost-effective transportation services and have the ability to mitigate the effect of greenhouse gas emissions. Additionally, PEVs are also useful as the vehicles have smooth acceleration and make less noise [3]. However, in contrast to the traditional vehicles, PEV adoption also poses several challenges as mentioned below.

### **1.1.1 Electricity demand curve**

The PEV adoption also influences the electricity demand curve due to the power required for their charging. The large-scale penetration of PEVs will add up the electricity demand putting pressure on the power grid. The electricity demand pattern will also change as more PEVs will come into use. The PEV is mainly

charged when it is parked at a parking place. Imagine a scenario, where all the office workers will plug-in their electric vehicle after returning to their home. This will suddenly raise the electricity demand in the evening. Moreover, the large and unpredictable nature of PEV charging requirement also leads to sudden fluctuations in the load curve. This may adversely impact the power electronics instruments like transformers or voltage controllers [4], [5] [6]. The sudden fluctuations also affect the devices connected to the same transmission line. Therefore, the PEV charging needs to be coordinated in such a way that it reduces peak power requirement.

Furthermore, the power requirement curve should be smooth to avoid the devices and instruments to be affected badly. Coordination of parked PEV charging can assist in improving load profile by distributing PEV charging of parked vehicles. The availability of different level of charging such as AC Level 1, AC Level 2 and AC Level 3 or DC charging can be leveraged to smoothen load profile [7]. For example, during peak load, the PEVs can be charged slowly using the low level of charging and fast charging when the load is low.

### **1.1.2 Charging time requirement**

The PEV charging, in contrast to traditional gasoline refilling, requires considerable time to charge. A large charging time hinders user from the adoption of vehicles that entirely rely on electricity as the source of energy. Therefore, the adoption of PEVs requires the large-scale development of charging stations in the smart city. The charging stations also need to be installed depending upon the expected traffic at different locations of the city, its proximity to different

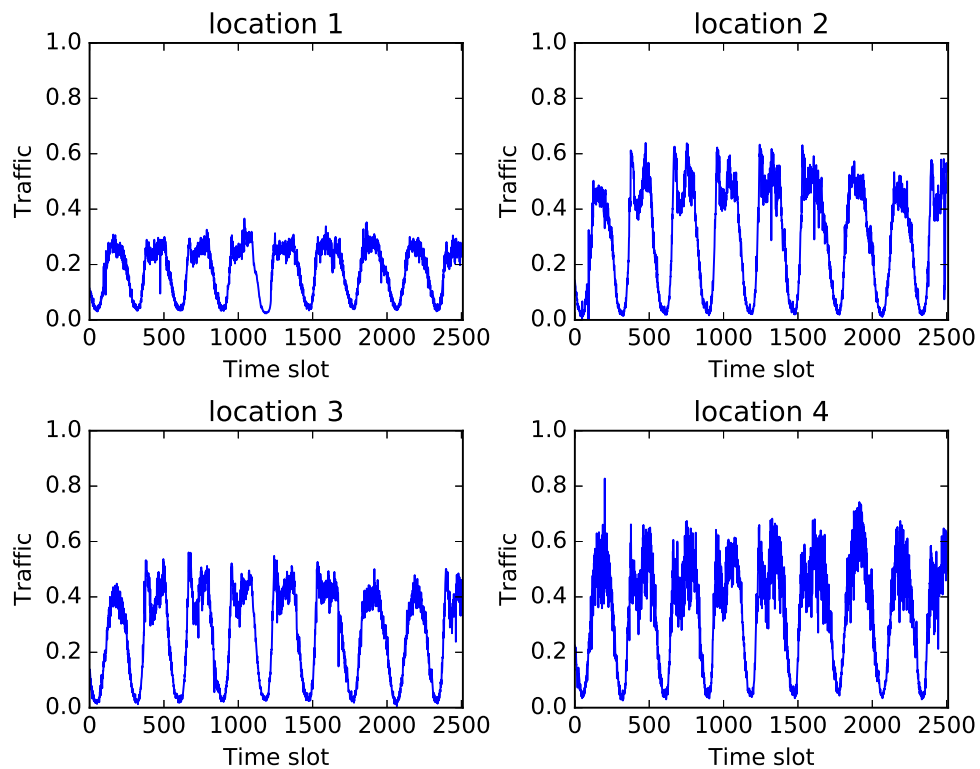
grids to avoid power losses and even user demand. The problem of PEV charging time is crucial mostly for the en route PEVs. Apart from installing charging stations, the coordination of en route PEV charging can significantly reduce their charging time. For example, if a large number of PEVs approaches few congested charging stations, then their charging time may be substantial. On the other hand, some charging stations may not be efficiently utilized concerning their capacity. Therefore, dispersing en route PEVs among a large number of charging stations have a potential to reduce the charging time with given charging demand and capacity. The problem of distributing charging stations to en route PEVs is also related to the traffic prediction problem. For instance, if the future traffic congestion near the location of the different charging station is known in advance, then it may assist in allocating charging stations to the PEVs. Thus, the accurate determination of traffic counts at different locations is a closely related problem that also needs to be addressed.

## **1.2 Anomaly detection**

The automated services in smart city necessitate very fine-grained anomaly detection for quality service provisioning. Anomaly detection helps in better decision making by applications. For example, application services like vehicle routing, ride-sharing services, PEV charging can use the anomalies in the data and make decisions based on the abnormal values. For example, depending upon traffic congestion, that is an anomaly, a vehicle can be re-routed to a different path. However, in smart city, due to growing number of sensors, the

anomaly detection problems are difficult because of different reasons as described below.

### 1.2.1 Scalability issue



**Figure 1.1:** Vehicular traffic variation

The volume and variability of data require a scalable and fine-grained solution to handle the anomaly detection problem. However, the anomaly detection for IoT require scalable solution independent of sensor values. To explain this, consider the vehicular traffic data obtained at 4 different locations in California I-680 freeway for the month of October 2017 and plotted in Figure 1.1. The data is measured in granularity of 5 minutes. As depicted in figure, the statistical

properties, that have often been used for anomaly detection, cannot be easily modeled and employed for all locations due to their variation. The presence of large number of distributed sensors makes the problem of anomaly detection difficult as a separate model need to be developed per sensor. To solve the problem, the development of an scalable and robust anomaly detection model is critical.

### **1.2.2 Distinguishing true anomaly with injected anomaly**

Another problem with the anomaly detection in the smart city is the injection of malicious anomalies. The maliciously injected anomaly can affect the performance of the automated applications. Optimization algorithm that relies on the data from sensors will detect a certain point as anomalous, while in actual the anomaly has not occurred. Thus, the optimization algorithm will make inefficient decisions because of the malicious anomalies in the data. The attacker can come up with a varied range of strategies to evade anomaly detection.

The opportunistic attacker can come with other strategies like targeting sensors providing correlated information. Since the statistical properties are often used for anomaly detection, the attack strategy may be such that the statistical properties are preserved. The attacker may optimize added noise data using the adversarial technique such that the error in the corresponding application performance is maximized. Such type of attacks are hard to detect even with the well-known methods [8], [9]. Thus, differentiating between real and maliciously generated anomalies is an important problem in smart cities that need to be researched out.

## 1.3 Thesis organization

The rest of this report is organized as follows.

- Chapter 2 describes a background study on connected vehicles, PEV charging, and anomaly detection methods.
- Chapter 3 describes the optimization algorithms for parked PEV charging. This chapter discusses a smart aggregator architecture that takes data from different vehicles and based on that makes optimization decision.
- Chapter 4 describes the optimization method for the en route PEV charging. In this chapter, it is shown that how predictive analytics can assist in making efficient charging decision for en route PEV charging.
- Chapter 5 presents an scalable anomaly detector for the smart city scenario. The chapter discusses the use of hierarchical clustering to impart scalability and machine learning for accurate anomaly detection.
- Chapter 6 discusses the effect of malicious anomalies in the performance of the traffic prediction application. The chapter describes how the prediction error varies as anomaly strength and percentage varies.
- Chapter 7 portrays the method to detect maliciously anomalies. The chapter describes machine learning methods that predict vehicular traffic and based on the predicted values malicious anomalies are found.
- Chapter 8 discusses the conclusions of the given research and puts forward the future research ideas.

## Chapter 2

# Background studies

The smart and connected vehicles are one of the significant components in a smart city. Business Insider (BI) estimates that the 94 million connected cars to be shipped in by 2021 [10]. Thus, an estimated annual growth of connected vehicles is around 35% from 21 million connected cars in 2016. With the recent advancement in Plugged-in Electric Vehicle (PEV), the smart and green transport is also gaining a lot of interest for researchers and businesses [11]. Several auto companies like Tesla, Nissan and Chevrolet have started PEV production and it is expected that others will also expand their market share of PEV production [12].

### 2.1 Connected vehicles in smart city

Vehicular networks are characterized by a highly dynamic topology with the nodes (vehicles) moving at high speeds. The vehicles also move in different network conditions which include interstate highways with lower Internet coverage, congested traffic conditions, with network becoming slower due to a

large number of connected vehicles. Traffic in cities with tall buildings or severe weather conditions may also affect network connectivity.

The connected vehicle architecture which was presumed to offer seamless connectivity for vehicle to vehicle (V2V) and vehicle to infrastructure (V2I) is evolving towards the seamless connectivity between vehicle to pedestrian devices, vehicle to charging stations and vehicle to smart grid network. Today's connected cars include either an on-board connection to the network or connection is through a remote device like smart-phone. Also, tools like GPS are now built-in the car itself. Applications like *GasBuddy* or *Spotify* are further enhancing the driving experience [13]. Although the connected electric vehicles are primarily concerned with bringing safety for drivers, there are several driving forces for bringing Internet connection in vehicles as outlined below.

**Safety:** Car crashes are one of the major concerns in today's world. A study done by National Highway Traffic Safety Administration (NHTSA) estimates that about 80% of car crashes can be prevented by adopting connected vehicles [14]. Connected vehicles ameliorate safety of vehicles by the use of adaptive cruise control and warning systems. Warnings about the speed limit, red light crossings, and lane change can be issued to the drivers in real-time. The turn assistance and movement near stop or yield sign can be made easier using vehicular networks [15].

**Emergency services:** If a collision occurs, the emergency services like ambulance and police are informed immediately. Vehicles near the place of emergency are directed to divert or clear the path. This helps in avoiding congestion near the accident site. The evacuation team or fire services can coordinate better

to make things faster. Thus, the time for emergency services to reach accident site can be drastically reduced in the system of connected vehicles.

**Location based services:** Location-based services can help both drivers and businesses. Services like information about nearby hotels, restaurants or shopping places, parking places for electric vehicle can be quickly provided to drivers. Location-based services are also beneficial for businesses as they can advertise to nearby drivers about the available resources and their cost.

**Energy management:** A vehicular network inter-connected to smart grids helps in better energy management as the grid can predict the traffic conditions in cities and based on that appropriate amount of energy can be generated for electric vehicle charging. Rather than charging all the vehicles at the same time in residential area or office parking lots, vehicles can be scheduled for charging to improve the electric load profile.

**Charging services:** The route of a vehicle can be optimized thus avoiding congested traffic route to save electric vehicle's battery. Information from vehicle's status of charge (SOC) can be used to decide whether to charge it using AC or DC voltage level at a parking place thus optimizing time and cost of the user. Also, charging station allocation to en route vehicles based on their status of charge (SOC) can reduce their charging time and minimize cost [16].

**Cognitive assistance:** Cognitive assistance help drivers to apply brakes or slow down depending upon real-time information [17]. It can assist drivers in avoiding any risky situation. It can also help new or student drivers by guiding them to control vehicles based on the real-time information from nearby vehicles and environments.

It is expected that the future of transportation will be governed by self-driving or autonomous cars [18]. Self-driving cars will require seamless Internet connectivity for its realization. Further, cars with solar roof have the potential to provide green and cheaper transport. Solar roofs can enable the flow of electricity from the car to the grid, thus the fleet of cars can act as an energy source supplying electricity to the grid while they are not in use.

However, the wide-scale adoption of the connected cars and its advancement towards self-driving cars is still evolving and presents various challenges. The following sections discuss the communication and computational needs that can enable scalable solutions to the connected vehicles.

## **2.2 PEV charging**

There has been work done in literature on PEV charging which can be categorized into two main parts - 1) Development of charging infrastructure 2) Scheduling of PEV charging.

### **2.2.1 Charging infrastructure development**

Several works have investigated the development of charging infrastructure for PEVs. In [19], Chen et. al. has investigated the location of optimal charging stations to minimize PEV users' charging station access cost. This work has used the parking information to find the optimal location of charging stations in a particular region thus reducing the cost of locating a charging station. In [20], Bayram et al. has proposed the two frameworks for charging station planning

which ensures grid reliability and is also economical for capacity planning. The paper has proposed to control users' charging behavior by controlling the electricity price. The paper has also considered the minimum amount of charging resources to facilitate suitable amount of QoS to be met. In [21], Zhang et. al. has considered the interaction between transportation and electrical network. Zhang et. al. has used this interaction for optimal planning of charging stations. The proposed station location also increases the driving range of a PEV. Shahraki et al. has described a model which captures public charging profile in [22]. The charging profile is used to optimally locate a charging station such that the number of miles a vehicle can travel can be increased. In [23], Wang et. al. proposed a multi-objective planning strategy for charging station planning based on traffic flow. The proposed method reduces power loss and voltage deviation from the smart grid. The methodology maximizes the number of PEVs that can be charged in a given charging infrastructure.

### **2.2.2 Scheduling of PEV charging**

Several works have investigated optimal PEV scheduling to minimize cost or improve grid performance. Nafi et al. [24] described a grid to vehicle load management scheme that allocates electric power to maximize the number of vehicles that can be served. Nafi et al. has considered charging schedules during which the grid load is minimum. The paper has used an SDN-based scheme that allocates charging schedules to PEVs. However, in certain scenarios some PEVs can be deferred for charging to further optimize grid management scheme. To consider this, Lu et al. [25] has proposed the automatic

scheduling of deferrable and interruptible PEV/PHEV load in smart grid such that the grid operates within safety limits and maximum number of vehicles can be accommodated for charging. Mentioned works by Nafi et al. [24] and Lu et al. [25] maximizes the number of PEVs that can be served. However, regarding PEV charging the electricity generation cost by a service provider is also a concern which has not been addressed in above works.

To solve the problem, Chen et al. [26] has presented an energy management scheme based on network switched charging system. The scheme uses an on-line scheduler which controls the chargers and optimizes the profit of the service provider. Chen et al. reduces the service provider's cost but does not discuss on load management. Deilami et al. [27] has presented a coordinated PEV charging method to improve voltage profile and minimize power losses. The framework presented in paper reduces electricity generation price by the use of dynamic electricity pricing scheme and PEV owner's priority to select the time at which PEV user need charging. The work handles both the problem of electricity load management and user's cost.

Large scale PEV charging has been considered by Ma et al. [28] that has used game theoretic-based decentralized approach for PEV scheduling. The approach is useful for the scenario where centralized aggregator-based coordination among PEVs is not possible. Mukherjee et al. [29] has proposed a scheduling policy where rather than using a single aggregator a distributed approach is adopted that use multiple aggregators. The method involves inter-aggregator communication to allocate charging schedule to PEVs. The method maximizes the profit of aggregators and number of PEVs that can be charged. A fully decentralized approach have been proposed by Mohammadi et al. [30] for large-scale

PEV charging. The given method reduces PEV charging cost while still considering the limitations of the available charging infrastructure. However, the mentioned works using decentralized approach, [28]-[30], have considered the flexible charging schedules by PEVs and do not consider the urgent charging requirement as of en route PEVs.

The works mentioned in literature have considered PEV charging process with the objective of either reducing cost (charging infrastructure development cost or users' electricity cost) or improving electric load profile. Although these factors are important with regard to PEV charging, the time an en route PEV takes to charge is an important concern which has not been discussed in reported literatures. Long waiting or charging time at the charging station hinders the large scale adoption of PEVs. In order to address this problem, we present an integrated architecture for en route PEV charging to minimize its average waiting and charging time at a charging station. The presented optimization unit offers a novel approach for charging station allocation that considers the different factors associated with the charging and discharging of an en route PEV.

## 2.3 Anomaly detection techniques

There have been several works in the literature on anomaly detection of time-series data. This section discusses some the popular anomaly detection techniques that have been frequently used by researchers. The anomaly detection methods can be broadly divided into three major categories - 1). Principal Component Analysis (PCA) based techniques, 2). Statistical methods, and 3). Clustering/Classification based techniques. We describe below each of the method

and their limitations in the IoT domain.

### 2.3.1 PCA-based anomaly detection techniques

Principal Component Analysis (PCA) is one of the well known unsupervised anomaly detection method. Often the data-set has high dimensions, and thus it is difficult to analyze and interpret. PCA is a dimensional reduction tool that transforms high dimensional data-set to a small dimension but containing most of the information of the original set. PCA procedure does so by transforming many correlated variables into a small number of the uncorrelated variables called principal components. It comes handy as an anomaly detection technique when data dimension is large. Following are some of the popular works that utilized PCA transformation for anomaly detection in large dimensional data.

Ling et al. has proposed a PCA based anomaly detection for network data analysis in [31]. The proposed architecture has local monitors and a centralized coordinator. The local monitor tracks the time-series data like TCP connection request per second or volume of traffic at a particular port per second. The global coordinator continuously monitors network-wide time-series data and make decisions regarding its health. Since coordinator gathers data from all monitors, it is of high volume and thus difficult to manage. To solve the problem, data rather than being continuously pushed to the coordinator, is sent only if it is required. This reduces the continuous transfer of data from every monitor to the coordinator thus reducing communication overhead. The collected data from local monitors is arranged in the form of a matrix. The coordinator

transforms the matrix to a lower dimension with the help of PCA to determine the influential factors. Based on the obtained PCA components, any outlier is detected to track the network-wide health. By using the stochastic matrix perturbation theory, the authors further discussed the trade-off between communication overhead and detection accuracy and found the upper bound on detection accuracy.

The outlier detection for IoT sensor data has been proposed by Yu et al. in [32]. The proposed method has a hierarchical system consisting cluster of IoT sensors, cluster head, and cloud. The mentioned scheme groups the IoT sensors by their spatial location using clustering algorithm like k-means. Every cluster is assigned a cluster head having strong computational power. The IoT sensors record the data and transfer it to the cluster head for processing. This reduces the computational burden on IoT sensors which are power constrained devices. The cluster head collects the data from corresponding IoT nodes to find the anomaly in the cluster using recursive-PCA (R-PCA) algorithm. The R-PCA is a modified version of PCA that iteratively updates the basis of PCA transformation based on the changes in IoT system. Since the parameters of PCA are continuously updated in real-time, it leads to the better data analysis. The outliers are detected using a metric called Squared Prediction Error (SPE) score which is defined as the residual values squared after the extraction of dominant components. High SPE score in a data point, after PCA transformation, signifies the presence of the anomaly. The clean data from the cluster head is further transmitted to the cloud for analysis and processing.

Netflix has recently proposed an outlier detection method (RAD) for their big data applications. The outlier detector solves the generic big data problems like

handling high cardinality data-set, minimizing false positives, seasonal variability, and is also useful when data is not normally distributed. The method uses a modified version of PCA, Robust PCA (RPCA), an algorithm for efficient anomaly detection in their cloud applications [33]. The RPCA recovers the principal components of the data matrix even if a fraction of the data is corrupt [34]. The robust version of PCA iteratively finds the Singular Value Decomposition (SVD) of the data-set. The threshold value is applied to the obtained SVD and based on that random noise and outliers are detected. Netflix has used the method in two of its major applications. Netflix involves a large number of daily banking transactions. It tracks those transactions both in real-time and batch mode and detects the failure (anomaly) in the transaction to assist the user. Every day a large number of peoples sign in the Netflix in their browser. Netflix determines anomaly in account sign-in process by utilizing the combination of country, language, browsers, etc. that is used in the login process.

PCA is one of the productive technique for anomaly detection and has been widely used for different application. However, PCA based transformation is based on several assumptions that limit its applications. For example, PCA considers that the dominant components are linearly correlated which cannot always be true. It also assumes that the principal components are orthogonal to each other. PCA is further based on the assumption that the data has high Signal to Noise Ratio (SNR) such that the components having high variance are considered as dominant while those with low variance are noise [35]. The mentioned assumptions cannot always hold, and thus the application of PCA is limited only to specific data-sets.

### 2.3.2 Anomaly detection based on statical characteristics

Another widely used method of anomaly detection is the statistical or distance-based approach. In statistical based approach, historical data patterns are analyzed to determine the properties like mean, mode, or median values. Often the statistics are recorded over the time window of different granularity. Based on the patterns, the threshold values of the data point at the different time are set. If the observed data surpasses threshold values, it is considered as anomalous. We describe some of the important methods that use statistical measures to detect the anomaly.

The importance of finding local outliers has been described by Markus et al. in [36]. The local outliers are the points that are not close to its neighbor. For this purpose, the work has described a Local Outlier Factor (LOF) for every data point. The LOF determines how much separated is an object from its neighborhood points. If a point is well close to its neighbor, then its LOF value is considered to be high. For example, if a point is deep inside a cluster, then its LOF value is one. For every other point, the method assigns an upper and lower bound on their LOF value. The LOF is a useful method for finding local outliers as local factors are sometimes more meaningful and provide insight into the data than the global factors. The LOF also avoids the use of binary factor regarding every point about whether it is an anomaly or not. It does so by assigning an upper and lower bound to the LOF value for points that are well separated from its neighbor. Using formal analysis, the effectiveness of the LOF has been described in the work.

For the high dimensional data-set, the concept of distance between two points

gets somewhat distorted. For such data-sets, the difference between farthest and nearest with respect to a certain point converges to zero even if all the dimensions are relevant. The problem becomes further complex if some of the dimensions contain irrelevant information. Therefore, anomaly detection using Euclidean distance is unsuitable if the data dimension is large as the concept of distance becomes vague. To avoid the problem, Kriegel et al. has proposed an Angular-Based Outlier Detection (ABOD) method in [37]. The ABOD is based on the concept that the angle of a query point with two other points in data-set is lower if the query point is outside a cluster of data. However, if the three points are inside the cluster, then the angle is large. The methodology measures the angular distance of a query point between every possible pair of points in the data-set. The obtained angular measurement is also weighted by dividing it with the product of distance of query point with the other two points. Scaling angle by the weight (product of distance) ensures that the Euclidean distance is important, but its influence is diminished. The variance of the weighted angle of query point with a pair of other points is the ABOF value. If measured ABOF value is large, then the point is considered as an anomaly. The ABOF based anomaly detection is found to be useful for high dimensional data-set. However, the proposed was computationally expensive. To improve the computation complexity, Pham et al. has proposed a projection based algorithm in [38] that runs in quadric time in contrast to the method proposed by Kriegel et al. that runs in cubic time. The algorithm can also be parallelized to further speed up its execution time.

Data falsification by an organized adversary poses a serious threat to business

models. Such adversaries may inject anomaly by launching different kind of attacks through altering data values but preserving statistical measures. To avoid this problem, Bhattacharjee et al. has proposed a semi-supervised anomaly detection technique in [39]. The work presented a robust statistical measure namely ratio of Harmonic Mean to Arithmetic Mean (HM-AM ratio) to infer the attack. HM-AM ratio is resilient against changes in mean value. The HM-AM ratio combined with HM and AM values finds the attack and its type efficiently. To mitigate the effect of the attack, a resilient mixture mean and standard deviation are used as an approximate consensus-based correction method. The given method is useful not only for point anomalies but group anomalies where false is injected in a large percentage of sensors under consideration. The method performs efficiently in real-time for detecting the anomaly in smart meter data. The simulation results for the diverse set of data validates the efficacy HM-AM ratio for detecting point and orchestrated anomaly.

With the rise of IoT devices, the data transferred to the cloud is growing at an immense scale. The time-series data has seasonal trends and often contains anomalies due to either internal or external factor. To detect anomalies in the cloud data, Vallies et al. has proposed a novel detector method in [40]. The authors discussed two statistical techniques that automatically detect anomalies in the cloud. The described method decomposes time-series data by filtering seasonal components. Then it applies Extreme Studentized Deviate (ESD) on the filtered time-series for anomaly detection. The mean and median in ESD are found to be highly sensitive to a large number of anomalies. Therefore, this work further describes the use of Seasonal-Hybrid-ESD (S-H-ESD) for anomaly detection. S-H-ESD uses median and Median Absolute Deviation (MAD) to

detect anomalies. S-H-ESD performs effectively even if 50% of data values are perturbed. The method found to be efficient for time-series data like tweets per second or CPU utilization per second. Although S-H-ESD can handle the large percentage of anomalies, it is also computationally expensive as compared to the ESD.

Statistical techniques are based on the assumption that all the data points are generated from specific distribution and follow a particular pattern. The presumption does not always hold. Especially for high dimensional data, the statistical techniques cannot be efficiently applied due to the above assumption. Second, there are different kinds of statistical methods and choice of specific statistical measure is difficult [41]. Therefore, relying solely on statistical method reduces the effectiveness of anomaly detector.

### **2.3.3 Clustering-based anomaly detection techniques**

For the case of anomaly detection often the data is of enormous volume. Due to a large size of the data, Supervised learning, in general, cannot be efficiently used since it requires labeled set of known data patterns and anomaly. Clustering is one of the popular unsupervised technique that can be applied for anomaly detection problem. The clustering methods collect objects with the similar pattern in groups or classes. The points within the cluster are very close to each other and show the similar pattern. In clustering based anomaly detector, the distance of any new point with respect to the points inside the cluster is examined. If the point lies within the expected cluster, it is considered as a regular point. If its distance from any of the cluster is considerable, then it

is classified as an anomaly. Following are some of the research on clustering based methods.

The high-dimensional IoT data may also contain noise patterns. Handling noisy data for anomaly detection is one of the challenging task as they may give rise to false alarm rate. To address this issue, Erfani et al. has proposed an anomaly detector that uses a hybrid of Deep Belief Network (DBN) and Support Vector Machine (SVM) while handling noise [42]. SVM is one of the popular clustering method and can be easily applied for anomaly detection. However, the problem with SVM is that as the number of dimensions increases it becomes computationally complex. The presence of noise makes the problem further complex, and thus its use is limited to low dimensional data. To overcome this issue, the author proposed the use of Deep Belief Neural Network (DBN) as a feature extractor method. The high dimensional noisy data is first processed using DBN to extract the features. The processed data is then fed to the SVM for clustering which is used for anomaly detection. The presented method effectively handles noisy data obtained from different sources. The authors also explains that the hybrid of DBN and SVM assists in replacing the non-linear kernel function in SVM with linear one thus further reducing computational complexity.

The use of the neural network for anomaly detection using clustering for the cloud data has been described by Pandeewari et al. in [43]. Pandeewari et al. has presented a fuzzy clustering based technique for anomaly detection in cloud data. The given method uses a combination of Fuzzy C-Means and Artificial Neural Network (FCM-ANN) methods that improve the detection accuracy

as compared to the techniques like Naïve Bayes classifier and simple ANN algorithm. The hybrid FCM-ANN method is effective even in the presence of low-frequency outliers and performs with a low false alarm rate.

In wireless sensor network anomalies can occur either in a point form from a single sensor, all measured values from one sensor, or from all set of sensor values. To handle the three types of anomalies, Rajesgarar et al. has described a distributed cluster-based approach for anomaly detection in [44] that also reduces communication overhead. The proposed method considers the sensor nodes to be arranged in the form of a tree structure. In the described approach, sensors individually detect the anomaly and also collaborates with other sensors to find the global anomaly. The individual sensor determines the groups in the measured data values using hyper-spherical cluster method. Based on the obtained clusters, it detects any local anomaly. The sensor nodes transmit a summary of their cluster to its parent node. The parent node combines the summary of all its child with its summary. It examines the combined summary for any anomaly and sends one step up in the level. In this way local, as well as global anomalies, are determined. The parent node also transfers the child node about any detected anomaly so that detection performance in the child get improved.

A technique to detect anomaly in spatiotemporal time-series data has been described by Albanese et al. in [45]. The author uses rough set and kernel set to develop an efficient anomaly detection method. The method describes Rough Outlier Set Extraction (ROSE) algorithm. The process defines an outlier in rough set based on upper and lower bound approximation. The objects in the set can neither be ruled out of its presence inside or outside of the set.

The author also discusses the idea of Kernel set. The Kernel set is the subset of original data-set and describes the data in terms of its structure. The work introduces two versions of ROSE. One uses the original data set and another kernel set. The analysis shows that the kernel set reduces the computation time of the algorithm.

Although clustering is one of the popular technique of anomaly detection, it cannot handle all kinds of anomaly and is not a comprehensive method. Any new data point that has not seen before, but is valid, can be classified as an anomaly [46] if it does not fall under any existing cluster. Therefore, if new valid patterns are frequent in time-series data, then clustering may provide lots of false alarm rate.

## Chapter 3

# Parked Plug-in Electric Vehicle (PEV) charging

The emission of greenhouse gases and rising petroleum prices have led to the adoption of renewable sources of energy. The introduction of Plug-in Electric Vehicle (PEV) which use electricity as the source of energy has the potential to electrify transport sector thus alleviating the dependency on fossil fuels. Beside environmental concerns, the energy required for PEV charging is cheaper as compared to gasoline-based fuel refilling. PEVs also have better performance with smooth acceleration and make less noise [3].

Although PEVs have several benefits, their wide-scale adoption is challenging since an extensive charging infrastructure need to be developed for their realization. With the anticipated growth of PEVs as a popular transportation choice, their confluence with smart grid and Electric Vehicle Supply Equipment (EVSE) is of utmost importance [16]. The large-scale penetration of PEVs will add up the electricity demand putting pressure on the power grid. Additionally, the extensive energy required for PEV charging, unpredictable nature of

PEV load, and their disparate charging requirement reshape the electricity demand curve of the consumer. Therefore, PEV charging need to be synergized such that it does not affect the electricity demand profile severely.

An uncoordinated PEV charging may cause sudden load change since many PEVs may be plugged in for charging within a small time interval. A sudden load change leads to voltage fluctuation which may affect the performance of appliances tied to the same transmission system [6]. An abrupt load variation also results in the early aging of power electronic instruments like transformers. Furthermore, it induces power loss and harmonic distortions [4], [5]. Beside unexpected load change, an uncoordinated PEV charging spurs electric load variation at the different time in a day, thus resulting in peak electricity demand differ to a considerable extent than the average electricity demand. The load variation decreases the value of the load factor, which is the ratio of average power demand to peak power demand. Contrarily, consumers with less difference between peak demand and average demand, one with high load factor, are charged less as compared to the consumers with low load factor [47]. Additionally, the electricity generation and management depend upon short-term or day ahead energy demand [48], [49]. An electric load profile with less variations eases the problem of the load forecasting such that the load prediction is better thus assisting in better supply side management.

Since reducing load variation is a key aspect for coordinate PEV charging, we investigate feasibility and mechanism of a centralized aggregator to schedule PEVs for charging. The proposed aggregator model makes charging decision on behalf of PEVs to reduce power variation and improve load factor.

### 3.1 System model

Figure 3.1 presents the proposed system model. The model consists of Plug-in Electric Vehicles (PEVs), Electric Vehicle Supply Equipments (EVSEs), and an aggregator. EVSE is used for providing electricity for PEV charging. The aggregator is a centralized controller which collects data from PEVs and EVSEs and schedules PEVs for charging based on their arrival time, departure time, and charging requirements. We consider the scheduling problem for large parking places where electric load due to PEV charging is large. For example, in a university campus or office building number of vehicles are of the order of 100 during working hours. Such large number of vehicles make sizable electric load. We examine the PEV scheduling problem for the scenario where arrival and departure time of PEVs at the parking place is known in advance. For example, in offices arrival or departure time of vehicles mainly depends upon working hours. Similarly, in the university campus, the arrival and departure depends upon class schedule.

We consider the futuristic scenario where every parking slot in the parking place has an EVSE for PEV charging. EVSE consists a connector which can be plugged into the PEV inlet for charging. The EVSE for the proposed model is considered to supply 120-volt Alternating current (AC) level 1, 220-volt AC level 2, and less known 220-volt AC level 3 voltage for charging. The different voltage level charge PEV with different rate [7]. A higher voltage level charge PEV in less time drawing more power from EVSE. The availability of variable voltage level charging is leveraged to flatten load profile by charging at the higher level when fewer vehicles are present and charging at the lower level

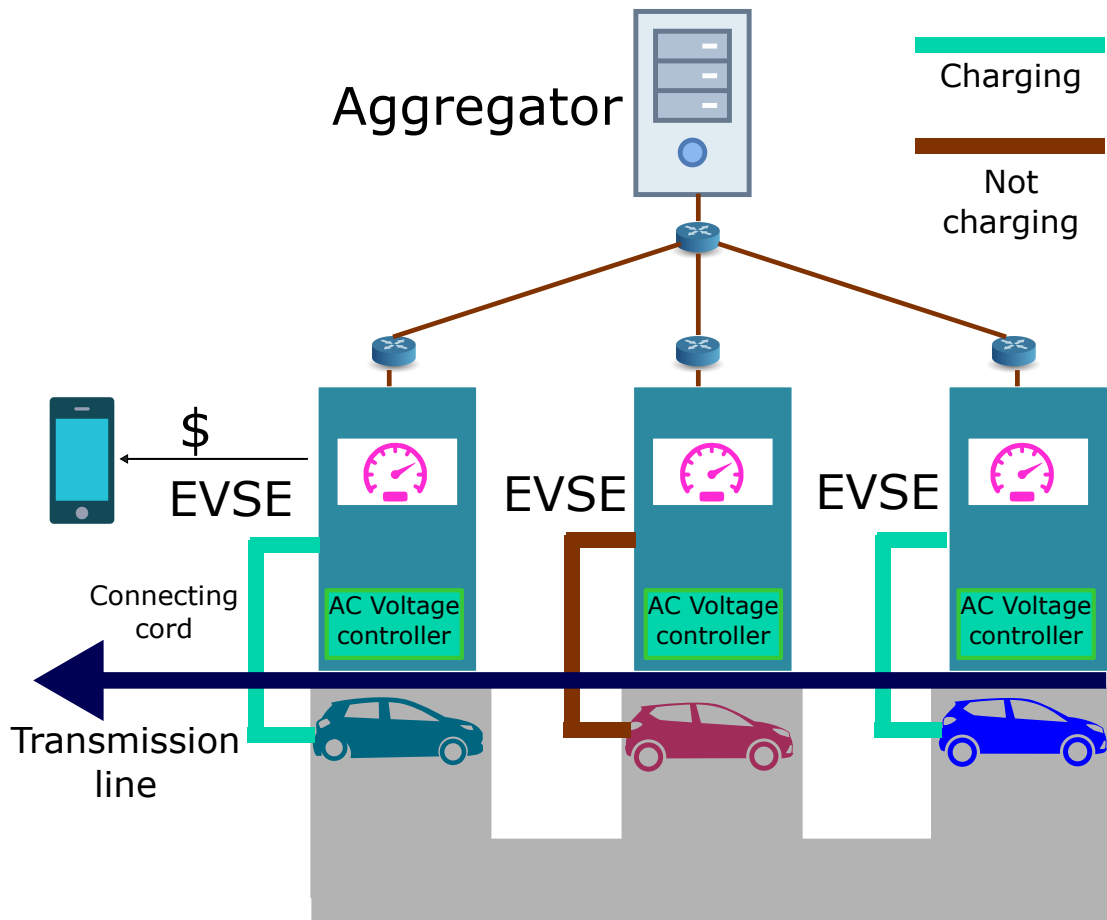


Figure 3.1: System model for parked PEV charging

when many PEVs need to be charged. The investigation of Combined Charging Systems (CCS) has enabled the same connector to be used for AC level 1 or level 2 charging [50], [51]. This eases the charging process when different level of charging is supplied by the same EVSE as connector need not be replaced based on the voltage supplied by EVSE. The user only has to plug in CCS connector to the PEV inlet, and the process of charging level selection can be automated. To monitor the charging status of the vehicle, EVSE establishes a communication link with PEV. The dedicated wire in CCS connector is used

for sharing information between PEV and EVSE [50]. AC voltage controller installed in EVSEs convert 120 (or 220) volt supply to 220 (or 120) volt supply. This enable EVSEs to provide different voltage for charging without having separate dedicated lines for 120 and 220 voltage.

The PEVs are installed with a Radio Frequency Identification (RFID) device for their unique identification and authorization. After the PEV is parked in a parking slot, RFID device is automatically scanned by sensors for its identification and authentication. The information about the parking slot within the parking place where a PEV is parked is sent to the aggregator. The user plugs in the charging connector to the PEV and leaves, but the charging process may not be immediately started. The charging decision is made by the aggregator. The PEV charging automatically begins at the time scheduled by the aggregator and connectors stops supplying electricity once PEV gets charged. The smart meter in EVSE tracks the amount of power consumed by the PEV. After charging process, the electricity price and the charging information is automatically sent to the user mobile phone.

## 3.2 Methodology

### 3.2.1 Problem statement

We consider a total number of  $N$  PEVs where each PEV is represented by a variable  $i$  such that  $i \in \{1, 2, \dots, N\}$ . We discretize time in slots each of length  $\tau$ . A particular time slot is represented by a variable  $j$  such that  $j \in \{1, 2, 3, \dots, M\}$  and  $M$  is the total number of slots required for charging all PEVs.

We represent  $V_l$  as the charging voltage supplied by EVSE at level  $l$ . The amount of current used by PEV depends upon voltage level  $V_l$  supplied by the EVSE for PEV charging and the PEV battery circuit that is specific to a PEV. We represent  $I_l^i$  as the current supplied by EVSE when PEV  $i$  is charged at voltage  $V_l$ . The required energy to charge PEV  $i$  called as a Status of Charge  $SOC_r^i$ . The arrival time and departure time of PEV  $i$  at the parking place is given by  $t_a^i$  and  $t_d^i$ . The number of slots required to charge PEV  $i$  at charging level  $l$ , represented as  $n_l^i$ , is given by equation 3.1. In equation 3.1, product  $V_l^i \times I_l^i$  is the power (energy per unit time) supplied by the EVSE.

If a PEV  $i$  arrives at time  $t_a^i$  and departs at time  $t_d^i$  then its arrival slot  $\tau_a^i$  and departure slot  $\tau_d^i$  are given by equations 3.2 and 3.3. The  $t_{in}$  is considered as the starting reference time. For example, if time slot length is 5 minutes and starting reference time is 7:00 AM then 7:00 AM-7:05 AM is zeroth slot, 7:05 AM-7:10 AM is first slot, 7:10 AM-7:15 AM is second slot and so on. If a PEV arrives at 7:03 AM then its arrival slot is considered as 7:05 AM-7:10 AM which is slot number 1. If it departs at 4:02 PM then its departure slot is 107 (3:55 PM-4:00 PM). If the PEV requires 30 slots to charge with some voltage level then these slots should be allocated between slot number 1 and 107 for its charging.

$$n_l^i = \lceil \frac{SOC_r^i}{V_l \times I_l^i \times |\tau|} \rceil \quad (3.1)$$

$$\tau_a^i = \lceil \frac{t_a^i - t_{in}}{|\tau|} \rceil \quad (3.2)$$

$$\tau_d^i = \lfloor \frac{t_d^i - t_{in}}{|\tau|} \rfloor \quad (3.3)$$

The net electricity load at the parking place in slot  $j$  is given by equation 3.4 subject to the constraints 3.5 and 3.6. The variable  $x_l$  is a binary variable which represents whether EVSE is charging at level  $l$  voltage. The inner summation in equation 3.4 adds over all charging levels. Equation 3.5 and 3.6 guarantees that only single charging level is used for charging a PEV. The outer summation adds the power requirement for every vehicle in parking lot in slot  $j$ . Variable  $y_i^j$  is a binary variable to determine if a PEV  $i$  is being charged in slot  $j$ . If the value of  $y_i^j$  is 1 then the PEV is being charged in slot  $j$ . Otherwise, the PEV is connected to EVSE but does not draw current.

$$P_j = \sum_{i=1}^N (\sum_{l=1}^L V_l I_l^i x_l) y_i^j \quad (3.4)$$

$$\sum_{l=1}^L x_l = 1 \quad (3.5)$$

$$x_m \times x_n = 0 \quad \forall m \neq n, m, n \in 1 \dots L \quad (3.6)$$

The change in PEV load between two consecutive time slots is given by equation 3.7.

$$\delta P_j = P_j - P_{j-1} \quad (3.7)$$

The optimization objective given by equations 3.8 and 3.9. Equation 3.8 minimizes average power deviation over various time slots and equation 3.9 maximizes load factor (ratio of average power to peak power).  $P_{avg}$  is the average of the power assigned at various time slots.

$$\min\left\{\frac{\sum_{j=1}^M \delta P_j}{M}\right\} \quad (3.8)$$

and

$$\max\left\{\frac{P_{avg}}{\max\{P_1, P_2, \dots, P_M\}}\right\} \quad (3.9)$$

### 3.2.2 PEV schedule using rectangle placement

We solve the problem of reducing electric load variation by scheduling PEVs such that they are charged according to the time decided by the proposed algorithms. For every PEV, we consider a rectangle whose length is the time PEV need for charging and height is the power required for its charging. For example, if a PEV  $i$  requires  $n_i^i$  slots to charge with the voltage level  $V_i$  and current  $I_i^i$  then for PEV  $i$  we consider a rectangle of length  $n_i^i \times |\tau|$  and height  $V_i \times I_i^i$ . The area of the rectangle which is the product of length (time) and height (power) represents required energy  $SOC_r^i$  for PEV charging. Since height of one rectangle is the power needed to charge one PEV, the net height of the rectangles, at a particular time slot, after they are placed horizontally to the time scale is the power required for charging all PEVs scheduled for charging at that time slot. Figure 3.2 illustrate the rectangle placing scenario for 3 PEVs. Rectangle A

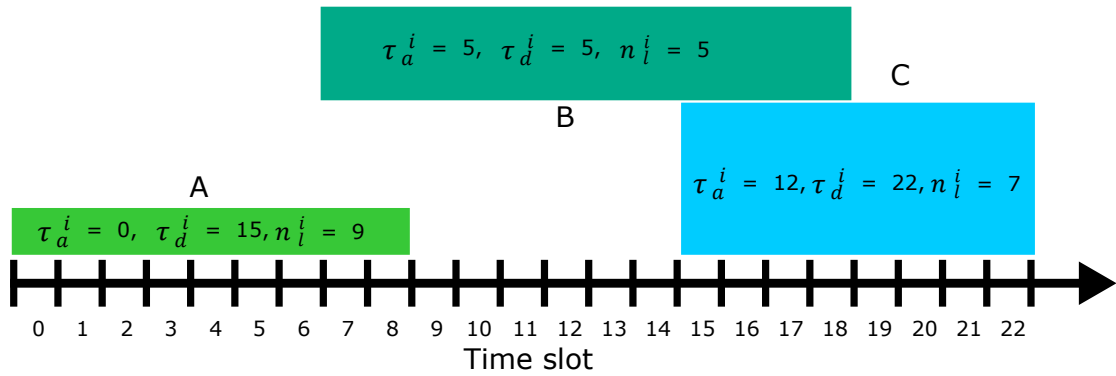


Figure 3.2: Rectangle placement

in figure requires 9 slots to charge. Its arrival and departure slots are 0 and 15 respectively. Thus, the PEV corresponding to the rectangle A need to be scheduled for charging between time slot 0 and 15 such that the rectangle A is to be placed between slots 0 and 15 occupying 9 slots. Similarly, rectangle B requires 12 time slots for charging and need to be placed between its arrival slot (5) and departure slot (20) and rectangle C is to be placed between slots 15 and 22 occupying 7 slots . If rectangles are placed as in Figure 3.2, the electric load at various time slots can be represented like in Figure 3.3. The graph in Figure 3.3 presents net required electric power at various time slots is called as *load profile*. For the given schedule, between time slots 0-6 only A is being charged, between slots 7-8 A and B are being charged, in interval 9-14 B is charged, in slots 15-19 both B and C are being charged, and between slots 20-22 C is charged.

The proposed algorithms assign slots for placing rectangles such that the deviation in height of the load profile is minimum. For instance, in Figure 3.3 the height (required power) is low between slots 0 and 6 and high between slots 15 and 19 resulting in deviation in height (required power) of the load profile. The presented algorithms place rectangles to minimize deviation in height at various time slots.

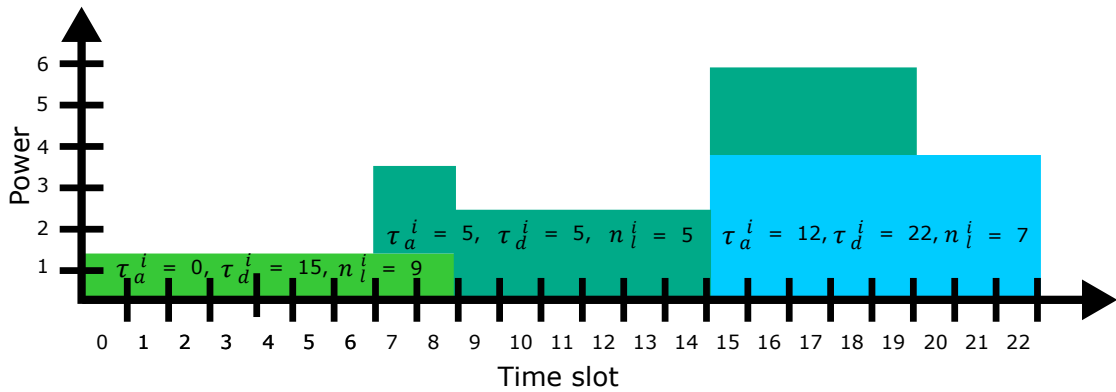


Figure 3.3: Total power

### Main algorithm

To solve the problem of reducing electric load variation between various time slots algorithm 1 is used. Algorithm 1 places the rectangles corresponding to every PEV to minimize deviation in height of the load profile. Algorithm 1 use algorithms 2-4 for optimal rectangle placement. Algorithm 1 takes list of vehicle charging request  $V$  as an input and provides charging schedule for every PEV in list. In step 1, algorithm 1 runs algorithm 2. Algorithm 2 does the initial rectangle placement corresponding to every PEV request. Initial rectangle placement algorithm gives priority to the arrival time of the PEV and use level 2 charging. Step 2 saves the assignment in a temporary variable  $tmpaSlotP$ . Based on the initial rectangle placement, step 3 determines the load factor  $LF_{curr}$  and step 4 sets the variable maximum load factor  $LF_{max}$  to the current load factor  $LF_{curr}$ . Step 5 finds the minimum height  $P_{ref}$  of the load profile.  $P_{ref}$  is also called as the reference height of the load profile. Step 4 determines the maximum height  $P_{max}$  of the load profile. Steps 8-15 are repeated in a loop. In step 8 algorithm 3 and in step 9 algorithm 4 are executed. The vehicle charging request list  $V$  and the value of  $P_{ref}$  are passed as parameters to

---

**Algorithm 1** Main algorithm
 

---

**Input:**  $V$  - Vehicle charging request list

**Output:**  $aSlotP$  - Assigned power at all slot

```

1: InitialRectanglePlacement( $V$ )
2:  $tmpaSlotP = aSlotP$ 
3:  $LF_{curr} = \frac{avg(aSlotP)}{max(aSlotP)}$ 
4:  $LF_{max} = LF_{curr}$ 
5:  $P_{ref} = min(aSlotP)$ 
6:  $P_{max} = max(aSlotP)$ 
7: while  $P_{ref} < P_{max}$  do
8:   VolageLevelSelection( $V, P_{ref}$ )
9:   ShiftRectangle( $V, P_{ref}$ )
10:   $LF_{curr} = \frac{avg(aSlotP)}{max(aSlotP)}$ 
11:  if  $LF_{curr} > LF_{max}$  then
12:     $LF_{max} = LF_{curr}$ 
13:     $aSlotP = tmpaSlotP$ 
14:  end if
15:   $P_{ref} = P_{ref} + h$ 
16: end while

```

---

both the algorithms. Algorithm 3 changes the charging level of some vehicles from level 2 to level 1 or level 3 to reduce variation in height of the load profile. Algorithm 4 shifts the position of rectangles to further flatten height of the load profile. Step 10 determines the load factor  $LF_{curr}$  after voltage level switching and rectangle shifting. If the new load factor  $LF_{curr}$  is greater than the previous maximum load factor then the values of maximum load factor  $LF_{max}$  and assigned slots for PEVs are updated (steps 11-14). In step 15, the variable reference height  $P_{ref}$  is incremented by a small value  $h$ . Since reference height  $P_{ref}$  is the minimum height of the load profile, the loop terminates when reference height is increased to such extent that it becomes greater than the maximum height  $P_{max}$  (obtained after the initial rectangle placement).

---

**Algorithm 2** Initial rectangle placement
 

---

**Input:**  $V$  - Vehicle charging request list

 $P_r$  - Reference power

**Output:**  $aSlotP$  - Assigned slots

 $LF$  - Load factor

```

1:  $V.sort(arrivalTime)$ 
2:  $aSlotP.all() = 0$ 
3: for  $i \leftarrow 1$  to  $|V|$  do
4:    $w = 0$ 
5:    $a = False$ 
6:   while  $a \neq True$  do
7:     for  $j \leftarrow V(i).\tau_a^i$  to  $V(i).\tau_d^i - n_l^i$  do
8:       if  $all(aSlotP(j : j + n_l^i) < w)$  then
9:          $all(aSlotP(j : j + n_l^i)) += V_i \times I_l^i$ 
10:         $a = True$ 
11:      else
12:         $w = w + wI$ 
13:      end if
14:    end for
15:  end while
16: end for

```

---

**Initial rectangle placement**

Algorithm 2 does the initial rectangle placement giving priority to the arrival time of PEVs. In step 1, the list of PEV charging requests  $V$  is sorted in increasing order of arrival time. The required power at every time slot is initially set to 0 in step 2 ( $aSlotP$  is the list that represents information about electric load at various time slots). Steps 4-15 are repeated for every PEV request in sorted list  $V$ .  $|V|$  denotes the number of elements in the set  $V$ . The algorithm uses a window  $w$ , the initial size of which is set to 0 in step 4. The variable  $a$  keeps the track of whether the rectangle is placed in load profile is initially set to *False* (step 5). For every PEV  $i$ , step 7 checks for the consecutive  $n_l^i$  slots between

arrival time  $\tau_a^i$  and departure time  $\tau_d^i$  where the height of the load profile is between 0 and  $w$  (values 0 and  $w$  are included). If such consecutive slots exist then the rectangle is placed on those slots, the net power in those slots is increased, and variable  $a$  is set to *True* (steps 8-10). If they do not exist then the window size is increased in step 12. Consecutive slots where rectangle can be placed are then searched for larger value of  $w$ . The process ensures that the rectangle is initially placed between arrival and departure slot of PEV over load profile where height is minimum.

### **Voltage level selection**

After initial rectangle placement, algorithm 3 switches the charging level of some PEVs from level 2 to level 1 or level 3 based on the height of the load profile at various slots. It modifies the load profile obtained from initial assignment by switching charging level voltage. Algorithm tries to bring the height of the load profile near the  $P_{ref}$  for every time slot. Steps 2-15 are repeated for every PEV to determine if its charging level can be switched to obtain better load profile. Variable  $aSlot$  contains the information about every PEV's charging schedule.  $sSlot_i$  is the starting slot number in which PEV  $i$  is scheduled for charging. Step 2 finds if all the slots assigned for a particular PEV has net height less than the  $P_{ref}$ . If this is the case then step 3 determines whether increasing voltage to level 3 still keeps the height of the load profile in those slots less than but closer to  $P_{ref}$ . If the condition is true then the voltage level for that PEV is switched to level 3 (steps 4-6). Steps 9-14 are similar to steps 2-8 except the portion in the load profile where height is greater than  $P_{ref}$  are being checked

---

**Algorithm 3** Voltage level selection
 

---

**Input:**  $V$  - Vehicle charging request list

 $P_{ref}$  - Reference power

**Output:**  $aSlotP$  - Assigned slots

 $LF$  - Load factor

```

1: for  $i \leftarrow 1$  to  $|V|$  do
2:   if  $all(aSlotP.(i)) < P_{ref}$  then
3:     if  $all(aSlotP.(i)([sSlot_i : n_i^i]) - V_i \times I_i^i + V_{i-1} \times I_{i-1}^i < P_{ref}$  then
4:        $all(aSlotP.(i)(sLot : n_i^i)) - = V_i \times I_i^i$ 
5:        $nSlot = \lceil SOC_r^i / (V_{i+1} \times I_{i+1}^i \times \tau) \rceil$ 
6:        $aSlotP.(i)[sSlot_i : nSlot] + = V_{i+1} \times I_{i+1}^i$ 
7:     end if
8:   end if
9:   if  $all(aSlotP.(i)) > P_{ref}$  then
10:    if  $all(aSlotP.(i)[sSlot_i : n_i^i]) - V_i \times I_i^i + V_{i-1} \times I_{i-1}^i > P_{ref}$  then
11:       $all(aSlotP.(i)[sLot : n_i^i]) - = V_i \times I_i^i$ 
12:       $nSlot = \lceil SOC_r^i / (V_{i-1} \times I_{i-1}^i \times \tau) \rceil$ 
13:       $aSlotP.(i)[sSlot_i : nSlot] + = V_{i-1} \times I_{i-1}^i$ 
14:    end if
15:  end if
16: end for

```

---

(steps 9 and 10). If the condition is satisfied then the charging level voltage for the PEV is switched to level 1 (steps 11-13).

**Rectangle shift**

Algorithm 4 shifts the rectangles from region where height of the load profile is greater than the  $P_{ref}$  to the region where height is less than  $P_{ref}$ . Steps 2-5 are repeated for every PEV. Step 2 determines if height of the load profile at assigned slots for a PEV is greater than  $P_{ref}$ . If the height is greater than  $P_{ref}$  then the rectangle is shifted towards the right, such that PEV charging ends at its departure slot. We shift some rectangles to the right because in step 1 the

**Algorithm 4** Shift rectangle**Input:**  $V$  - Vehicle charging request list $P_{ref}$  - Reference power**Output:**  $aSlotP$  - Assigned slots $LF$  - Load factor

- 1: **for**  $i \leftarrow 1$  **to**  $|V|$  **do**
- 2:   **if**  $all(aSlotP[sSlot_i : n_l^i]) > P_{ref}$  **then**
- 3:      $all(aSlotP[sSlot_i : n_l^i] - = V_l \times I_l^i$
- 4:      $all(aSlotP[\tau_d^i - n_l^i : \tau_d^i] + = V_l \times I_l^i$
- 5:   **end if**
- 6: **end for**

**Table 3.1:** Simulation parameters for parked PEV charging

Parameter	Value
Arrival time	7 AM - 5 PM
Departure time	8 AM - 2 AM
$SOC_r^i$	15 kWh - 49 kWh
AC level 1	120 V, 16.6mA/33.3mA
AC level 2	220 V, 72.72mA/90.9mA/109mA
AC level 3	220 V, 218.18mA/245.45mA/272.72mA

priority was given to the arrival time (arrival time is in left). To balance that effect, this step shifts some rectangles to right to make height constant.

The algorithm 3 and 4 run for different values of  $P_{ref}$  obtained from algorithm 1 for optimum rectangle placement. After these algorithms are run for all possible values of  $P_{ref}$ , the final position of rectangles is used to schedule PEV charging.

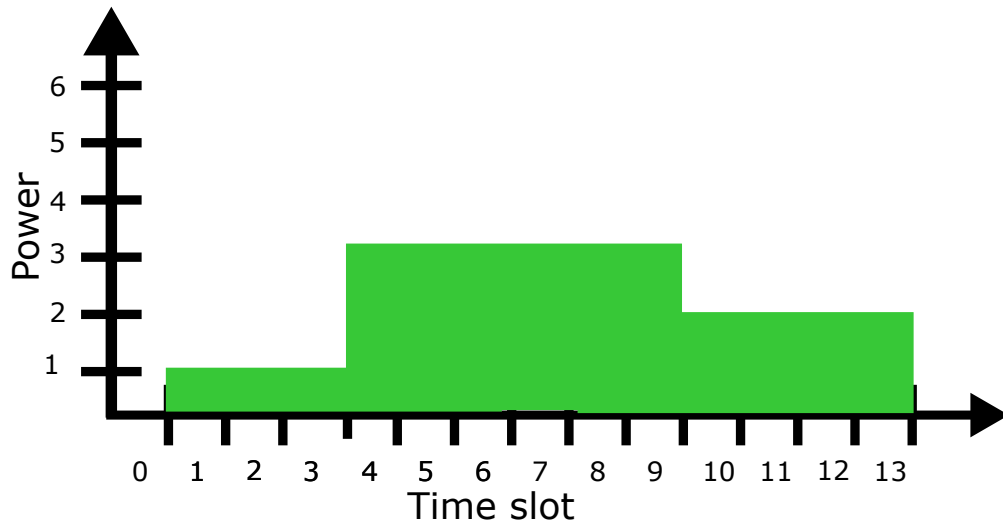
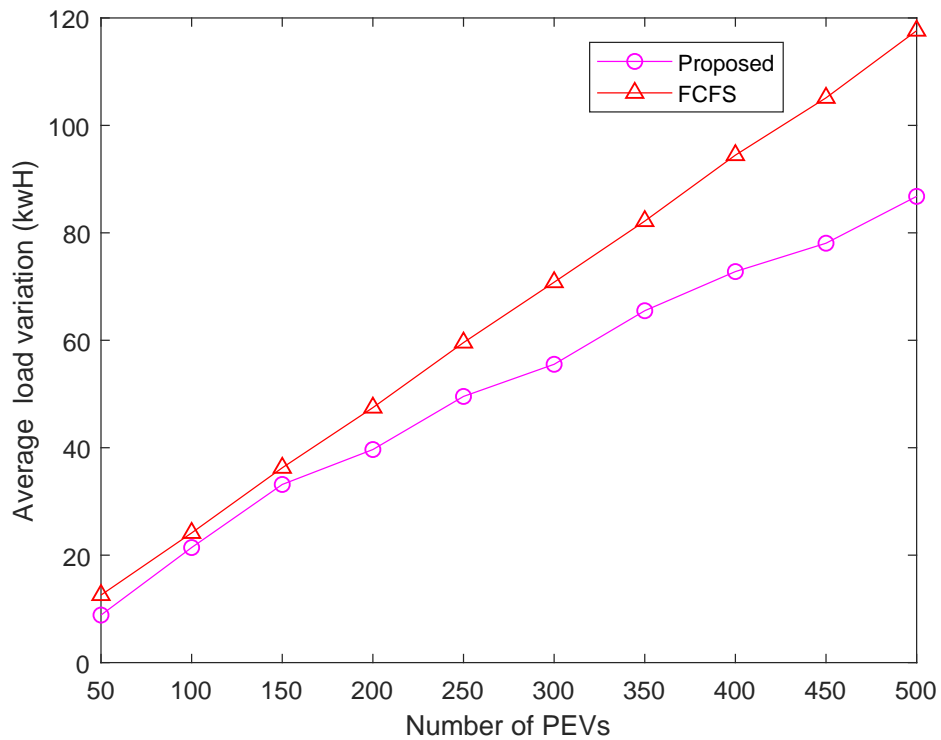


Figure 3.4: Load profile

### 3.3 Results

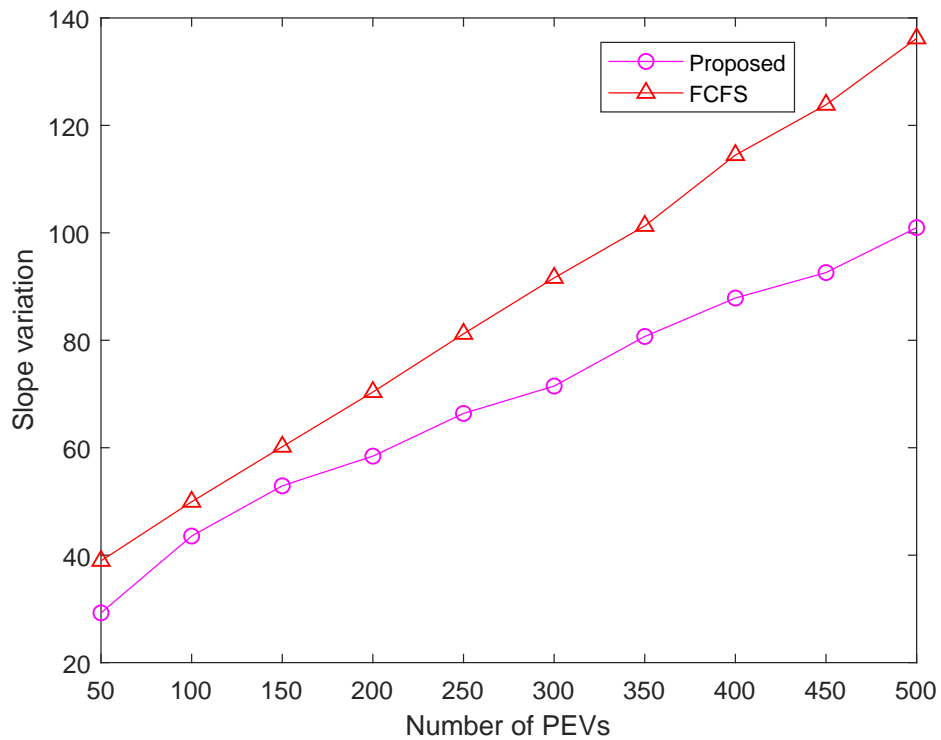
We conducted an extensive set of experiments in python based framework to determine the deviation in power at various time slots. The proposed algorithms are compared with the First come, First serve (FCFS) based PEV charging. Simulation results are compared for different metrics to determine the extent to which PEV load varies in different time slots. To determine steepness in power curve, we introduce the metric *slope-variation*. Slope-variation is defined as the ratio of cumulative power variations in different time slots and the number of times the load profile changes. For example, in Figure 3.4 the height of the load profile changes to 2 units at the start of slot 4 and 1 unit in slot 10. The slope-variation for the mentioned example is average of cumulative height change at these two points  $((2 + 1)/2)$ .



**Figure 3.5:** Average power variation per time slot as number of PEVs are varied

### 3.3.1 Simulation parameters

Table 3.1 presents the simulation parameters for the experiments. Parameters were selected randomly between chosen range. Table also gives the voltage and current values for different levels of PEV charging. For example, AC level 1 charging use 120 V. Some PEVs for AC level 1 charging require 16.6 mA of current and some other need 33.3 mA. The length of time slot is chosen as 5 minutes.

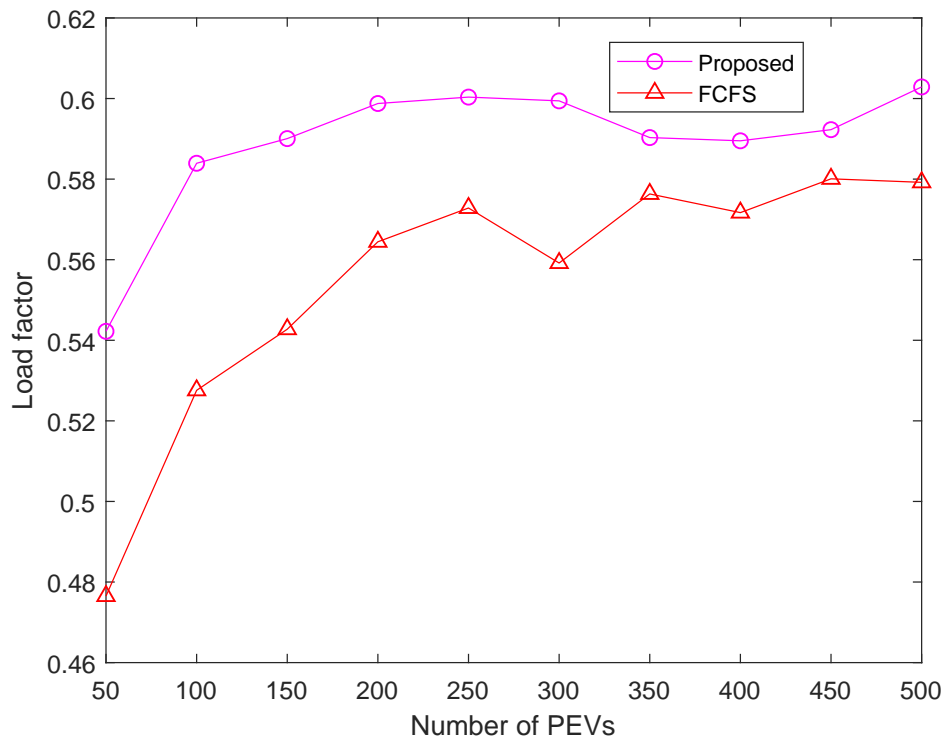


**Figure 3.6:** Average slope variation as number of PEVs are varied

### 3.3.2 Analysis of power variation

Figure 3.5 depicts the average power change over all time slots. As shown in the figure, the average power change is lower using proposed algorithms as compared to FCFS based PEV charging. For less number of PEVs, the gap between presented and FCFS algorithm is small. However, as number of PEVs increase the proposed methods shows better performance.

Figure 3.6 compares the slope-variation obtained from the proposed approach and the FCFS algorithm. The figure illustrate that for the proposed approach, the slope varies gradually when compared with FCFS. Here also, the gap between proposed and traditional method increases with the number of PEVs.



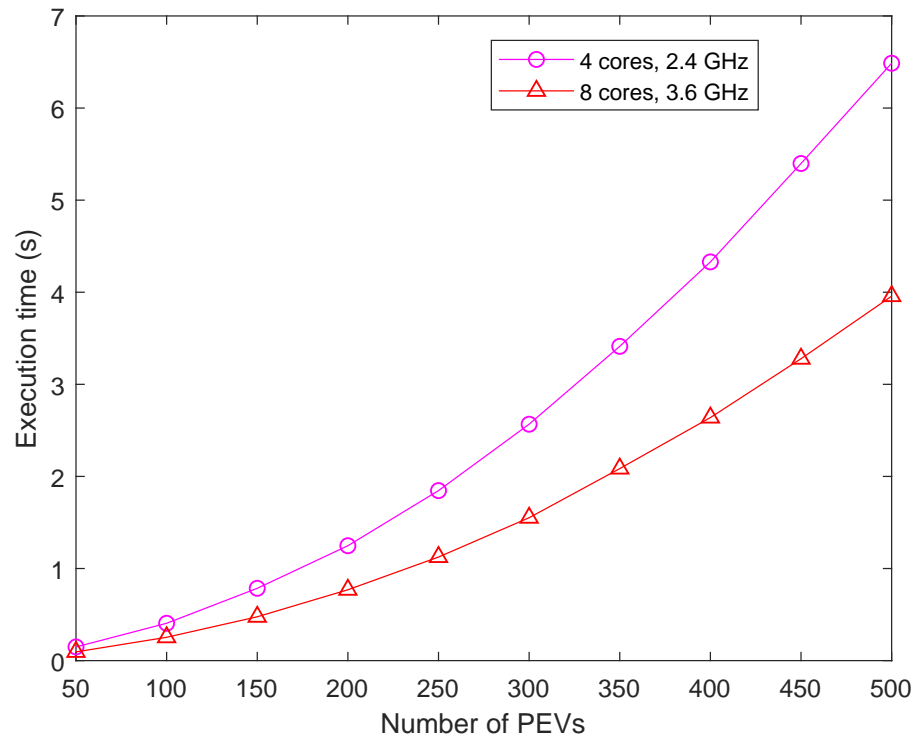
**Figure 3.7:** Load factor as number of PEVs are varied

Therefore, the importance of our approach lies in the large scale PEV charging at parking place.

Figure 3.7 compares the load factor between the proposed and FCFS method. The proposed algorithms show a performance gain in terms of load factor. Henceforth, the given method is also economically beneficial due to high value of the load factor.

### 3.3.3 Timing analysis

Figure 3.8 depicts the cumulative execution time of the presented algorithms as the number of PEVs are varied. We run our algorithms in two different CPU



**Figure 3.8:** Execution time

configurations (8 cores, 3.6 GHz and 4 cores, 2.4 GHz) in ubuntu operating system. Figure interprets that the run time of the algorithm is less than 6 seconds for both configurations. The lower execution time makes them feasible to implement in the aggregator.

### 3.4 Observations and remarks

This chapter highlighted the importance of reducing load variation due to PEV charging at a parking lot. We presented the smart aggregator architecture that automates PEV charging process. Our proposed rectangle placement based algorithms assign the time at which PEVs should be charged. We also presented

charging voltage level selection to find the voltage at which a PEV should be charged. Rectangle placement combined with charging level selection makes load profile smooth as compared to traditional FCFS based charging. The simulation results characterize that our method reduces power deviation and load factor. Furthermore, our proposed architecture is feasible to implement in aggregator due to its lower execution time.

## Chapter 4

# En route Plug-in Electric Vehicle (PEV) charging

The problem of en route Plug-in Electric Vehicle (PEV) charging is complex due to their mobility and requirement of low charge time as a user may not want to wait long near a charging station. Since PEV charging depends upon Electric Vehicle Supply Equipments (EVSEs) installed in a charging station, and energy provided by the Smart Grid (SG), an efficient charging solution requires PEVs, EVSEs, and SG to coordinate for sharing information with each other [52]. However, due to their spatial separation and dynamic connectivity, the communication network between them cannot be managed using traditional decentralized architecture. Consequently, the network management for tying PEV, EVSE, and SG for their coordination and information sharing entails alternative method.

Apart from network management, an en route PEV needs to be allocated a charging station in case it is running out of its battery power. Allocation should

be done in a manner such that the wait and charge time of a PEV at a charging station is low. The charging station located near a place having high PEV traffic is expected to have a long wait time for PEV charging because of frequent charging requests. However, if the traffic density at a location is known in advance then some PEVs can be allocated charging station located near low traffic congestion region thus reducing average wait time of PEVs at a charging station. Therefore, advance traffic prediction is essential for determining the location at which a PEV should be charged so that the time it spends in a charging station is low.

Moreover, PEV discharging and charging process depends upon factors like its speed, battery drainage rate, EVSE charging rate, and wait time at a station. The mentioned variables are dynamic and varies within a broad range. Henceforth, the allocation of charging station to PEVs such that their cumulative charging time is low is a complicated task. Therefore, an optimization method which considers these multiple dynamic parameters is required for allocating a pertinent charging station to PEVs that reduces their charging time.

Thus, the major challenges regarding en route PEV charging is the investigation of an efficient network for information exchange, development of an optimization method that reduces the charging time of PEVs at the charging station, and finding an efficient prediction unit that can help optimization unit in making better decision. To address the given problems, this research investigates an inter-wined COP architecture as shown in Figure 4.1 consisting of a Communication unit, a Prediction unit, and an Optimization unit. We envision Communication unit that has potential to enable efficient information sharing between spatially separated end units by adopting Software-defined Network

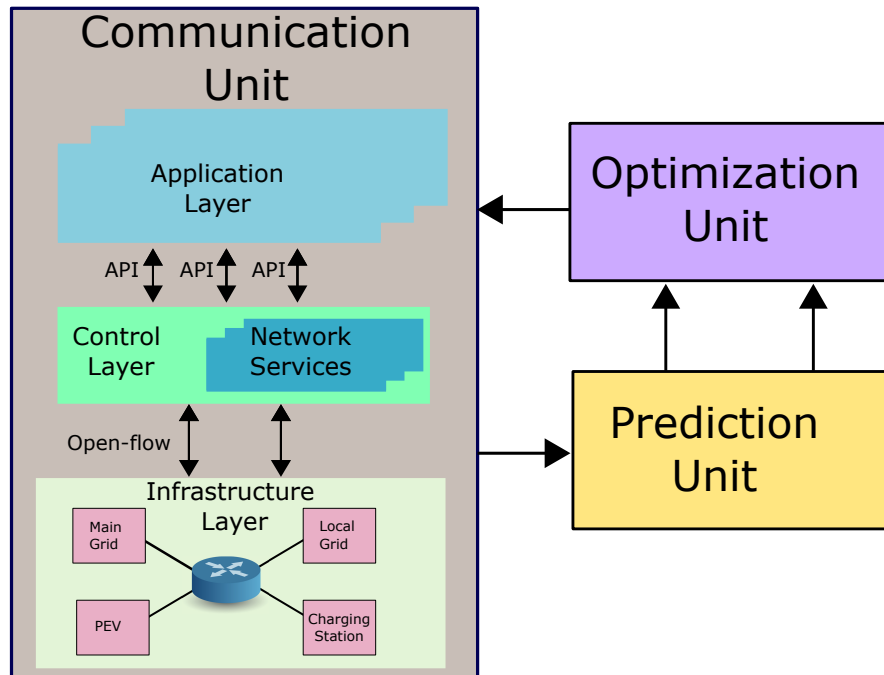
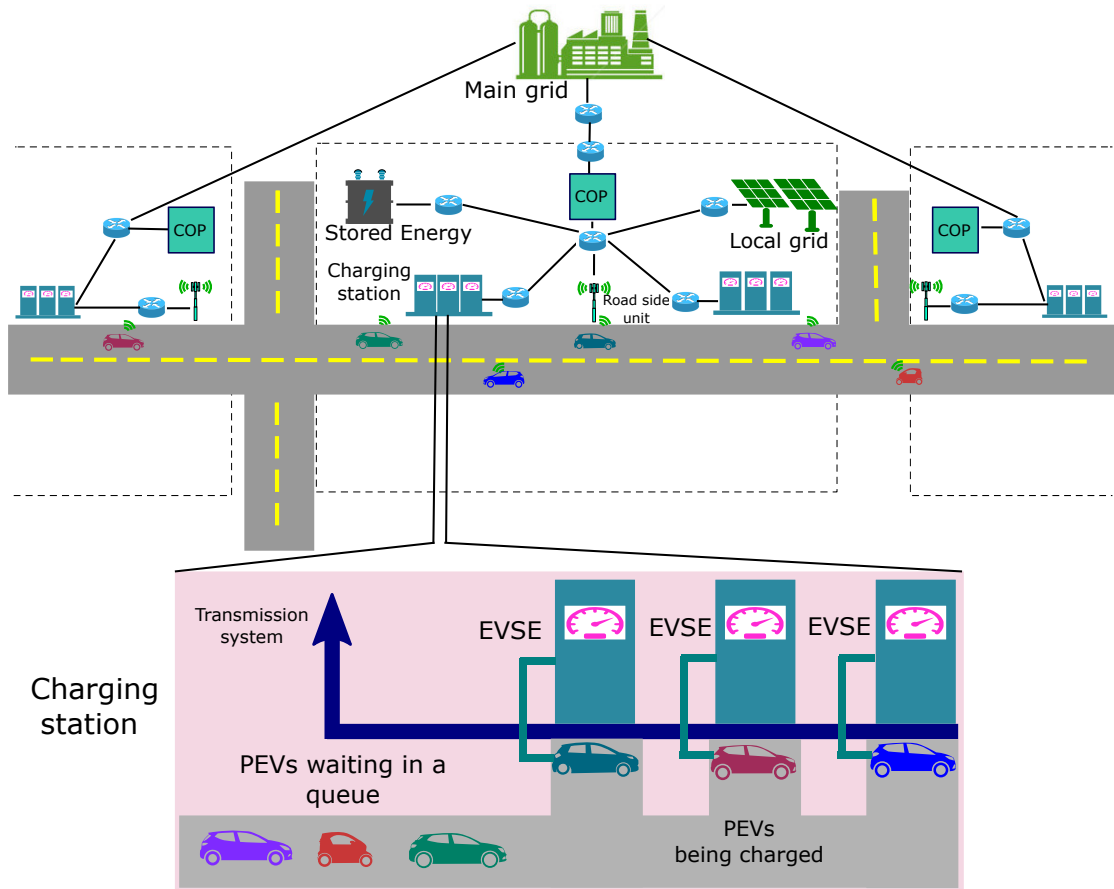


Figure 4.1: COP architecture

(SDN)-based framework. Optimization unit allocates charging station to different PEVs based on the parameters associated with the charging and discharging of PEV battery. Prediction unit estimates traffic near different charging stations to obtain wait time, and therefore helps optimization unit in making better decisions.

## 4.1 System model for en route PEV charging

The proposed system model for allocating charging stations to PEVs is depicted in Figure 4.2. Similar to [24] and [53], it has been assumed that the charging rate of a PEV depends on the rate at which EVSE is supplying power. Also, the EVSEs installed in a single charging station are considered to be similar having



**Figure 4.2:** System model for en route PEV charging

same charging rate. However, two EVSEs in different charging station may have different charging rate.

Figure 4.2 shows en route PEVs moving along the road. PEVs charge themselves at charging station which are also situated along the road. EVSEs provide power to PEVs and there may be multiple number of EVSEs in a single charging station. Electric supply may come from different units scattered in the region. The whole region is divided into multiple zones. Every zone consists of an integrated Communication, Optimization, and Prediction (COP) unit which is responsible for coordination between different end units like PEVs, EVSEs,

and smart grid and for optimally allocating charging stations to PEVs. Loop detectors are installed along the road to measure the traffic volume at different locations. The measured data from the loop detectors is sent to the centralized COP unit. Figure 4.2 also depicts the charging station model. The power needed to charge PEVs is supplied through the charging bus which is connected to the transmission system. The PEVs arriving at a charging station wait in a queue. As soon as the EVSE slot is empty, the PEV first in the queue occupies that space for charging. We define *effective charging time* of a PEV as the sum of the time during which PEV waits in a queue and the actual charging time at an EVSE.

#### 4.1.1 Smart grid and charging station architecture

The power for PEV charging comes from various units that includes main grid, local generation unit, and storage unit.

To understand power flow in a charging station, we consider bus in the transmission system by the variable  $q$  such that  $q \in B$  and  $B = \{1, 2, \dots, Q\}$ . The total number of bus are given as  $Q$ .  $p$  is a particular element of  $B$  and represents the bus connected to the charging station. Thus, if the value of  $q$  is equal to  $p$  then  $q$  is the bus connected to the charging station. The voltage in a bus  $q$  at time  $t$  is given as  $V_{q,t}$ .  $\theta_{q,t}$  and  $\theta_{p,t}$  are the angle of the voltage  $V_{q,t}$  and  $V_{p,t}$  respectively. The real and reactive power flowing in bus  $p$  at time  $t$  is represented as  $P_{p,t}$  and  $X_{p,t}$  respectively.  $Y_{p,q}$  is the admittance between bus  $q$  and bus  $p$  and  $\delta_{p,q}$  is the angle between reactive and resistive components of  $Y_{p,q}$ . The bus  $p$  may also

be connected to a local power generation unit which generates real and reactive power as  $P_{G,t}$  and  $X_{G,t}$  respectively at time  $t$ .

Variable  $e$  represents a particular EVSE in a charging station such that  $e \in \{1, 2, \dots, m\}$  and the total number of EVSEs in the charging station are  $m$ .  $PEV_t^r$  and  $PEV_t^x$  are the real and reactive load at an EVSE due to PEV charging at time  $t$ . Binary variable  $x_e$  denotes whether a particular EVSE is connected to a PEV. Thus, if the value of  $x_e$  is 1 then the PEV is connected to EVSE and draws current from it.  $\widehat{P}_t$  and  $\widehat{X}_t$  are the cumulative real and reactive load at time  $t$  due to other devices which may also be connected to the bus  $p$ .

The net power flowing in bus  $p$  must be equal to power generated at bus  $p$  subtracted by the power consumed by any load connected to the bus  $p$ . Also, the power flowing in bus  $p$  depends upon the voltage difference and admittance between bus  $p$  and other buses that may be connected to the bus  $p$ .

Therefore, from the power flow analysis [54], the relation between power flowing, power generated, and power consumed in the bus  $p$  is provided by the equations 4.1 and 4.2.

$$P_{p,t} = P_{G,t} - \sum_{e=1}^m PEV_t^r \times x_e - \widehat{P}_t \quad (4.1a)$$

$$P_{p,t} = \sum_Q |V_{q,t}| \times |V_{p,t}| \times |Y_{p,q}| \times \cos(\theta_{p,t} - \theta_{q,t} - \delta_{p,q}), \quad (4.1b)$$

$$X_{p,t} = X_{G,t} - \sum_{e=1}^m PEV_t^x \times x_e - \widehat{X}_t \quad (4.2a)$$

$$X_{p,t} = \sum_Q |V_{q,t}| \times |V_{p,t}| \times |Y_{p,q}| \times \sin(\theta_{p,t} - \theta_{q,t} - \delta_{p,q}), \quad (4.2b)$$

subject to the constrains given by equation 4.3-4.5.

$$|V_p^{min}| \leq |V_{p,t}| \leq |V_p^{max}| \quad (4.3)$$

$$|P_p^{min}| \leq |P_{p,t}| \leq |P_p^{max}| \quad (4.4)$$

$$|X_p^{min}| \leq |X_{p,t}| \leq |X_p^{max}| \quad (4.5)$$

where  $V_p^{min}$ ,  $P_p^{min}$  and  $X_p^{min}$  are the minimum voltage, real power, and reactive power that should flow in bus  $p$  for its proper operation.

$P_p^{max}$ ,  $V_p^{max}$ , and  $X_p^{max}$  are the maximum voltage, real power, and reactive power that bus  $p$  can handle.

For tracking the energy supplied from EVSE to PEV, EVSE sets up the communication link to PEV. The charging station also has a smart meter for tracking amount of energy supplied to it for charging all PEVs. EVSEs also has a smart meter that tracks the amount of electricity supplied to a PEV. We describe below the centralized COP architecture that has potential to provide efficient communication between end units in above architecture.

### 4.1.2 COP architecture

The COP architecture consist of communication, prediction, and optimization units as shown in Figure 4.1. Communication unit consists of an SDN controller responsible for managing network connectivity between PEVs, EVSEs, local, and main grid. The network of PEVs is highly dynamic because the connection between nodes connect and break very frequently due to the mobility of vehicles. In traditional IP-based networks, the intelligence is distributed throughout the network elements like routers and switches. Therefore, network policies need to be configured or altered in every router and switch making network management a cumbersome task. It is also difficult to manage and monitor such a large scale and spatially separated network due to its decentralized architecture. SDN has layered architecture with application, control, and infrastructure layers as shown in Figure 4.1. In SDN domain, routers and switches have a shallow level of intelligence and are merely dumb devices. They only forward packets to other network elements-based upon the entry in a table which is called as a flow table. Every router and switch has its associated flow table that contains the information about what to do with a specific packet arriving at that node. The logically centralized, but physically distributed, SDN controller has topological information of the whole network and decides the policies to be implemented in every node of the network [55]. The SDN controller decides about those policies and correspondingly fills the entries in the flow tables for every node. Since management of whole network is done by the logically centralized SDN controller, the model is scalable because any policy implementation or re-configurations need to be done in SDN controller only, not in every router and switch. Also, since the network of PEVs, EVSEs, and smart grid has both wired

and wireless components, SDN-based network solution can be easily applied to such a dynamic network because of its flexible architecture. Thus, the requests made by the PEVs, the connection between Road Side Units (RSUs) and EVSEs, and between EVSEs to local and main grids can be efficiently managed using SDN controller.

Data prediction unit predicts the short and long-term traffic count based on the spatial and temporal traffic information. Prediction unit obtains the input from loop detectors installed in a freeway and based on that determines future traffic flow. Prediction unit communicates the predicted traffic to the optimization unit. Optimization unit in turn also receives the charging requests from PEVs. The PEV charging requests are collected over an optimization period having length of  $\tau_i$ . For PEV charging requests in period  $\tau_i$ , the optimization algorithm allocates charging station to those set of PEVs. The optimization algorithm runs again after the time interval  $\tau_{i+1}$  to allocate charging station to new set of PEVs making request in the interval  $\tau_{i+1}$ . Thus, optimization algorithm runs periodically after every time interval  $\tau_i, \tau_{i+1}, \tau_{i+2}$  and so on and allocates charging stations to PEVs making request in corresponding time interval. Optimization unit shares the information about allocated charging stations to SDN controller. SDN controller then routes this information to the PEVs and allocated charging stations through an optimized communication link.

## 4.2 Methodology

We describe the prediction unit and optimization unit design for predicting traffic and subsequently allocating charging stations to PEVs. The prediction and

optimization unit processes the information received from PEVs and charging stations. Based on the received information, the units allocate charging station to PEVs.

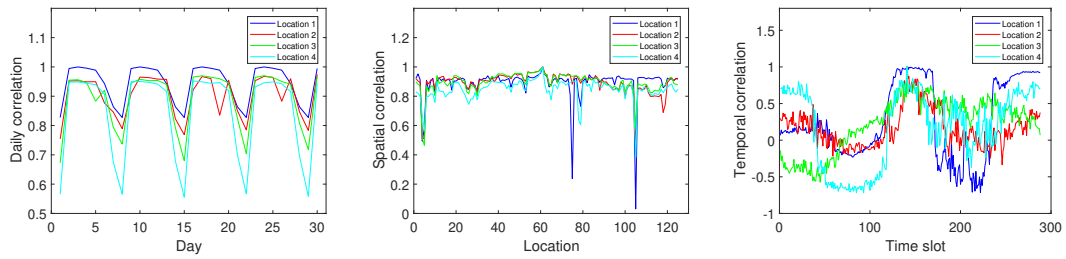
#### 4.2.1 Prediction unit design

This section describes the prediction unit for determining traffic flow in advance. We first extract the daily, spatial, and temporal correlation in traffic flow data then use the analysis for developing the traffic prediction model.

The data used for correlation analysis and traffic prediction is obtained from Performance Measurement System (PeMS) [56]. The PeMS collects the traffic flow in California highways through the loop detectors. The loop detectors collect the traffic flow information every 5 minutes throughout the day and for all days. We call the interval of 5 minutes as one time slot such that there are total number of  $24 * 12$  time slots in a particular day. We collect the traffic flow for October 2017 for the freeway segment of I-680.

##### Correlation analysis

Freeway traffic often has specific patterns which varies with time and location. To determine those patterns, we did cross-correlation analysis to extract temporal and spatial behavior in freeway traffic. We determine the trend in daily traffic at a particular location. For correlation analysis, we consider a matrix  $A$  of dimension  $(31, 24 * 12)$  containing the traffic information at a certain location. Each row of the matrix  $A$  contains the traffic count for a particular day for all



(a) Flow correlation between different days (b) Flow correlation between different locations (c) Flow correlation between different time slots

**Figure 4.3:** Correlation analysis

time slots. We chose  $A_{ref}$  as a particular row of  $A$  such that  $A_{ref}$  contains the traffic information for a particular day of the month at a particular location. To determine the correlation between daily traffic, the cross-correlation between row  $A_{ref}$  and other rows of matrix  $A$  are plotted for 4 different locations in Figure 4.3a. Figure 4.3a show the daily correlation (Pearson's Correlation coefficient) for those locations. The figure depicts that the Pearson's correlation coefficient has a peak value. The value of the coefficient reduces from the peak value and then again increases showing periodicity in its behavior. However, the periodic behavior is not perfect, and it deviates at some point of time and location. Henceforth, albeit the same day traffic in previous weeks can provide information about the expected traffic in the current week, yet the information is not sufficient to provide accurate traffic behavior since the patterns are not perfectly periodic.

Figure 4.3b show spatial correlation information in traffic pattern. We find the correlation of traffic count between different locations for the same day. To obtain this information, the matrix  $A$  is updated to contain the traffic information at different locations for all time slots for a particular day. Hence, a particular row of  $A$  contains the traffic count at a particular location for every time slot

in a single day. Similar to daily correlation information, the Pearson's correlation coefficient is obtained between different rows of matrix  $A$  with respect to a particular row  $A_{ref}$  of  $A$ . The value of correlation coefficient is 1 if the correlation is determined for reference row  $A_{ref}$  with itself. Figure shows that the locations near the reference location (point in the graph where value of correlation is 1) has higher coefficient as compared to the locations away from the reference point. Therefore, at the same point of the time, traffic at nearby points are expected to have similar pattern showing high spatial correlation. Further, the value of the correlation coefficient is high for almost all the locations. The high correlation value enables the same trained algorithm to be used at different points for traffic prediction. Since the proposed method has a centralized prediction unit which gathers the information from every location in a particular zone and has to predict the traffic for those locations, the centralized unit is overwhelmed with the computational burden. Over-and-above, the training process of the prediction algorithms is, in general, computationally intensive. Therefore, the idea that the algorithm is trained at one location and then the same trained parameters are used for predicting traffic at other locations, due to high spatial correlation, reduces the computational demand of the centralized prediction unit.

Further, matrix  $A$  is updated to contain the traffic count for different days for every time slot at a particular location. Hereby, its dimensions are  $(24 * 12, 31)$ . The particular row of  $A$  contains traffic count at a certain time slot for every day of the month. The correlation analysis is depicted in Figure 4.3c. The point in the graph showing correlation coefficient 1 is the correlation of  $A_{ref}$  with respect to itself. The temporal correlation is high near the reference time slot and

decreases as the time difference from reference slot increases. Thus, the correlation between nearby time slots with the reference time slot is higher showing the historical traffic can be utilized to some extent to predict traffic at a particular time. The correlation coefficient also fluctuates randomly in the given plots. The region near  $A_{ref}$  is flat in location 1, but it has a steep slope at other locations. Additionally, the degree of steepness is different at different locations. Therefore, although the average traffic at a certain time slot depends on previous time slots, the dependency decreases for the larger gap between time slots. The behavior also has randomness. Correspondingly, the historical traffic information can be leveraged to predict the traffic volume, the information requires advance method for traffic prediction due to its random behavior.

The aforementioned correlation analysis delineates that the traffic flow has weekly periodicity, spatial, and temporal correlation. While weekly periodicity and temporal correlation enable to predict the traffic but it requires advance methods due to non-linearity in periodicity as well as in correlation analysis. Therefore, to predict the traffic, we use deep neural network-based on the backpropagation algorithm [57]. The high spatial correlation suggests that the deep neural network can be trained at one location and the learned parameters of the network to be used for traffic prediction at other locations. We describe below the neural network that is used for traffic prediction.

### **Neural network for deep learning**

The neural network is organized in the form of a layered graph [57]. We use variable  $l$  to represent the layer of the neural network such that  $l \in \{1, 2, \dots, L\}$ ,

where  $L$  are the total number of layers in the neural network. First layer of the neural network is called as input layer and accepts the input data. The  $L_{th}$  layer is the output layer and provides the output given the input in the first layer. There can be a number of layers between input and output layer and those layers are called as hidden layers. The nodes in the graph are called as neurons. In this section, we use variable  $i$ ,  $j$ , and  $k$  such that  $i$ ,  $j$ , and  $k$  are the set of natural numbers and used to represent the particular instance of any variable. For example, we represent  $x_i$  and  $y_i$  as the  $i_{th}$  instance of input and output vector respectively. The connection between  $j_{th}$  neuron in layer  $l$  and  $i_{th}$  neuron in layer  $l - 1$  has a weight  $w_{j,i}^l$ . The net input to the  $j_{th}$  neuron in layer  $l$  is given as  $net_j^l = \sum_i w_{j,i}^l \times o_i^{l-1}$ . Here,  $o_i^{l-1}$  is the output from  $i_{th}$  neuron in layer  $l - 1$  and summation is performed over all the neurons of layer  $l - 1$ . The neuron processes the input using non-linear activation function  $f$  such that the output of  $j_{th}$  neuron in layer  $l$  is given as  $o_j^l = f(net_j^l)$ .

Neural network operates in two phases. In the first phase, the network is trained by feeding it with known input and output patterns. Once the network is trained, the unknown input data is fed to get the output at the  $L_{th}$  layer. During the training of network, we present a set of known input and output vectors  $x_i$  and  $y_i$ , and based on the prediction error at the output neuron, the weights  $w_{j,i}^l$  in all the connecting links are adjusted such that the desired output is obtained at the output node. Once this adjustment is made, the network is presented with another set of input and output pattern and the process is repeated till the prediction error becomes very small. For each pattern, we

calculate error as in equation 4.6.

$$E = y_i \times \log o_j^l + (1 - y_i) \times \log(1 - o_j^l) \quad (4.6)$$

where  $y_i$  is the target output, and the value of  $l$  to calculate error is  $L$ . We achieve convergence towards improved values of the weights by calculating the incremental change in weight  $\Delta w_{j,i}^l$  which is considered to be proportional to the partial derivative of the error function with respect to weight  $w_{j,i}^l$  as  $\Delta w_{j,i}^l = -\eta \frac{\partial E}{\partial w_{j,i}^l}$ , where  $\eta$  is the learning rate.

For every neuron connection  $w_{j,i}^l$ , the partial derivative of the error with respect to weight can be expressed as in equation 4.7.

$$\frac{\partial E}{\partial w_{j,i}^l} = \frac{\partial E}{\partial net_j^l} \times \frac{\partial net_j^l}{\partial w_{j,i}^l} \quad (4.7)$$

Since  $net_j^l = \sum_i w_{j,i}^l \times o_i^{l-1}$ ,  $\frac{\partial net_j^l}{\partial w_{j,i}^l} = o_i^{l-1}$ . If we define  $\delta_j^l = -\frac{\partial E}{\partial net_j^l}$  then  $\Delta w_{j,i}^l$  is given by the equation 4.8.

$$\Delta w_{j,i}^l = \eta \times \delta_j^l \times o_i^{l-1} \quad (4.8)$$

To find the value of the  $\delta_j^l$ ,  $\frac{\partial E}{\partial net_j^l}$  can further expressed as  $\frac{\partial E}{\partial o_j^l} \times \frac{\partial o_j^l}{\partial net_j^l}$ .

For the output neuron  $\frac{\partial E}{\partial o_j^l}$  is given as  $\frac{y_i}{o_j^l} - \frac{1 - y_i}{1 - o_j^l}$  (derivative of equation 4.6 with respect to  $o_j^l$ ) and  $\frac{\partial o_j^l}{\partial net_j^l}$  is given as  $f'(net_j^l)$ , since  $o_j^l = f(net_j^l)$ , where  $f'$  is the derivative of activation function  $f$ . Thus, for the output node incremental weight change is given as in equation 4.9.

$$\Delta w_{j,i}^l = \eta \times \left( \frac{y_i}{o_j^l} - \frac{1 - y_i}{1 - o_j^l} \right) \times f'(net_j^l) \times o_i^{l-1} \quad (4.9)$$

However, the neurons in hidden layers do not affect the output directly. They affect the activation in the next layer of the network. Thus, to find  $\Delta w_{j,i}^l$ , where  $w_{j,i}^l$  is now the weight of the neuron in hidden layer we need to find  $\frac{\partial E}{\partial o_j^l}$  indirectly using next layer activation. Therefore, to evaluate  $\frac{\partial E}{\partial o_j^l}$ , we have  $-\frac{\partial E}{\partial o_j^l} = -\sum_k \frac{\partial E}{\partial net_k^{l+1}} \times \frac{\partial net_k^{l+1}}{\partial o_j^l}$  ( $k$  is a neuron in layer  $l + 1$ ).

Since  $net_k^{l+1} = \sum_j w_{k,j}^{l+1} \times o_j^l$ ,  $-\frac{\partial E}{\partial o_j^l} = \sum_k -\frac{\partial E}{\partial net_k^{l+1}} \times w_{k,j}^{l+1} = \sum_k \delta_k^{l+1} \times w_{k,j}^{l+1}$ . In this case,  $\delta_j^l = f'(net_j^l) \times \sum_k \delta_k^{l+1} \times w_{k,j}^{l+1}$ . Thus, for the inner layer neurons the incremental weight change is given by the equation 4.10.

$$\Delta w_{j,i}^l = \eta \times f'(net_j^l) \times \sum_k (\delta_k^{l+1} \times w_{k,i}^k) \times o_i^{l-1} \quad (4.10)$$

In equation 4.10, the term  $f'(net_j^l) \times \sum_k (\delta_k^{l+1} \times w_{k,i}^k)$  is the  $\delta_j^l$  for hidden layer  $l$  and is evaluated in terms of  $\delta_k^{l+1}$  in layer  $l + 1$ . That is deltas at an internal node can be calculated in terms of deltas at the upper layer. The weights are updated as in equation 4.11, where  $\Delta w_{j,i}^l$  is calculated using equation 4.9 or 4.10 depending on whether neuron is an output neuron or hidden unit neuron.

$$w_{j,i}^l = w_{j,i}^l - \eta \times \Delta w_{j,i}^l \quad (4.11)$$

### Input feature vector selection

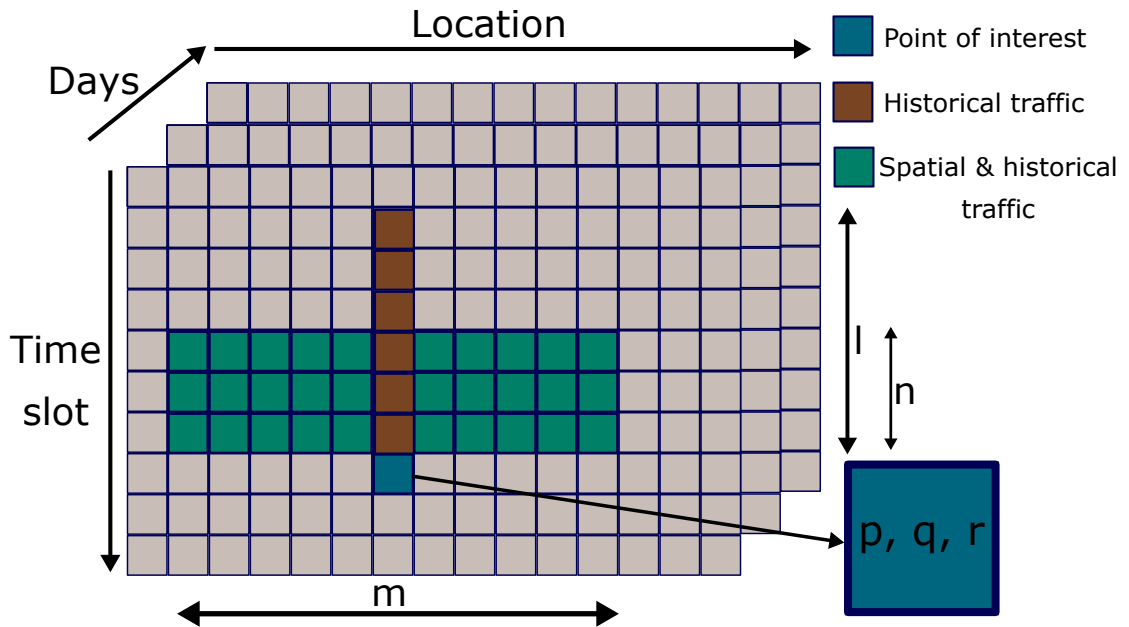


Figure 4.4: Flow matrix

The deep neural network is fed with known input and output patterns for the training purpose. The input is collected in a vector  $x_i$  which is called as an input feature vector. Hereby, we utilize the periodicity, spatial, and temporal correlation in traffic flow to get input feature vector for traffic prediction. We collect the relation between temporal and spatial traffic information using flow matrix  $F$ . Figure 4.4 shows the three-dimensional flow matrix containing traffic flow information. Suppose the traffic needs to be predicted for the time slot having coordinates  $\{p, q, r\}$ . The temporal traffic information is determined from the rectangle of length  $l$ . The length of the rectangle  $l$  represents previous  $l$  time slots if traffic needs to be predicted at time slot  $\{p, q, r\}$  such that time slots  $\{p-l, q, r\}$  to  $\{p-1, q, r\}$  are the part of the input feature vector  $x_i$ . The traffic count for the same day of the week in previous  $s$  weeks is appended to the

vector  $x_i$ . Further, the spatial and temporal traffic information is also combined from the rectangle of width  $m$  and height  $n$  as shown in the figure (ignoring overlapped region which is already appended) such that  $n$  are the previous time slots and  $m$  are the nearby locations. Thus, the traffic near  $m$  points in previous  $n$  slots is also appended in the vector  $x_i$ . The input vectors for different values of coordinates  $\{p, q, r\}$  (points where traffic needs to be predicted) are combined in an input matrix  $I$  as in equation 4.12.

$$I = [x_1^T, x_2^T \dots x_{IO}^T] \quad (4.12)$$

where,  $x_i^T$  is the transpose of vector  $x_i$ . The known traffic for the input  $x_i$  is  $y_i$  (traffic at location  $\{p, q, r\}$ ). The output array containing known traffic is given by equation 4.13. Here  $IO$  are the total number of input and output patterns.

$$Y = [y_1, y_2, \dots, y_{IO}] \quad (4.13)$$

The number of columns of matrix  $I$  is equal to the number of columns in the matrix  $Y$  such that for a particular column in  $I$  the corresponding column in  $Y$  is the known output traffic given that input.

### Backpropagation learning

The training of deep neural network requires many parameters to be adjusted for its efficient implementation. The parameters include, number of layers  $L$  of the deep neural network, number of neurons in each hidden layers  $n_h^l$  ( $l$  is a particular layer of the network), learning rate  $\eta$  of the back-propagation

algorithm, number of iterations for which algorithm should run  $iter$ , activation function for hidden or output neuron  $f$ , and initial weight  $w_{j,i}^l$  between every neuron  $j$  in layer  $l$  and neuron  $i$  in layer  $l - 1$  [57]. We considered the following state space for choosing various parameters:

$$L \in \{3, 8\}, n_h^l \in \left\{ \frac{|x_i|}{2}, |x_i * 2| \right\}, \eta \in \{0.1, 2\}, iter \in \{1000, 100000\},$$

$$f \in \{Sigmoid, Tanh, relu, leaky relu\}$$

Even for above small sets, the selection of optimum parameters is a very complex problem. Due to large state space, an exhaustive search of above parameters to find an optimum set of parameters is infeasible. Since in prediction problem the objective is to predict traffic flow with low error and less training time, we follow a combination of random and greedy search to select parameters for our training model. Parameters are searched within above set such that the traffic is predicted with a high accuracy and low training time. Before training the network, the matrix  $F$  is normalized using min-max method.

We randomly initialize weights between  $\{-1, 1\}$  having the normal distribution. The number of layers in the deep neural network are greedily searched in the range mentioned above. The learning rate  $\eta$  and the number of iterations for every value of the  $\eta$  are examined for the different number of layers. Similarly, the sigmoid activation function  $f$  is chosen for the hidden layer and relu activation for the output layer.

The neural network is fed with training set input and output patterns individually or in a batch mode. In a batch mode, the whole data set is divided into batches of small size and each batch of data is fed into the network one by one.

---

**Algorithm 5** Charging station set for PEVs
 

---

**Input:** 1). Set of charging requests in one optimization interval

2). Location of charging stations

**Output:** Tuple  $M$ - Potential charging stations for PEVs

```

1: for  $i \leftarrow 1$  to  $N$  do
2:    $d_i^{max} = \frac{SOC_i^{curr} - SOC_i^{min}}{r_i^{dc}}$ 
3:   for  $j \leftarrow 1$  to  $m$  do
4:     if  $\gamma_i == 1$  then
5:       if  $l_j^c > l_i$  and  $l_j^c < l_i + d_i^{max}$  then
6:          $M_i.append(CS_j)$ 
7:       end if
8:     else
9:       if  $l_j^c < l_i$  and  $l_j^c > l_i - d_i^{max}$  then
10:         $M_i.append(CS_j)$ 
11:      end if
12:    end if
13:  end for
14:   $M.append(M_i)$ 
15: end for

```

---

Based on the average error in a batch, the weights are adjusted between neuron connections. However, in both cases weights are adjusted several times. To avoid this, we combine the input training pattern as in equation 4.12 and 4.13 and present all the patterns at once. Then, based on the average value of the error on whole dataset, the weights are updated for every iteration. The advantage of the approach is the lower training time because the weights are not updated many times like training with individual input and output pattern or as in batch mode.

### 4.2.2 Optimization unit design

This section presents the optimization unit design for allocating charging station to PEVs. The optimization unit receives charging requests from PEVs and

allocates charging station based upon multiple parameters related to charging and discharging of PEVs. For the forthcoming discussion, if  $M$  is a set, we use the notation  $M(l)$  to denote  $l_{th}$  element of the set  $M$ ,  $M(l).index$  as the index of the element  $M(l)$  in set  $M$ , and  $||M||$  as the cardinality of the set  $M$ .

---

**Algorithm 6** Optimal Charging station allocation to PEVs

---

**Input:**  $M$ - Potential charging station set,  $l_j^c$ - Location of charging stations

**Output:**  $S$ - Set of assigned requests

```

1:  $M.sort(cardinality)$ 
2: for  $i \leftarrow 1$  to  $N$  do
3:    $M_i.sort(chargingTime)$ 
4: end for
5: for  $i \leftarrow 1$  to  $N$  do
6:    $tmpStation = M_i(1)$ ,  $k = 1$ 
7:   for  $j \leftarrow k$  to  $||M_i||$  do
8:      $P = M$ 
9:      $d1 = \sum_{i+1}^N t_i^c(1), UpEffTime(tmpSt, R_i, M)$ ,  $d2 = \sum_{i+1}^N t_i^c(1)$ 
10:     $diffFirstEle = d2 - d1$ ,  $M = P$ 
11:    for  $l \leftarrow k + 1$  to  $||M_i||$  do
12:       $P = M$ 
13:       $d1' = \sum_{i+1}^N t_i^c(1), UpEffTime(tmpSt, R_i, M)$ ,  $d2' = \sum_{i+1}^N t_i^c(1)$ 
14:       $diffFirstEle' = d2' - d1'$ ,  $M = P$ 
15:       $FirstEleCh = diffFirstEle' - diffFirstEle$ 
16:       $diffSameSet = t_i^c(l) - t_i^c(k)$ 
17:      if  $diffSameSet < FirstEleCh$  then
18:         $tmpStation = M_i(l)$ ,  $k = l$ 
19:      end if
20:    end for
21:  end for
22:   $S_i = tmpSt, UpEffTime(S_i, R_i, M)$ 
23:   $S.append(S_i)$ 
24: end for

```

---

### Problem statement

We represent PEV charging request as  $i$  such that  $i \in \{1, 2, \dots, N\}$  and  $N$  are the total number of requests. To represent different charging stations variable

$CS_j$  is used so that  $j \in \{1, 2, \dots, m\}$  and the number of charging stations in a particular zone are  $m$ . Variable  $l_j^c$  is used to represent the location of  $j_{th}$  charging station. Charging rate of EVSE at station  $j$  is represented as  $r_j^c$ . The information set of a vehicle making a request  $i$  at time  $t$ ,  $R_i$ , is depicted in set  $(t_i, l_i, \gamma_i, s_i^{avg}, SOC_i^{curr}, SOC_i^{min}, SOC_i^{req}, r_i^{dc})$ , such that the time at which request is made is given by  $t_i$ ,  $l_i$  is the location of the vehicle at time  $t_i$ ,  $\gamma_i$  the direction in which PEV is traveling, and  $s_i^{avg}$  is the average speed of the vehicle.  $\gamma_i$  a binary variable representing the direction of travel. Here, we assume highway as a one dimensional axis with increasing value of location while traveling left to right. If a PEV is traveling towards right then the value of  $\gamma_i$  is considered to be equal to 1. The charging level of PEV is given as  $SOC_i^{curr}$ ,  $SOC_i^{min}$ , and  $SOC_i^{req}$  to represent current, minimum and required status of charge of PEV. Variable  $r_i^{dc}$  represents the rate at which PEV  $i$  discharges. Equation 4.14 evaluates the maximum possible distance  $d_i^{max}$  a vehicle can travel depending upon its charging level.

$$d_i^{max} = \frac{SOC_i^{curr} - SOC_i^{min}}{r_i^{dc}}. \quad (4.14)$$

The optimization period is considered to have a length of  $\tau$ . The requests are collected in one optimization interval and allocated charging stations. Equation 4.15 represents the time-slot  $\tau_n$  when a vehicle with charging request  $i$  reaches the charging station  $j$ . Equation 4.16 provides the sum of the travel time and effective charging time of  $i_{th}$  vehicle at  $j_{th}$  charging station, where  $t_{j,n}^w$

is the wait time at charging station  $j$  during time slot  $\tau_n$ .

$$\tau_n = \left\lceil \frac{t_i + \frac{|l_j^c - l_i|}{S_i^{avg}}}{\tau} \right\rceil \quad (4.15)$$

$$T_{i,j} = \frac{|l_j^c - l_i|}{S_i^{avg}} + t_{j,n}^w + \{SOC_i^{req} - (SOC_i^{curr} - r_i^{dc} \times |l_j^c - l_i|)\} \times r_j^c \quad (4.16)$$

The optimization objective is to minimize the function represented in equations 4.17.

$$\min \left\{ \frac{\sum_{i=1}^N t_{j,n}^w + \{SOC_i^{req} - (SOC_i^{curr} - r_i^{dc} \times |l_j^c - l_i|)\} \times r_j^c}{N} \right\} \quad (4.17)$$

subject to the following constrains represented by equations 4.18 and 4.19.

$$\{l_i \leq l_j^c \leq l_i + d_i^{max}\} \times \gamma_i \quad (4.18)$$

$$\{l_i - d_i^{max} \leq l_j^c \leq l_i\} \times (1 - \gamma_i) \quad (4.19)$$

Equation 4.18 and 4.19 are the constraints that need to be satisfied between the charging stations location, location of vehicle, and the maximum distance it can travel satisfying its SOC requirement.

We use the neural network-based traffic prediction model to estimate the initial wait time near different charging stations. The initial wait time near a charging stations is evaluated using equation 4.20, where  $\Gamma$  is the arbitrarily chosen

constant and  $y_{j,n}^{\hat{}}$  is the predicted traffic at  $j^{th}$  charging station and time slot  $\tau_n$ .

$$t_{j,n}^w = \Gamma \times y_{j,n}^{\hat{}}, \quad (4.20)$$

### Set cardinality-based search

To solve the above optimization problem, algorithms 5 and 6 are used. For effective charging time update algorithm 6 employs algorithm 7.

Depending upon its  $SOC_i^{curr}$  and  $SOC_i^{min}$ , a PEV can be charged in some possible charging station. For PEV charging request  $i$ , we represent such possible stations using set  $M_i$ . To determine  $M_i$  for each request  $i$ , we use algorithm 5. Algorithm 5 has a loop that iterates between steps 1-15 for all the charging requests  $i$ . The maximum distance a vehicle can cover while keeping its charging level more than  $SOC_i^{min}$  is found in step 2. For every charging station the steps 3-13 are repeated to find potential number of charging stations. The direction in which a PEV is traveling is determined in step 4. The charging station  $CS_j$  is added in set  $M_i$  if its location is between the PEV location and the distance  $d_i^{max}$  it can travel (steps 5, 6, 9, and 10).  $M_i$  is added to the tuple  $M$  in step 14. Tuple  $M$  of potential charging station sets for every request is used in algorithm 6 and 7 for charging station allocation to the PEVs.

On the basis of tuple  $M$ , algorithm 6 finds the optimal charging station for every PEV request. The tuple  $M$  is sorted (increasing order) according to the cardinality (number of element) in the set  $M_i$  (step 1). While allocating charging station to PEV we give priority to request with less number of potential charging stations. Therefore, we allocate charging station to requests in order of their

position in sorted tuple  $M$  such that we allocate charging station to first element of tuple  $M$  first and then proceed to other elements one by one in order. In steps 2-4, charging stations in set  $M_i$  are sorted in increasing order of their effective charging time for request  $i$ .

---

**Algorithm 7** Update effective charging time

---

**Input:**  $CS_b, R_a, M$

**Output:**  $M$

```

1: for  $i \leftarrow a + 1$  to  $N$  do
2:   for  $j \leftarrow 1$  to  $||M_i||$  do
3:     if  $M_i(j) = CS_b$  then
4:       if EffCharging overlaps then
5:         AddEffChargingTime
6:       end if
7:     end if
8:   end for
9:    $M_i.sort(chargingTime)$ 
10: end for

```

---

Steps 5-24 are repeated for every charging request. For a given PEV charging request, we temporarily allocate first element of set  $M_i$  as a charging station in step 6. We use variable  $k$  for storing the index of the assigned charging station (temporary) in set  $M_i$ . Thus the initial value of  $k$  equal to 1 that gets updated when temporarily assigned station changes. For the request under consideration, steps 7-21 are iterated for each element of  $M_i$ . In step 8, the value of the tuple  $M$  is saved using the variable  $P$ . The sum of effective charging time of

the first element of all requests which are at higher position (having higher cardinality), than the request being considered, in set  $M$  is calculated in step 9. In step 9, we consider  $t_i^c(k)$  as the effective charging time of  $i_{th}$  request element  $k$  of  $M_i$ . Subsequently, effective charging time of requests at higher position in  $M$  is changed assuming temporary station is allocated to the request. The change in effective charging time is done by calling algorithm 7. We pass the assigned station, request, and set of potential station as input parameters to the algorithm 7 (step 9). Sum of the effective charging time of first element at higher position in  $M$  are found again (step 9) to determine the increase in cumulative charging time in step 10. Then,  $M$  is assigned to its original value  $P$  (step 10). Similar to steps 8-10, steps 12-14 are executed with the difference is that the charging station being considered are have index higher than  $k$ . Step 15 finds the difference between values obtained in steps 10 and 14. We find how much assigning a station with higher index  $k$  affects the cumulative effective charging time of first potential station for all sets  $M_i$  at higher position in the set  $M$  than the request being considered. If the increase is less than the difference while assigning a station at higher index in same set  $M_i$  then the one with higher  $k$  is considered for a particular request (steps 17-19). Once all the possibilities are checked, the temporary station is allocated to the given request. It is saved using variable  $S_i$  and the effective charging times of  $M$  is updated. The charging station is added to a set  $S$  (step 23). The updated effective charging time after the whole process is employed for optimal allocation in next interval.

To update the effective charging time in  $M$  the algorithm 7 is used. The algorithm 7 takes the input parameters either from steps 9, 13, or 22 of algorithm 6

and assigns those parameters to the variable  $CS_b$  (assigned station),  $R_a$  (assigned request), and  $M$ . For requests at higher position in set  $M$  the steps 1-10 are repeatedly processed. For every such requests steps 2-8 checks all possible charging stations in set  $M_i$ . Then, it is checked if the station accepted as argument in algorithm is present in potential charging station set. If the charging station is also present in given set then it is checked in step 4 that if wait time and charge time for the two requests at the same station coincides. The overlapping occurs if the time at which vehicle being verified arrives at the particular station lies between the time when allocated vehicle's arrival time and the time when its charging gets complete. The additional wait time of the station being checked is obtained by subtracting arrival time of vehicle that is being checked and the allocated vehicle's charging completion time. The additional wait time is added to the given station in step 5. If some vehicle arrives at the given station for charging then the updated wait time is used. Since we have added the effecting charging time in a charging station the set of potential charging stations need to be sorted again in order of increasing effective charging time. This process of sorting is done in step 9.

### **Complexity analysis**

We evaluate the complexity of the exhaustive search method that find the optimal solution of the problem and the proposed method as outlined below.

*Exhaustive search:* For the worst case analysis, every PEV request can have all

charging stations as potential stations where they can be charged. The best solution requires an exhaustive search for all possible combinations of charging stations and requests. Thus, first request can have  $m$  possible charging station allocation. For, every allocation of charging station to request 1, request 2 can have  $m$  possible allocations totaling  $m * m$  allocation for first 2 requests. Similarly, for  $N$  requests, the total number of searches are  $m * m * m \dots m * m$  ( $N$  times)  $= m^N$  allocations. Thus, the complexity for exhaustive search to find the optimal solution is  $m^N$ . Therefore, optimal solution can not be obtained in polynomial time using exhaustive search.

*Proposed Method:* To evaluate the complexity of the proposed method, we need to find the complexity due to all three algorithms and add them together to get the highest order term. Further, in our analysis, we use the fact that a nested loop with  $N$  steps in both inner and outer loop contribute  $O(N^2)$  due to  $N^2$  steps and sorting process consisting of  $N$  elements contribute  $O(N \log N)$  [58].

First, we consider algorithm 5 that involves two loops; the inner loop executes  $m$  times and outer one  $N$  times thus having complexity of order  $O(mN)$ . Algorithm 6 does the sorting in steps 1-4 that have complexity of  $O(N \log(N)) + O(Nm \log(m))$ . Algorithm 6 then repeats for every request  $N$  times. In every iteration, we have summation term (steps 9 and 13) that processes request from current request to other subsequent request. Therefore, these steps repeat  $N + N - 1 + N - 2 + \dots + 2 + 1 = \frac{N(N-1)}{2}$  times, thus individually having complexity of  $O(N^2)$ . The algorithm 7 is called in steps 9, 13, and 22 that contributes  $O(mN^2)$  due to combined effect of algorithm 6 and steps 1-8 of algorithm 7. Algorithm 6 and step 9 of algorithm 7 contributes  $O(N^2 m \log m)$  due to sorting process  $N^2$

times. Thus, the complexity due to summation steps in algorithm 6 and updating charging time becomes  $O(N^2) + O(mN^2) + O(N^2m \log m) = O(N^2m \log m)$  (dropping lower order terms). Further, summation and updating of charging time steps are within the loops in steps 7-21 that execute  $m^2$  times. Therefore, complexity due to algorithm 6 and 7 should be multiplied by  $m^2$  (except sorting terms that are outside inner loops in steps 7-21) so that it becomes  $O(N \log(N)) + O(Nm \log(m)) + O(m^2 N^2 m \log m) = O(m^3 N^2 \log m)$ . Adding this complexity with that of algorithm 5, the net complexity due to all 3 algorithms becomes  $O(mN) + O(m^3 N^2 \log m) = O(m^3 N^2 \log m)$ . Thus, the proposed method is solvable in polynomial time due to reduced search space.

### 4.3 Results

Both deep neural network and optimization algorithms are implemented using simulation model in python. The freeway segment in California I-680 has been used for the analysis. We used the realistic traffic trace from the month of October 2017. The location of charging stations are randomly selected in the freeway segment. For our analysis, we considered all the vehicles as PEVs.

For the neural network model, we selected parameters using random and greedy search, as explained in section 4.1.3, such that the prediction error is low. Thus we use a 3 layer neural network with 25 neurons in each hidden units. The learning rate  $\eta$  and number of iterations are selected as 0.5 and 10000 respectively. We chose the values of  $l = 6$ ,  $m = 5$ ,  $n = 3$  (Figure 4.4), and  $s = 2$  (number of previous weeks). Relu activation is selected for hidden units and

sigmoid for output unit. We consider about 80% of data set for training the neural network and 20% for the testing the performance of the network.

**Table 4.1:** Simulation parameters for en route PEV charging

Parameter	Value
SOC (current)	8 - 15 kWh
SOC (minimum)	2 - 5 kWh
SOC (required)	15 - 50 kWh
Vehicle location	0 - 100 mile
Request time	0 - 15 minute
PEV avg. speed	30 - 40 mph
Discharge rate	0.1-2.5kWh/mile
Charging rate	1 - 5 kWh
Station location	20 - 120 miles

For optimization unit, we collected requests in one time slot and found out the average effective charging time of the PEVs if they were allocated charging stations based on the proposed method. Table 4.1 shows the parameters associated with the PEV charging and discharging process that are used in experiments. Within the given range the parameters are chosen arbitrarily. The requests are collected in a period 15 minutes and assigned charging station. We run the proposed optimization algorithm for 100 iterations and found the average effective time. To determine if the given algorithms can be practically implemented we also found their run-time.

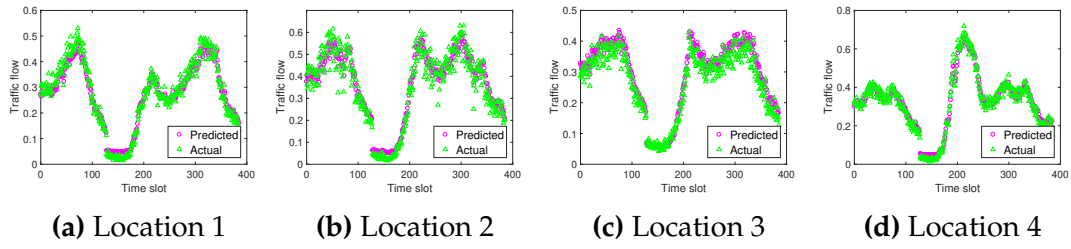
### 4.3.1 Performance metrics

For the prediction unit we compare the actual traffic  $y_{j,n}$  and predicted traffic  $\hat{y}_{j,n}$  at  $j$ th charging station and different time slots  $\tau_n$ . The difference between actual and predicted traffic  $y_{j,n} - \hat{y}_{j,n}$  is also plotted. For different locations and multi-step ahead predictions, we also evaluate the average of the absolute value of errors and Root Mean Square Error (RMSE) between  $y_{j,n}$  and  $\hat{y}_{j,n}$ .

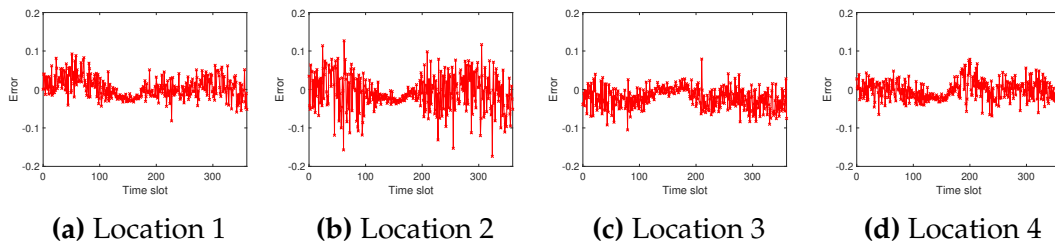
For optimization unit, we compare the *effective charging time* obtained using shortest distance-based method and proposed method. In shortest distance-based method, PEVs charge themselves at the nearest charging station in their direction of travel.

### 4.3.2 Prediction unit results

We evaluate the performance of the proposed neural network algorithm when training is done at one location and the trained network is used for prediction at different locations. Figure 4.5 compares the actual traffic  $y_{j,n}$  and predicted traffic  $\hat{y}_{j,n}$ . Figure 4.6 captures the error between actual and predicted values. As depicted in Figure 4.5, for the linear region of the graph the difference between actual and predicted values is almost negligible. However, the difference becomes large near the peak values of the curve. The prediction error varies between 0-15% for different locations. Error reaches around 10-15% when there is sharp variation in traffic flow. The average and RMSE error as shown in table 4.2. Figure 4.7 and 4.8 portrays the variation in RMSE and average error



**Figure 4.5:** A comparison of actual and predicted traffic



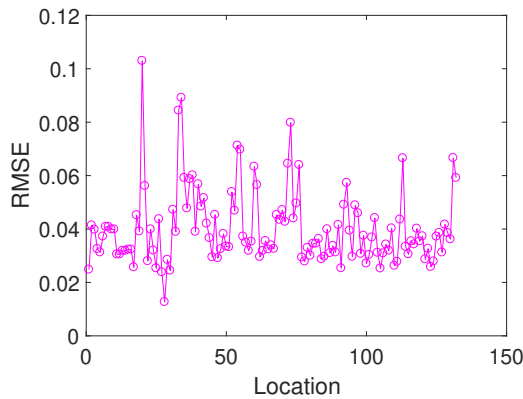
**Figure 4.6:** Prediction error

at 136 different locations while training is done at a single place. Figures depict that the RMSE and average error values follow similar patterns at different locations and also accurately predict traffic due to low error values. The average error varies between 1% to 8% and RMSE error between 0.01 to  $\approx 0.1$  as depicted in figures.

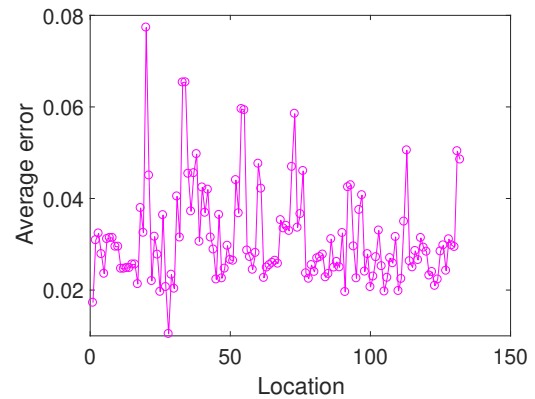
**Table 4.2:** Performance for training at one location and prediction at other

Location	RMSE	Avg. error
Location 1	0.0280	0.0220
Location 2	0.0469	0.0368
Location 3	0.0345	0.0272
Location 4	0.0258	0.0209

Figure 4.9 and 4.10 show the performance of the algorithm when predicted data



**Figure 4.7:** RMSE values at different locations

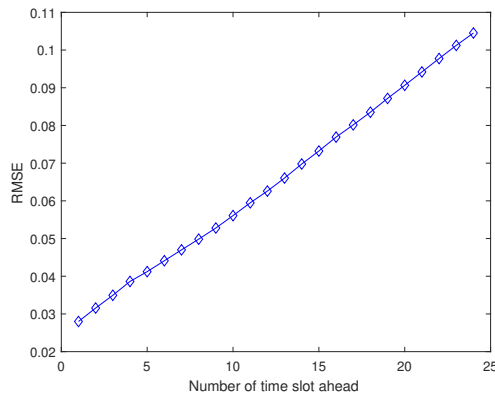


**Figure 4.8:** Average error at different locations

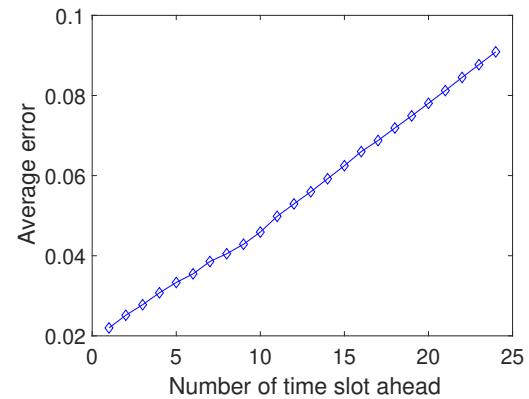
is used for the prediction for subsequent time slots. We use predicted data to make up to 2 hours of advance prediction. Figures show that the average and rmse error increases linearly with the number of time slots. However, the maximum error that the network reaches is 10% which is still in an acceptable limit. Thus, the prediction performance varies with the average error between 3% to about 10% when predicted data is used for the prediction.

### 4.3.3 Optimization unit results

We first change the number of charging requests and observe how the average effective charging time changes. Figure 4.11 shows the corresponding variation. There is a remarkable difference between the proposed method and the decentralized minimum distance-based method. Specifically, for 50 requests the average effective charging time using proposed method is 0.65 times while for 500 requests it is about 0.037 times than that of minimum distance-based method.



**Figure 4.9:** RMSE with multi-step ahead prediction

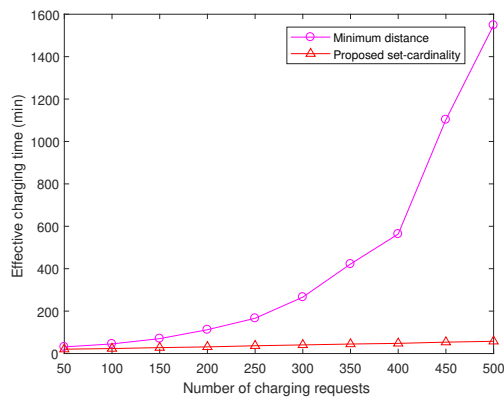


**Figure 4.10:** Average error with multi-step ahead prediction

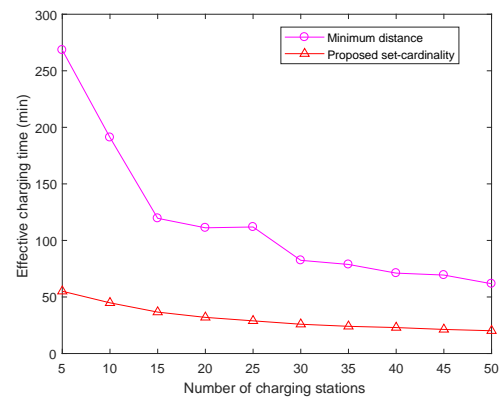
It can also be observed from the figure that the decentralized approach has exponential variation thus charging time increases rapidly on increasing number of vehicles. On the other hand, the proposed method has a linear variation. This is because rather than PEVs going to few congested charging station our approach disperses charging station allocation among charging stations based on their charging and discharging parameters. Therefore, the proposed method performs much better for large number of requests as compared to minimum distance-based approach.

Figure 4.12 show the variation in average effective charging time as number of charging stations are varied for the constant number of PEV charging requests. As expected the effective charging time decreases for both minimum distance and set cardinality-based methods. It is 0.20 and 0.326 times using proposed method than shortest distance method for 5 and 50 stations respectively. However, the graph for set cardinality method becomes almost parallel to the horizontal axis and there is not much decrease in the effective charging time. Thus, from increasing number of stations from 30-50 the proposed method shows the

reduction of only 5.7 minutes. This deduce that for the scenario of the proposed method less number of charging stations can be used without much degradation in the performance in terms of the charging time. Therefore, proposed method not only decreases the charging time but also economically efficient since it uses less number of charging stations to achieve better performance.



**Figure 4.11:** Average charging time as number of vehicles are varied with constant number of charging stations

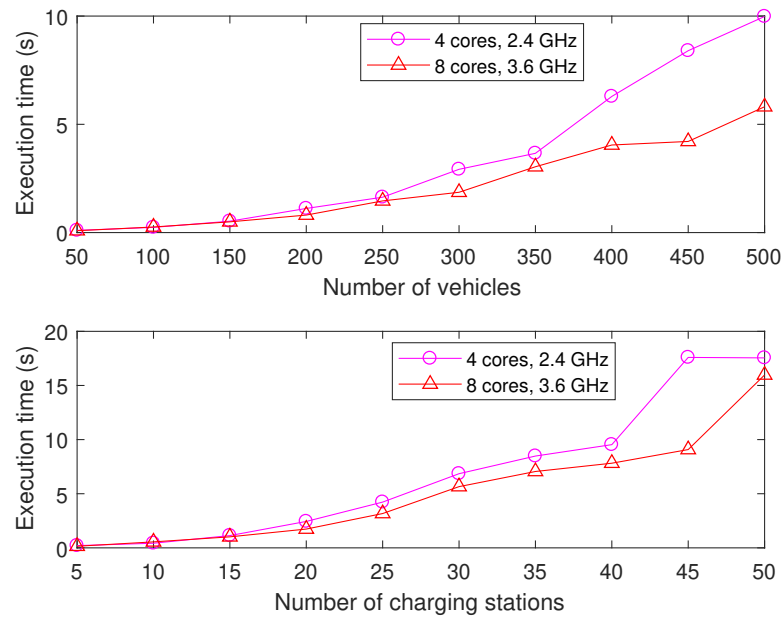


**Figure 4.12:** Average charging time as number of charging stations are varied with constant number of vehicles

#### 4.3.4 Timing analysis

**Table 4.3:** Prediction unit timing analysis

Location	$T_{train}$ (s)	$T_{pred}$ (s)
Location 1	304.98	0.0020
Location 2	361.61	0.0007
Location 3	361.61	0.0017
Location 4	426.27	0.0010



**Figure 4.13:** Execution time of the proposed algorithms

Table 4.3 show the training ( $T_{train}$ ) and prediction ( $T_{pred}$ ) time for the neural network at four different locations. The training time is around 5 minutes and prediction time is around one-hundredth of seconds when algorithm is run in 4 core, 2.4 GHz processor, and ubuntu operating system . Since the network needs to be trained only for single location, the training time of 5 minutes is in feasible limit and does not introduce much computational overhead in the performance of the prediction unit.

In Figure 4.13, the run-time of the optimization method is plotted. We run proposed algorithms in 8 cores, 3.6 GHz and 4 cores, 2.4 GHz CPU processors. The ubuntu operating system was used for running the model. It is observed that the maximum execution time is 30 second that is negligible considering that the requests are collected every 15 minutes. The figure depicts that the runtime of the algorithm is less than 30 seconds for both configurations.

## 4.4 Observations and remarks

This chapter highlights the importance of a scalable SDN-based network for large scale and spatially separated dynamic network. The proposed prediction unit not only estimates the future traffic accurately but is also efficient to implement since it does not require training at multiple locations. The use of predicted data to estimate long-term traffic also achieves good accuracy. Our set cardinality-based optimization unit outperforms than the traditional minimum distance-based charging station selection. Simulation results validate the effectiveness and implementation feasibility of the proposed method. The prediction unit can further be improved by estimating other parameters like power requirement at different charging stations, travel time estimation, and cost prediction under dynamic pricing scenario. Thus, a user can be informed in advance where he should charge his vehicle based on several estimated parameters.

The optimization unit can also be extended further. The optimization problem considering the charging rate dependent on both PEV battery and EVSE can be investigated. Also, in the state-of-the art technology, the PEVs can be charged using different electric voltage levels, namely Alternating Current (AC) levels 1, 2, or 3 and Direct Current (DC) level 1 or 2 [7], [59]. Therefore, the efficient use of the different charging levels, for en route PEVs, to reduce charging time needs to be researched out.

## Chapter 5

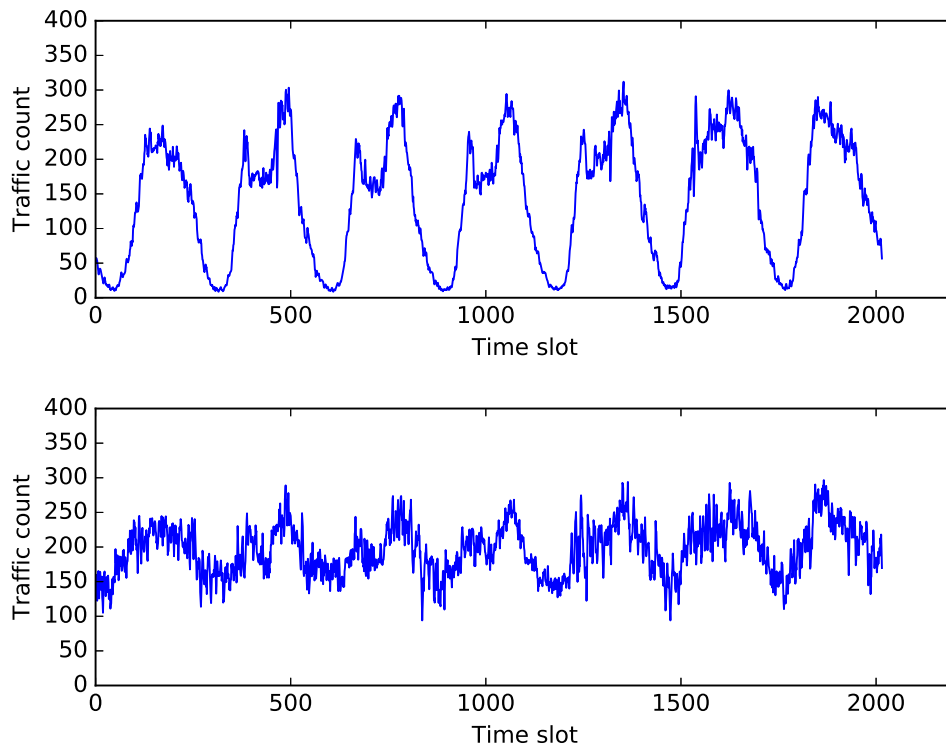
# Anomaly detection

The chapter 3 and 4 have described the PEV charging applications in the smart city. The given PEV charging applications depend upon data like traffic count, SOC level, vehicle speed, location etc. The data is transferred to a centralized location where it is used to make a decision. However, there can be anomalies in the data. Anomalies are any variation in the usual data pattern. For example, anomaly in traffic sensor data can occur due to traffic congestion. In the presence of anomalies, the application needs to make an alternate decision like traffic rerouting. Therefore, detection of anomalies in the data is an important problem. Further, it is also possible that an adversary may take control of sensor data and manipulate it in the sensor itself or while data traverses from sensor to cloud. Such manipulation will give rise to malicious anomalies. If malicious anomalies are present then the application will make unwanted alternate decisions. This will have an adverse effect in the performance of the application. Therefore, finding anomalies and determining whether it is due to variation in surrounding environmental conditions or malicious manipulation of sensor

data is a critical problem for efficient provisioning of smart city-based applications. Due to its importance, from this chapter, we will discuss the anomaly detection problem in the context of smart city in more detail.

Smart city, based on Internet of Things (IoT), envisions the presence of ubiquitous devices and sensors for continuous measuring of environmental parameters [60], [59], [61]. In the context of IoT, detecting anomalies is difficult due to the volume and variability of the data. Also, a large number of sensors are present in IoT, making problem further difficult. The IoT devices generate data having diverse patterns and characteristics. Therefore, developing anomaly detector satisfying the need of every sensor is difficult. Figure 5.1 shows the vehicular traffic near two different locations. It can be observed that the two time-series differ significantly. Therefore, an anomaly detector developed for one of them cannot be effectively used for the other. Thus, the traditional anomaly detectors, that does not consider the diversity in the data and presence of a large number of sensors, cannot be effectively used in the IoT domain. If conventional methods are applied in the IoT domain, then a separate model is required for each sensor. Such a system suffers from scalability issue due to the presence of a large number of sensors in IoT. Therefore, for IoT, a scalable anomaly detector needs to be researched out.

Further, several state-of-the-art anomaly detectors relies on different assumptions about the sample data-set. For example, the conventional Grubb's test assumes that the sample size should be large enough for high detection accuracy [62]. The assumption about the sample size of data cannot always hold, and therefore, it is not an effective method for every condition. Anomaly detectors relying on statistical properties of sample data assume that the data comes from



**Figure 5.1:** Traffic count at different locations

a particular distribution [63]. The assumption cannot always be true. Therefore, although the statistical characteristics of a sample data-set are useful for detecting anomalies, relying solely on them reduces their effectiveness. Many anomaly detectors use supervised learning on training data having labeled (or known) anomaly patterns [64]. However, due to diversity and volume of data, the labeled anomaly patterns may not always be available. Therefore, supervised learning on existing anomaly patterns is not an efficient technique.

In this chapter, we spell out an anomaly detector that is scalable and does not depend on the underlying assumptions about the sample data or labeled anomalies. We employ Hierarchical clustering to impart scalability to the anomaly

detector. Hierarchical clustering finds the correlated sensors and forms clusters. The sensors in a cluster have similar measurement patterns and an anomaly detector developed for one of them can be used for every sensor in a cluster. For detecting anomalies, we partition time-series into segments so that the effect of data distribution is minimized. We use robust statistics M-estimators coupled with the LSTM neural network to detect anomalies in a time series segment. The proposed method that combines the M-estimators and LSTM neural network can effectively detect up to 50% anomalies in every time-series segment.

## **5.1 Preliminaries**

Before proceeding to the proposed methodology, we explain below the problem statements and the motivations for using Hierarchical clustering and Long Short-Term Memory (LSTM) neural networks, in the context of the proposed scalable anomaly detector for Internet of Things (IoT) sensors in smart city.

### **5.1.1 Problem statements**

The anomalies in time-series data are the points in the sample that does not have an expected value. The problem to find anomalies in IoT is divided into two parts.

The different IoT sensors provide a diverse range of time-series patterns. There are also a large number of sensors in IoT. Therefore, the first problem is to determine correlated sensors in IoT. We propose to use Hierarchical clustering to find the sensors that have correlated measurements. Once correlated sensors

are obtained, the second step is to develop an anomaly detector for a correlated cluster of sensors. Therefore, the second problem is to accurately find anomalous points in the sample. We use Long Short-Term Memory (LSTM) neural network coupled with the robust statistical analysis to find the anomalous points or anomalies in IoT.

### 5.1.2 Hierarchical clustering

Hierarchical clustering is one of the many clustering techniques that is used for the data analysis purpose [65]. We use Hierarchical clustering since it is flexible and has fewer assumptions about the underlying data patterns. The methods like K-means algorithm assumes that the number of clusters is known in advance. The density-based methods like Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Ordering Points to Identify the Clustering Structure (OPTICS) are based on the different density between clusters [66]. The distribution-based algorithm like expectation maximization method assumes that the data comes from a particular distribution like normal or gaussian [67]. Support Vector Machines (SVM) is also a popular mechanism, but it is computationally expensive [68].

In the given context, we have a large number of sensors providing disparate time-series patterns. The problem is to determine sensors that provide similar measurement patterns. Thus, it is not known in advance the number of correlated clusters of sensors. Additionally, we consider that sample distribution is not known in advance. The clustering method should also be less

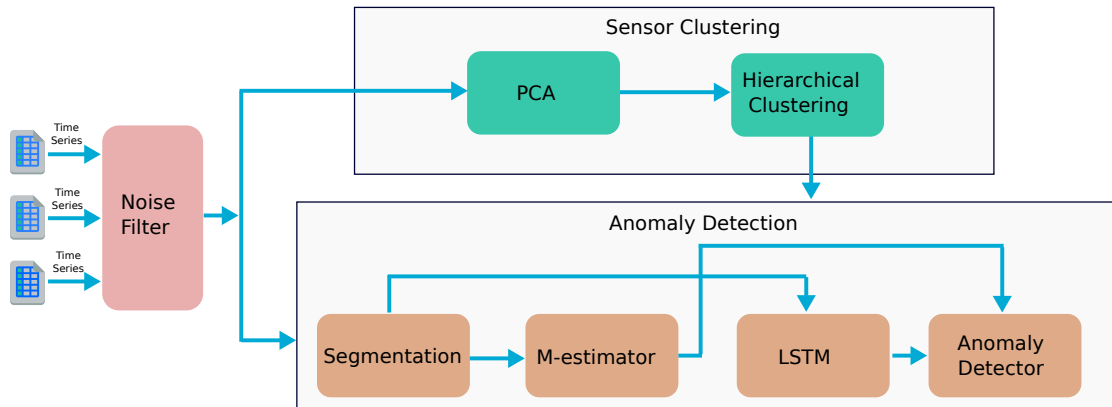
computation-intensive. Since, the Hierarchical clustering is a light-weight and non-parametric technique, it aptly fits in our problem.

In the Hierarchical clustering method, initially, all the data points are considered as an individual cluster. Subsequently, it repeatedly performs the two steps. First, it identifies the two clusters that are close to each other. Then, it merges those clusters to form one big cluster. The process is repeated until the entire data-set is collected in a single cluster. Depending upon the variation in the distance between two clusters that are being merged, the user can determine the number of clusters for an application.

### **5.1.3 LSTM neural network**

There are different methods for time-series analysis. For example, the Auto Regressive Integrated Moving Average (ARIMA) is often used for time-series analysis. However, LSTM network is found to be efficient in capturing long-term relations between temporally separated data points in sequential data. LSTM is suitable for time-series data analysis as it has memory units. Thus, for non-linear and time-series data-sets the LSTM is often suitable as compared to other methods [69]. Therefore, this chapter has used LSTM for the anomaly detector problem.

LSTM neural network contain cells for processing data where each cell has input, output, and forget gate. The gates keep track of amount of information transferred from input to output and next cell. The input gate controls the extent up to which input value is passed to the cell, forget gate controls the information to be remembered and passed to the next cell, and output gate controls



**Figure 5.2:** Architecture describing anomaly detection process

the amount of information from input and previous gate to be used as the output [69]. Thus, the cell in the LSTM neural network acts as its memory and helps in extracting long-term dependencies in the data.

## 5.2 Anomaly detector architecture

Figure 5.2 shows the overview of the proposed model. The proposed model is deployed at the centralized cloud unit. The measured data from spatially distributed sensors is transferred to the centralized unit for use in different applications [56]. In cloud, before using data for a specific application, it is analyzed using the given model to detect the presence of anomalies.

The proposed model consists of the two modules, namely the sensor clustering module and anomaly detection module. As shown in the figure, the time-series data is first passed through the filter to remove noise and is then fed to the clustering and the anomaly detection modules.

The clustering process is performed offline on the recorded time-series data from every sensor. We assume that the data used for the sensor clustering does not contain any anomaly. Before clustering, we use the Principal Component Analysis (PCA) on time-series to reduce its dimensions. Thus, the data fed to the Hierarchical clustering has lower dimensionality and makes clustering process computationally efficient. After PCA analysis, the Hierarchical clustering determines the correlated sensors.

The anomaly detection module is performed both offline and online. During offline phase, the LSTM neural network is trained on known data-set for its optimized performance. During online phase, the module is used for detecting anomalies. Anomaly detection module contains a statistical module (M-estimator) and LSTM neural network module. The time-series data is first segmented, and then on every segment statistical analysis is performed. The time-series segment is also fed to the LSTM neural network for estimating range in every segment. The LSTM neural network uses the processed information from clustering module. The processed data from both the M-estimator module and LSTM neural network module is combined in the anomaly detection to find the anomalies in every segment. The following section describes the different modules and their functioning in detail.

### 5.3 Methodology

We present below the detailed description of the proposed anomaly detector. In our analysis, the time is discretized into the length of the interval  $\tau$ , and every such interval is called a time-slot. We use variable  $i$ ,  $j$ , and  $k$  to denote

location, day, and time-slot respectively. The sensor at location  $i$  is represented using variable  $s_i$ . Variable  $L$  represents the total number of sensors such that  $i \in \{1, 2, \dots, L\}$ . The data is stored in a matrix  $n$ . The values in matrix  $n$  are transformed by normalizing them between 0 and 1 using the min-max scaling.

The measured value at location  $i$ , day  $j$ , and time-slot  $k$  is represented using variable  $n_{i,j,k}$ . Subscript “\_” is used to represent time-series. If one or more subscript ( $i$ ,  $j$ , or  $k$ ) in  $n_{i,j,k}$  is replaced by “\_” then the corresponding variable represents the time-series vector containing all the values of the replaced variable. For example, the time-series representing measured sensor data at location  $i$  on day  $j$  is given as  $n_{i,j,-}$ . Here, the variable for the time-slot ( $k$ ) is replaced by “\_” and thus represents the data for every time-slot. The first element of vector  $n_{i,j,-}$  is the sensor data in the first time-slot, the second is the second data in second time-slot and so on. Similarly,  $n_{i,-,-}$  is one big vector that represents the sensor data in a time-series format at location  $i$  for all days and time-slots. To represent the values for a range of time-slots between  $k$  and  $k + x$  ( $x$  is an integer), we use  $k\_k + x$  in the subscript. Thus, the vector  $n_{i,j,k\_k+x}$  represents the sensor data between time slots  $k$  to  $k + x$ . Similar notation is used for representing range of days and locations. In this chapter, we use symbol “ $\times$ ” to denote scalar multiplication between two numbers or between a number and a vector/matrix. Dot product and element-wise multiplication between two vectors or matrix is represented using symbol “.” and “ $*$ ” respectively.

The time-series data ( $n_{i,j,-}$ ) contains noise. To reduce the effect of noise, the  $n_{i,j,-}$  is passed through the moving average filter. As shown in Figure 5.2, the time-series is passed through a noise removal filter. Filtered data is used in clustering as well as in the anomaly detection module. For detecting anomalies, the first

step is to find the cluster of correlated sensors and is described in the following section.

### 5.3.1 Clustering module

We reduce time-series vector near every sensor to a low dimensional representation using Principal Component Analysis (PCA). Reducing time-series to a low dimensional representation makes Hierarchical clustering in the subsequent step computationally efficient. Thus, the Hierarchical clustering, instead of using entire time-series vector, uses its low dimensional representation and is computationally efficient.

#### Principal Component Analysis

For clustering, we use the time-series data for one week from every sensor. The vector representing weekly time-series data is represented using variable  $v_i$ , where  $v_i = n_{i,1-7,-}$ . The vector  $v_i$ , for every  $i$ , is processed using Principal Component Analysis (PCA) and reduced to a low cardinality representation while preserving patterns in the original sample.

PCA is an unsupervised machine learning algorithm that is used for data analysis [70]. It reveals the hidden structure in high dimensional data-sets and provides a low-rank approximate matrix, given the original matrix. Since it is an efficient dimension reduction technique, we use it to extract the dominant components of the time-series vectors.

We use the matrix  $V$ , such that  $V = \{v_1, v_2, \dots, v_L\}$ , for reducing the cardinality of every  $v_i$  vector.  $v_i$  is a row of  $V$  and represents the sample data for one sensor. Since there are  $L$  number of sensors, the number of rows in matrix  $V$  are  $L$ . The number of columns is the cardinality of vector  $v_i$  and is represented as  $D$ . The primary objective of PCA is to find a  $d$  dimensional coordinate system, where  $d < D$ . The coordinates in the new subspace are orthogonal to each other. Also, they are linear combinations of the sample data and maintains maximum variability in data points. Mathematically, the problem is to find  $\hat{V}$  such that  $\hat{V} = V.P$ , where the dimensions of  $P$  are  $D \times d$ . For the given time-series representation, we determine the different principal components such that the first principal component is one in the direction provided by equation 5.1.

$$p_1 = \underset{\|p\|=1}{\operatorname{arg\,max}} \{ \sum_i (v_i \cdot p)^2 \} \quad (5.1)$$

The value of  $p$  that maximizes the right hand side of the equation 5.1 is the first principal component  $p_1$ . In equation 5.1, the term  $v_i \cdot p$  is the projection of vector  $v_i$  in the direction of unit vector  $p$ . The objective is to find the direction  $p_1$  that maximizes the variability of the vectors  $v_i$ . Proceeding similarly, if first  $d - 1$  principal components are determined, the  $d_{th}$  principal component is one of the residuals. The residual is obtained by removing the data mapped into the  $d - 1$  directions. Thus,  $d_{th}$  principal component  $p_d$  is obtained as given in equation 5.2.

$$p_d = \underset{\|p\|=1}{\operatorname{arg\,max}} \| (V - \sum_1^{d-1} V \cdot (p_i \cdot p_i^T)) \cdot p \|^2 \quad (5.2)$$

Once the principal orthogonal axes are determined, the transformed data in their direction is obtained using equation 5.3.

$$\hat{V} = V.P \quad (5.3)$$

It can be proved that the first  $d$  principal components of  $V$  are the top  $d$  eigenvalues in the covariance matrix of  $V$ . However, the proof is beyond the scope of this chapter and can be found in [70]. For the given data, using PCA, we have reduced the cardinality of vector  $v_i$  by 61%.

### Hierarchical clustering

The matrix  $\hat{V}$  is used to perform Hierarchical clustering and find correlated sensors. Hierarchical clustering is an unsupervised non-parametric clustering method that can be used to group objects according to similar patterns and put them in a cluster [71]. Every row of  $\hat{V}$  represents reduced data sample for a sensor. Since the number of sensors are  $L$ , the number of rows in  $\hat{V}$  is also equal to  $L$ . The clustering process for finding similar sensors is explained in algorithm 8.

---

#### Algorithm 8 Hierarchical clustering

---

**Input:** Reduced matrix-  $\hat{V}$

**Output:** Clusters-  $c$

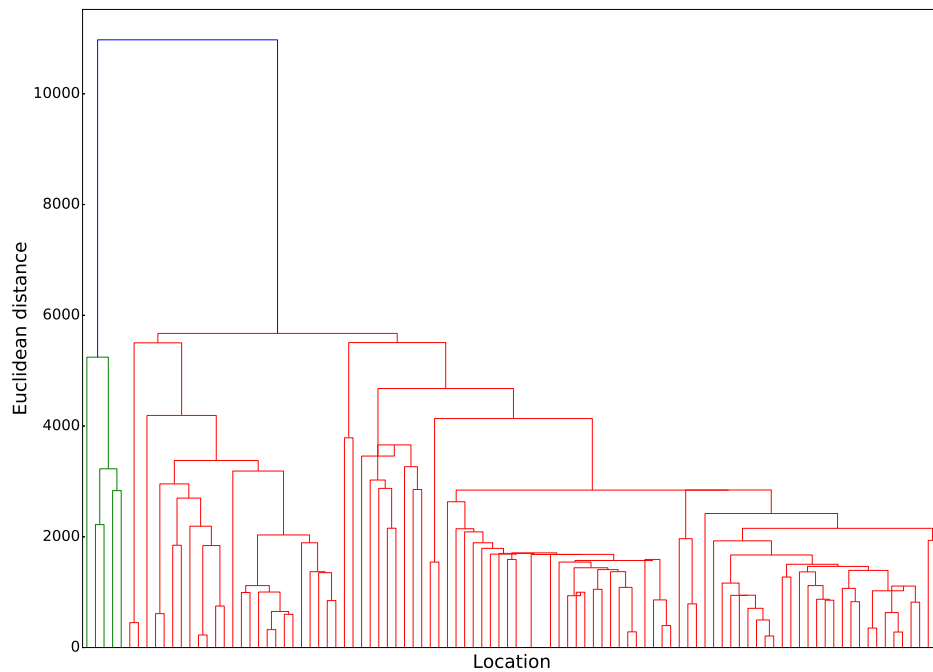
- 1: Assign each data sample to a single cluster.
  - 2: **repeat**
  - 3: Find the two most similar clusters.
  - 4: Merge clusters obtained in step 2.
  - 5: **until** A single cluster is obtained.
-

The clustering process begins by assigning every row of  $\hat{V}$  to a cluster. Thus, in the beginning, we have as many clusters as the number of sensors  $L$  such that every cluster has a single element. In successive steps, the two clusters which are similar to each other are merged to form a single cluster. Thus, as the algorithm progresses, the number of clusters decreases. The merging process continues until a single cluster is formed.

To find the similarity between two clusters, euclidean distance between cluster centroids is used. The two clusters having minimum euclidean distance between their centroids are merged in each step. For a cluster having single element, centroid is the value of multidimensional vector (row of  $\hat{V}$ ). If it has more than one element, its centroid is obtained by taking the average or center point. The average is obtained across each principal directions (columns of  $\hat{V}$ ). Thus, if any two row in  $\hat{V}$  are in a cluster then its centroid is obtained as the vector whose elements are the average of values in every column of the corresponding rows.

### **Dendrogram and number of clusters**

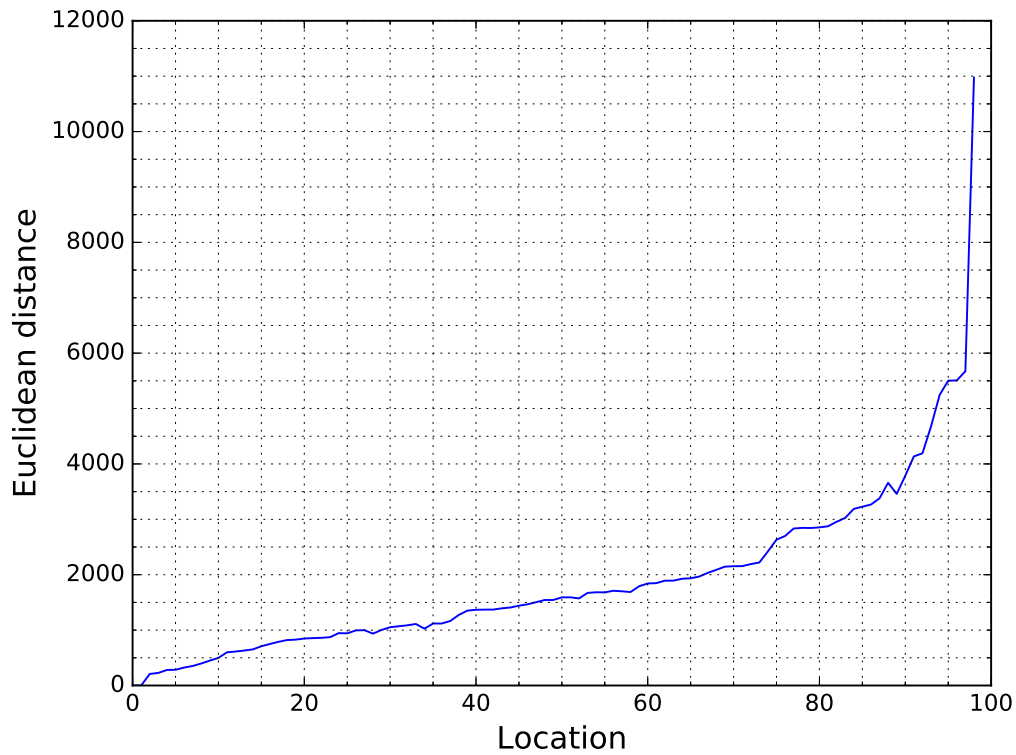
Figure 5.3 shows the dendrogram plot explaining cluster formation process for a set of vehicular traffic sensors. The lowest end of the plot (leaf) represent single sensors. The vertical lines that are combined to a single point show the two clusters being merged. The vertical axis represents the euclidean distance between the cluster centroids. The difference in height of the vertical lines, that are combined, is the distance between two clusters. If a horizontal line is drawn through the plot, then the number of clusters, before the merge at the particular



**Figure 5.3:** Dendrogram explaining cluster formation

level, is equal to half the number of times the horizontal line cuts the vertical lines.

The number of clusters are selected such that the distance between the clusters being merged is less than a certain threshold value. If the threshold distance is low (at the lower end of the dendrogram plot), large number of clusters are formed. In this case, the samples within clusters will be very similar to each other. On the other hand, if the threshold distance is significant, the number of clusters will be less, but the samples within the cluster will have high variability. Thus, the number of clusters depends upon the chosen threshold value. The variation in the euclidean distance between clusters being merged is plotted in Figure 5.4. To select the number of clusters, the point where there is an abrupt



**Figure 5.4:** Euclidean distance between clusters

change in the slope of the graph is chosen as the threshold distance. As we can see from the Figure 5.4, the slope of the curve has an edge at three points, where distance approaches 2500, 3500, and 5500.

One of these points can be chosen as the threshold distance to determine the number of clusters. As explained above, for lower threshold distance 2500, the number of clusters will be more with less number of samples in a cluster, but points within a cluster will be close to each other. For higher threshold value (5500) the cluster size will be significant with high variability of sample points within cluster elements.

### Cluster center

The cluster center can be obtained using various alternatives, like ward, average, single, complete, or centroid [71]. In this chapter, we chose centroid because it was found to provide the highest cophenetic correlation coefficient  $cc$ . A higher cophenetic correlation coefficient implies that the clusters better represent the original data-set [72].

The cophenetic correlation value is determined as the linear correlation between the original pairwise distance between the sample  $\hat{V}$  and the distances obtained from the dendrogram plot. Thus, if  $Y$  represents the vector containing pairwise distance of the elements in  $\hat{V}$  and  $Z$  represents the euclidean distances between clusters where two points are first merged, then the cophenetic correlation coefficient is obtained as in equation 5.4. Here,  $Y_{st}$  is the distance between  $s_{th}$  and  $t_{th}$  original observation in  $\hat{V}$  and  $Z_{st}$  is the distance between height of the dendrogram plot when  $s_{th}$  and  $t_{th}$  point are first combined in a cluster. Variables  $y$  and  $z$  are the average of  $Y$  and  $Z$  respectively.

$$cc = \frac{\sum_{s<t}(Y_{st} - y) \times (Z_{st} - z)}{\sqrt{\sum_{s<t}(Y_{st} - y)^2 \times \sum_{st}(Z_{st} - z)^2}} \quad (5.4)$$

### 5.3.2 Anomaly detection

The obtained clusters are used for detecting anomalies in time-series data. The anomaly detector is developed for every cluster of sensors. The given model is scalable because an anomaly detector is used for many sensors. We describe below the combination of LSTM neural network and M-estimator for detecting anomalies in time-series data. The following analysis is for a cluster of sensors.

The trained LSTM neural network can be used for any sensor in the particular cluster. Thus, there are as many models as the number of clusters obtained during clustering analysis.

### Segmentation

As shown in Figure 5.2, for detecting anomalies, we first segment time-series. The time-series vectors are split into length of equal intervals  $\Gamma$ . The segments are used for detecting anomalies. In a segment, we assume that 50% of the samples near median are true values. Thus, we verify for anomalies in lower and upper ends of the segment. For example, if values in a time-series segment are sorted in increasing order of magnitude, then half of the values in the middle portion are considered as true values. The other half (25% at lower and 25% at upper ends) values may contain anomalies. A particular segment starting at time-slot  $k$  is represented using the variable  $s_k = \{n_{i,j,k}, n_{i,j,k+1}, \dots, n_{i,j,k+\Gamma}\}$ . For convenience, we omit the subscript  $i$  and  $j$  in  $s_k$ . The segments of time-series are fed to the M-estimator for statistical analysis. Time-series segments are also used by the LSTM neural network for estimating deviations in it. The statistical analysis and estimated deviation are combined to find the anomalies.

### M-Estimator

The statistical analysis is performed on a time-series segment. We determine the M-estimator value for a segment. M-estimator is the robust statistical measure of a sample data-set [63]. The critical properties of the M-estimator is that it is resilient in the presence of anomalies and does not depend on samples having

normal distribution. Thus, even if the anomalies are present, the M-estimator value is expected to be constant. In contrast, statistical properties like mean is not robust against anomalies because the true sample mean changes even in the presence of a single anomaly. For a segment, the M-estimator is obtained as the solution of the equation 5.5.

$$\sum_{k=k}^{k+\Gamma} \xi\left(\frac{n_{i,j,k} - \mu}{\sigma(s_k)}\right) = 0 \quad (5.5)$$

The denominator  $\sigma(s_k)$  is a function on  $s_k$  and gives initial estimate of the solution. The solution  $\mu$  of the equation is the robust M-estimator  $\mu_k$  for segment  $s_k$ .  $\xi$  is a real valued Huber function  $\xi(a) = a \times \min(1, \frac{b}{|a|})$ , where  $b$  is a constant.

The M-estimator is used to find the deviation  $d_k$  for a time-series segment. For segment  $s_k$ , the deviation from the robust M-estimator  $\mu_k$  is found using equation 5.6.

$$d_k = \max(|n_{i,j,k} - \mu_k|, |n_{i,j,k+1} - \mu_k|, \dots, |n_{i,j,k+\Gamma} - \mu_k|) \quad (5.6)$$

We use quantities  $s_k$ ,  $\mu_k$ , and  $d_k$  for training the LSTM neural network. That is based on the values  $s_k$ ,  $\mu_k$ , and  $d_k$ , a neural network model is developed that assists in determining anomalies in time-series segments. The detailed analysis of LSTM neural network and subsequently the anomaly detector is provided in the following sections.

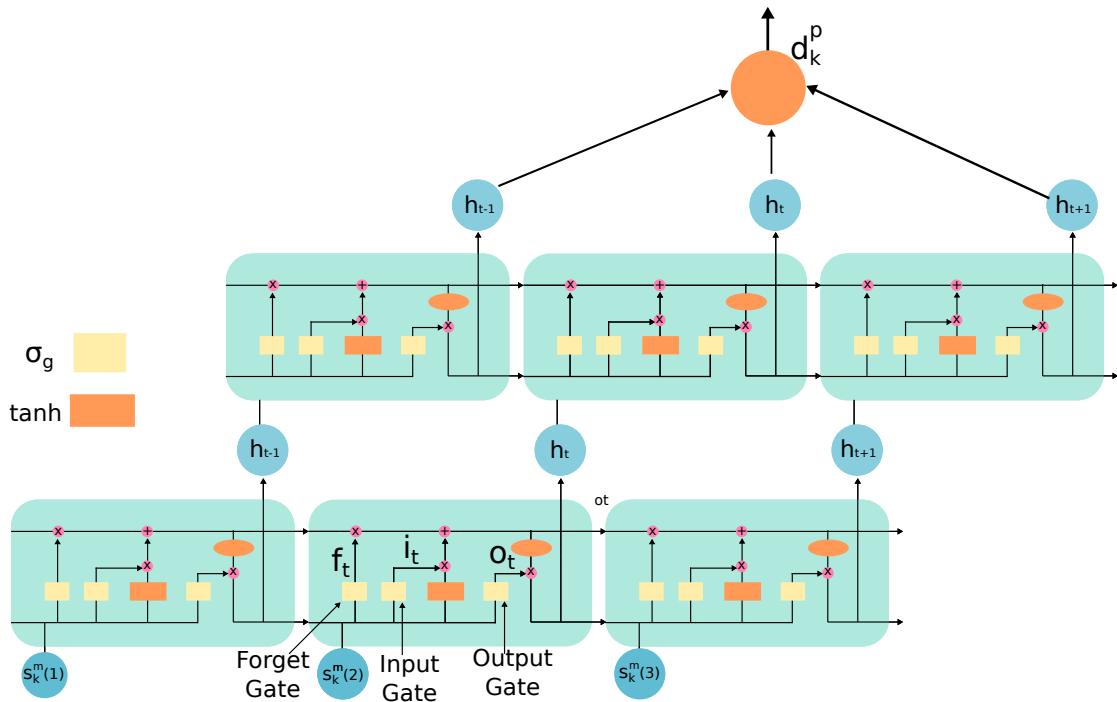


Figure 5.5: LSTM neural network

### LSTM neural network

We describe the LSTM neural network that is used for predicting deviation from robust M-estimator. Equation 5.6 shows the actual deviation from M-estimator in a time-series segment. However, in the presence of anomalies, actual deviation in the segment is different as calculated using equation 5.6 because the sample points are not true values but anomalies. Thus, the proposed method predicts the deviation from the M-estimator  $d_k^p$ , rather calculating it using equation 5.6.

From segment  $s_k$ , we find the vector  $s_k^m$ , containing 50% of  $s_k$  elements near the median. The elements of the vector  $s_k^m$  are represented as  $s_k^m(1)$ ,  $s_k^m(2)$ , ..., and so on. Since the cardinality of  $s_k$  is  $\Gamma$ , the vector  $s_k^m$  has the cardinality  $\Gamma/2$ . The prediction problem is, given the input  $s_k^m$ , design a LSTM neural network to

accurately estimate  $d_k^p$ .

The LSTM neural network architecture is shown in Figure 5.5. The LSTM neural network has cells that process the inputs fed to it. The cells are sequentially joined to each other to form a chain as shown in the figure. Each such chain is called a layer of the LSTM neural network. The multiple layers are stacked above each other such that output of a layer is the input for layer above it. The cell has a memory called as its state. The LSTM cell processes the data based on an external input, output of the previous cell, and state of the previous cell [73].

The variable  $t$  is used to represent a cell. Thus, the cells adjacent to  $t$ , in same layer, are  $t - 1$  and  $t + 1$ . Every cell has forget, input, and output gates. Forget gate determines the information from the previous cell to be discarded in the current cell. The input gate determines the information from the current input to be saved. The output gate determines the amount of information to be passed to the next stage. Each gate receives external input (an element of vector  $s_k^m$ ) and output from the previous cell. If  $r$  is a number between 1 and  $\Gamma/2$ , gates process the input as given in equation 5.7, to get the outputs,  $f_t$ ,  $i_t$ , and  $o_t$ , of forget, input, and output gates respectively, for cell  $t$ .

$$f_t = \sigma_g \times (W_f \cdot s_i^m(r) + U_f \cdot h_{t-1} + b_f) \quad (5.7a)$$

$$i_t = \sigma_g \times (W_i \cdot s_i^m(r) + U_i \cdot h_{t-1} + b_i) \quad (5.7b)$$

$$o_t = \sigma_g \times (W_o \cdot s_i^m(r) + U_o \cdot h_{t-1} + b_o) \quad (5.7c)$$

Here,  $W_f$ ,  $W_i$ , and  $W_o$  are the weight matrices between external input and the

three gates.  $U_f$ ,  $U_i$ , and  $U_o$  are the weights between output from cell  $t - 1$  and the gates.  $b_f$ ,  $b_i$ , and  $b_o$  are the bias vectors,  $h_{t-1}$  is the output from the cell  $t - 1$ , and  $\sigma_g$  is the sigmoid activation function.

The memory of LSTM cell (cell state) is evaluated using equation 5.8, where  $W_c$  and  $U_c$  are the weight matrices and  $b_c$  is the bias vector. The cell state depends upon current external input  $s_i^m(r)$ , output of  $t - 1$  cell  $h_{t-1}$ , and state of the  $t - 1$  cell  $C_{t-1}$ . As depicted in equation 5.8b, the cell  $t$  forgets some information from state of the cell  $t - 1$  (term  $f_t * C_{t-1}$ ) and keeps some information from current input and output of the cell  $t - 1$  (term  $i_t * \tilde{C}_t$ ), to get the updated state  $C_t$ .

$$\tilde{C}_t = \tanh(W_c \cdot s_i^m(r) + U_c \cdot h_{t-1} + b_c) \quad (5.8a)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (5.8b)$$

The output of a cell depends upon the output gate ( $o_t$ ) and cell state  $C_t$ , is given in equation 5.9.

$$h_t = o_t * \tanh(C_t) \quad (5.9)$$

The above analysis is for a single cell in a layer (input layer). The outputs of cells in a layer are the external inputs to the cells in layer stacked above it.

### LSTM neural network design

In the given prediction problem, the vector  $s_k^m$  is the input and  $d_k^p$  is the output to the LSTM neural network. The cells in the input layer of LSTM neural network are added based on the cardinality of the vector  $s_k^m$ . Since, for an input vector  $s_k^m$ , there is a single valued output  $d_k^p$ , a neuron is added at the output. The outputs from the uppermost LSTM layer is fed as input to the feed-forward neuron. The output of the neuron is the required predicted deviation  $d_k^p$ . Due to data normalization, the output value ranges between 0 to 1. Therefore, the activation function of the output neuron is chosen as  $\tanh$ , as it is a continuous function and its range include required limits between 0 and 1. Between input layer and output neuron, three LSTM layers are added so that the network accurately predicts the  $d_k^p$  value. The activation function for LSTM cells is also chosen as  $\tanh$ .

Initially, the LSTM neural network operates in training phase to set its weights and biases to optimum values. Once the network is trained, it is used for predicting deviation. The training phase is an offline process that uses the known input vectors  $s_k^m$  and output values  $d_k$ . The quantities  $s_k^m$  and  $d_k$  are collected for one week data. During the training phase, the weights and biases associated with the network are adjusted so that the difference between predicted deviation  $d_k^p$  and the actual deviation  $d_k$  is minimum. Thus, the input  $s_k^m$  is fed to the network to obtain the output  $d_k^p$ . Then, based on the mean squared error between predicted  $d_k^p$  and actual  $d_k$ , the weights of the neural network are adjusted. For neural network training, we tested different optimization methods,

like Adam, Adagrad, Adadelata, Gradient descent, and RMSprop [74]. Our analysis resulted in using Adam optimization since it converges fast as compared to other methods. For training purpose, all the input vectors are collected in a matrix  $X$  such that a row of  $X$  is one input vector  $s_k^m$ . The network is trained for 10000 iteration. To avoid the over-fitting of the network, we selected a dropout of 20% [75]. Thus, in every iteration, 20% of the weights in each layer are ignored and not updated. Once the weights are optimally updated such that the error between expected and known output is very low, they are saved and represent the proposed trained model for a cluster of sensors.

The trained model is used in prediction phase, when the anomaly detector operates. During prediction phase, a vector  $s_k^m$  is fed as input to the neural network. The output provides the estimated value of the deviation from the M-estimator  $d_k^p$ . The following section describes how the M-estimator and predicted deviation  $d_k^p$  are combined for detecting anomalies.

### **Anomaly detector**

The process of finding anomalies using M-estimator and LSTM neural network is described in the algorithm 9. The algorithm takes time-series segment  $s_k$  as input. It uses a hyper-parameter  $\alpha$  to determine the expected range of values within which a segment lies. The algorithm employs trained LSTM neural network models. There are as many LSTM neural network models as there are the number of clusters. The algorithm provides the number of positive  $\eta_p$ , negative  $\eta_n$ , and the total number of anomalies  $\eta_t$ . Positive anomalies  $\eta_p$  have value

---

**Algorithm 9** Anomaly detector algorithm
 

---

**Input:** Time-series segment-  $s_k$   
 Hyper-parameter-  $\alpha_k$   
 Trained LSTM neural network models

**Output:** Number of anomalies-  $\eta_p, \eta_n, \eta_t$

- 1:  $\mu_k = \text{Find\_Mestimator}(s_k)$
- 2:  $d_k^p = \text{Find\_Deviation}(s_k^m)$
- 3:  $\eta_p, \eta_n, \eta_t = 0$
- 4: **for**  $r \leftarrow 1$  **to**  $\Gamma$  **do**
- 5:   **if**  $s_k(r) < \mu_k - \alpha \times d_k^p$  **then**
- 6:      $\eta_n = \eta_n + 1, \eta = \eta + 1$
- 7:   **end if**
- 8:   **if**  $s_k(r) > \mu_k + \alpha \times d_k^p$  **then**
- 9:      $\eta_p = \eta_p + 1, \eta = \eta + 1$
- 10:   **end if**
- 11: **end for**

---

higher than the expected maximum value of a segment. Negative anomalies  $\eta_n$  have value less than the expected minimum value.

In step 1, algorithm determines the M-estimator  $\mu_k$  for the time-series segment  $s_k$ . Here, it should be noted that  $s_k$  may contain anomalies. Thus, using M-estimator is useful as it is robust against the presence of anomalies. Then, step 2 finds the deviation  $d_k^p$  using the LSTM neural network model. For determining deviation, out of different LSTM neural network models, one for the cluster in which segment  $s_k$  belongs is chosen. The number of anomalies is set to zero in step 3.

After that, steps 4-11 iterates for every element of  $s_k$ . In step 5, the algorithm checks whether an element is less than the expected minimum value. The expected minimum value is set to  $\mu_k - \alpha \times d_k^p$ . If the point is less than the expected minimum value, it is considered as a negative anomaly. Correspondingly, the negative and total number of anomalies are incremented in step 6. Similarly,

step 8 verifies if a point is more than the expected maximum value  $\mu_k + \alpha \times d_k^p$ . If the condition is satisfied, the values of the positive and the total number of anomalies are incremented in step 9. Tunable hyper-parameter  $\alpha$  is adjusted to get the different range of expected minimum and maximum values.

## 5.4 Results

This section describes in detail the evaluation results of the proposed method. We test the performance of the anomaly detector using two different data-sets; 1)vehicular traffic count and 2) environmental pollutant Carbon Mono-oxide (CO) level.

### 5.4.1 Materials and methods

The vehicular traffic data used for testing the given method is collected from the Performance Measurement System (PeMs). PeMs provides freeway traffic at California Highways [56]. The pollutant level is obtained from the California Air Resource Board [76] website. We implemented the simulation atmosphere in a Python-based framework. In our frame-work, different python tools and libraries are employed. The clustering module is implemented using Python-Scikit tool [77]. The LSTM neural network is developed using Keras framework running on top of the TensorFlow environment [74]. The simulation is executed in Ubuntu 16.04 operating system running on Intel Core-i5 processor at 2.60 GHz and containing four cores.

We simulate anomalies using the mean of the standard deviations in the training data segments. The average of the standard deviations in all  $s_k$  vectors is obtained for training sample. The anomalies proportional to the obtained average are injected in the test data.

### 5.4.2 Evaluation metrics

The efficacy of the proposed method is evaluated using precision  $P$ , recall  $R$ , and F-measure  $F$ . The precision is the ratio of true positives  $tp$  and the sum of true positives  $tp$  and false positives  $fp$  (equation 5.10).

$$P = \frac{tp}{tp + fp} \quad (5.10)$$

The recall  $R$  is the ratio of true positives  $tp$  and the sum of true positives  $tp$  and false negatives  $fn$  and is calculated using equation 5.11.

$$R = \frac{tp}{tp + fn} \quad (5.11)$$

The F-measure is obtained using Precision and Recall values as given by equation 5.12.

$$F = 2 \times \frac{P \times R}{P + R} \quad (5.12)$$

To find F-measure, we put equal weight to precision and recall values. The generalized form of F-measure is given in equation 5.13, where the parameter  $\beta \geq 0$ . The F-measure is said to be recall-oriented, if  $\beta < 1$ , and precision oriented, if  $\beta > 1$  [40].

$$F_g = (1 + \beta^2) \times \frac{P \times R}{\beta^2 \times P + R} \quad (5.13)$$

### 5.4.3 Performance evaluation

We evaluate performance as the magnitude of anomalies increases for both the data-sets. In table 5.1 and 5.2, the first column represents the strength of the injected anomalies. For example, *mean\_std\_5* is the actual value of data plus 0.5 times the mean of recorded standard deviation in the training data.

**Table 5.1:** Analysis for positive anomalies

Strength	Vehicular traffic			Air pollutant		
	<i>P</i>	<i>R</i>	<i>F</i>	<i>P</i>	<i>R</i>	<i>F</i>
<i>mean_std_5</i>	0.89	0.55	0.68	0.95	0.7	0.81
<i>mean_std_2</i>	0.91	0.72	0.8	0.96	0.86	0.91
<i>mean_std_3</i>	0.93	0.89	0.91	0.97	0.99	0.98
<i>mean_std_4</i>	0.93	0.96	0.94	0.97	1.0	0.98
<i>mean_std_5</i>	0.93	0.99	0.96	0.97	1.0	0.98
<i>mean_std_6</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_7</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_8</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_9</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_10</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_11</i>	0.93	1	0.96	0.97	1.0	0.98
<i>mean_std_12</i>	0.93	1	0.96	0.97	1.0	0.98

Table 5.1 depicts the performance measure in the presence of positive anomalies. From the table, we can observe that as the magnitude increases, the performance of the anomaly detector improves for both vehicular traffic and pollution level data. It can be observed that the recall values are close to 1 in most of the rows. It led to conclude that the number of false negatives are negligible. The precision values are also high for different magnitudes. The maximum

precision value for vehicular traffic is 93% while for air pollutant is 97%. The F-measure approaches 96% for vehicular traffic and 98% for air pollutant data. Thus, the metrics show that the given method accurately detects the anomalies for different data-sets and has a negligible number of false negatives.

**Table 5.2:** Analysis for negative anomalies

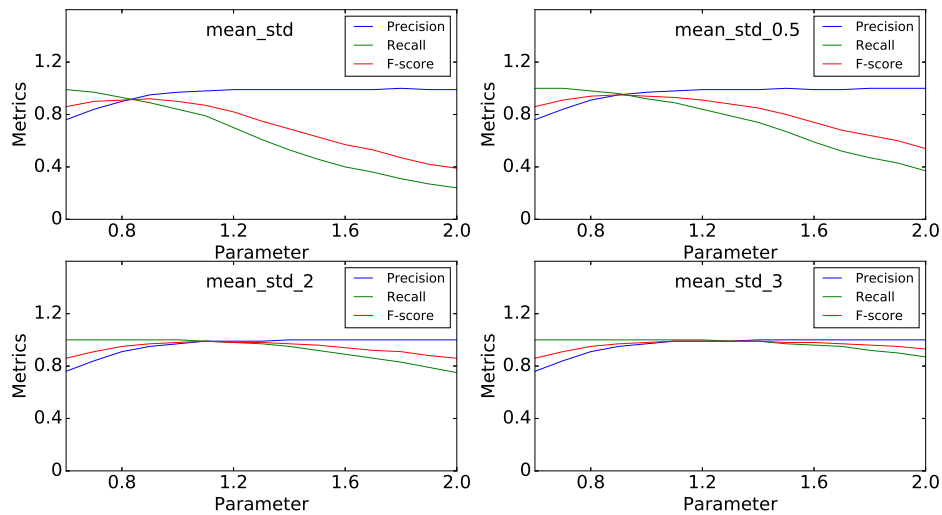
Strength	Vehicular traffic			Air pollutant		
	<i>P</i>	<i>R</i>	<i>F</i>	<i>P</i>	<i>R</i>	<i>F</i>
<i>mean_std_5</i>	0.95	0.5	0.66	0.86	0.22	0.35
<i>mean_std_2</i>	0.97	0.72	0.83	0.97	0.29	0.45
<i>mean_std_3</i>	0.99	0.9	0.94	0.99	0.76	0.86
<i>mean_std_4</i>	0.98	0.95	0.96	0.99	1.0	0.99
<i>mean_std_5</i>	0.97	0.99	0.98	0.98	1.0	0.99
<i>mean_std_6</i>	0.98	0.98	0.98	0.97	1.0	0.98
<i>mean_std_7</i>	0.99	0.98	0.98	0.97	1.0	0.98
<i>mean_std_8</i>	0.98	0.99	0.98	0.97	1.0	0.98
<i>mean_std_9</i>	0.99	0.99	0.99	0.96	1.0	0.98
<i>mean_std_10</i>	0.99	1	0.99	0.96	1.0	0.98
<i>mean_std_11</i>	0.98	1	0.99	0.97	1.0	0.98
<i>mean_std_12</i>	0.98	1	0.99	0.97	1.0	0.98

Table 5.2 presents the performance analysis for negative anomalies. The algorithm effectively captures the negative anomalies too in the sample. The precision values approach 95% for vehicular traffic count and 99% for air pollutant data. For high injected magnitude, the recall values become 100% for both the data-sets. The corresponding F-measure also approaches to 98% for traffic and pollutant data. Thus, for negative anomalies also the performance measure in terms of precision, recall, and F-measure values are high.

Further, we observe the performance when both positive and negative anomalies are present. For this case, the percentage of anomalies is varied, and the metrics values are given in table 5.3. We observe that the performance in terms

**Table 5.3:** Analysis as percentage of anomalies vary

% anomalies	Vehicular traffic			Air pollutant		
	$P$	$R$	$F$	$P$	$R$	$F$
8.3%	0.88	0.97	0.92	0.85	1	0.92
16.7%	0.98	0.96	0.97	0.93	0.97	0.95
25%	0.99	0.96	0.97	0.96	0.95	0.95
33.3%	1	0.96	0.98	1	0.97	0.98
41.7%	1	0.95	0.97	0.99	0.95	0.97
50%	1	0.96	0.98	1	0.98	0.99

**Figure 5.6:** Comparison of performance metrics as  $\alpha$  varies

of the three metrics improve as the number of anomalies increases. For example, the precision approaches to around 1 for both data-sets.

Thus, the table 5.1-5.3 concludes that the given method accurately detects anomalies, has less number of false positives, and as the percentage of anomalies increases accuracy increases.

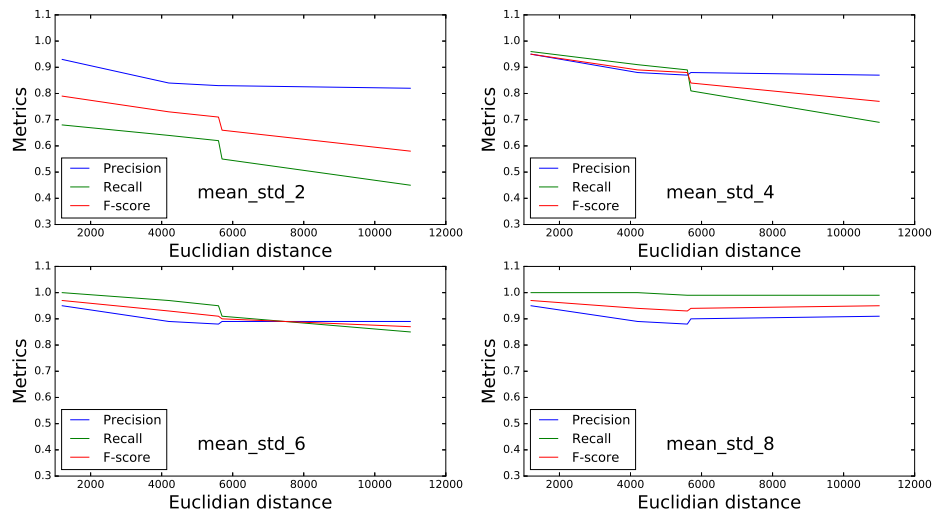
#### 5.4.4 Performance analysis as parameter $\alpha$ is tuned

Figure 5.6 shows the variation in performance metrics as the parameter  $\alpha$  in algorithm 9 is tuned. We plot the graphs for 4 different strength of injected anomalies. The figure portrays that for a low value of  $\alpha$ , recall is better than the precision. However, as the value of  $\alpha$  increases, the precision becomes higher than the recall. Also, as the magnitude of anomalies increases, the plots become parallel to the horizontal axis, and the performance of the anomaly detector improves. Thus, the value of  $\alpha$ , where the three metrics are equal to each other, increases on increasing the injection magnitude.

The figure infers that for an application requiring low false negatives, the  $\alpha$  should be set to a low value. On the other hand, applications requiring low false positives,  $\alpha$  can be set to a high value. Thus, the parameter  $\alpha$  can be tuned, satisfying the need of an application. Further, for effectively detecting anomalies of low strength,  $\alpha$  can be tuned to a small value. However, for detecting anomalies having high strength,  $\alpha$  can be set to a high value.

#### 5.4.5 Performance analysis as cluster size varies

Figure 5.7 shows the performance variation as the threshold euclidean distance increases for different magnitudes of anomaly strength. As we presented earlier in section 5.1.3, on increasing the euclidean distance, the number of sensors in a cluster increases and thus the system scales up. Figure interprets that the accuracy decreases as the threshold euclidean distance (cluster size) increases. For small cluster size, the accuracy is greater as the model can better represent less

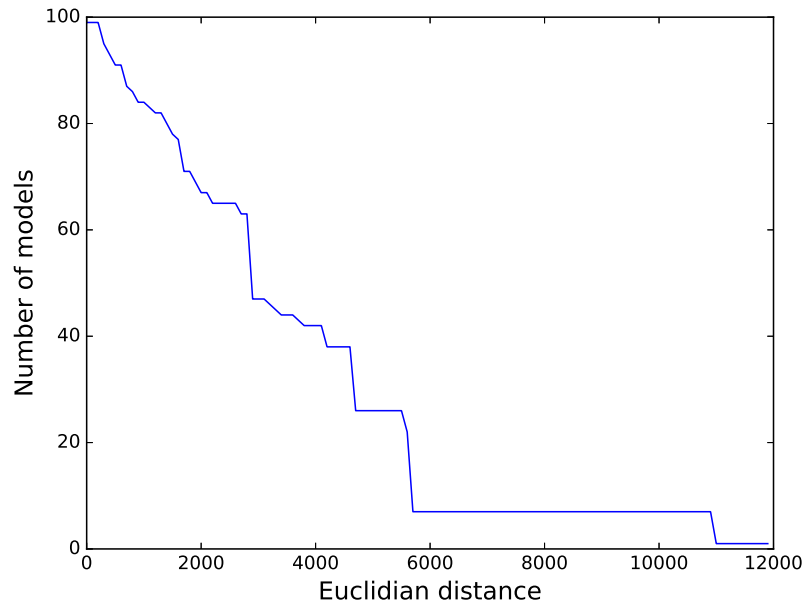


**Figure 5.7:** Performance metrics as euclidian distance varies

number of sensors within a cluster. For large cluster size, the sensor diversity within a cluster increases, and due to this, the accuracy decreases. Furthermore, for high strength of injected anomalies, there is a smaller decrease in accuracy as euclidean distance increases. This is because those anomalies tend to be easily detected.

Figure 5.8 shows the number of models required to be developed as euclidean distance increases. For low euclidean distance, the number of clusters is large, and thus the number of required models are large. Therefore, computational effort is high. For high euclidean distance, the number of clusters is small, and thus the less number of models are required, and the computational effort is low.

Thus, Figures 5.7 and 5.8 show that as the threshold euclidean distance or cluster size increases, the accuracy decreases, and the number of models to be developed becomes low, requiring less computational efforts. Therefore, there is

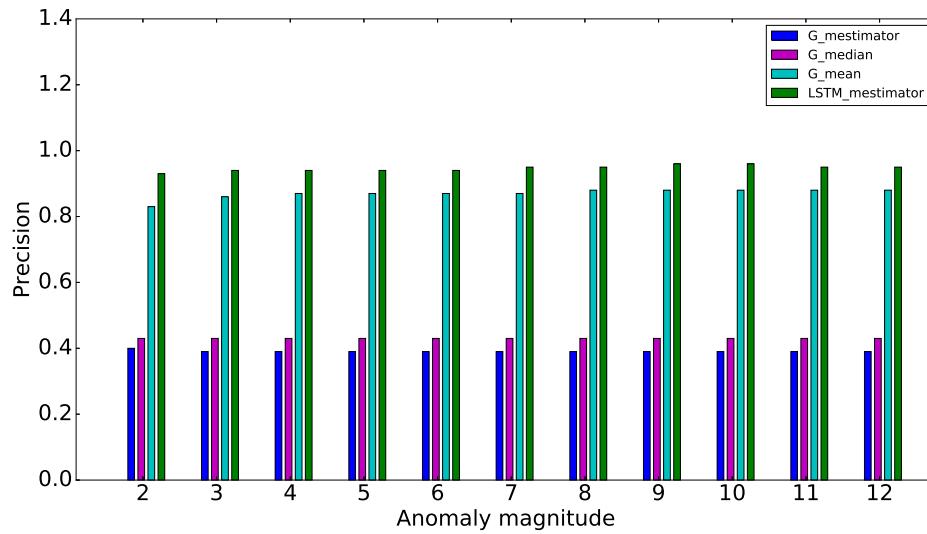


**Figure 5.8:** Number of models as euclidian distance is varied

a trade-off between performance and computational efforts.

#### 5.4.6 Comparison with other approaches

We compare the proposed method with statistical-based techniques. The Generalized Extreme Studentized Deviate (GESD) test is used as a benchmark for the performance comparison [62]. Traditionally, GESD uses the mean and standard deviation of the sample as a statistical measure. The median and median absolute deviation (MAD) have also been used in GESD test [78]. We evaluate the proposed method against the above methods such that using mean (and standard deviation) and median (and MAD) in GESD. Furthermore, we also test the performance using M-estimator and deviation from it in GESD. The



**Figure 5.9:** Precision comparison for traffic data

comparison for the different metrics is provided in Figures 5.9-5.11 for vehicular data and in Figure 5.12-5.14 for pollutant data.

Figures 5.9 and 5.12 show that the proposed LSTM neural network and M-estimator based method outperforms the GESD-based techniques irrespective of the statistical measure chosen. For GESD, using mean and standard deviation as statistical properties provides the highest precision values, that is lower than the proposed method. It should be noted that, by tuning  $\alpha$ , the precision can be further improved, but at the cost of the recall. The GESD-based methods do not have such tunable parameter, and thus, the obtained precision values are fixed.

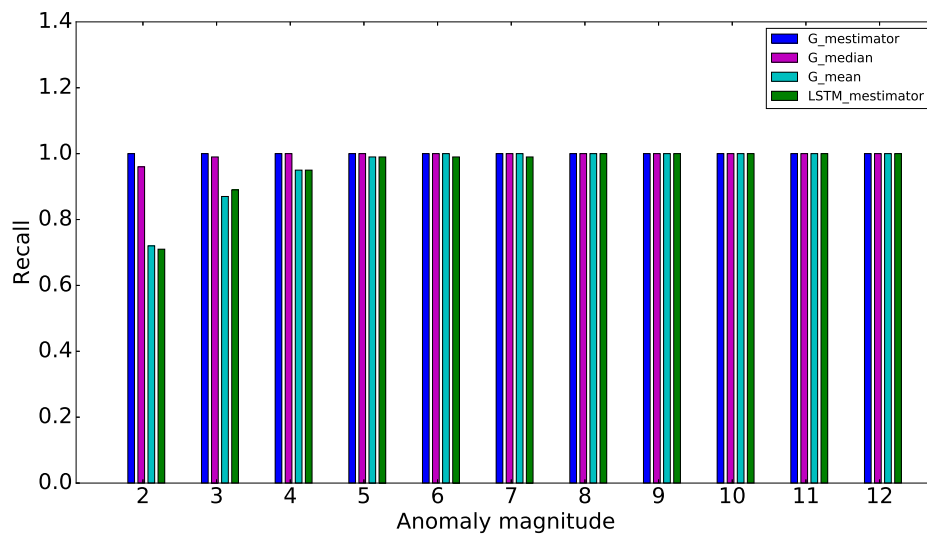


Figure 5.10: Recall comparison for traffic data

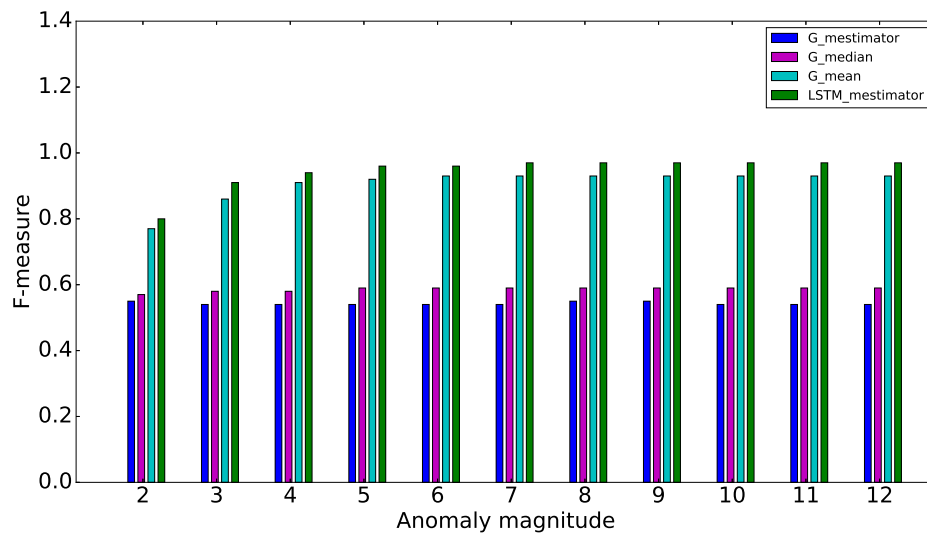
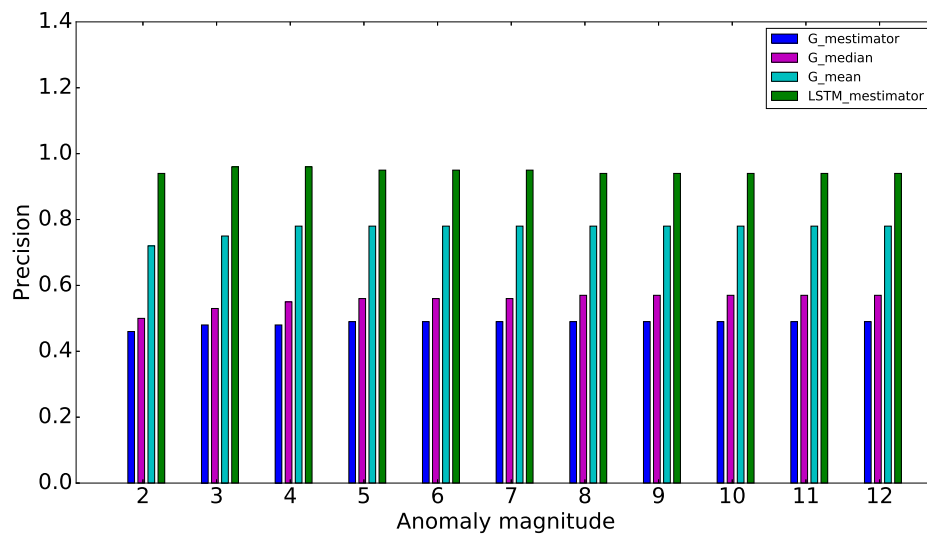
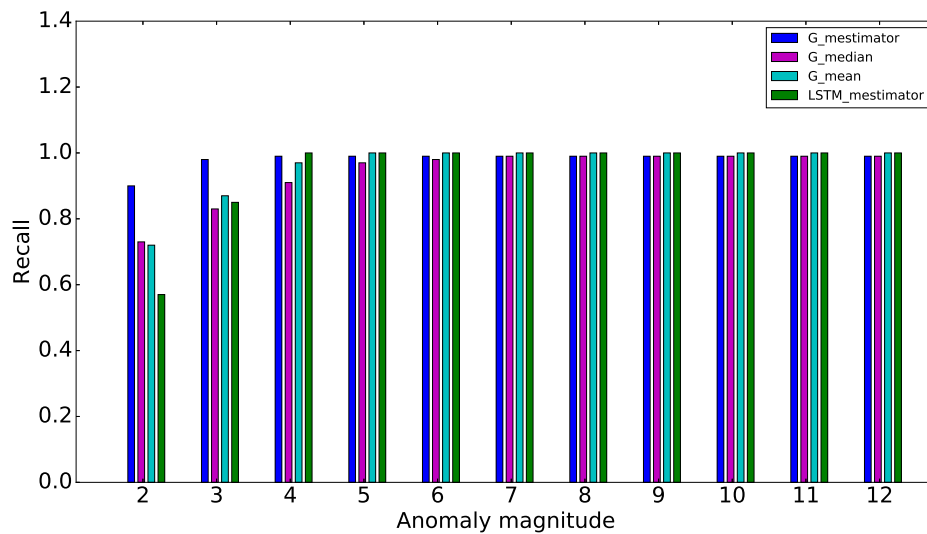


Figure 5.11: F-measure comparison for traffic data



**Figure 5.12:** Precision comparison for pollutant data

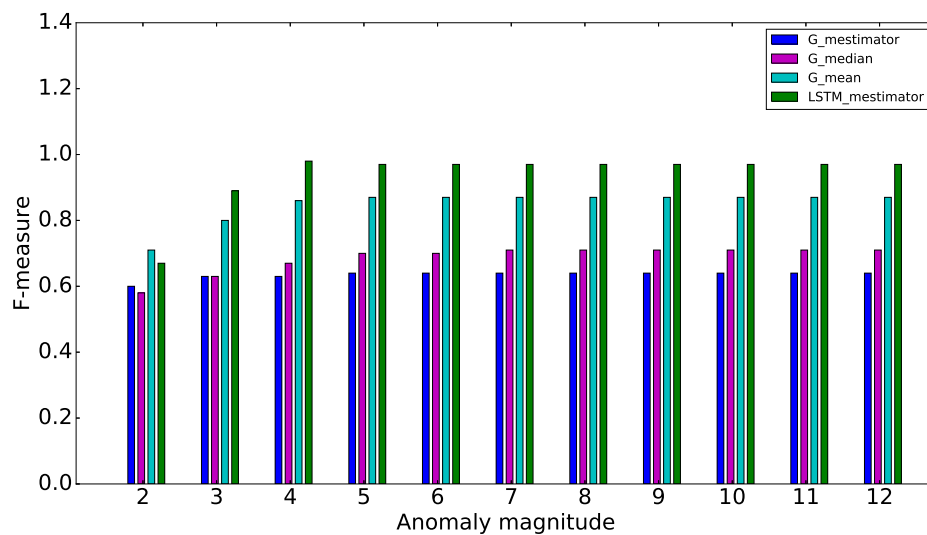


**Figure 5.13:** Recall comparison for pollutant data

Figures 5.10 and 5.13 compares the recall values for both the data-sets. As shown in the figure, for the low strength of the anomalies, using GESD with M-estimator and deviation from it provides the highest recall. For high strength,

the recall is comparable for different methods. However, as shown in figures 5.9, 5.11, 5.12, and 5.14, precision and F-measures are lowest when using GESD with M-estimators. Thus, although the GESD with M-estimator and deviation from it provides comparable recall, it is not a viable option since the performance of the other two metrics deteriorates significantly.

Figures 5.11 and 5.14 compares the F-measure of the GESD-based techniques and the proposed method. The figures show that regarding F-measure, the given method outperforms different GESD-based techniques. Therefore, as presented, the proposed anomaly detector has better performance as compared to the statistical-based techniques in terms of precision, recall, and F-measures. Also, the proposed method is scalable and can be tailored according to the application requirements. The flexibility is not available in the statistical-based techniques, and thus the given architecture is better than contemporary methods.



**Figure 5.14:** F-measure comparison for pollutant data

### 5.4.7 Computation complexity

In the given anomaly detector, first, we train the LSTM neural network, and then it is used for the operation. While training phase is computation-intensive and takes a longer time, it is performed only once. The average time the anomaly detector takes, during the operating phase, to perform a test on a segment is 0.18 seconds. Thus, operating phase still takes a pretty low execution time. Furthermore, we tested the performance using limited computing resources. The deployment of the algorithm in the cloud is expected to further reduce execution time because of the presence of a large pool of computing resources.

## 5.5 Observations and remarks

This chapter has highlighted the importance of a scalable anomaly detector for smart city-based automated applications. We presented an anomaly detector that not only accurately determines the anomalies but is also scalable. The performance of the proposed architecture is tested on two different data-sets. The obtained results show that our method has high precision, recall, and F-measure values. The given anomaly detector can be tuned according to the application's performance requirement, and either precision or recall values can be further improved. The performance is also tested as the number of sensors in a cluster increases. There is a trade-off between performance and computational efforts as the cluster size increases. The proposed method outperforms the contemporary methods that use statistical properties of the sample.

## Chapter 6

# Performance analysis of a traffic prediction application in the presence of malicious anomalies

The anomalies in time-series data can occur due to either variation in surrounding environmental patterns or the data falsification attacks. Data falsification attacks are the kind of vulnerabilities where some measured values are intelligently altered so that the associated system behaves abnormally [39]. Such attack can severely hurt the performance of the prediction method and affect the proactive intelligent decision making by smart city based applications.

Anomalies due to data falsification attack can either be limited to few sensors or an orchestrated attack on several sensors can be launched. Since the sensor data may be spatially and temporally correlated, an attacker can intelligently target correlated group of sensor nodes. The attack can be launched by nefarious adversaries, like business rivals or organized criminals [79], seeking long-term benefits. These adversaries are equipped with the resources that can bypass the

cryptographic security mechanism [39]. For example, an adversary can alter predicted traffic count near some electric vehicle charging station by changing historical traffic count values in the correlated group of sensors. This can lead to mismatch between energy supplied from the smart grid and actual energy demand at a charging station. Also, the estimation of the wait time for an electric vehicle at a specific charging station may go wrong resulting in charging service provider giving incorrect information to the customers. Albeit, the customer's trust in that service provider can be affected resulting in revenue loss for that company.

The problem of determining anomalous data points due to data falsification attack in traffic count is complicated because of several reasons. The amount of data from the sensors is of enormous quantity such that manually analyzing the data is infeasible. Additionally, the possible types of attacks that an attacker can introduce is generally not known in advance. An attacker can intelligently attack a group of sensors without affecting the statistical properties, like mean, mode, and periodicity, of the data points to a considerable extent. Moreover, the sensor data like traffic count is affected by external factors like climatic conditions, events, construction, or emergency situations. These conditions may cause the sensor data to behave abnormally. Therefore, distinguishing whether the anomaly is due to data falsification attack or external factors makes detection of such vulnerabilities complex.

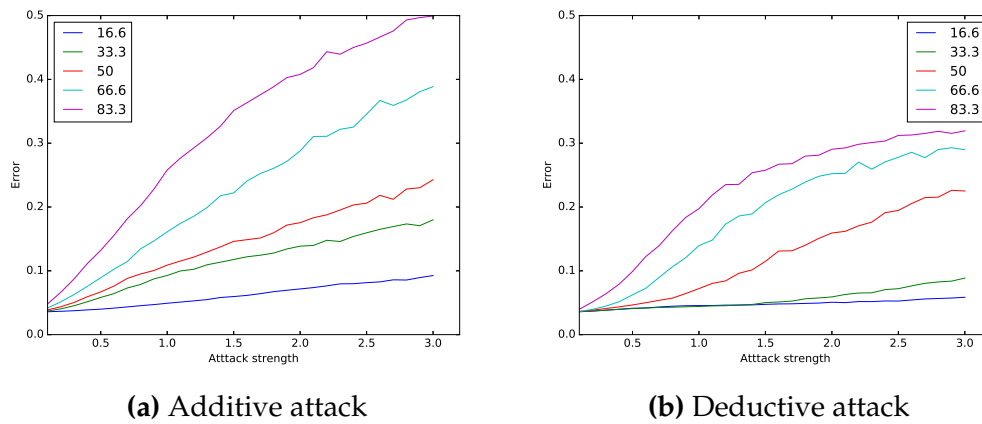
This chapter analyzes the problem of malicious anomalies due to data falsification attack for the traffic prediction system. Since deep learning has been found as an efficient technique to predict both short and long-term traffic accurately, [80], [81], [82], the chapter uses a multilayer neural network as our traffic

prediction algorithm. The chapter investigates the impact of data falsification attack on the performance of the deep neural network algorithm. The chapter also discuss the amount false injection that an attacker should introduce to cause any noticeable error in prediction performance.

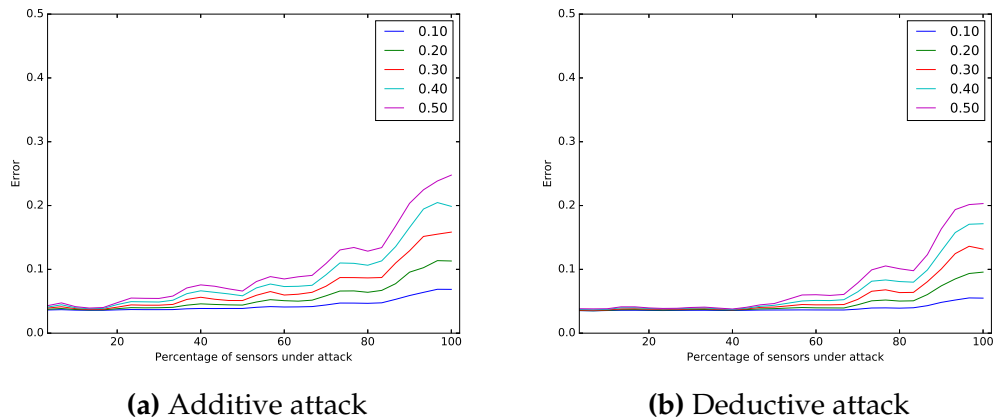
## 6.1 System model for performance analysis

System model consists of en route vehicles in a specific part of the city. The road network is divided into zones based on their spatial location. Loop detectors are installed along the road to measure traffic volume and speed of the vehicles. The loop detectors continuously measure the data and the aggregate data over a periodic time interval  $\tau$  is sent to the centralized edge server. Edge server brings the cloud computing capabilities near the user and is used for analysis and storage of the data [83]. The proposed edge server has analytical modules like advance traffic prediction that predicts the future traffic counts and thus supports applications like charging station allocation for PEV charging [16], efficient traffic routing, or traffic speed management [84]. The processed information is transmitted either to the vehicle's dashboard or the user's mobile phone. The filtered information from the edge server is sent to the remote cloud for making long-term analytical decisions like city planning and management.

An adversary may target a group of loop detectors over a time interval  $\Gamma_a$  such that data sensed from the loop detectors is corrupted, and the incorrect information is sent to the edge server. Because of that, the analytics hosted at the edge platform performs adversely affecting the performance of different services. Additionally, the small anomalies added at the sensors get cumulated at



**Figure 6.1:** Effect of attack as magnitude of attack is varied



**Figure 6.2:** Effect of attack as percentage of values in input vector are varied

the cloud and may also impact the long-term decision making at the cloud. To address this issue, the chapter proposes an anomaly detector that is placed in the edge server. The incoming data from loop detectors is first analyzed by the anomaly detector to find if it is a clean data. After analysis, filtered data is sent to the analytic applications for assisting different services.

## 6.2 Prediction performance

The chapter evaluate the performance of the Deep Neural Network (DNN) under the condition that an adversary modifies the sensor data. DNN has been widely used for traffic road-side traffic prediction [82]. In the given analysis, multi-layer perceptron based DNN trained using backpropagation algorithm. The given neural network uses the historical traffic from a group of sensors to predict future traffic at a particular location. The network provides an accuracy of 95% when clean data is used as an input to the neural network. The experiment modifies the input to the DNN, during its operation, randomly, either by a positive (additive attack) or negative (deductive attack) value. The analysis assume that the attack is not introduced during the training phase of the network. The attack is injected while the neural network is used for the prediction application. Experiments examine the error variation at the output node as attack strength or percentage of values in the input vector is varied.

Figure 6.1 shows the performance of the DNN for the case when attack strength is varied for different percentage of values in the input vector. Here, it should be noted that the input vector contains the data coming from the loop detectors (that may be under attack). As expected the prediction error increases for the higher percentage of input vector values under attack and greater attack strength. Figure 6.1a and 6.1b show that to cause significant variation in error either attack strength has to be high or a large percentage of values in the input vector have to be altered. For example, in Figure 6.1a if an adversary injects an attack of strength 0.5 (normalized value) per input value then if 83.3% of values in the input vector are changed the error goes above 10%. Otherwise, for the

low percentage of input vector values under attack, the prediction error is in the range of 5%.

Figure 6.2a and 6.2b present the scenario when the percentage of sensors under attack are varied for different attack strength. The figure depicts that irrespective of attack strength, the error does not increase for less percentage of input vector values under attack. However, for both the additive and deductive attacks, as the number of sensors increases above a threshold point, the error suddenly starts increasing. Specifically, for additive attacks, at least 20% of input vector values feeding input to the DNN need to be targeted. Similarly, for the deductive attack, the number of input vector values targeted should be around 40%.

### **6.3 Observations and remarks**

The analysis in this chapter reveals that the malicious anomalies can significantly alter the performance of an application. An attack in correlated set of sensors for a longer period of time may significantly affect the performance of such applications. Thus, detection of malicious anomalies is an important problem to prevent applications from making adverse decisions. In the next chapter, the detection of such anomalies is discussed in more detail.

## Chapter 7

# Malicious anomaly detection

The evolution of the smart city has led to the big data era where automatic detection of anomalies is an important problem that need be investigated. Detecting anomalies can be useful to any automated IoT-enabled applications as they assist in making accurate optimal decisions. The system utilizes these anomalies in making an intelligent decision based on surrounding environmental conditions. For example, a sudden increase in traffic between two routers can be detected as an anomaly by a Software-defined Network (SDN) controller, that may reroute other important data traffic to some different nodes. Similarly, any intrusion at a physical place is detected as an anomaly to trigger an alarm so that the user can take a proactive action. Thus, IoT that encompasses the presence of sensors all around us can assist in accurate monitoring of environmental data and based on that applications make automated decisions [60].

However, there is also a possibility that a malicious adversary can take control of the sensors, manipulating its data, thus giving rise to false anomalies. This may have a long term impact on the side of the application provider [39]. Thus, although the anomalies due to variation in environmental conditions are

useful to an application, the anomalies due to compromised sensors have adverse effect on application performance. Therefore, finding anomalous values in sample data due to such malicious manipulation is a significant problem in the context of smart city. Although, there have been several works done on finding anomalies due to external environmental conditions, a limited number of research works consider the anomalies arising from malicious manipulation of sample data. It is important to find such anomalies as once it is detected, a proactive remedial measure can be taken without affecting system performance.

This chapter attempts to determine the anomalies due to compromised sensor data. Our approach is different from state-of-the-art as it does not rely on the properties of sample data that is being under test. We employ data mining on diverse heterogeneous time-series from different sources to find anomalies. This helps us in detecting anomalies due to the malicious tampering of data. Our approach is based on the idea that heterogeneous sample data are often correlated. A variation in one dataset must be accompanied by corresponding variation in the correlated one. For example, vehicular traffic at a certain location and noise level at the same location are correlated to each other [85]. The change in vehicular traffic must be accompanied by some change in noise level at the same location. A variation in one of them but not in other must lead to suspect the presence of a malicious manipulation.

Thus, in the given chapter, we build models to predict a certain time-series data from another correlated set of time-series. The predicted value is used for detecting malicious anomalies. The existing works in the literature have used the statistical properties in the same dataset to detect the anomalies [31], [86],

[43]. These methods are suitable for the anomaly detection due to abnormal variation arising in environmental data. Our approach can efficiently capture the anomalies due to malicious manipulation of the sensor data.

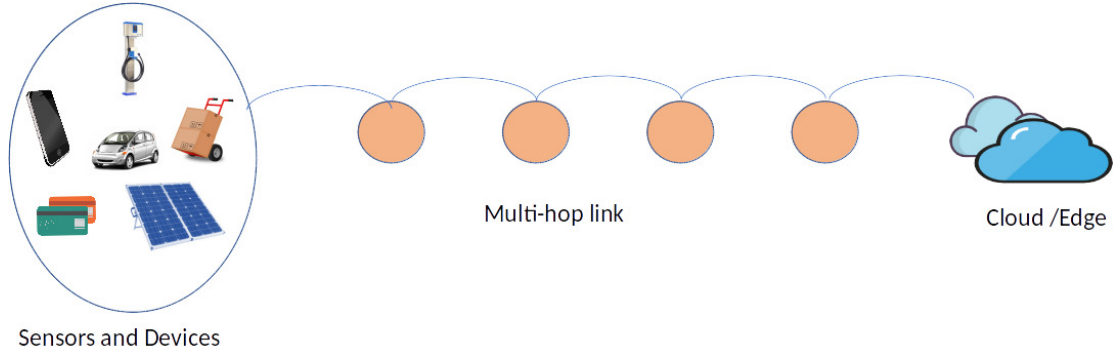
To solve the problem, we design a neural network inspired from the Ensemble learning [87] and use it in conjunction with rule-based statistical analysis. Our proposed neural network predicts a certain quantity using surrounding environmental parameters. Since we are using surrounding parameters, any variation in those conditions are captured in predicted test parameter. Then, based on the predicted values of the test parameter, we perform statistical analysis to determine whether there is an anomaly. Thus, any anomaly due to malicious data perturbation is captured.

## **7.1 Attack model, problem statement, and solution motivation**

In this section, we formally describe the attack model and problem statement. Then, we proceed to discuss motivations of the given solution to detect the malicious anomalies.

### **7.1.1 Attack model**

Figure 7.1 shows the typical computation model in smart city where sensors and devices transmit their data to the cloud or edge for processing. Normally, the data is transferred through a multi-hop link. For example, in connected



**Figure 7.1:** The model describing the attack surface

vehicle scenario, the information sensed by a vehicle is transmitted to other nearby vehicles, then Road Side Units (RSUs), and then to the nearby edge server. It is possible that an intermediate node manipulates the data so that it is not equal to the value measured by a sensor.

$$Opt(s_i^r) = y \quad (7.1a)$$

$$s_i^r + \delta = s_i \quad (7.1b)$$

$$Opt(s_i^r + \delta) = y + \Delta \quad (7.1c)$$

The effect of such manipulation is described in equation 7.1. We consider sensor  $s$  that measures certain quantity after every time-interval  $\tau_s$ . Sensor measures some parameter  $s_i^r$  at time  $i$ , where  $i \in \mathbb{N}$  and represents the  $i_{th}$  time-slot in units of  $\tau_s$ . In equation 7.1a, an application makes a decision  $y$ , based on an optimization algorithm  $Opt$ , on the value of the environmental parameter  $s_i^r$ . If an adversary manipulates the value of  $s_i^r$  by quantity  $\delta$ , the new value becomes  $s_i$  (equation 7.1b). The optimization method, in the presence of such malicious

data manipulation, will make a decision  $y + \Delta$  instead of  $y$ . Thus, the expected output of an application will be altered if such anomalies are present in the data.

### 7.1.2 Problem statement

In above attack scenario, the value of data used by an application is not equal to the actual value measured by the sensor, due to manipulation in intermediate nodes. The problem thus converges to determine if the ground truth sensor value is equal to the value used by an application service. Thus, given the value  $s_i$ , used in an application, whether this is equal to  $s_i^r$ . That is to determine whether equation 7.2 holds true.

$$s_i^r \sim s_i \tag{7.2}$$

If the values are similar, the measured sensor value and the one used by application service provider are similar. This leads that the data has not been modified while it traverses from sensor to cloud. On the other hand, if there is difference between two values, a malicious manipulation is there.

### 7.1.3 Solution motivation

Anomaly detection is one of the imperative problem in data science. To find if the measured sensor data contains anomaly, researcher rely on statistical characteristics and distribution in the sample. In anomaly detection problem, in general, a distance-based technique is used where a point is compared with

nearby or expected value. Then, based on the distance between two it is classified as an anomaly. However, such approach does not consider whether the deviation from an expected value is due to variation in environmental condition or due to malicious data manipulation. It is possible that a certain value is anomalous due to either surrounding conditions or malicious manipulation by an adversary.

In order to address the above problem, in this chapter, we use heterogeneous environmental parameters and attempt to estimate the sensor values  $s_i^r$  using surrounding environmental parameters. Our solution is motivated by the fact that a certain test parameter and surrounding environmental conditions are correlated. For example, environmental pollutant level at a certain location depends upon vehicular traffic density at the same location [85]. A high vehicular density region must have high pollutant level. Similarly, vehicular traffic and noise pollutant are correlated [85]. Thus, one can be predicted from other and thus can also be used for finding any malicious data manipulation. If environmental conditions are constant but the test variable changes then it can be suspected that there is a data manipulation. Hence, our solution uses data from different surrounding environmental sources, having varied measurement scale, distribution, and densities, to predict the  $s_i^r$  values. Based on the estimated value, a statistical method is applied to determine if the anomalies exist in the data. Since we use surrounding environmental conditions to predict a certain test variable, any variation in environmental condition is reflected in the predicted quantity. Therefore, our method will predict the anomalies only due to malicious data manipulation.

### 7.1.4 Ensemble learning

The proposed method uses a neural network that is inspired from the ensemble learning. Before describing our neural network architecture, the ensemble learning is explained as follows.

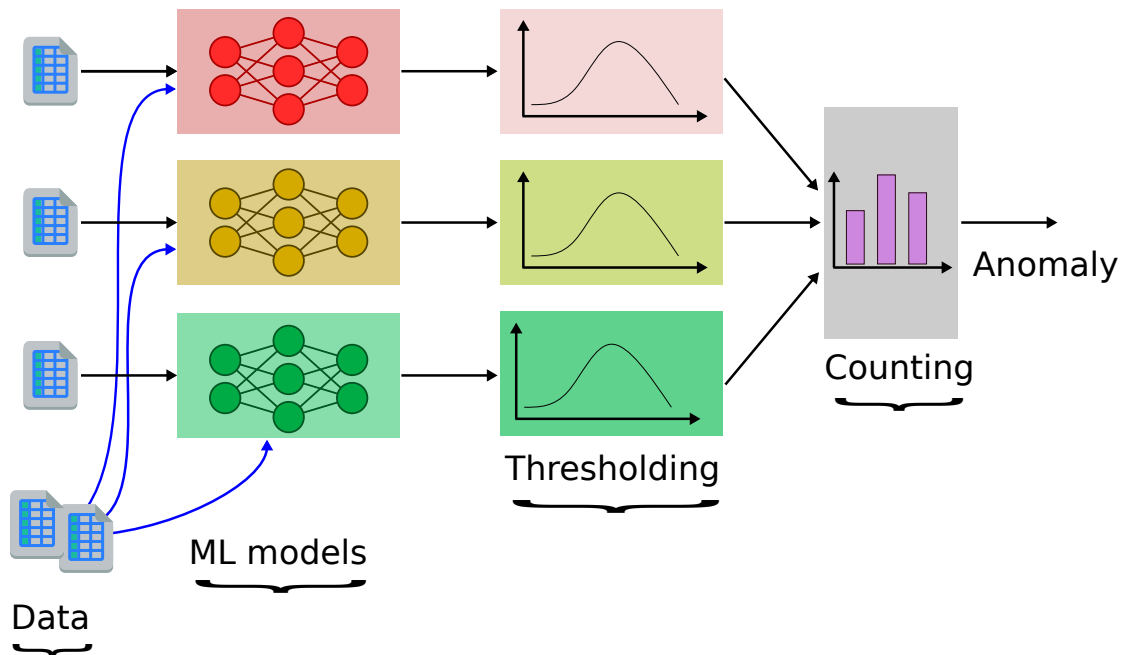
Ensemble learning is a machine learning technique where multiple learners are combined to solve a problem. In contrast to ordinary methods of Machine learning, that trains a single model to get an optimal parameter, ensemble learning combines different models to get a better result. Thus, in ensemble learning a set of hypothesis are trained and then combined to estimate the result [87]. The ensemble learning is efficient because it boosts the multiple weak learners to get a strong learner for a better accuracy. The individual learner in Ensemble learning method are called as the base learner. These learners are combined either in sequential or parallel. To obtain the final prediction, either majority voting or weighted averaging is used. For ensembles to produce a better prediction, the base learners should be accurate and diverse in nature. Diversity enables the ensemble learning to be more generalized in nature. The generalization capability of the ensemble learning lies in the fact that given a training data, it is often not possible to predict the best machine learning model for a given problem. Therefore, combining multiple learners is a better choice. Also, the search process of different algorithms vary. Thus, even if we know the best algorithm to be used in a given situation, the search process may not provide optimal solution. Thus, to compensate this the different learners are combined to get optimal performance. Hence, the ensemble learning is a powerful method to ensure better generalization capability in a training process [88].

The capability of the ensemble learning method is based on the diversity in different base learners. Therefore, its ability to correct the errors of some of the members is based on the fact that different learners make different errors on same set of inputs. Thus, the combination of these different learners reduces the overall prediction error. Diversity can be introduced using different methods. For example, the training data can be partitioned and different learners are trained using different datasets. The different learners may be developed using different algorithms. Also, the model parameters can be different to introduce diversity in base learners [87].

## 7.2 System architecture for malicious anomaly detection

Figure 7.2 shows the proposed architecture to detect if the malicious anomalies are present in the data. Input to the given architecture is data obtained from surrounding conditions. Output provides whether a certain point of the test variable is anomalous. For every input parameter, a ML model is developed and trained to predict the test parameter. These models are called as the base learners and are specific to certain input variable.

As shown in the figure, some input parameters are fed across all the ML Models. The base learners separately predict the value of the test parameter. Then, in the next stage, the predicted value is compared with the actual measured value of the test parameter. If the difference between predicted and test parameter value is greater than a certain threshold, the particular model classifies



**Figure 7.2:** Malicious anomaly detector architecture

the point as anomalous. The results of thresholding process are combined together and counted. If more than a certain number of units classifies a point as anomaly, the point is considered as maliciously tampered.

The data obtained from different sensors are heterogeneous in timescale and measurement units. The next sections describes how the data from different sensors are combined together to make prediction and then for anomaly detection.

### 7.2.1 Data preprocessing

We consider two types of environmental parameters. One set of parameter is used to employ diversity in the machine learning. These parameters are fed separately to different base learners. Such parameters are represented using

variable  $P_l$ , where  $l \in \{1, 2, \dots, L\}$ . Here,  $l$  is a variable that represents a particular base learner. The total number of base learners are represented by variable  $L$ . The sensor measuring the  $P_l$  values provide the measurement after every  $\tau_l$  unit of time. The values in set  $P_l$  are given as  $\{p_1^l, p_2^l, \dots\}$ . Similarly, the other group of parameters are represented by variable  $R_m$ , where  $m \in \{1, 2, \dots, M\}$ . Parameters  $R_m$  are used as input to all the base learners. There are total number of  $M$  such parameters. The values of set  $R_m$  are measured after  $\tau_m$  time units and are given as  $\{r_1^1, r_2^2, \dots\}$ .

The parameters in set  $R_m$  affects the values in  $P_l$ , and thus the combined effect of  $P_l$  and  $R_m$  influences the test variable. For example, the set  $P_l$  may contain a particular vehicular related pollutant and set  $R_m$  wind speed data. In this case, wind speed at a particular location affects the environmental pollutant level. Since environmental pollutant at a certain location depends upon vehicular traffic at that location, a reverse-engineering can be performed to predict vehicular traffic (test variable) using environmental pollutant and wind speed.

The data obtained from the sensors may have missing values. These missing sensor values are replaced using mean imputation technique. Since the different sensors provide measurement in different time scale ( $\tau_l$  and  $\tau_m$ ), it is converted to a common timescale scale  $\tau$ .

The common timescale  $\tau$  is selected such that  $\min(\text{set}(\tau_l), \text{set}(\tau_m), \tau_s) \leq \tau \leq \max(\text{set}(\tau_l), \text{set}(\tau_m), \tau_s)$ . Here,  $\tau_s$  is the time period after which test parameter provides measurement. The value  $\tau$  follows the constraint that the  $\frac{\tau_i}{\tau}$  is an integer  $\forall \{\text{set}(\tau_l), \text{set}(\tau_m), \tau_s\} > \tau$  and  $\frac{\tau}{\tau_i}$  is an integer  $\forall \{\text{set}(\tau_l), \text{set}(\tau_m), \tau_s\} < \tau$ . The

processed data for parameters  $P_l$ ,  $R_m$ , and test data  $S$  in timescale  $\tau$  are represented either using equations 7.3 or 7.4 depending upon whether  $\tau_l, \tau_m, \tau_s > \tau$  or  $\tau_l, \tau_m, \tau_s < \tau$  respectively.

$$p_i^l = \sum_{j=\tau_l/\tau}^{\tau_l/\tau} p_j^l \quad (7.3a)$$

$$r_i^m = \sum_{j=\tau_m/\tau}^{\tau_m/\tau} r_j^m \quad (7.3b)$$

$$s_i = \sum_{j=\tau_s/\tau}^{\tau_s/\tau} s_j \quad (7.3c)$$

$$p_i^l = p_i^l \rightarrow p_{i+\tau/\tau_l}^l \quad (7.4a)$$

$$r_i^m = r_i^m \rightarrow r_{i+\tau/\tau_m}^m \quad (7.4b)$$

$$s_i = s_i \rightarrow s_{i+\tau/\tau_s} \quad (7.4c)$$

After time scaling, the values of parameters are normalized in the range between  $0 \rightarrow 1$  using min-max method. Thus, for a particular parameter value  $p_i^l$ , the normalized value is obtained as  $\frac{p_i^l - \min(P_l)}{\max(P_l) - \min(P_l)}$ . The same operation is performed for parameters  $R_m$  and test sensor data  $S$ . Also, as parameters  $P_l$  and  $R_m$  represent different quantities, they are converted to a common distribution using equation 7.5, where  $std(P_l)$  is the standard deviation of normalized values in  $P_l$ . The similar operation is performed for the values of  $R_m$  and  $S$ .

$$p_i^l = \frac{p_i^l - \text{mean}(P_l)}{\text{std}(P_l)} \quad (7.5)$$

## 7.2.2 Machine learning model

The transformed datasets  $P_l$  and  $R_m$  are used for developing neural network models to predict the test parameter  $S$ . During training phase, the known values of datasets  $P_l$ ,  $R_m$ , and  $S$  are used to update model parameters. Here, as explained in earlier section, it is considered that the parameters  $P_l$  and  $R_m$  collectively affects the values of  $S$ . During test phase,  $P_l$  and  $R_m$  are input to the individual base learner to obtain the values of parameter  $S$ . Then, based on the obtained predicted value, anomaly test is performed to determine whether data is maliciously manipulated.

For neural network training and testing, first the input and output vectors are determined. For this purpose, a parameter  $\Gamma$  is considered. The  $\Gamma$  depicts the samples of time-series of each environmental parameter fed to the neural network. We consider that  $\Gamma$  is same for every environmental parameter. Thus, the input vector for a model  $l$  is obtained using equation 7.6. In the given equation, symbol  $\rightarrow$  is used to represent the consecutive values of a particular parameter in the given range.

$$x_i^l = \{set(p_i^l \rightarrow p_{i+\Gamma-1}^l) \cup set(r_i^1 \rightarrow r_{i+\Gamma-1}^1) \cup set(r_i^2 \rightarrow r_{i+\Gamma-1}^2) \dots \cup set(r_i^M \rightarrow r_{i+\Gamma-1}^M)\} \quad (7.6)$$

For the set of values  $x_i^l$ , the known measured value of sensor  $s$  at time slot  $i+\Gamma-1$  is  $s_{i+\Gamma-1}^l$ . The vector  $x_i^l$  and  $s_{i+\Gamma-1}^l$  are used for training the base learner  $l$ .

Algorithm 10 describes the process of base learner selection. Algorithm takes

---

**Algorithm 10** Model selection
 

---

**Input:**  $set(x_i^l)$ ,  $set(s_{i+\Gamma-1}^l)$ - Set of input and output values

$\vartheta_1, \vartheta_2 \dots \vartheta_j$ - Set of base algorithm methods

**Output:**  $C_1, C_2 \dots C_L$ - Base learners

- 1: **for**  $l \leftarrow 1$  **to**  $L$  **do**
  - 2:   Obtain training data  $X_l^{train} \subset set(x_i^l)$  and  $Y_l^{train} \subset set(s_{i+\Gamma-1}^l)$
  - 3:   Obtain test data  $X_l^{test} \subset set(x_i^l)$  and  $Y_l^{test} \subset set(s_{i+\Gamma-1}^l)$  such that  $X_l^{train} \cap X_l^{test} = NULL$  and  $Y_l^{train} \cap Y_l^{test} = NULL$
  - 4:   **for**  $k \leftarrow 1$  **to**  $K$  **do**
  - 5:     Find hypothesis  $h_k$  for  $\vartheta_j$  on  $X_l^{train}$  and  $Y_l^{train}$
  - 6:     Find error  $rmse_l = rmse(Y_l^{test}, h_k(X_l^{test}))$
  - 7:     **if**  $rmse_l < min(rmse_l)$  **then**
  - 8:       Replace  $C_l$  with  $\vartheta_j$
  - 9:     **end if**
  - 10:   **end for**
  - 11: **end for**
- 

training data and based on that provides the base learner models  $C_l$  for different values of  $l$ . To find  $C_l$ , we use various neural network algorithms represented by variable  $\vartheta_j$ , where  $j \in \{1, 2, \dots, J\}$ .

The different neural network algorithms  $\vartheta_j$  chosen in this chapter are Multi-layer Perceptrons (MLP), Long-Short Term Neural Network (LSTM), Convolution Neural Network (CNN), and CNN combined with LSTM. The prime objective of the given method is given input, output, and  $\vartheta_j$ , determine the neural network model that minimizes Root Mean Square Error (RMSE)  $rmse_l$  for the base learner  $l$ . For every base learner, algorithm checks the model  $\vartheta_j$  and trains it to find the optimal weights of the network. The algorithm searches for all the values of  $\vartheta_j$  and chooses one that provides minimum RMSE error.

The obtained model  $C_l$  and its trained hypothesis function  $h_l$  are used during operating phase to detect the anomalies. During operating phase, input  $x_i^l$  is fed to a particular base learner to obtain the output  $h_l(x_i^l)$ . The predicted output is

used in subsequent stages to detect anomalies.

### 7.2.3 Thresholding

To determine whether the particular value is anomalous, we use the predicted sensor value from different base learners. If the difference between predicted value and the value used in optimization algorithm is greater than a threshold, it is classified as an anomaly (equation 7.7). In equation 7.7, the  $\beta$  is a tunable parameter and the  $e_l$  determines the status of anomaly. If the value of  $e_l$  is 1, the corresponding point is classified as anomaly based on predicted output of the base learner  $l$ .

$$e_l = (|s_i - h_l(x_i^l)| < \beta \times th) \quad (7.7)$$

### 7.2.4 Counter

Values  $e_l$  are added to determine the anomalous sample values. For this purpose, we use equation 7.8 to combine the output after the thresholding process. We use parameter  $K$ , where  $1 \leq K \leq L$ , for deciding if a point in the sample is anomaly. A particular value is anomaly if the sum of  $e_l$  values is greater than or equal to  $K$  such that equation 7.8 evaluates to be true. The parameter  $K$  can be tuned according to the application requirement. A low value of  $K$  signifies that the anomaly detector is a strict and even a single predicted anomalous output, after thresholding, classifies the point as anomaly. A high  $K$  value, close to  $L$ ,

classifies a value as anomaly if majority of outputs after thresholding predicts the point as anomalous.

$$e_1 + e_2 \dots + e_l \geq K \quad (7.8)$$

## 7.3 Evaluation results

In this section, we describe the experimental results to evaluate the performance of the proposed method. We divide our result into two parts. First, we discuss the prediction performance using different neural networks. Subsequently, we describe the performance of the anomaly detector. However, before describing the results, we present the description of data and model parameters.

### 7.3.1 Data

To verify the effectiveness of the given model, we use the environmental pollutants from California. Pollutants considered are Carbon Monoxide (CO), Carbon dioxide ( $CO_2$ ), Black Carbon (BC), PM25HR, and Nitric Oxides (NOX). We also chose temperature and wind speed as input to the neural network. Different base learner are fed with different pollutant. Temperature and wind speed data are provided as input to all the base learners. The test parameter is the vehicular traffic at the same location. We predict the vehicular traffic at a point using various base learners that use above parameters as input and then based on that decide whether the measured value is anomalous. The environmental

pollutants, weather, and temperature data is collected from California Air Resource Board [76]. The data is obtained for a location near San Jose, California. The traffic data is collected at a nearby freeway point from the Performance Measurement System (PeMs) [56]. Both the datasets are collected for the period of one month.

### 7.3.2 Base learner parameters

This section describes the parameters used to train the neural networks.

The different neural network-based base learners  $\vartheta_j$  are used and compared to find the optimal base learner. Specifically, our approach used Multilayer Perceptrons (MLP), Long-Short Term Neural Network (LSTM), Convolution Neural Network (CNN), and a hybrid of LSTM and CNN model.

First, we chose the parameter  $\Gamma$  for selecting cardinality of the input vector. A low value of  $\Gamma$  will result in lower accuracy as less amount of information is used in input. High  $\Gamma$  value makes the training process computation intensive as the size of inputs are large. Therefore, we balance between very low and high value to chose  $\Gamma$  as 16. Thus, the cardinality of the input vectors is  $16 \times 3$  because three parameters, environment pollutant, wind speed, and temperature are input to a base learner. We kept the cardinality of the input vectors consistent throughout different algorithms.

The number of epochs to train the neural network are chosen as 1500. The learning rate is initially set to the value 0.001. While training the models, we observe the mean absolute error, if it does not decrease for 50 consecutive epochs then

the learning rate is reduced by the factor of half. The minimum possible learning rate is considered to be 0.0001. Another important parameter is batch size. Rather than training individual input vectors, the weights are updated according to average error obtained in a batch of input vectors. Feeding the data in batches also decreases the run time as the hardware is optimally utilized. Therefore, we kept the batch size of 256. The above parameters are kept consistent for different  $\vartheta_l$ . We describe below the model parameters specific to the different neural networks.

### **MLP**

For MLP neural network, we chose four hidden layers. The number of units in different layers are chosen as 48, 72, 48, 32, 16, and 1. Thus, we gradually decrease the number of units in hidden layers from 72 to 1. We tried with different activation functions and found that *relu* activation is providing minimum *rmse<sub>l</sub>* value, and thus it is used as neuron activation for input and hidden layers. Linear activation is used for output neuron. In hidden layers, we use a dropout of 40% to avoid the model bias. The network is trained using *adam* optimization method.

### **LSTM**

LSTM has layered architecture. We add 5 LSTM layers followed by a single neuron for dense unit. The number of LSTM units are chosen as 128 in the input layer and 50 in the intermediate layers. The activation function for LSTM units is selected as *softmax*. For output neuron, a linear activation function is

used. In every layer, a dropout of 40% is chosen. For LSTM training, *adam* optimization was found to be suitable due to low training time and better accuracy.

## CNN

CNN is another popular neural network that is used for predictive analytics. Since we have single dimensional input vectors, the 1D convolution is performed using CNN. First we chose two convolution layer and then a max-pooling layer. In these layers, filter size of 64 and kernel size of 3 is used. Subsequently, two more convolution and a global-average pooling layer is added. Here, filter size and kernel size are selected as 128 and 3 respectively. We employ *relu* activation in hidden layers. For dense layer, we used *softmax* activation function. The dropout of 50% is used after global-average pooling layer. For CNN training, *rmsprop* optimizer is used.

## CNN\_LSTM

The hybrid of CNN and LSTM contains a CNN network followed by a LSTM network. The CNN network is similar to one in the above subsection. After that, we add two LSTM layers with 50 units in each layer. The activation function in LSTM units is chosen as *sigmoid*. The dropout between CNN and LSTM is 50%, while between two LSTM layers is 20%. A single output neuron having *sigmoid* activation is added after LSTM network. The combined network is trained using *rmsprop* optimization.

### 7.3.3 Evaluation metrics

We evaluate the prediction and anomaly detection results using different metrics. The prediction performance is evaluated for different base learners using mean square error  $MSE_l$  and mean average error  $MAE_l$  as described in equations 7.9, where  $y_i$  is the actual sensor value at time-slot  $i$ .

$$MSE_l = \frac{\sum_i (h_l(x_i^l) - y_i)^2}{N} \quad (7.9a)$$

$$MAE_l = \frac{\sum_i |h_l(x_i^l) - y_i|}{N} \quad (7.9b)$$

$$(7.9c)$$

The performance of anomaly detector is evaluated using Precision, Recall, and F-Measure values. Precision  $P$  is the ratio of ground truth anomalies and total number of detected anomalies and is given in equation 7.10. The ground truth anomalies that are detected are represented using true positive  $tp$  value. The anomalies that are not detected by given method are called as false positives  $fp$ .

$$P = \frac{tp}{tp + fp} \quad (7.10)$$

The recall value  $R$  is given as in equation 7.11. It is determined as the ratio of true positive and the sum of true positive and false negatives  $fn$ . The false negatives are the correct sample values but are detected as anomalies. The precision and recall values are combined to get the F-measure values as given in equation 7.12. Here, to calculate F-measure, both precision and recall are given equal weights. The general form of F-measure is given in equation 7.13, where

ML model	MAE	MSE
MLP	0.13	0.03
LSTM	0.22	0.07
CNN	0.17	0.04
CNN_LSTM	0.15	0.04

**Table 7.1:** Mean Average and Mean Squared error for CO as input variable

if  $\beta > 1$ , the more weight is given to recall but if  $\beta < 1$  it is inclined towards precision.

$$R = \frac{tp}{tp + fn} \quad (7.11)$$

$$F = 2 \times \left( \frac{P \times R}{P + R} \right) \quad (7.12)$$

$$F = (1 + \beta^2) \times \left( \frac{P \times R}{\beta^2 \times P + R} \right), \text{ where } \beta > 0 \quad (7.13)$$

### 7.3.4 Prediction results

We compared the prediction performance of different neural networks. The prediction results for MLP, LSTM, CNN, and CNN\_LSTM for different pollutants are depicted in tables 7.1-7.5.

The tables show that the average error for different neural networks is between 13% to 22%. The mean squared error is between 3% to 7%. For different cases, we find that the LSTM network performs poorly. Also, the LSTM neural network is computationally inefficient as compared to the MLP or CNN. Hence,

ML model	MAE	MSE
MLP	0.13	0.03
LSTM	0.22	0.07
CNN	0.17	0.05
CNN_LSTM	0.16	0.04

**Table 7.2:** Mean Average and Mean Squared error for BC as input variable

ML model	MAE	MSE
MLP	0.12	0.03
LSTM	0.22	0.07
CNN	0.15	0.03
CNN_LSTM	0.15	0.04

**Table 7.3:** Mean Average and Mean Squared error for NO<sub>2</sub> as input variable

LSTM is not a feasible solution, being inaccurate as well as computationally inefficient. The CNN and CNN\_LSTM show comparable performance for different pollutants. For example, when CO and BC are used as the input, the difference in error is around 2% and 1% respectively. For other pollutants, the MAE is same for CNN and CNN\_LSTM. However, CNN\_LSTM has LSTM layers and thus it is computationally inefficient as compared to the CNN only network. Thus, among two the preference is given to the CNN. Furthermore, the MLP and CNN also has somewhat comparable performance. The maximum difference between the MAE is 3%. The computational complexity of MLP is also somewhat lower than the CNN. Thus, for the give dataset we prefer MLP for base learners as for every pollutant it provides the minimum error.

ML model	MAE	MSE
MLP	0.13	0.03
LSTM	0.22	0.07
CNN	0.16	0.04
CNN_LSTM	0.16	0.04

**Table 7.4:** Mean Average and Mean Squared error for NOX as input variable

ML model	MAE	MSE
MLP	0.14	0.03
LSTM	0.22	0.07
CNN	0.16	0.04
CNN_LSTM	0.16	0.04

**Table 7.5:** Mean Average and Mean Squared error for PM25HR as input variable

### 7.3.5 Anomaly detection

This section describes the anomaly detection results. For the analysis purpose, we inject malicious anomalies in the data. The injected anomalies are proportional to the mean standard deviation in the training data. The anomalies are introduced in the test data. We analyze the performance as the anomaly magnitude, parameter  $\beta$ , and parameter  $K$  are varied.

#### Analysis as magnitude is varied

Table 7.6 shows the metric values as the anomaly magnitude is varied. The values of  $\beta$  and  $K$  are set as 3. As the table depicts, the precision values are between 99% and 100%. That is the anomalies are efficiently detected using the

given method for various anomaly strengths. The recall is around 76% for the different strength of the anomalies. Thus, we see that for the given case the proposed method accurately detects the anomalies but the recall is lower. If we compare the F-measure then it is around 86%, that shows the method performs decently.

	Precision	Recall	F-measure
mean_std_dev_0.5	0.99	0.76	0.86
mean_std_dev_1	0.99	0.76	0.86
mean_std_dev_1.5	1.0	0.76	0.86
mean_std_dev_2.0	1.0	0.76	0.86
mean_std_dev_2.5	1.0	0.76	0.86
mean_std_dev_3	1.0	0.76	0.86
mean_std_dev_3.5	1.0	0.76	0.86
mean_std_dev_4	1.0	0.76	0.86
mean_std_dev_4.5	1.0	0.76	0.86
mean_std_dev_5	1.0	0.76	0.86

**Table 7.6:** Performance measurement as anomaly magnitude is varied

### Analysis as parameter $\beta$ is varied

Table 7.7 shows the metric values for different values of  $\beta$ . The magnitude of anomalies is set as 1.5 times the mean of the standard deviation and the  $K$  is set to 3. The table shows that as the  $\beta$  is increased the precision decreases while recall increases. The highest values of precision and recall are both 100%. However, if precision is 100%, the recall has minimum value of 56%. On the other hand, if recall is 100%, the precision is at its lowest level 59%. Thus, one comes at the cost of the other. From the table, it can be said that when the

value of  $\beta$  is 5, the optimal performance is obtained with the highest value of F-measure.

	Precision	Recall	F-measure
1	1.0	0.56	0.72
2	1.0	0.64	0.78
3	1.0	0.76	0.86
4	0.98	0.83	0.9
5	0.95	0.88	0.91
6	0.84	0.94	0.89
7	0.77	0.97	0.86
8	0.71	0.99	0.83
9	0.66	1.0	0.8
10	0.59	1.0	0.74

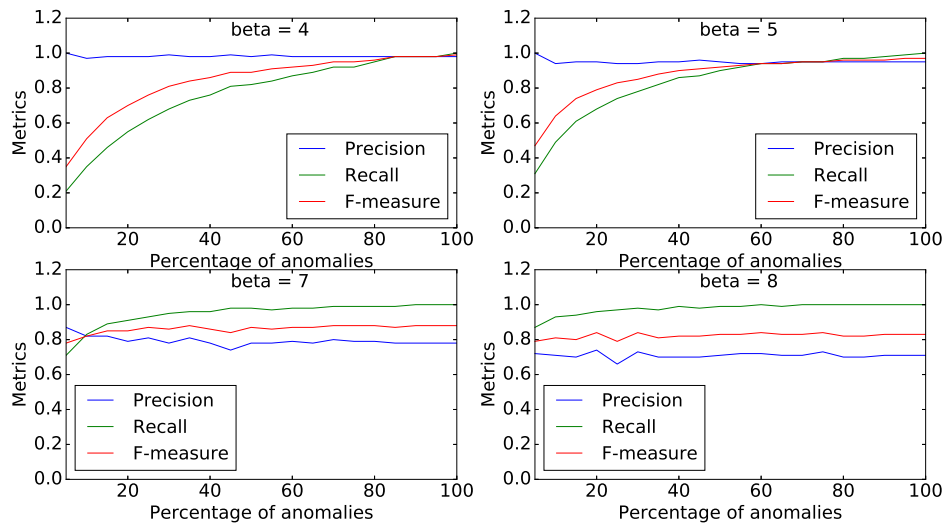
**Table 7.7:** Performance measurement as parameter  $\beta$  is varied

### Analysis as parameter $K$ is varied

Table 7.8 presents the performance metrics as the parameter  $K$  is varied. The anomaly strength is chosen as 1.5 times the standard deviation value and the value of  $\beta$  is set as 5. The table depicts that as  $K$  increases, the precision decreases. The recall increases on increasing  $K$ . The increase in recall value is significant and it has lowest value 77% when the value of  $K$  is 1 and increases to 94% when the value of  $K$  is 5. The F-measure values also increases on increasing  $K$ . The minimum value of F-measure is 86% when  $K$  is 1. The F-measure value increases to 93% when  $K$  is set to 5. Thus, we can conclude that the increases of  $K$  has significant effect on the accuracy of the model as both the recall and F-measure increases.

	Precision	Recall	F-measure
1	0.97	0.77	0.86
2	0.96	0.85	0.9
3	0.95	0.88	0.91
4	0.94	0.91	0.92
5	0.93	0.94	0.93

**Table 7.8:** Performance measurement as parameter  $K$  is varied



**Figure 7.3:** Metrics variation as percentage of anomalies vary

### Analysis as percentage of anomalies vary

Figure 7.3 shows the variation in metric values as the number of anomalies vary. We analyze when data has very less number of anomalies scattered over a certain duration. These points are on the left portion of the horizontal axis in the graph. We also analyze when there is larger penetration of anomalies in a certain duration. Thus, in the right portion of the graphs anomalies are present almost throughout the time-series. As we can see from the figure, for point

anomalies, the recall values are lower. As the number of anomalies increase, the metric values increases mainly because our method has high precision.

## **7.4 Observations and remarks**

This chapter has presented the importance of malicious anomaly detection in sensor data. The chapter portrayed an attack model where adversary can manipulate the sensor data to cause unwanted decision making by an application service. To detect such anomalies, we provided a neural network architecture that first predicts the test variable and then finds anomaly using a statistical test. The proposed model uses different neural network for different environmental variables. One of the limitations of the given model is that the prediction error is high. The model can be improved using better machine learning algorithms and using spatial-temporal information to predict test parameter.

## Chapter 8

# Conclusions and future research direction

The availability of sensor data in smart city can be potentially used for the development of many applications and services. In this thesis, application services for Plug-in Electric Vehicle Charging in smart city are presented and described in detail. The thesis described the various constraints like electricity requirement and longer charging time related to PEV charging. The thesis has presented a rectangle placement based algorithm to schedule the PEV for charging in parking places. For en route PEVs, set-cardinality based optimization algorithm is presented to reduce average time a PEV spends waiting and charging at a station. Our algorithms have better performance as compared to the traditional methods of PEV Charging.

Another smart-city problem, related to the efficient use of sensor data in various applications, that we considered is on anomaly detection. The effective detection of anomalies in smart city is important as it helps in making application

services take proactive decisions. The thesis has presented an anomaly detection model that is not only accurate but also scalable and can be used for large number of sensors. Further, we also analyzed the effect of malicious anomalies for predictive analytics-based applications. Since, malicious anomalies significantly affect the performance of applications, in an adverse manner, the thesis presented a method to detect them.

Our parked PEV charging methods can further be improved by considering dynamic arrival and departure of vehicles. For this purpose, machine learning-based predictive analytics can be integrated to the given method. The prediction mechanism may predict the number of PEVs arriving and departing at a particular time. Based on that vehicles can be chosen for optimal charging to minimize influence in the grid. The en route PEV charging can be further improved by considering constraints related to the energy requirement.

This thesis has considered the optimization and anomaly detection problems separately. However, in real-world application deployment, the two problems need to be considered together. Thus, an application service should first analyze the sensor data to detect real and malicious anomalies. The malicious anomalies must be discarded. The real anomalies should be used in the optimization algorithm. Thus, any application service must be developed in such a way to make decision based on anomalies. Our PEV charging applications can thus further be improved by integrating the anomaly detector in it.

## Bibliography

- [1] J. Pan, R. Jain, S. Paul, T. Vu, A. Saifullah, and M. Sha, “An internet of things framework for smart energy in buildings: Designs, prototype, and experiments”, *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 527–537, Dec. 2015.
- [2] Y. Sun, H. Song, A. J. Jara, and R. Bie, “Internet of things and big data analytics for smart and connected communities”, *IEEE Access Journal*, vol. 4, pp. 766–773, Feb. 2016.
- [3] *Pev benefits*. [Online]. Available: <http://mbeva.org/why-drive-clean/pev-benefits/>.
- [4] A. C. Zambroni de Souza and D. Q. Oliveira, “Plug-in electric vehicles management in smart distribution systems”, in *Plug In Electric Vehicles in Smart Grids: Energy Management*, S. Rajakaruna, F. Shahnian, and A. Ghosh, Eds. Singapore: Springer, 2015, pp. 59–77.
- [5] S. Barker, A. Mishra, D. Irwin, P. Shenoy, and J. Albrecht, “Smartcap: Flattening peak electricity demand in smart homes”, in *Proc. of the IEEE International Conference on Pervasive Computing and Communications*, Lugano, Switzerland, Mar. 2012, pp. 67–75.

- [6] P. Gayatri, G. D. Sukumar, and J. Jithendranath, "Effect of load change on source parameters in power system", in *Proc. of the Power, Control, Communication and Computational Technologies for Sustainable Growth*, Kurnool, India, Dec. 2015, pp. 178–182.
- [7] *Electric vehicle charging technology analysis and standards*. [Online]. Available: <http://www.fsec.ucf.edu/en/publications/pdf/FSEC-CR-1996-15.pdf>.
- [8] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods", in *Proc. of the ACM Workshop on Artificial Intelligence and Security*, Dallas, Texas, USA, Nov. 2017, pp. 3–14.
- [9] D. Hendrycks and K. Gimpel, *Early methods for detecting adversarial images*, 2016. arXiv: 1608.00530 [cs.LG].
- [10] A. Meola, *Automotive industry trends: Iot connected smart cars and vehicles*, Dec. 2016. [Online]. Available: <http://www.businessinsider.com/internet-of-things-connected-smart-cars-2016-10>.
- [11] F. Mwasilu, J. Justo, E. Kim, T. Do, and J. Jung, "Electric vehicles and smart grid interaction: A review on vehicle to grid and renewable energy sources integration", *Renewable and Sustainable Energy Reviews*, vol. 34, pp. 501–516, Jun. 2014.
- [12] E. Coalition, *State of the plug-in electric vehicle market*, 2016. [Online]. Available: <http://www.electrificationcoalition.org/sites/default/files/>.
- [13] V. Caswell, *5 mobile apps to increase driving experience*, Mar. 2016. [Online]. Available: <http://www.trueautoprotection.com/auto-warranty-and-maintenance-blog/topic/spotify>.

- [14] N. study, "u.s. dot advances deployment of connected vehicle technology to prevent hundreds of thousands of crashes", Jul. 2016. [Online]. Available: <https://www.nhtsa.gov/press-releases>.
- [15] S. Sultan, M. Al-Doori, A. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network", *Journal of Network and Computer Applications*, vol. 37, pp. 380–392, Jan. 2014.
- [16] R. Shukla and S. Sengupta, "A novel software-defined network based approach for charging station allocation to plugged-in electric vehicles", in *Proc. of IEEE International Symposium on Network Computing and Applications*, Boston, Massachusetts, Oct. 2017, pp. 437–441.
- [17] Z. Chen, L. Jiang, W. Hu, K. Ha, B. Amos, P. Pillai, A. Hauptmann, and S. Mahadev, "Early implementation experience with wearable cognitive assistance applications", in *Proc. of the Workshop on Wearable Systems and Applications*, ser. WearSys '15, Florence, Italy, May 2015, pp. 33–38.
- [18] M. Gerla, E. K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds", in *Proc. of the IEEE World Forum on Internet of Things*, Mar. 2014, pp. 241–246.
- [19] T. D. Chen, K. M. Kockelman, W. J. M. J. Fellow, and M. Khan, "The electric vehicle charging station location problem: A parking-based assignment method for seattle", in *Annual meeting of Transportation Research Board*, vol. 340, Jan. 2013, pp. 13–1254.
- [20] I. S. Bayram, A. Tajer, M. Abdallah, and K. Qaraqe, "Capacity planning frameworks for electric vehicle charging stations with multiclass customers", *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1934–1943, Jul. 2015.

- [21] H. Zhang, S. J. Moura, Z. Hu, and Y. Song, "Pev fast-charging station siting and sizing on coupled transportation and power networks", *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2595–2605, Oct. 2018.
- [22] N. Shahraki, H. Cai, M. Turkey, and M. Xu, "Optimal locations of electric public charging stations using real world vehicle travel patterns", *Transportation Research Part D: Transport and Environment*, vol. 41, pp. 165–176, Dec. 2015.
- [23] G. Wang, Z. Xu, F. Wen, and K. P. Wong, "Traffic-constrained multiobjective planning of electric-vehicle charging stations", *IEEE Transactions on Power Delivery*, vol. 28, no. 4, pp. 2363–2372, Oct. 2013.
- [24] N. S. Nafi, K. Ahmed, M. Datta, and M. A. Gregory, "A novel software defined wireless sensor network based grid to vehicle load management system", in *Proc. of the International Conference on Signal Processing and Communication Systems*, Dec. 2016, pp. 1–6.
- [25] H. Lu, G. Pang, and G. Kesidis, "Automated scheduling of deferrable pev/phev load by power-profile unevenness", in *Proc. of the IEEE International Conference on Smart Grid Communications*, Oct. 2013, pp. 235–240.
- [26] S. Chen and L. Tong, "Iems for large scale charging of electric vehicles: Architecture and optimal online scheduling", in *Proc. of the IEEE International Conference on Smart Grid Communications*, Nov. 2012, pp. 629–634.
- [27] S. Deilami, A. S. Masoum, P. S. Moses, and M. A. S. Masoum, "Real-time coordination of plug-in electric vehicle charging in smart grids to minimize power losses and improve voltage profile", *IEEE Transactions on Smart Grid*, vol. 2, no. 3, pp. 456–467, Sep. 2011.

- [28] Z. Ma, D. S. Callaway, and I. A. Hiskens, "Decentralized charging control of large populations of plug-in electric vehicles", *IEEE Transactions on Control Systems Technology*, vol. 21, no. 1, pp. 67–78, Jan. 2013.
- [29] J. C. Mukherjee and A. Gupta, "Distributed charge scheduling of plug-in electric vehicles using inter-aggregator collaboration", *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 331–341, Jan. 2017.
- [30] J. Mohammadi, G. Hug, and S. Kar, "A fully distributed cooperative charging approach for plug-in electric vehicles", *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3507–3518, Jul. 2018.
- [31] L. Huang, X. Nguyen, M. Garofalakis, M. I. Jordan, A. Joseph, and N. Taft, "In-network pca and anomaly detection", in *Proc. of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06, Canada, Dec. 2006, pp. 617–624.
- [32] T. Yu, X. Wang, and A. Shami, "Recursive principal component analysis-based data outlier detection and sensor data aggregation in iot systems", *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2207–2216, Dec. 2017.
- [33] *Rad - outlier detection on big data*. [Online]. Available: <https://medium.com/netflix-techblog/rad-outlier-detection-on-big-data-d6b0494371cc>.
- [34] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?", *Journal of the ACM*, vol. 58, no. 3, 11:1–11:37, Jun. 2011.
- [35] J. Shlens, "A tutorial on principal component analysis", *CoRR*, 2014. [Online]. Available: <http://arxiv.org/abs/1404.1100>.
- [36] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers", in *Proc. of the ACM SIGMOD International*

- Conference on Management of Data*, Dallas, Texas, USA, May 2000, pp. 93–104.
- [37] H. Kriegel, M. S. Hubert, and A. Zimek, “Angle-based outlier detection in high-dimensional data”, in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, USA, 2008, pp. 444–452.
- [38] N. Pham and R. Pagh, “A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data”, in *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Beijing, China, 2012, pp. 877–885.
- [39] S. Bhattacharjee, A. Thakur, and S. K. Das, “Towards fast and semi-supervised identification of smart meters launching data falsification attacks”, in *Proc. of the 2018 Asia Conference on Computer and Communications Security*, Incheon, Republic of Korea, 2018, pp. 173–185.
- [40] O. Vallis, J. Hochenbaum, and A. Kejariwal, “A novel technique for long-term anomaly detection in the cloud”, in *Proc. of the USENIX Conference on Hot Topics in Cloud Computing*, Philadelphia, PA, 2014, p. 15.
- [41] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey”, *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009.
- [42] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, “High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning”, *Journal of Pattern Recognition*, vol. 58, pp. 121–134, Oct. 2016.
- [43] N. Pandeewari and G. Kumar, “Anomaly detection system in cloud environment using fuzzy clustering based ann”, *Mobile Networks and Applications*, vol. 21, no. 3, pp. 494–505, Jun. 2016.

- [44] S. Rajasegarar, C. Leckie, and M. Palaniswami, "Hyperspherical cluster based distributed anomaly detection in wireless sensor networks", *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1833–1847, Jan. 2014.
- [45] A. Albanese, S. K. Pal, and A. Petrosino, "Rough sets, kernel set, and spatiotemporal outlier detection", *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 194–207, Jan. 2014.
- [46] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques", *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, Jan. 2016.
- [47] *The hidden costs of low load factor*. [Online]. Available: <https://www.csu.org/CSUDocuments/hiddencostslowloadfactor.pdf>.
- [48] J. Taylor, L. de Menezes, and P. McSharry, "A comparison of univariate methods for forecasting electricity demand up to a day ahead", *International Journal of Forecasting*, vol. 22, no. 1, pp. 1–16, Nov. 2005.
- [49] S. Fan and R. J. Hyndman, "Short-term load forecasting based on a semi-parametric additive model", *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 134–141, Feb. 2012.
- [50] *Sae j1772*. [Online]. Available: [http://ecee.colorado.edu/~ecen5017/lectures/CU/L40\\_out.pdf](http://ecee.colorado.edu/~ecen5017/lectures/CU/L40_out.pdf).
- [51] *Electric vehicle charging guide*. [Online]. Available: <https://chargehub.com/en/electric-car-charging-guide.html>.
- [52] N. Chen, M. Wang, N. Zhang, X. S. Shen, and D. Zhao, "Sdn-based framework for the pev integrated smart grid", *IEEE Network*, vol. 31, no. 2, pp. 14–21, Mar. 2017.

- [53] D. Steen, T. Le Anh, M. Ortega-Vasquez, O. Carlson, L. Bertling, and V. Neimane, "Scheduling charging of electric vehicles for optimal distribution systems planning and operation", in *Proc. of the International Conference on Electricity Distribution*, Frankfurt, Germany, Jun. 2011, pp. 6–9.
- [54] L. Jun, "Power flow and power flow calculation research in power system", in *Intelligence Computation and Evolutionary Computation*, Berlin, Heidelberg: Springer, 2013, pp. 439–445.
- [55] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking", *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 27–51, Jun. 2015.
- [56] *California department of transportation*. [Online]. Available: <http://pems.dot.ca.gov/>.
- [57] L. V. Fausett *et al.*, *Fundamentals of neural networks: Architectures, algorithms, and applications*. Prentice-Hall Englewood Cliffs, 1994, vol. 3.
- [58] *Time complexity*. [Online]. Available: <https://wiki.python.org/moin/TimeComplexity>.
- [59] R. M. Shukla, S. Sengupta, and A. N. Patra, "Smart plug-in electric vehicle charging to reduce electric load variation at a parking place", in *Proc. of the IEEE Annual Computing and Communication Workshop and Conference*, Las Vegas, NV, USA, Jan. 2018, pp. 632–638.
- [60] R. M. Shukla, S. Sengupta, and M. Chatterjee, "Software-defined network and cloud-edge collaboration for smart and connected vehicles", in *Proc. of the Workshop Program of the International Conference on Distributed Computing and Networking*, Varanasi, India, Jan. 2018, 6:1–6:6.

- [61] R. M. Shukla, P. Kansakar, and A. Munir, "A neural network-based appliance scheduling methodology for smart homes and buildings with multiple power sources", in *Proc. of the IEEE International Symposium on Nano-electronic and Information Systems*, Gwalior, India, Dec. 2016, pp. 166–171.
- [62] *Generalized esd test for outliers*. [Online]. Available: <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35h3.html>.
- [63] P. J. Rousseeuw and M. Hubert, "Anomaly detection by robust statistics", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, 2018.
- [64] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series", in *Proc. of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Presses universitaires de Louvain, Bruges, Belgium, 2015.
- [65] *Hierarchical clustering algorithms*. [Online]. Available: [https://home.deib.polimi.it/matteucc/Clustering/tutorial\\_html/hierarchical.html](https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html).
- [66] H. K. Kanagala and V. V. Jaya Rama Krishnaiah, "A comparative study of k-means, dbscan and optics", in *Proc. of the International Conference on Computer Communication and Informatics*, Coimbatore, India, Jan. 2016, pp. 1–6.
- [67] X. Jin and J. Han, "Expectation maximization clustering", in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer, 2010, pp. 382–383.
- [68] X. Zhang, "Support vector machines", in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer, 2010, pp. 941–946.

- [69] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial", *Journal of Computer*, no. 3, pp. 31–44, 1996.
- [70] I. Jolliffe, "Principal component analysis", in *International Encyclopedia of Statistical Science*, M. Lovric, Ed. Berlin, Heidelberg: Springer, 2011, pp. 1094–1096.
- [71] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms", *The Computer Journal*, vol. 26, no. 4, pp. 354–359, 1983.
- [72] J. S. Farris, "On the cophenetic correlation coefficient", *Journal of Systematic Zoology*, vol. 18, no. 3, pp. 279–285, 1969.
- [73] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data", *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 187–197, May 2015.
- [74] *Keras documentation*. [Online]. Available: <https://keras.io/optimizers/>.
- [75] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014.
- [76] *California air resource board*. [Online]. Available: <https://ww3.arb.ca.gov/html/ds.htm/>.
- [77] *Scikit-learn*. [Online]. Available: <https://scikit-learn.org/stable/>.
- [78] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A novel technique for long-term anomaly detection in the cloud", in *Proc. of the USENIX Conference on Hot Topics in Cloud Computing*, Philadelphia, PA: USENIX, 2014, p. 15.

- [79] *Ami system security requirements*. [Online]. Available: [https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/14-AMI\\_System\\_Security\\_Requirements\\_updated.pdf](https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/14-AMI_System_Security_Requirements_updated.pdf).
- [80] J. Guo, Z. Wang, and H. Chen, "On-line multi-step prediction of short term traffic flow based on gru neural network", in *Proc. of the International Conference on Intelligent Information Processing*, Bangkok, Thailand, Jul. 2017, 11:1–11:6.
- [81] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, "A hybrid deep learning based traffic flow prediction method and its understanding", *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 166–180, May 2018.
- [82] N. G. Polson and V. O. Sokolov, "Deep learning for short-term traffic flow prediction", *Transportation Research Part C: Emerging Technologies*, vol. 79, pp. 1–17, 2017.
- [83] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas, "Mobile-edge computing", in *ETSI White Paper*, Sep. 2014.
- [84] *A higher technical standard for the austrian autobahn*. [Online]. Available: <https://www.cisco.com/c/dam/en/us/products/collateral/security/us-asfinag-case-study.pdf>.
- [85] Z. Ross, I. Kheirbek, J. E. Clougherty, K. Ito, T. Matte, S. Markowitz, and H. Eisl, "Noise, air pollutants and traffic: Continuous measurement and correlation at a high-traffic location in new york city", *Journal of Environmental Research*, vol. 111, no. 8, pp. 1054–1063, 2011.

- [86] M. Cheng, Q. Xu, J. Lv, W. Liu, Q. Li, and J. Wang, "Ms-lstm: A multi-scale lstm model for bgp anomaly detection", in *Proc. of the IEEE International Conference on Network Protocols*, Singapore, Nov. 2016, pp. 1–6.
- [87] C. Zhang and Y. Ma, *Ensemble machine learning: Methods and applications*. Springer, 2012.
- [88] H. Lappalainen and J. W. Miskin, "Ensemble learning", in *Advances in Independent Component Analysis*, M. Girolami, Ed. London: Springer, 2000, pp. 75–92.