DATA-DRIVEN FRAMEWORK FOR ENERGY MANAGEMENT IN EXTENDED RANGE ELECTRIC VEHICLES USED IN PACKAGE DELIVERY APPLICATIONS

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE
UNIVERSITY OF MINNESOTA
BY

Pengyue Wang

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

ADVISOR: WILLIAM NORTHROP

AUGUST 2020

Acknowledgement

I would like to thank my advisor Dr. William Northrop for his support during my graduate study. I am sincerely grateful to have such an intelligent and caring advisor for my Ph. D. study. Without his guidance, it would be impossible for me to finish all the research work, and this thesis would not be possible. I am especially grateful to him for encouraging me to learn new things and come up with new ideas for our research projects. His feedback was the most essential part of all the work presented in this thesis. I also appreciate the research assistant positions he provided in the last four years and his generous sponsorship of the international conferences I attended.

For my research project, I am thankful to Dr. Shashi Shekhar for his advising throughout my graduate study and Yan Li for his partnership. I would like to acknowledge Shawn Haag for his help in the research project. I would also like to thank Dr. Zongxuan Sun and Dr. Rajesh Rajamani for their support on my doctoral committee.

I would like to thank Zonghu Han, Rui Luo and Heng Wang for being great roommates as well as great friends in the last four years. I enjoy the time spending with them together and I appreciate the help they provide to help me go through some difficult times. I would also like to thank Ying Lin for his help for many things. Although I cannot list all my friends in the University of Minnesota, I am deeply grateful to all of you.

Finally, I would like to thank my parents Tianju Chen and Yanjun Wang, and my girlfriend Ran Li. My parent's support is the foundation of my graduate study and I am grateful to them for always trusting me. Ran is always supportive of my study and her encouragement is invaluable to me. I cannot imagine my life without her.

Abstract

Plug-in Hybrid Electric Vehicles (PHEVs) have potential to achieve high fuel efficiency and reduce on-road emissions compared to engine-powered vehicles when using well-designed Energy Management Strategies (EMSs). The EMS of PHEVs has been a research focus for many years and optimal or near optimal performance has been achieved using control-oriented approaches like Dynamic Programming (DP) and Model Predictive Control (MPC). These approaches either require accurate predictive models for the trip information during driving cycles or detailed velocity profiles in advance. However, such detailed information is not feasible to obtain in some real-world applications like the delivery vehicle application studied in this work. Here, data-driven approaches were developed and tested over real-world trips with the help of two-way Vehicle-to-Cloud (V2C) connectivity.

First, the EMS problem was formulated as a probability density estimation problem and solved by Bayesian inference. The Bayesian algorithm deals with the condition where only small amounts of data are available and sequential parameter estimation problem elegantly, which matches the characteristics of the data generated by delivery vehicles. The predicted value of the parameter for the next trip is determined by the carefully designed prior information and all the available data of the vehicle so far. The parameter is updated before the delivery tasks using the latest trip information and stays static during the trip. This method was demonstrated on 13 vehicles with 155 real-world delivery trips in total and achieved an average of 8.9% energy efficiency improvement with respect to MPGe (miles per gallon equivalent).

For vehicles with sufficient data that can represent the characteristics of future delivery trips, the EMS problem was formulated as a sequential decision-making problem under uncertainty and solved by deep reinforcement learning (DRL) algorithms. An intelligent agent was trained by interacting with the simulated environment built based on the vehicle model and historical trips. After training and validation, optimized parameter in the EMS was updated by the trained intelligent agent during the trip. This method was demonstrated on 3 vehicles with 36 real-world delivery trips in total and achieved an average of 20.8% energy efficiency improvement in MPGe.

Finally, I investigated three problems that could be encountered when the developed DRL algorithms are deployed in real-world applications: model uncertainty, environment uncertainty and adversarial attacks. For model uncertainty, an uncertainty-aware DRL agent was developed, enabled by the technique of Bayesian ensemble. Given a state, the agent quantifies the uncertainty about the output action, which means although actions will be calculated for all input states, the high uncertainty associated with unfamiliar or novel states is captured. For environment uncertainty, a risk-aware DRL agent was built based on distributional RL algorithms. Instead of making decisions based on expected returns as standard RL algorithms, actions were chosen with respect to conditional value at risk, which gives more flexibility to the user and can be adapted according to different application scenarios. Lastly, the influence of adversarial attacks on the developed neural network based DRL agents was quantified. My work shows that to apply DRL agents on real-world transportation systems, adversarial examples in the form of cyber-attack should be considered carefully.

TABLE OF CONTENTS

List of Tables ..... *v*
List of Figures ..... *vi*
1. Introduction *1*
1.1. Background and Motivation **1**
1.2. Literature Review **4**
1.3. Outline and Contribution **14**
2. Problem Statement For Extended Range Electric Vehicle Energy Management Strategy *16*
2.1. Vehicle Configuration **16**
2.2. Data Description and Preprocessing **17**
2.3. Energy Management Strategy Introduction **21**
2.4. Problem to Solve **25**
3. Bayesian Inference Based Energy Management *29*
3.1. Probability Density Estimation Problem Formulation **29**
3.2. Maximum Likelihood and Bayesian Inference **30**
3.3. Vehicle Model **37**
3.4. Designing of the prior **41**
3.5. Results and Discussion **46**
4. Deep Reinforcement Learning Based Energy Management *52*
4.1. Reinforcement Learning **52**
4.2. Feedforward Neural Networks **55**
4.3. Reinforcement Learning Problem Formulation **59**
4.4. Value-based Method **61**
4.5. Actor-Critic-based Method **67**
4.6. Results and Discussion **70**
5. Safe and Robust Algorithms *87*
5.1. Uncertainties in Reinforcement Learning **89**
5.2. Risk-aware Energy Management **93**
5.3. Uncertainty-aware Energy Management **115**
5.4. Adversarial Attacks on Deep RL-based Energy Management **128**
6. Conclusion *141*
Bibliography *146*

LIST OF TABLES

Table 2.1: Vehicle Specifications 17

Table 3.1: Vehicle model parameters 37
Table 3.2: Bayesian model parameters 43
Table 3.3: Fuel efficiency improvement 51

Table 4.1: Neural network structures and hyperparameters for DQN 71
Table 4.2: Neural network structures and hyperparameters for DDPG 74
Table 4.3: Statistics of the three selected routes 80
Table 4.4: Summary of testing results 86

Table 5.1: Neural network structures and hyperparameters for D3PG 97
Table 5.2: Comparison of QR loss and MSE under two problem settings 106
Table 5.3: Tradeoff between fuel use and risk of running out of battery 114

LIST OF FIGURES

Figure 1.1: Energy flow in the U.S., 2018 (Quadrillion Btu) [1].....2
Figure 1.2: Illustration of how real-world velocity profiles are generated. 8
Figure 1.3: Illustration comparing rule-based (RB) EMS strategies to optimization-based (OB) strategies for human driven vehicle systems. 13

Figure 2.1: Powertrain configuration of the studied EREV. 16
Figure 2.2: Comparison of raw data and preprocessed vehicle speed and distance data. 21
Figure 2.3: Comparison of $SOCref$ and real-time SOC (upper) and fuel use (lower) with different $Lset$ settings for one measured delivery trip.....24
Figure 2.4 Distribution of distance and energy intensity for an example EREV.....26
Figure 2.5: Four historical GPS trajectories of an example delivery vehicle. 27
Figure 2.6: A typical delivery trip data with high $Lset$ setting. 28

Figure 3.1: Illustration of how to determine the $Lset$ from a known Gaussian distribution of one vehicle. 30
Figure 3.2: Validation of the vehicle model by comparing the raw SOC data and simulated SOC data.....41
Figure 3.3: Distribution of the mean (left) and standard deviation (right) of trip distance for all delivery vehicles used to design the prior from the collected dataset. 42
Figure 3.4: Bayesian $Lset$ and the best $Lset$ curves for four vehicles. 44
Figure 3.5: Initial and final t-distribution and the normalized actual data.....45
Figure 3.6: T-distributions with different number of data. 45
Figure 3.7: Components of the developed Bayesian inference framework. 46
Figure 3.8: Bayesian $Lset$, best $Lset$ and baseline $Lset$ curves for vehicle T11 (upper) and T12 (lower).....47
Figure 3.9: Illustration of how fuel is saved in a real delivery trip.....48
Figure 3.10: Engine operation frequency for Bayesian EMS and baseline case. 48
Figure 3.11: Fuel use and minimal SOC under different $Lset$ for a trip of vehicle D. ...49

Figure 3.12: Fuel use and MPGe comparison for test trips of vehicle T11. 50
Figure 3.13: Fuel use and MPGe comparison for test trips of vehicle T12. 50

Figure 4.1: An example of a feedforward neural network with one hidden layer. The bias parameters have a fixed value of 1..... 56
Figure 4.2: Illustration of one update step in Algorithm 1..... 66
Figure 4.3: Illustration of one update step in Algorithm 2..... 69
Figure 4.4: Learning curves of 3 learning rates (upper) and batch sizes (lower). 70
Figure 4.5: Performance of DQN on a test trip with 40 miles. 71
Figure 4.6: Performance of DQN on a test trip with 50 miles. 72
Figure 4.7: Action-values for state A (left) and state B (right). 72
Figure 4.8: Performance of DQN on low initial $Lset$ values. 73
Figure 4.9: Comparison of MPGe by using DQN and baseline settings on test trips..... 74
Figure 4.10: Velocity profiles of two example test trips with a distance of 35 miles (upper) and 50 miles (lower)..... 75
Figure 4.11: Performance of DDPG on a test trip with 50 miles. 75
Figure 4.12: Performance of DDPG on a test trip with 35 miles. 75
Figure 4.13: Action-values of state A (left) and state B (right) calculated by the critic for 13 example actions in the predefined range of [-20, +20]. The red bar indicates the action that is calculated by the actor. 76
Figure 4.14: Performance of DDPG on low initial $Lset$ values. 77
Figure 4.15: Performance of DDPG on low initial SOC values. 77
Figure 4.16: Comparison of MPGe by using DDPG and baseline settings on test trips. 78
Figure 4.17: Characteristics of the delivery trips of three testing vehicles. 79
Figure 4.18: Example GPS trajectories of testing vehicle R1, R2 and R3. 82
Figure 4.19: Diagram illustrating the testing framework of DDPG. 82
Figure 4.20: Velocity profile of the example test trip. 83
Figure 4.21: SOC, $SOCref$ and $Lset$ profile for the example test trip. 83

Figure 4.22: SOC, fuel use and engine state comparison between RL method and baseline method. In the last figure, two “on” positions are labeled on the y-axis to differentiate the two conditions.....84
Figure 4.23: MPGe and fuel use comparison of all test trips of R1, R2 and R3.....85
Figure 4.24: Mean and one standard deviation of MPGe improvement of R1, R2 and R3. 86

Figure 5.1: Ten NNs are trained. The prediction of each individual NN is shown on the left figure. The mean prediction is shown on the right figure with one and two standard deviation represented by the shaded region. 90
Figure 5.2: A one-step decision-making problem showing the importance of estimating the uncertainty associated with different actions. The T represents terminal states. 92
Figure 5.3: Multi-modality (left) and high variance (right) return distributions. 92
Figure 5.4: An example return distribution for a state-action pair with $N = 8$. 94
Figure 5.5: One step in the for loop in Algorithm 3. The blue arrow and red arrow indicate the distribution of the probability is according to its distances with the nearest two atoms. 95
Figure 5.6: Performance of the D3PG on three test trips with distances of 35 miles(upper), 51 miles (middle) and 50 miles (lower). The states associated with A, B and C are all at time step 26000s. The dashed lines help to see how close is the real-time SOC with the value of 10%.....98
Figure 5.7: The Q-values of three actions under states A (left), B (middle) and C (right). The actor action is 0 for states A and B (do not change the $Lset$) and +6.5 for state C. .99
Figure 5.8: Return distributions of an actor action and two example actions for state A (left), state B (middle) and state C (right). 100
Figure 5.9: Return distribution of actor action (left), +15 (middle) and -15 (right) for state C. 100
Figure 5.10: The variance of return distributions of three actions under state A(left), state B(middle) and state C(right). 102

Figure 5.11: Illustration of the differences between DQN and IQN. The DQN outputs the expected value for a given state-action pair while IQN outputs samples from the implicitly modeled return distribution whose mean is the output of DQN. 103
Figure 5.12: A simple example illustrating how to sample returns $Z\tau(s,a)$ from the implicitly modeled return distribution. Assume five τ values: 0.1, 0.3…0.9 are sampled from the uniform distribution $U([0,1])$, five values of $Z\tau(s,a)$ are calculated by feeding the five τ values into the estimated $QFZs,a\tau$. With enough number of samples, the mean and other statistics of the underlying return distribution can be estimated accurately. . 104
Figure 5.13: Comparison of fitting an artificial dataset with linear regression and quantile regression with 0.9, 0.5 and 0.1 quantiles. For each x, twenty y values are sampled from a gamma distribution. For linear regression (LR), the MSE is minimized, which tries to estimate the conditional mean of the underlying assumed Gaussian distribution for each input x. For quantile regression, there is no assumption about the underlying distribution and the conditional quantile values are estimated. 105
Figure 5.14: Illustration of CVaR with $\eta = 1.0$ (left) and $\eta = 0.5$ (right). It can be observed that although the expected value of return distribution $Z(s,a1)$ is higher than that of $Z(s,a2)$ (the position of the vertical dashed line), it does not hold true for $\eta = 0.5$ due to its high variance (annotated by arrows). 107
Figure 5.15: Illustration of the proposed risk-aware framework. 108
Figure 5.16: Performance of the trained algorithm on a test trip with a distance of 35 miles with $\eta = 1.0$, $\eta = 0.5$ and $\eta = 0.1$. To make the figure clear, the $Lset$ corresponding to $\eta = 0.1$ is not shown. 109
Figure 5.17: Return distributions with mean and variance for all possible actions in state A. The smooth line is fitted with a kernel density estimator for better visualization, and the mean and variance were still calculated with the original data. The y-axis represents the probability density so that the value could be higher than 1. The red dot represents the mean value of each distribution. 109
Figure 5.18: Performance of the trained algorithm on a test trip with a distance of 48 miles with $\eta = 1.0$ (upper), $\eta = 0.5$ (middle) and $\eta = 0.1$ (lower). 111

Figure 5.19: Return distributions with mean and variance for all possible actions in state B. The red dots indicate expected values of the estimated return distributions. 112
Figure 5.20: Highest CVaR among possible actions along the two test trips. 113
Figure 5.21: Distance and energy intensity distribution of training trips and training-testing trips combined. 116
Figure 5.22: Comparison of two GPS trajectories with 43.1 and 44.3 miles. 116
Figure 5.23: Illustration of the concepts of frequently visited states, infrequently visited states and unvisited states in a two-dimensional state space. 118
Figure 5.24: Performance of the trained algorithm on a training trip with a distance of 43.1 miles (upper) and the estimated uncertainties along the trip (lower). 124
Figure 5.25: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state A. 124
Figure 5.26: Performance of the trained algorithm on a training trip with a distance of 52.9 miles (upper) and the estimated uncertainties along the trip (lower). 125
Figure 5.27: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state B. 125
Figure 5.28: Performance of the trained algorithm on a test trip with a distance of 57.4 miles (upper) and the estimated uncertainties along the trip (lower). 126
Figure 5.29: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state C. 127
Figure 5.30: Illustration of the proposed uncertainty-aware framework. 128
Figure 5.31: Illustration of the adversarial attacks. The low-dimensional state representation st provided by the EREV is processed by the adversary before sending to the DRL agent on the cloud. 130
Figure 5.32: Performance of the DQN under two kinds of random noise with different values of ε. The error bars indicate one standard deviation. 133
Figure 5.33: Performance of the DQN under FGSM and its two variations. 134
Figure 5.34: Performance of the DQN under the FGSM and FD methods. 135
Figure 5.35: Performance of the DQN under the strategy of transfer across policies. .. 136

Figure 5.36: Performance of the DQN under the strategy of transfer across algorithms. 136
Figure 5.37: Performance of DQN on the recorded trip shown in Figure 3.5 under adversarial examples generated by $L2$ method with $\varepsilon = 0.05$. 137
Figure 5.38: Performance of DQN on the recorded trip shown in Figure 3.6 under adversarial examples generated by FGSM method with $\varepsilon = 0.02$. 137
Figure 5.39: Action-values comparison for state A. The left is for the original state and the right is for the perturbed state. 138
Figure 5.40: Action-values comparison for state B. The left is for the original state and the right is for the perturbed state. 138

1. INTRODUCTION

1.1. *Background and Motivation*

Optimization of the global transportation system has been a popular research topic for many years. The transportation sector consumes a significant amount of energy and is one of the largest sources of pollution globally. For example, vehicles in the U.S. accounted for about 28% of the total energy used in 2018 as shown in Figure 1.1 [1] and emitted a similar portion of anthropogenic carbon dioxide into the environment. Further, air quality is a significant concern for residents of cities all over the world. Emissions from transportation sector is the biggest contributing factor to air pollution. Nearly 33% of carbon dioxide emissions, 24% of methane emissions and 65% of nitrous oxide emissions resulted from fossil fuel combustion for transportation activities [2]. Vehicles consume about 83% of energy in the transportation sector in 2009 in the U.S. [3], motivating the fact that reducing fuel use in on-road vehicles has become an important topic for study.

To alleviate inefficient energy use and emissions caused by combustion of fuels in transportation, hybrid electric vehicles (HEVs) have been introduced and have become more and more widely used. HEVs use regenerative braking, allow the engine to operate within a more efficient range, and can use electrical energy from the grid to improve vehicle emissions and reduce overall energy use. HEVs are powered by at least two energy sources. For most HEVs, these consist of batteries and internal combustion engines (ICEs). Plug-in HEVs (PHEVs) are equipped with powerful electric motors and high-capacity

batteries which can be recharged directly from the grid which allows them to achieve higher fuel efficiency on a gasoline equivalent basis [4]. However, although PHEVs have the potential to achieve high fuel efficiencies and low on-road emissions, an effective energy management strategy (EMS) is necessary to yield the highest improvement over conventionally powered vehicles [5]. The EMS manages how to use the energy properly from the two sources. Usually, the objective of an EMS is to minimize the fuel use [6]. Co-optimization of energy management with other metrics like emissions [7] and battery life [8][9] is also a popular research topic published in current literature.

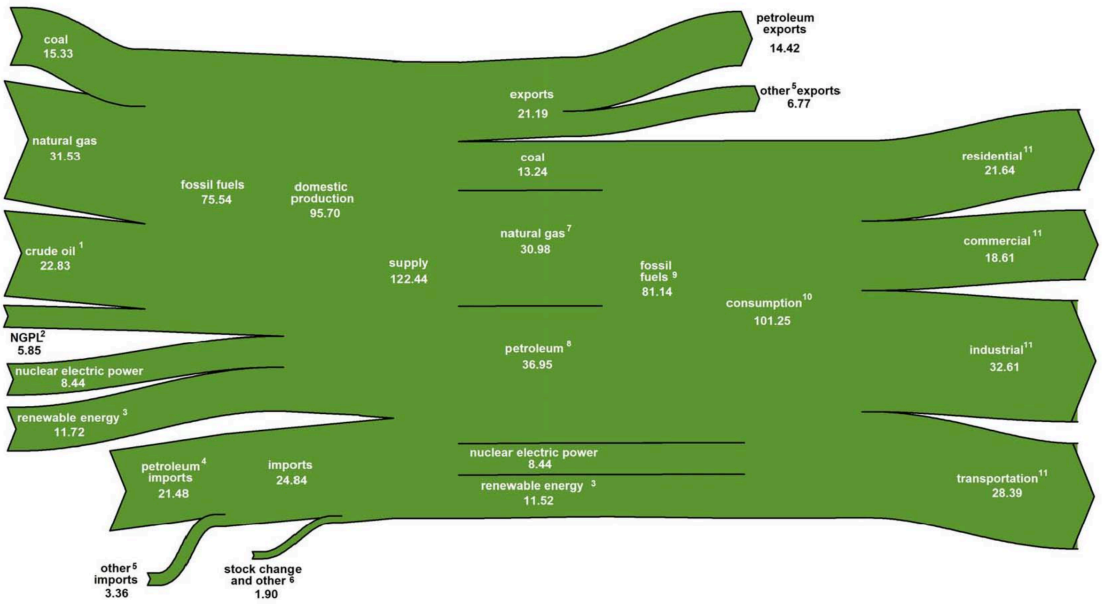


Figure 1.1: Energy flow in the U.S., 2018 (Quadrillion Btu) [1].

The central goal of this project is to improve the fuel efficiency of in-use extended range electric vehicles (EREVs) used for package delivery through real-time EMS optimization using two-way vehicle-to-cloud (V2C) connectivity. Such connectivity is a key component of the field of Intelligent Transportation Systems (ITS). The big data generated from V2C, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I)

connectivity have profound impacts on the design and application of ITS, which can make it more efficient, safer and profitable [10]. Also, data-driven methods have been widely used in various applications in the area of ITS with different degree of success [11].

The EREV is a sub-category of PHEV as it can be charged directly from the electrical grid. It has a high capacity battery which serves as the main energy source. The small range-extending (REx) ICE is not connected to the wheels but instead is used to charge the battery when the EMS considers it is necessary. The addition of a small range extender eliminates the "range anxiety" experienced by the vehicle owner [12] and the total energy efficiency can be improved through judicious application of the REx [13][14].

Most large delivery fleets are equipped with telemetry systems and record trip data like velocity, distance and GPS position. Most vehicle manufacturers also have the ability to update vehicle software and change control parameters on the vehicle directly with their telemetry system. The delivery EREV fleet used in this work records data every five seconds for each vehicle, but higher time resolution is possible using with current telemetry technology, but is limited by data transmission costs. Although comprehensive data are generally collected by telemetry systems, this system is generally not connected to the vehicle EMS to use historical data or real-time trip information to improve fuel efficiency. The main goal of this body of work is to bridge this gap by developing a predictive, intelligent EMS that combines data-driven approaches and physics-based drive cycle simulations to significantly improve the fuel efficiency of EREV vehicles used in delivery applications.

1.2. *Literature Review*

Energy management strategies can be mainly divided into rule-based (RB) methods and optimization-based (OB) methods [5][18][19]. RB methods are widely used industrially as they are robust, easy to implement, can use simple hardware, and have low computational requirements compared with OB methods. OB methods are well-known for the potential to achieve optimal or near optimal performances with respect to fuel efficiencies or other predefined optimization targets. Therefore, a preponderance of existing literature focuses on OB methods.

The feasibility of an EMS mainly dependents on two factors: the computational cost and the assumptions and information needed for execution. The high fuel efficiency achieved by OB methods depends heavily on these two factors. Firstly, OB methods require considerable computational resources [5]. For example, in [20], X. Zeng et al. developed a stochastic model predictive control-based EMS which achieved high fuel efficiency. However, the computation time for each stochastic dynamic programming (DP) execution ranged from 10 to 100 seconds, an unfeasible duration for real-world applications during vehicle operation. Second, OB methods require either detailed trip information or broad assumptions to predict future trip information like the vehicle's second-by-second velocity profile [5]. Accurate predictions about the future are very difficult to make, especially if vehicles are controlled by human drivers in actual traffic conditions. This difficulty has motivated considerable previous research regarding driving style recognition as has been summarized in [21].

Two approaches have been proposed to estimate future trip information for OB methods. The first is to make assumptions about the future trip and build predictive models based on those assumptions. The second is to use information from advanced transportation infrastructure like Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) and Vehicle-to-Cloud (V2C) connectivity to provide short-term "look-ahead" data.

A wide variety of assumptions have been applied to incorporate predictive models for enabling OB EMS methods in real-time with varying degrees of success. In [22], future driver power demands were modeled as Markov Chains which made the real-time optimization of stochastic model predictive control (SMPC) feasible. In [23], results from DP served as training data for neural networks (NNs) which were used as online controllers. First, DP was used to get the optimal control strategy on six standard driving cycles. Then, two neural networks were used to imitate the optimal behavior, given some designed inputs under two conditions of the known and unknown remaining distance. After training, the performance of the trained NN controllers was shown on six driving cycles which represented different driving conditions. In [24], a real-time EMS based on Pontryagin's minimum principle (PMP) was developed and tested on three standard driving cycles and one real driving cycle. X. Zeng et al. [20] modeled road grade, speed limit and stops using three transition matrices. The computational complexity of the proposed stochastic DP (SDP) algorithm was highly dependent on the sizes of the transition matrices. In [25], the real-time operation of equivalent consumption minimization strategy (ECMS) was realized through the radial basis function NN used to predict energy demand in trip segments. Driving conditions and styles were assumed to

be the same in this work. In [26], an adaptive ECMS for real-time use was developed on standard driving cycles. X. Zeng et al. [27] derived a lookup table for online using by analyzing the results from offline SDP algorithm which benefited from recognizing existing frequent routes. S. Zhang et al. [28] used a Bayes NN to predict the movement of preceding vehicles with the help of information from vehicle-to-vehicle (V2V) communication. S. Xie et al. [29] placed attention on algorithmic time efficiency in an effort to develop a low-cost controller. They compared NN and Markov chain to predict velocity and chose the latter to predict future bus trajectories which facilitated the real-time application of SMPC. In [30], a NN was trained to learn from a near-optimal control strategy on seven standard driving cycles calculated from an OB method and tested on an unseen standard driving cycle. Trip preview data with different horizons were assumed to be available for the application of model predictive control (MPC) on several standard driving cycles in [31].

Incorporating more real-time traffic information from external sources has also been investigated by a number of researchers, an example of the second kind of method to estimate future trip information mentioned previously. The benefits of including road grade information for ECMS and DP were shown in [32] assuming vehicle velocity was constant. S. Kermani et al. [33] assumed the route information was available in advance and showed global optimization methods can be used in real-time for a given bus route. The benefits of information provided by intelligent transportation system (ITS) were investigated in [34]. C. Sun et al. [35] used dynamic traffic feedback data and assumed that all vehicles can provide the required information. The velocity profile of the vehicle

was assumed to be the same as the traffic flow, which was calculated by the average of all vehicles that on that road segment. SOC trajectories were planned by DP every 300 seconds. In [36], a real-time ECMS was enabled by a NN used for velocity prediction, using the information from V2V and V2I connectivity. X. Qi et al. [37] assumed that a short-term velocity prediction model would be available in the future and developed an online EMS based on an evolutionary algorithm. D. Chen et al. [38] performed SOC planning for short horizon optimal control using sparse traffic information over a given route and demonstrated the near optimal fuel efficiency on standard driving cycles. Compared with DP, they reduced the computation time from hours to less than a minute. In [39], it was assumed that an intelligent transportation system could provide the speed and road elevation profile so that optimization by DP would be possible.

Based on this review, it is clear that researchers have successfully demonstrated the potential of OB EMS in real-time applications using predictive methods and information from advanced transportation infrastructures. However, significant deficiencies remain to effectively use them in a human controlled transportation system, thus reducing their feasibility.

The main drawback for using OB methods in real-world applications is that it is nearly impossible to know the velocity profile in advance as it is determined by many complex and stochastic factors. Figure 1.2 shows a simplified version of how velocity profiles are generated in real-world scenarios. There are mainly two groups of factors that influence vehicle velocity profile: the environment and the driver. The environment consists of factors like weather, road condition, other drivers on the road as well as events like

holidays, sports events or concerts. For the driver, driving style and the driver's behavioral condition, or mood, play important roles.

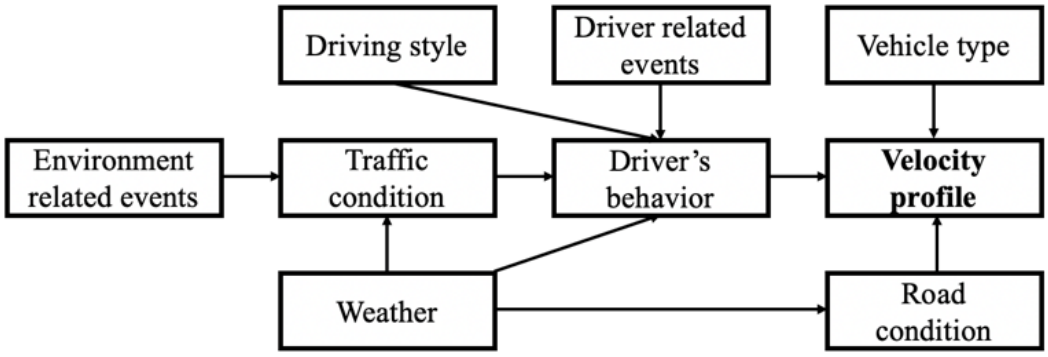


Figure 1.2: Illustration of how real-world velocity profiles are generated.

Velocity profile and driver behavior prediction is itself, a research area. McNew [40] simplified velocity profiles into launch, cruise, and deceleration and used kernel regression to predict the driver's intended cruising speed on a closed course. This was driver-specific and was applicable to rushed or relaxed driving. F. Bender et al. [41] made predictions for garbage trucks whose routes were repeated. They found that the stop positions were very close, and the velocity profiles associated with these positions were very similar. M. Platho et al. [42] focused on road interactions. They first decomposed the situation into small sets of related road users and made predictions for the different sets by different models. S. Cairano et al. [43] used a Markov Chain to model the driver's behavior. S. Lefevre et al. [44] compared parametric and non-parametric approaches for velocity profile prediction during highway driving with lane keeping situations. J. Park et al. [45] used neural networks to predict short-term traffic speed at the sensor location considering factors like day of week and traffic congestion levels. The velocity profile was constructed by the predicted speed at all sensors along the selected route. J. Ziegmann

et al. [46] used kernel regression and recurrent neural networks to predict velocity profiles on experimental routes. B. Jiang et al. [47] assumed the real-time traffic information was accessible and the vehicles that required speed prediction were connected through on-board smartphone communications on a vehicular network. Further, their model was built only for morning rush hour on one particular route.

Route significantly influences the velocity profiles due to variable traffic conditions and the reality that features of different roads can be very diverse. For some real-world problems, the route is not fixed before the trip like in the delivery vehicle trips studied in this work. For a fixed starting point to a fixed destination, the driver can choose different routes according to personal preferences under different traffic conditions. For a package delivery trip, the problem would be more complex if the order of deliveries can be changed freely or partially changed. Therefore, to predict future trip information for this kind of problem, route should be first predicted. Route prediction itself is a difficult topic, as it is highly dependent on assumptions.

R. Simmons et al. [48] assumed driving was largely routine and used a hidden Markov model to model the transition of roads. Although they achieved about 98% accuracy, 95% of the predictions were forced transitions which means there was only one possible way. For the rest of 5% predictions, the performance was about 70%. J. Froehlich et al. [49] also assumed the trips were repeated. During the trip, the algorithm used the GPS trace to find the best match in the database. However, they found that even after a month of observation, 40% of the driver's trip still had new routes, which influenced the model's accuracy largely. Similarly, V. Larsson et al. [50] also tried to identify commuter routes

from historical driving data. Both of the last two papers constructed similarity matrices and used the hierarchical clustering method.

Based on the aforementioned literature, it is evident that to predict high fidelity future trip information like velocity profile, the most critical factors are how much information is available and what simplifications can be made. For example, traffic conditions are highly complex to model as they are determined by all the drivers on the road. To remove the influence of such factors, one can investigate the prediction model on a closed course [40] or experimental route [46]. Also, by assuming that advanced communication infrastructures such as V2V and V2I are available, the information on traffic conditions can be used directly [36].

In recent years, continuous progress has been made in applying model-free, data-driven reinforcement learning (RL) algorithms in the area of EMS. First, the near optimal performance under known trip information is demonstrated on comprehensive standard driving cycles and realistic driving cycles. In [51]-[54], action-values were stored as tables and the entries were updated according to value-based RL algorithms such as Q-learning. The transition pairs used in the update equations were generated by running the vehicle models on standard driving cycles. A parametric study to find hyperparameters for tabular-based Q-learning algorithm was performed in [55] based on one standard driving cycle, showing the importance of training process design and parameter tuning. In [56], deep Q-learning (DQN) was used. The main difference was, instead of storing the action-values in a table, a NN was used as the function approximator for the action-values. The ability to converge to near optimal performance was shown on several standard driving

cycles. Velocity predictors were used with Q-learning algorithms in [57], showing potential benefits on two recorded realistic driving cycles. In [58], the performance of Q-learning with tables and NNs were compared on standard driving cycles. Also, the adaptability of the DQN was tested on a test trip, which was a combination of four standard driving cycles. A technique called double Q-learning was utilized in [59]. It demonstrated that this technique was able to help with both optimization performance and convergence rate. All the above introduced literature tested the performance of the RL-based controller on trips that the algorithms were trained on. In [60], an actor-critic based DRL method called deep deterministic policy gradient (DDPG) was used. This work showed the trained DRL agent could perform well on unseen test trip by training the algorithm on one training cycle and testing it on one test cycle which was a combination of six standard driving cycles. Researchers in [61] wanted to design a RL-based online learning architecture that was able to adapt the controller with different driving conditions. However, the reported performance was still on the trip that the algorithm was trained on.

Another interesting idea to adapt the RL algorithm to different driving cycles were shown in [62] and [63]. The main idea was, first, obtain a RL-based control strategy offline based on a transition probability matrix (TPM) extracted from standard driving cycles or recorded previous driving cycles. When the control strategy was used online, the TPM was kept updated according to the ongoing trip information. The difference between the updated TPM and the offline TPM was monitored, and if the difference was greater than a predefined threshold, a new control strategy would be calculated according to the new TPM. The main underlying assumption was, the updated TPM which was constructed by

the information from both offline recorded driving cycles and online driving cycles up to that time step, could represent the future driving condition well. This assumption should be further investigated. Also, the quality and robustness of the online calculated RL-based strategy should be considered as there was no time testing it. There are specific applications where the RL-based controllers are tested on unseen trips that are not used to train the controller. In [64], the tabular Q-learning controller was trained on six trip data on a commuter route and tested on the seventh trip. In [65], a fixed bus route was used to perform the study and showed generalization abilities on new trips. For more literature and possible research directions with RL, please refer to the survey paper [66].

Although RB EMS's cannot achieve the highest fuel efficiency in theory, their full potential for in-use HEVs and EREVs has not been realized. Parameters in the predefined rules are usually tuned on standard driving cycles or combinations of standard driving cycles [67]. New efficient RB methods were developed on standard driving cycles [68][69]. Also, RB control strategies can be learned from results of OB methods on given driving cycles. For example, global optimal solutions on standard driving cycles were first calculated by DP, then rules were extracted from the optimal solution [70]-[72]. Results from standard driving cycles can provide valuable information about the performance and characteristics of an RB EMS. However, standard cycles cannot accurately represent the very large range of real driving vocations. In [73], a predictive RB blended model was developed. Fuel economy improvement was demonstrated compared to a conventional RB method in a simulation environment that included uncertainty in energy demand.

However, this method required distance and road type of the route in advance, which is not applicable in many real-world applications.

RB methods are compared to OB methods in the schematic given in Figure 1.3, illustrating the tradeoff between feasibility for human driven vehicles and fuel efficiency improvement. A perfect model has both high feasibility and results in high fuel efficiency improvement. RB EMS methods tend to result in lower fuel efficiency improvements but are feasible to implement, whereas OB methods can attain high fuel efficiency improvement but have low feasibility due to the reasons previously discussed. For example, DP is a typical strategy employed in OB methods and can achieve a theoretical highest fuel efficiency with known detailed future trip information and at high computational expense.

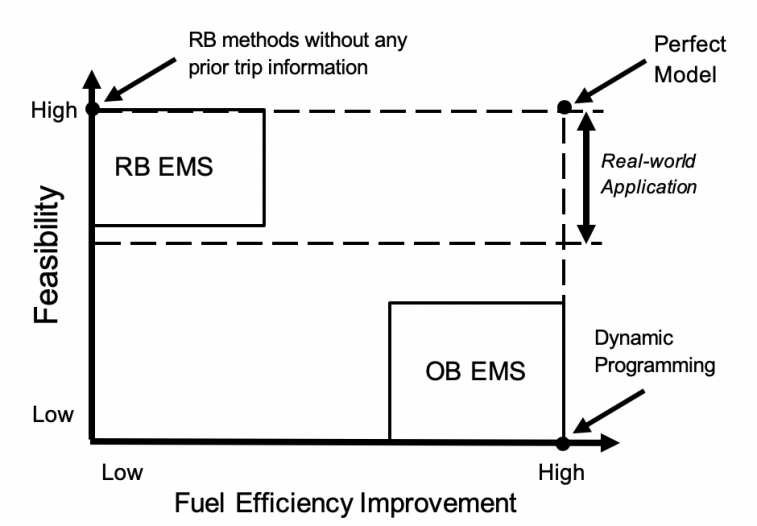


Figure 1.3: Illustration comparing rule-based (RB) EMS strategies to optimization-based (OB) strategies for human driven vehicle systems.

1.3. *Outline and Contribution*

The remainder of this dissertation is structured as follows:

In Chapter 2, the vehicle configuration, collected data and the in-use energy management strategy of the delivery fleet is introduced. The main difficulties in solving this problem are then discussed.

Chapter 3 begins by formalizing the EMS problem as a probability density estimation problem. Then, maximum likelihood and Bayesian Inference are introduced, which can be used to solve the density estimation problem. The advantage of Bayesian inference over maximum likelihood in this problem setting is discussed. Also, the derivation of the simplified vehicle model used in this work is presented. For the results part, the designing of the prior which is used in the Bayesian Inference is first introduced, then the real-world testing results are shown in detail.

Chapter 4 begins by introducing the main idea and key components of reinforcement learning as well as feedforward neural networks, which are the foundations of deep reinforcement learning (DRL) algorithms. Then, our energy management problem is formulated as a RL problem by describing the problem as a Markov decision process (MDP). To solve the MDP, two DRL algorithms with different characteristics are used. Simulation results and real-world testing results are summarized at last part of this chapter.

Chapter 5 consists of three parts. The first two parts are about two types of uncertainty that play important roles in real-world problem settings. First, risk-aware algorithms that are used to deal with the environmental uncertainty is shown. Second, an uncertainty-aware algorithm that is designed to capture the model uncertainty is discussed. The third

part of this chapter investigates the impacts of adversarial attacks on DRL-based energy management systems.

Finally, Chapter 6 summarizes the data-driven methods used in this dissertation and potential future research directions.

2. PROBLEM STATEMENT FOR EXTENDED RANGE ELECTRIC VEHICLE ENERGY MANAGEMENT STRATEGY

2.1. *Vehicle Configuration*

The configuration of the EREV powertrain studied in this work is given in Figure 2.1. The motive power of the vehicle is provided by an electric motor which uses stored energy from a high-capacity battery. The internal combustion engine (ICE) serves as a range extender (REx). It is used to charge the main battery using a generator. There is no mechanical connection between the REx output shaft and the vehicle drive shaft so that the ICE is completely decoupled from vehicle operation. Vehicle specifications are given in Table 2.1.

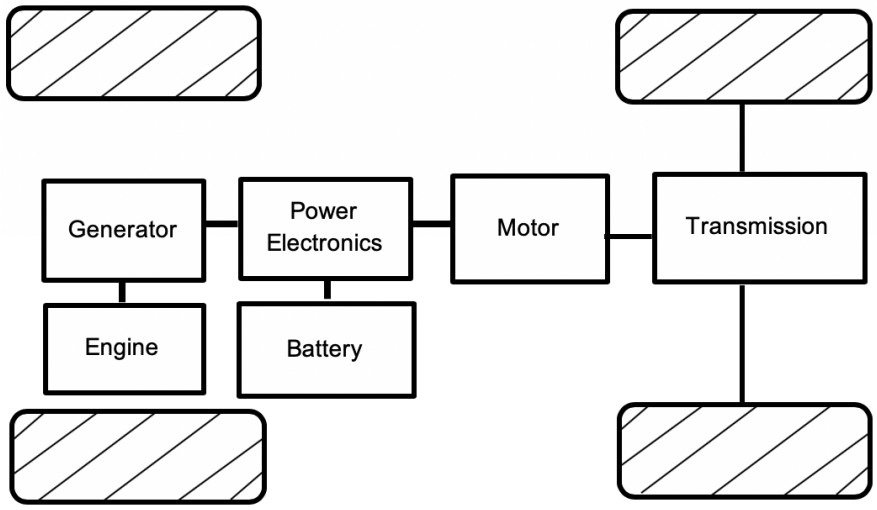


Figure 2.1: Powertrain configuration of the studied EREV.

Table 2.1: Vehicle Specifications

	Parameters	Value	Unit
Vehicle	Curb weight	5080	kg
	Typical weight	6800	kg
Motor	Maximum power	235	kW
	Maximum torque	2030	Nm
	Continuous power	130	kW
	Continuous torque	680	Nm
	Normal operating range	0-3500	rpm
Li-ion Battery	Usable capacity	56	kWh
	Voltage range	300-400	V
	Nominal voltage	385	V
Engine	Displacement	0.647	L
	Compression ratio	10.6	/
	Working power	11	kW

2.2. *Data Description and Preprocessing*

Data recording system and database introduction

On-board diagnostics measurement data were collected from in-use EREV delivery vehicles. Measured parameters included the vehicle's movement (e.g. velocity, distance), operating condition of powertrain components (e.g. voltage, current and state of charge of battery pack, speed and torque of the traction motor and on and off status of engine) and others (e.g. ambient temperature, humidity, altitude, heater condition and signal strength). While the vehicle was operating, 260 parameters per vehicle in total were recorded including the timestamp and the vehicle's location every five seconds.

Data from the vehicles were stored in a secure Oracle spatial database instance with support for geometry objects and spatial indexes. The database schema consisted of three main tables: Vehicle, Trip Summary and Drive Trip. The Vehicle table recorded properties of each vehicle, such as the make, model and year. Every record (row) in the Trip Summary table is a summary of a single delivery trip, which contains attributes such as the date, time, duration, distance and fuel use. Each record in this table is associated with a Drive Trip table. The Drive Trip table records all data of the vehicle during one delivery trip. Each row in the table describes the 260 parameters of a vehicle at one spatial location with a timestamp.

Data Preprocessing

Data quality is crucial to the accuracy of the vehicle simulation and developed algorithms. However, the raw data used in this work have three challenges; first, the resolution is low as the data are recorded every 5 seconds and significant changes in driving activity can occur in this time period. Second, there is occasional latency in the distance and velocity data due to variable signal strength. Third, there are missing values in the recoded data. The low-resolution problem makes the data piecewise constant which is not realistic as the velocity profile should be smooth in reality. The latency problem causes a stepped profile shape for both the velocity profile and distance profile. In addition, the recorded velocity value occasionally remains at zero while the distance data is increasing. In this condition, the zero-velocity value should be corrected by the distance data since the distance records are more reliable. To solve these problems, linear interpolation and a Gaussian filter were used to increase the resolution and smooth the

data [15]. The information in the distance profile is used to correct the zero-velocity problem iteratively.

The trip-level data preprocessing procedure used in the work is as follows:

Step 1

To fill in missing values, do zero-filling for the velocity profile and forward-filling for the distance profile;

Step 2

For both profiles, interpolate the five second data into one second data linearly;

Step 3

Use Gaussian filters to process the distance and velocity profile to get smoothed distance profile and velocity profile, the degree of smoothness is determined by σ_1 and σ_2 in the Gaussian filters; σ_1 and σ_2 are both set to be 3 to provide the degree of smoothness such that the acceleration calculated from the smoothed velocity profile and the velocity calculated from the smoothed distance profile are in normal vehicle running range.

Step 4

Calculate a new velocity profile from smoothed distance profile by second order finite difference method (for the first and last data point, velocity is zero):

$$v^{new}(t) = \frac{d(t + \Delta t) - d(t - \Delta t)}{2\Delta t}. \tag{2.1}$$

Step 5

Compare every point of the smoothed velocity profile and the corresponding point in the new velocity profile calculated from the smoothed distance profile, and update all

points that the value is 0 in the smoothed velocity profile and the value is not 0 at new velocity profile into the non-zero value multiplies by a factor ε, which initializes as 1;

Step 6

Calculate new distance profile by the smoothed and corrected velocity profile. If the final distance calculated has an error smaller than 500 meters, the preprocessing is finished. Otherwise, go back to step 5 and update ε according to the value of error until the stopping criteria is satisfied.

Since the actual velocity profile should be continuous and the velocity and acceleration cannot be too large, a Gaussian filter is used to infer the distance and velocity information within the five second data resolution. Also, the smoothing process significantly improves the data quality of the distance profile such that a new velocity profile can be found in Step 4. Without the Gaussian filter, the velocity calculated from distance profile would yield unrealistically high velocities at the points where distance changes. Also, at some data points, the acceleration calculated from the unsmoothed velocity profile would be too high. Step 5 corrects for wrong velocity values. However, the velocity value calculated from the smoothed distance profile is not accurate, requiring a factor to appropriately scale the velocity trajectory. This procedure refines the data on a trip level iteratively.

Figure 2.2 compares part of the raw velocity data and preprocessed data of one trip and shows the distance calculated by the raw data with zero-filling and by the preprocessed data. It can be seen that the distance calculated by the preprocessed data agrees with the raw distance data very well with a cumulated error less than 500 meters. Although the smoothing process in step 3 is similar to a standard filter process, a physical

check according to distance data is performed in the procedure, which differentiates the method from previous work [35].

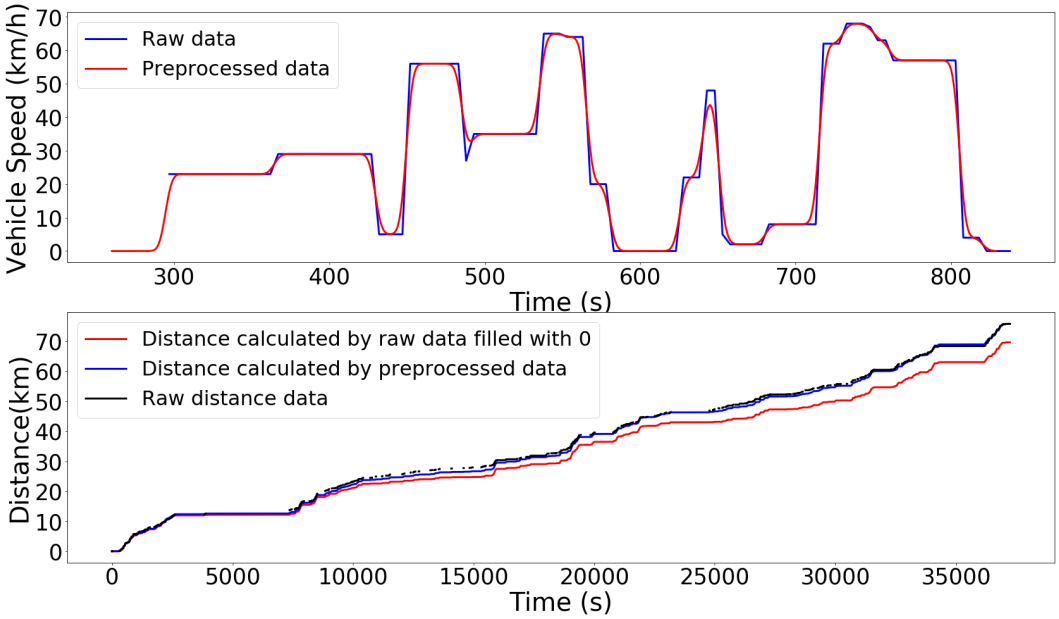


Figure 2.2: Comparison of raw data and preprocessed vehicle speed and distance data.

2.3. *Energy Management Strategy Introduction*

The developed rule-based EMS is thermostatic [16], meaning that the powertrain switches between two modes to optimize fuel consumption: REx-on or REx-off. The REx engine operates at one predetermined high efficiency speed and load condition to provide reliable, safe, and low noise operation; thus, the vehicle does not use a power-split strategy like in parallel HEV architectures. For trips that exceeds the all-electric-range (AER) of the EREV, the goal of the EMS is to optimize when, and how frequently to operate the REx engine to minimize the fuel use and achieve a target end battery state of charge (SOC) at the end of a trip, SOC_{tev}. For short trips, no fuel should be used, which means the vehicle operates as a pure electric vehicle (EV). Engine on-off control logic is based on

the measured real-time SOC, which is similar to the series powertrain configuration in [17] and [60]: the engine will be turned on if the actual SOC is lower than the SOC reference value (SOC_{ref}). An intuitive derivation of the calculation of SOC_{ref} is as follows.

The energy reference value can be expressed as:

$$E_{ref} = E_{total} - E_{consumption}, \tag{2.2}$$

where E_{total} represents the total energy, the vehicle needs to have to finish the goal, which comes from the battery and the engine. $E_{consumption}$ is the energy consumption on the given trip thus far. Consequently, E_{ref} represents how much energy need to be left to finish the goal.

To express the terms in the SOC expression, both sides are divided by the E_{total}:

$$\frac{E_{ref}}{E_{total}} = 100\% - \frac{E_{consumption}}{E_{total}}. \tag{2.3}$$

E_{total} consists of two parts: usable energy and unusable energy. The latter corresponds to the energy behind the target end battery state of charge SOC_{tev}:

$$E_{total} = E_{usable} + E_{unusable} = E_{usable} + SOC_{tev} \times E_{total}. \tag{2.4}$$

The usable energy is assumed to be able to run the vehicle for an expected total trip distance of d_e with expected energy intensity e_e, which is the energy use per unit distance (kW-hr/mile):

$$E_{usable} = e_e \times d_e. \tag{2.5}$$

The energy consumption term is expressed as the product of current distance and the energy intensity so far for the actual trip:

$$E_{consumption} = e_t \times d_t. \tag{2.6}$$

From (2.4) and (2.5), E_{total} has the form of:

$$E_{total} = \frac{e_e \times d_e}{100\% - SOC_{tev}}. \tag{2.7}$$

By substituting (2.6) and (2.7) into (2.3), and representing the SOC_{ref} as the ratio of E_{ref} and E_{total}:

$$SOC_{ref} = 100\% - (100\% - SOC_{tev}) \times \frac{e_t \times d_t}{e_e \times d_e}. \tag{2.8}$$

The desired SOC_{tev} in this work was set 10%. The energy intensity term of the equation can be expressed through a new variable defined as L_{set}:

$$L_{set} = \frac{e_e \times d_e}{e_t}. \tag{2.9}$$

Equation (2.8) now simplifies to (2.10) which is being used in the delivery fleet:

$$SOC_{ref} = 100\% \times \left(1 - 0.9\frac{d_t}{L_{set}}\right). \tag{2.10}$$

The SOC_{ref} decreases linearly with d_t while driving when the value of L_{set} is fixed. Furthermore, to reduce fuel consumption in short trips and prevent charging the battery too frequently, which will degrade its life, if the calculated SOC_{ref} is larger than 60%, it is set to 60% in this work. Consequently, for short trips and the beginning of long trips

where SOC never drops below 60%, the REx will not operate. SOC_{ref} represents how much battery energy is expected to be left when the vehicle has traveled for d_t given predetermined L_{set}. Similar SOC reference values are also used in [17][60] for blended mode control for an OB EMS. The single parameter in (2.10) to be optimized is L_{set}. At the beginning of a trip, the vehicle will first operate in charge depleting (CD) mode and will switch to a combination of CD mode and charge sustaining (CS) mode such that the actual SOC follows the reference value with the aid of the REx engine. This control strategy can be classified as a blended RB method as it is designed to make the SOC achieve the lowest value at the end of the trip set by SOC_{tev}. Figure 2.3 shows an example comparison of battery SOC and corresponding SOC_{ref} with two L_{set} settings. It can be observed that the lower L_{set} leads to a lower fuel consumption and final battery state of charge.

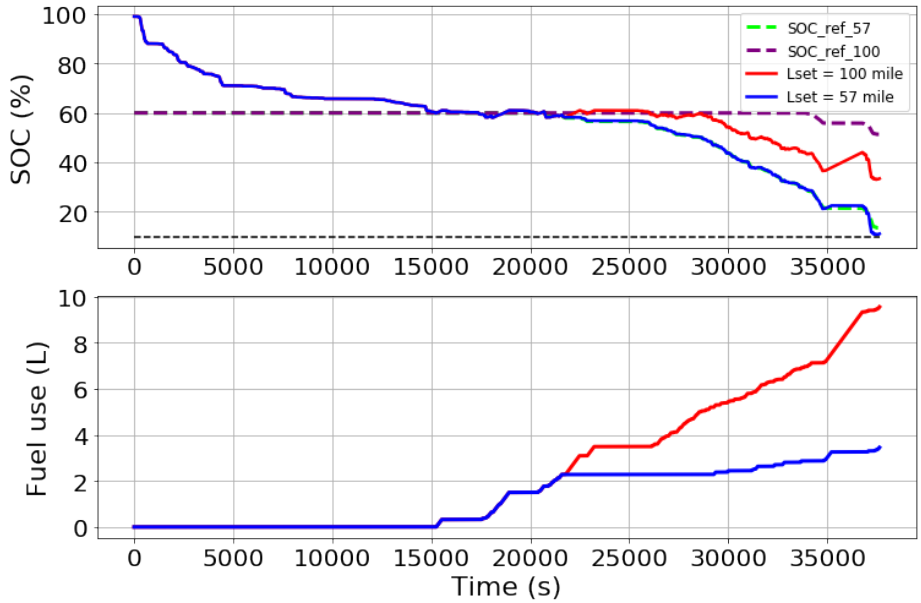


Figure 2.3: Comparison of SOC_{ref} and real-time SOC (upper) and fuel use (lower) with different L_{set} settings for one measured delivery trip.

2.4. *Problem to Solve*

To reduce fuel consumption on a future trip, it is evident that L_{set} should be preprogrammed according to trip distance and energy intensity. To save cost, the vehicle should charge at the depot using grid electricity at night, minimizing fuel consumption and on-road emissions. Setting L_{set} to minimize REx operation is the key problem to solve in this work. Ideally, if the trip distance and energy intensity of the trip are known in advance, the L_{set} can be preprogrammed according to these two values so that the vehicle finishes the trip with a 10% SOC value for long trips and operates as a pure EV for short trips. However, L_{set} is difficult to determine for at least two reasons. First, it is difficult to estimate the trip distance accurately *a priori*. Vehicles in different delivery areas have very different distributions of trip distances day-to-day. Also, for an individual vehicle, the trip distances in actual routes vary from the scheduled distance and differ day-to-day based on delivery demand, even though the vehicles might traverse the same region each day. Second, it is difficult to estimate energy intensity before a given trip as it relates to many factors like traffic condition, weather and the behavior of the driver. In Figure 2.4, the distribution of distance and energy intensity of an example EREV is shown. In Figure 2.5, four historical GPS trajectories of the same example delivery vehicle is used to show the similarities and differences of trajectories of the same vehicle. Due to the reasons discussed above, the values of L_{set} for the real-world delivery fleet are set high (100 miles) to prevent potential delays of delivery tasks or dangerous conditions like running out of battery during highway driving. Figure 2.6 shows a typical trip with high L_{set} setting: the

final battery SOC is much higher than 10%, which means some or all of the fuel consumed during the delivery trip was unnecessary.

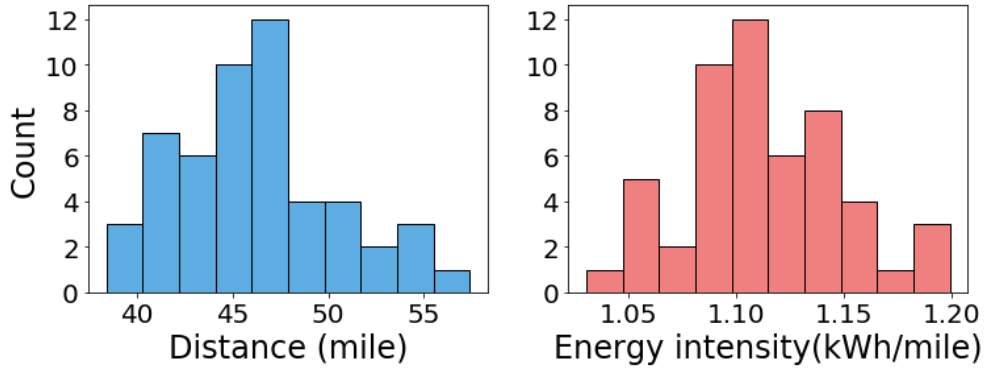
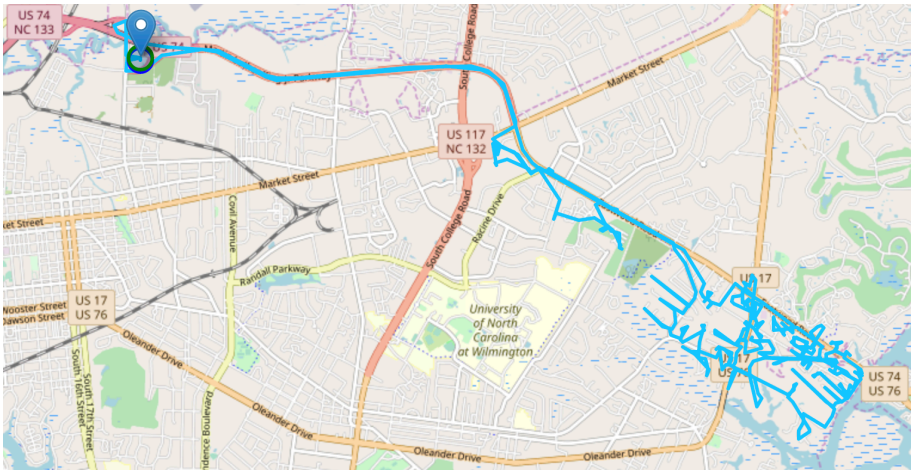
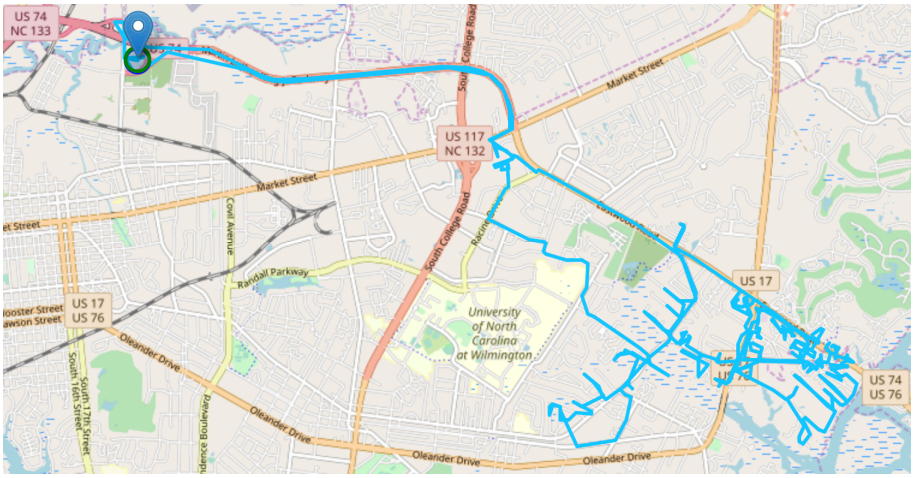


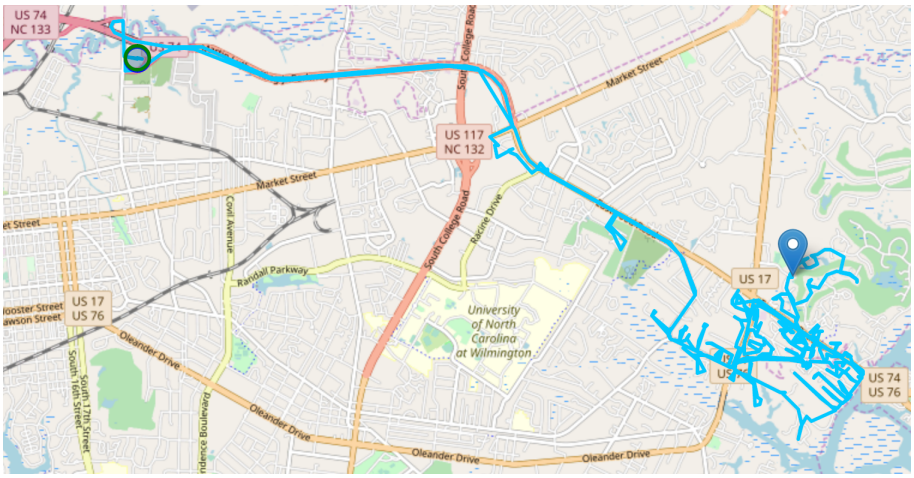
Figure 2.4 Distribution of distance and energy intensity for an example EREV.



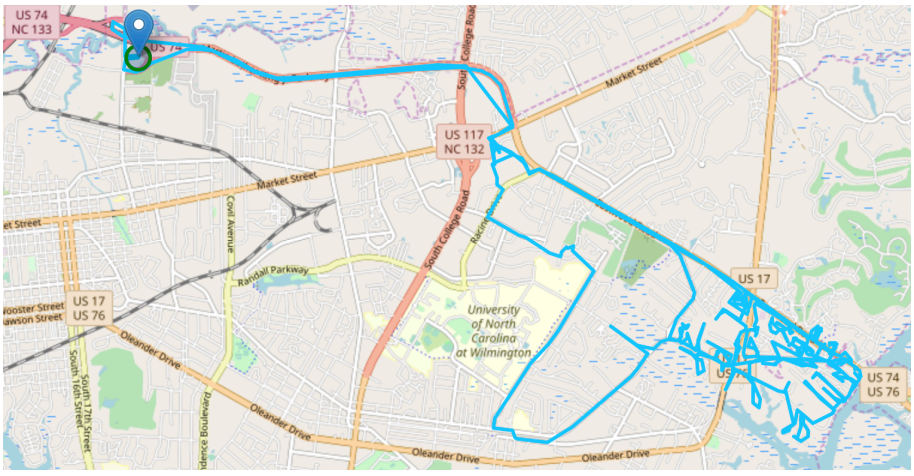
(a)



(b)



(c)



(d)

Figure 2.5: Four historical GPS trajectories of an example delivery vehicle.

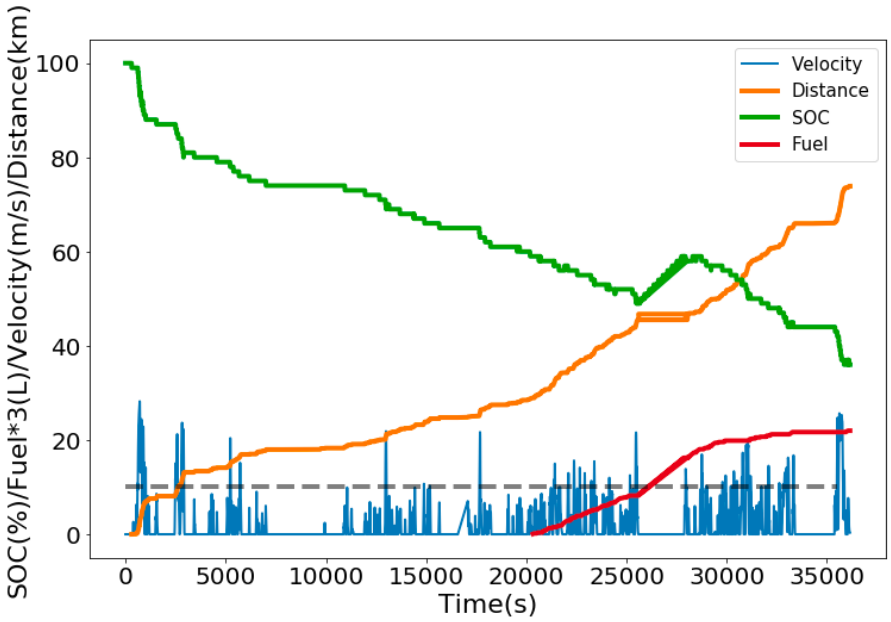


Figure 2.6: A typical delivery trip data with high L_{set} setting.

3. BAYESIAN INFERENCE BASED ENERGY MANAGEMENT

3.1. *Probability Density Estimation Problem Formulation*

For an individual vehicle, there exists a best L_{set} that achieves the highest fuel efficiency without compromising the drivability for each trip. This value can be calculated after the trip as the full velocity profile is available. It is a natural idea to estimate a better L_{set} for the next trip of the vehicle by considering all the previous best L_{set} of the vehicle. However, it is not straightforward to determine how to estimate a proper L_{set} for the next trip as there is uncertainty in future trips. To deal with the uncertainty, the distribution of the best L_{set} of each individual vehicle is modeled as a Gaussian distribution as shown in Figure 3.1. If the distribution is known, the L_{set} is calculated through the cumulative density function (CDF). As shown in Figure 3.1, the L_{set} for the next trip of the vehicle is the value where the CDF equals 0.99; i.e., the L_{set} is the 99^{th} percentile of the distribution. This conservative calculation method ensures that the probability of running out of battery is very low if the real data follows the Gaussian assumption. Therefore, our goal is to estimate the parameters in the Gaussian distribution using the available best L_{set} data for each individual vehicle, which can be considered as a probability density estimation problem in the area of pattern recognition and machine learning.

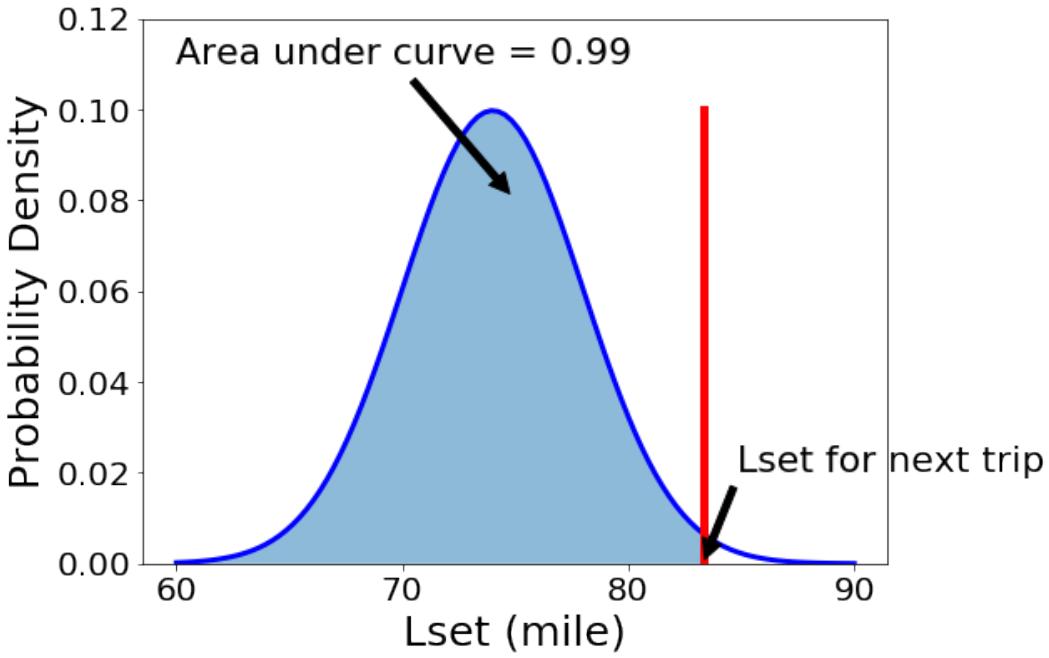


Figure 3.1: Illustration of how to determine the L_{set} from a known Gaussian distribution of one vehicle.

3.2. *Maximum Likelihood and Bayesian Inference*

For a 1-dimensional Gaussian distribution, there are two parameters to be determined: mean μ and variance σ^2. The naïve and straightforward estimation method is maximizing likelihood using the available data. The distribution of the best L_{set} associated with each vehicle is assumed to follow a Gaussian distribution with unknown mean and unknown precision: $p(L_{set}) \sim N(\mu, \lambda)$ where λ is the precision defined as the reciprocal of variance: $\lambda = \frac{1}{\sigma^2}$. Using precision instead of the variance facilitates the derivation of the Bayesian inference method.

To simplify the notation, $L_{set}^{[n]}$ and $\hat{L}_{set}^{[n]}$ represent actual best L_{set} and predicted L_{set} for the nth trip. Given historical data from N trips, the likelihood can be written in the form given in (3.1) assuming the data are independent and identically distributed (i.i.d.).

$$p\left(L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]} \middle| \mu, \lambda\right) = \frac{\lambda^{\frac{N}{2}}}{(2\pi)^{\frac{N}{2}}} \exp\left[-\frac{\lambda}{2}\sum_{i=1}^{N}\left(L_{set}^{[i]} - \mu\right)^2\right]. \tag{3.1}$$

Maximizing the likelihood with respect to the unknown parameters μ and λ gives:

$$\hat{\mu}_{mle} = \frac{1}{N}\sum_{n=1}^{N} L_{set}^{[n]},$$

$$\hat{\lambda}_{mle} = \frac{1}{\hat{\sigma}_{mle}^2} = \frac{1}{\frac{1}{N}\sum_{n=1}^{N}\left(L_{set}^{[n]} - \hat{\mu}_{mle}\right)^2}. \tag{3.2}$$

This method works well and gives good estimates if the amount of data is large. However, for new vehicles or for vehicles driving new route profiles, the number of trips is very small or zero so that it is difficult to have a good estimation of the distribution, and the statistical strength of such a prediction will be low. To deal with this problem, the parameters in the Gaussian distribution are estimated using a Bayesian algorithm [74]. The parameters are determined by both data and prior knowledge. Every time new trip data is available, distribution parameters are updated adaptively. Once the parameters are updated, the L_{set} can be calculated conservatively by the CDF of the posterior predictive distribution. The following derivation mainly follows [75] and more detailed information can be found in [76]-[78].

If $\hat{L}_{set}$ is calculated using the distribution estimated only on the historical data by maximizing the likelihood (3.2) yielding point estimates of μ and λ, when the size of data

is small or there is no data, the calculated $\hat{L}_{set}$ will be highly unstable, leading to potentially undesirable vehicle performance. To solve this problem, a prior is introduced by considering the distribution of μ and λ to make the model more conservative.

With the introduced prior distribution of μ and λ: $p(\mu, \lambda)$, the posterior distribution of μ and λ: $p\left(\mu, \lambda \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right)$ can be determined by incorporating the historical data. As $\text{posterior} \propto \text{prior} \times \text{likelihood}$ [78], the form of posterior is given by (3.3), which models the distribution of μ and λ instead of just providing point estimates, i.e., instead of estimating μ and λ as single values, the distribution of them are modeled.

$$p\left(\mu, \lambda \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right) \propto p(\mu, \lambda) \cdot p\left(L_{set}^{[1]}, L_{set}^{[2]} \dots \dots L_{set}^{[N]} \middle| \mu, \lambda\right). \tag{3.3}$$

By introducing a prior distribution, μ and λ is estimated based on both the information from data and prior knowledge. This can give us a more conservative estimation for small N. The concept of conjugate prior from Bayesian probability theory is used, which considerably simplifies the analysis and makes the calculation of posterior distribution tractable. If a prior distribution is conjugate to the likelihood function of a given distribution, the posterior distribution will have the same form of distribution as the prior [75]-[78]. The conjugate prior for a Gaussian distribution with unknown mean and unknown precision is the Normal-Gamma distribution [76]: $p(\mu, \lambda) \sim Normal - Gamma(\mu_0, \kappa_0, a_0, b_0)$.

There are 4 parameters in the prior. μ_0 is the prior estimate of the mean. κ_0 is the size of pseudo samples from which μ_0 is estimated. a_0 is half the size of pseudo samples from which the prior precision is estimated. b_0 represents the prior estimate of the precision.

With use of the conjugate prior, the posterior distribution is also Normal-Gamma: $p\left(\mu, \lambda | L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right) \sim Normal - Gamma(\mu_N, \kappa_N, a_N, b_N)$, where:

$$\mu_N = \frac{\kappa_0 \mu_0 + Nm}{\kappa_N},$$

$$\kappa_N = \kappa_0 + N,$$

$$a_N = \frac{1}{2}(2a_0 + N),$$

$$b_N = b_0 + \frac{N}{2}s^2 + \frac{\kappa_0 N}{2\kappa_N}(m - \mu_0)^2. \tag{3.4}$$

m and s^2 are the sample mean and variance of best L_{set} of available trips. As can be observed in (3.4), the posterior mean μ_N is the weighted sum of the prior estimate of the mean and the sample mean of the available data. Posterior precision is also determined by the prior estimate and the statistic of the data. The estimate of the posterior mean and precision are based on κ_N and $2a_N$ samples respectively, which are both the sum of the number of real data and the number of pseudo samples. The parameters become the prior if the number of data N is 0, e.g. $\mu_N = \mu_0$ if $N = 0$ according to (3.4). It can be observed that the influence of the prior information decreases as the number of available data increases.

After the posterior distribution of μ and λ are obtained, the next $\hat{L}_{set}$ can be predicted by using $p(\hat{L}_{set}|\mu, \lambda)$, considering all possible values of μ and λ according to the posterior distribution $p\left(\mu, \lambda \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right)$ by integrating over μ and λ:

$$p\left(\hat{L}_{set} \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right) = \iint p(\hat{L}_{set}|\mu, \lambda) p\left(\mu, \lambda \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right) d\mu d\lambda$$

$$\sim t_{2a_N}\left(\mu_N, \frac{b_N(\kappa_N+1)}{a_N\kappa_N}\right). \tag{3.5}$$

Intuitively, this is a weighted sum of infinite number of Gaussians, and the weight is determined by the posterior distribution. Given the prior and historical data, the posterior predictive model for the next $\hat{L}_{set}$ is a t-distribution after integration. To be clear, L_{set} is assumed to follow a Gaussian distribution. However, the μ and λ are unknown such that all possibilities of μ and λ are considered by the posterior distribution $p\left(\mu, \lambda \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots L_{set}^{[N]}\right)$ and integrated over μ and λ. The t-distribution is like an infinite sum of Gaussians [75][76].

Robustness is one of the main characteristics of a t-distribution. It has longer 'tails' than Gaussian distribution, which means the position and shape of the t-distribution is less sensitive to outliers [76][77]. This property is an advantageous in this application as it can prevent the shape and position of the distribution from being influenced largely by some very short trips.

After the parameters in the prior are determined, the posterior predictive model $p\left(\hat{L}_{set} \middle| L_{set}^{[1]}, L_{set}^{[2]} \dots \dots L_{set}^{[N]}\right)$ is determined. This t-distribution is used to calculate the $\hat{L}_{set}$ conservatively for the next trip. After the data of the next trip is observed, the previous posterior becomes the prior and the distribution is updated according to the new observation.

The procedure for calculating $\hat{L}_{set}$ and updating parameters is described as follows:

Initialization step

The initial t-distribution is $t_{2a_0}(\mu_0, \frac{b_0(\kappa_0+1)}{a_0\kappa_0})$, which is completely determined by the prior as there is no available trip information ($N = 0$), leading to:

$$\mu_N = \frac{\kappa_0\mu_0 + Nm}{\kappa_N} = \mu_0,$$

$$\kappa_N = \kappa_0 + N = \kappa_0,$$

$$a_N = \frac{1}{2}(2a_0 + N) = a_0,$$

$$b_N = b_0 + \frac{N}{2}s^2 + \frac{\kappa_0 N}{2\kappa_N}(m - \mu_0)^2 = b_0. \tag{3.6}$$

Prediction step

The prediction step is based on the CDF of the t-distribution as shown in Figure 3.1. The value of the CDF evaluated at $\hat{L}_{set}$, is the probability that the next actual best L_{set} will take a value less than or equal to the predicted $\hat{L}_{set}$:

$$CDF_{L_{set}}(\hat{L}_{set}) = P(L_{set} \leq \hat{L}_{set}). \tag{3.7}$$

$\hat{L}_{set}$ is determined by setting the CDF = 0.99, which means L_{set} will be smaller than $\hat{L}_{set}$ with a probability of 0.99 under the assumption. From this point, it can be seen that, the calculated $\hat{L}_{set}$ will be higher than the actual ideal L_{set} by a margin in most trips. For real-world driving, low $\hat{L}_{set}$ leading to a very low SOC during a trip should be avoided to a high confidence level even at the expense of smaller improvement in fuel economy.

Update step

After a new trip is observed, the parameters in the prior are updated by the parameters in the posterior; i.e., after new data is recorded, the previous posterior information becomes prior for the new information:

$$\mu_0^{new} = \mu_N^{old},$$
$$\kappa_0^{new} = \kappa_N^{old},$$
$$a_0^{new} = a_N^{old},$$
$$b_0^{new} = b_N^{old}. \tag{3.8}$$

The parameters in the posterior are then updated according to the new data L_{set} and the updated prior:

$$\kappa_N^{new} = \kappa_0^{new} + 1,$$
$$\mu_N^{new} = \frac{\kappa_0^{new}\mu_0^{new} + L_{set}}{\kappa_N^{new}},$$
$$a_N^{new} = \frac{1}{2}(2a_0^{new} + 1),$$
$$b_N^{new} = b_0^{new} + \frac{\kappa_0^{new}}{2\kappa_N^{new}}(L_{set} - \mu_0^{new})^2. \tag{3.9}$$

After updating the parameters in the t-distribution, $\hat{L}_{set}$ for the next trip can be calculated by the prediction step.

Table 3.1: Vehicle model parameters

Symbol	Parameter	Unit
F_d	Total force demand of the vehicle	N
c_{rr}	Coefficient of rolling resistance	/
c_d	Coefficient of air resistance	/
P_d	Total power demand of the vehicle	W
P_b	Battery power	W
P_e	Engine power	W
η_{btw}	Efficiency from battery to wheel	/
η_{etw}	Efficiency from engine to wheel	/
ρ	Air density	kg/m^3
A	Frontal area	m^2
m	Total vehicle mass	kg
g	Gravity constant	N/kg
a	Acceleration	m/s^2
v	Velocity	m/s
t	Time	s
θ	Road slope	rad
V_{oc}	Open circuit voltage	V
I	Current	A
R_0	Battery internal resistance	ohm
Q	Battery capacity	Ah
f	Cumulated fuel use	L
C_c	Battery charging rate	%/s
C_f	Fuel consumption rate	L/s

3.3. *Vehicle Model*

A vehicle model is necessary as it is used to calculate the best L_{set} for each trip and guides updates to the Bayesian algorithm. Also, fuel efficiency improvement is estimated by calculating the fuel use for different L_{set} values.

Vehicle dynamics

A simplified vehicle model is derived here and used in this work. The used notation is summarized in Table 3.1. The vehicle force demand can be written in the following form:

$$F_d = ma + c_{rr}mgcos(\theta) + \frac{1}{2}c_d A\rho v^2 + mgsin(\theta). \quad (3.10)$$

Ignoring the road grade, it can be simplified to:

$$F_d = ma + c_{rr}mg + \frac{1}{2}c_d A\rho v^2. \quad (3.11)$$

Since power can be calculated as $P = Fv$, the power demand can be written as:

$$P_d = mav + c_{rr}mgv + \frac{1}{2}c_d A\rho v^3. \quad (3.12)$$

The power in the case of an EREV is provided solely by the motor, which uses energy from the battery and the engine:

$$P_d = P_b\eta_{btw} + P_e\eta_{etw}. \quad (3.13)$$

From (3.12) and (3.13), the battery power can be written as:

$$P_b = \frac{\left(c_{rr}mgv + \frac{1}{2}c_d A\rho v^3 + mav\right)}{\eta_{btw}} - \frac{P_e\eta_{etw}}{\eta_{btw}}. \quad (3.14)$$

By assuming $\eta_{btw}, \eta_{etw}, m, g, c_{rr}, A, c_w, \rho$ are all constants and the fact P_e is a constant (ignoring the transition process from on to off and off to on), this equation can be rewritten including the dependence on time t as:

$$P_b(t) = Av(t) + Bv^3(t) + Ca(t)v(t) - D, \quad (3.15)$$

where A, B, C and D are combinations of constants:

$$A = \frac{c_{rr}mg}{\eta_{btw}},$$

$$B = \frac{\frac{1}{2} c_d A \rho}{\eta_{btw}},$$

$$C = \frac{m}{\eta_{btw}},$$

$$D = \frac{P_e \eta_{etw}}{\eta_{btw}}. \tag{3.16}$$

Battery model

A simplified battery model is used to model the battery pack:

$$P_b(t) = V_{oc}(s) I(t) - R_0(s) I^2(t), \tag{3.17}$$

$V_{oc}(s)$ and $R_0(s)$ depends on SOC which is denoted as s. The derivative of s is proportional to current at the battery terminals:

$$\dot{s}(t) = -\frac{1}{Q} I(t). \tag{3.18}$$

If the current is found from the battery power equation and substituted into the above equation (3.17) results:

$$\dot{s}(t) = -\frac{V_{oc}(s) - \sqrt{{V_{oc}}^2(s) - 4R_0(s)P_b(t)}}{2R_0(s)Q}, \tag{3.19}$$

$V_{oc}(s)$ can be modeled as a piecewise linear function of the SOC and $R_0(s)$ can be modeled as a constant $R_0(s) = R_0$. By combining (3.15) and (3.19), velocity profiles can be used as an input to calculate the SOC profile step by step given the initial SOC:

$$s(t + \Delta t) = s(t) + \dot{s}(t)\Delta t. \tag{3.20}$$

If the vehicle is stopped and the engine is on, the SOC update is simply:

$$s(t + \Delta t) = s(t) + C_c \Delta t, \tag{3.21}$$

where $\Delta t = 1s$, which means the step size is 1 second.

Engine Model

As the engine only works at a predefined fixed condition, the fuel rate and engine charging power are both constant. The transition process from off to on and on to off is ignored. Therefore, when the engine is turned on, the fuel use can be calculated as:

$$f(t + \Delta t) = f(t) + C_{fuel\ rate} \Delta t. \tag{3.22}$$

Validation

The accuracy of the vehicle model is very important for the developed framework. As the engine on/off control logic is based on the SOC value, validation of the model is based on the SOC curve. Starting with the developed vehicle model, parameters were calibrated for different vehicles on each delivery area using several trips. After calibration, the model performed consistently on the other trips for the same vehicle. Errors arose from simplifications in the model including ignoring wind speed and road grade, and assuming constant vehicle components efficiencies. Also, noisy and low-resolution raw data introduced error even after the preprocessing process. Furthermore, the SOC value in the raw data itself contained some level of error as the SOC value was not measured directly. Some degree of error is inevitable in all vehicle measurement datasets. As an example, raw SOC data and simulated SOC data for one EREV are shown in Figure 3.2 for four actual trips. For each simulated trip in this study, the mean relative error of the calculated SOC curve is less than 5% compared with recorded SOC curve. Considering raw data

quality and model complexity as well as the goal of determining fuel consumption under different L_{set}, the accuracy of the model is deemed adequate.

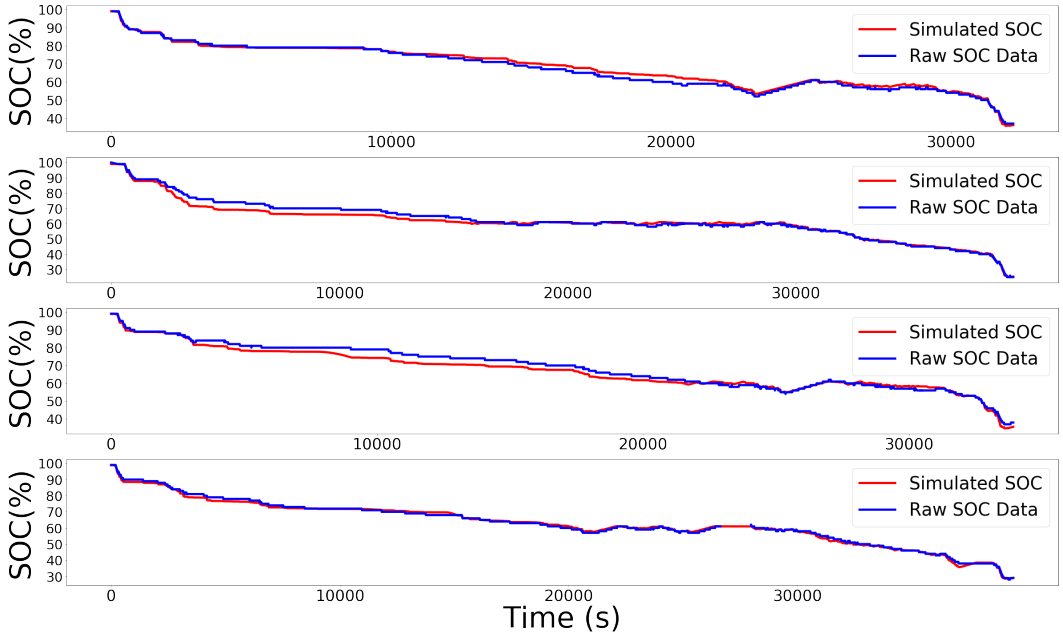


Figure 3.2: Validation of the vehicle model by comparing the raw SOC data and simulated SOC data.

3.4. *Designing of the prior*

In this section, a prior is designed for use in the Bayesian inference algorithm by determining the parameters $\mu_0, \kappa_0, a_0, b_0$ from the collected vehicle data so that the $\hat{L}_{set}$ calculated is conservative, especially for small N or when no data are available for new vehicles. The origin $\hat{L}_{set}$ can be used as an initial condition (100 miles). The parameters are determined using data from 78 vehicles with more than 12,000 accumulated trips in total. The 78 vehicles in different delivery areas have various mean distance and standard deviation as shown in Figure 3.3.

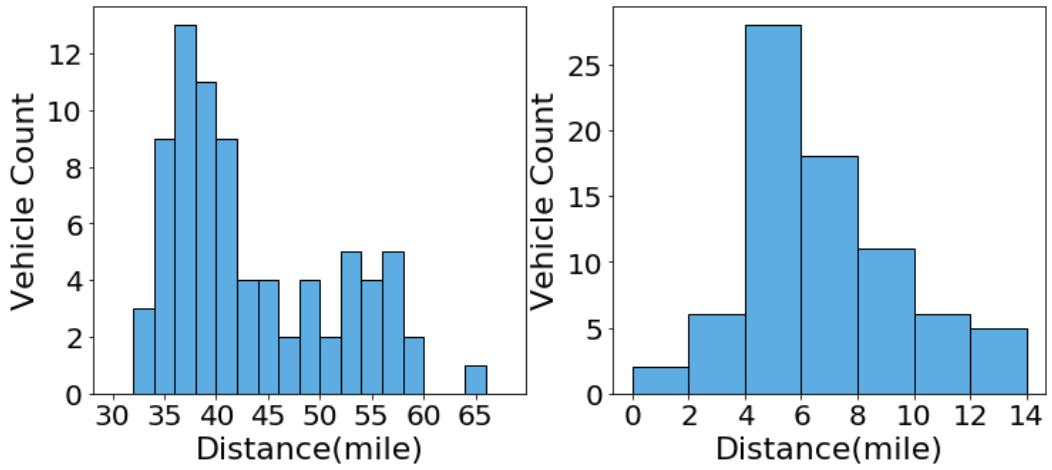


Figure 3.3: Distribution of the mean (left) and standard deviation (right) of trip distance for all delivery vehicles used to design the prior from the collected dataset.

The chosen parameters should satisfy two conditions: first, the corresponding initial t-distribution should yield a $\hat{L}_{set}$ that is about 100 miles, which means if there is no data available, the original setting will be used; second, the series of updated $\hat{L}_{set}$ values should be no lower than the actual series of L_{set} for all historical trips for each vehicle. The difference between the calculated $\hat{L}_{set}$ curve and the actual best L_{set} curve needs to be minimized to reduce fuel use under the constraint of $\hat{L}_{set}$ always no lower than best L_{set}. Considering this goal, the parameters were varied to reduce the gap between the two curves until the two curves touched for one of the vehicles. The final values of the parameters in the prior are shown in Table 3.2.

In Table 3.2, the prior mean is 74 and is estimated from 5 pseudo samples. The prior precision is 0.01, which is estimated using 50 pseudo samples (the relation to the parameters in the prior is $\kappa_0 = n_{\mu_0}, a_0 = n_{\lambda_0}/2, b_0 = n_{\lambda_0}/2\lambda_0$).

Figure 3.4 shows the calculated $\hat{L}_{set}$ curve and best L_{set} curve for four vehicles, which are used to design the prior. Vehicle A and B represent about 75% of other vehicles; the calculated $\hat{L}_{set}$ is clearly higher than the actual L_{set} for all trips. Vehicle C and D illustrate conditions where the calculated $\hat{L}_{set}$ and the actual L_{set} are very close, which represents about 25% of all vehicles. It can be observed that there is a margin between these two curves. As the margin gets smaller, more fuel can be saved as the prediction becomes closer to the best value. However, the risk of underestimating the L_{set} also increases. The

Table 3.2: Bayesian model parameters

Parameter	Value
μ_0	74
n_{μ_0}	5
λ_0	0.01
n_{λ_0}	50

designed prior minimizes the gap while avoiding the condition of calculated $\hat{L}_{set}$ being smaller than the actual L_{set} for more than 12,000 trips of the 78 vehicles.

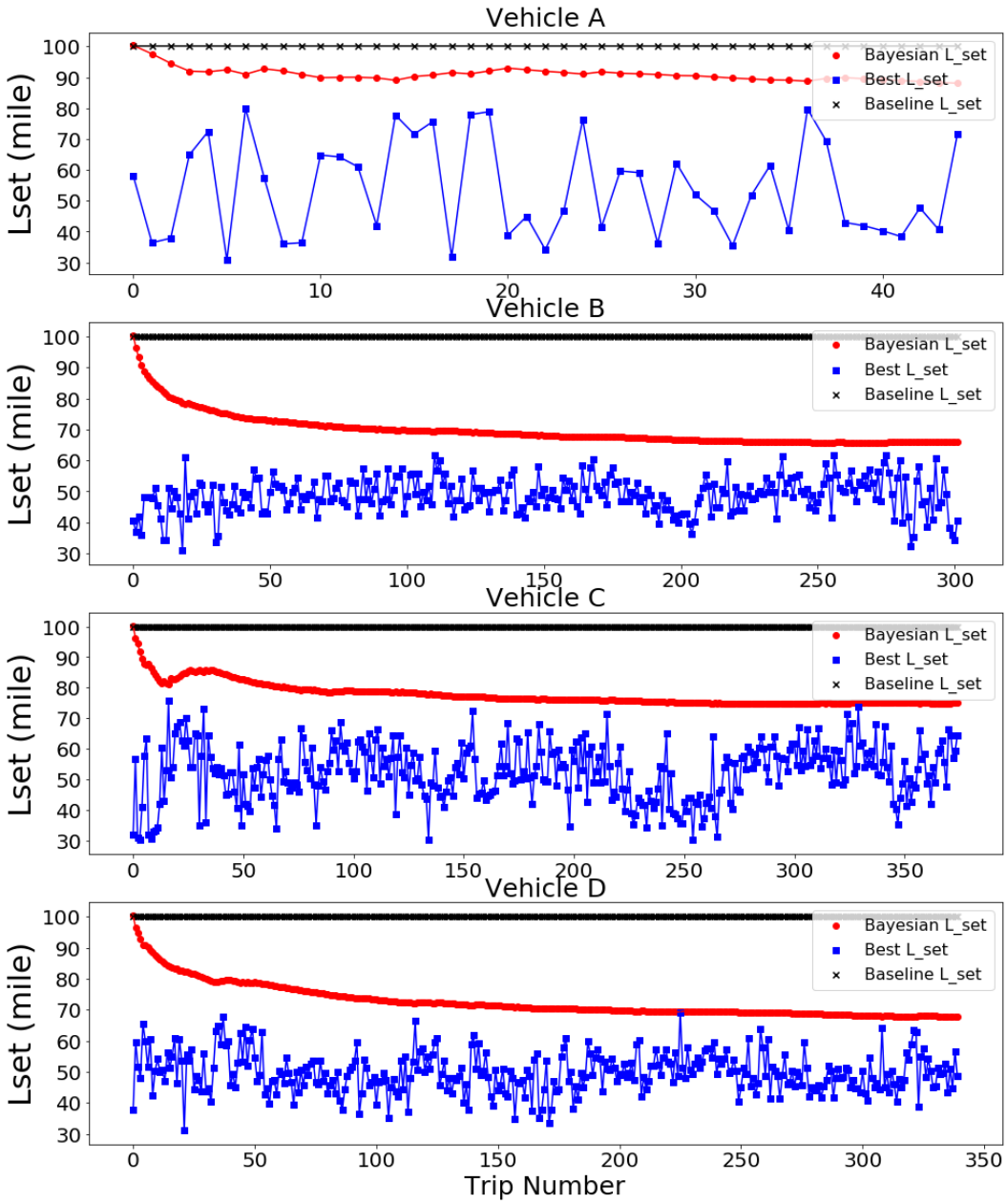


Figure 3.4: Bayesian L_{set} and the best L_{set} curves for four vehicles.

The initial posterior predictive distribution only determined by the prior and final distribution using all trip data of vehicle D is shown in Figure 3.5. Also, the actual best L_{set} data is shown in the form of normalized histogram. It can be shown that the real best L_{set} distribution can be represented by the calculated distribution well after enough data

is available. Several intermediate distributions are shown in Figure 3.6. It can be observed that as the number of data increases, the estimated distribution can better represent the real data distribution. When the number of data is small, the updated distribution does not significantly deviate from the prior, guaranteeing a stable and conservative $\hat{L}_{set}$. For example, even though the first best L_{set} data of vehicle D is found to be less than 40 miles, which is much lower than the baseline 100 miles, the $\hat{L}_{set}$ calculated from the updated t-distribution is about 96 miles as there is only one historical trip available.

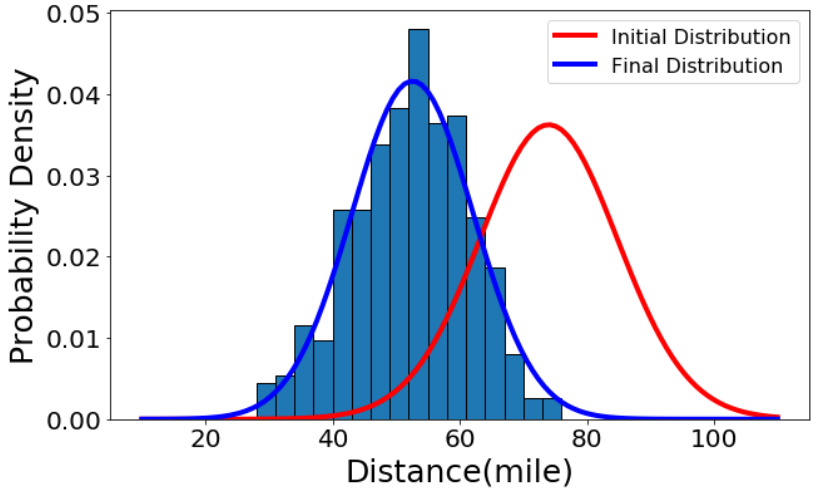


Figure 3.5: Initial and final t-distribution and the normalized actual data.

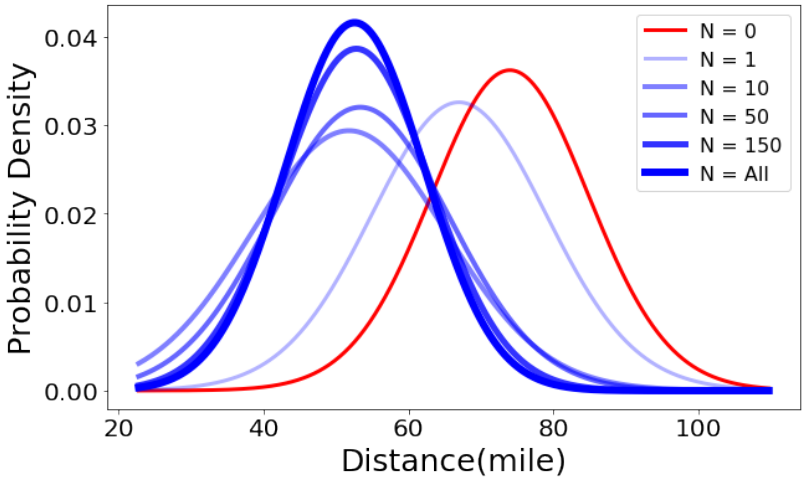


Figure 3.6: T-distributions with different number of data.

3.5. *Results and Discussion*

The fuel efficiency improvement achieved by the EMS framework is quantified by fuel use and the mile per gallon equivalent (MPGe). MPGe is estimated by the equation [79]:

$$MPGe = \frac{distance(mile)}{fuel\ use(gallon) + \frac{electic\ energy\ use(kwh)}{33.7\left(\frac{kwh}{gallon}\right)}}. \tag{3.23}$$

In this paper, fuel efficiency improvement is demonstrated on 13 vehicles with 155 real-world delivery trips in total. The 13 vehicles are relative new vehicles and are not used to design the prior. Figure 3.7 illustrates the components of the developed framework.

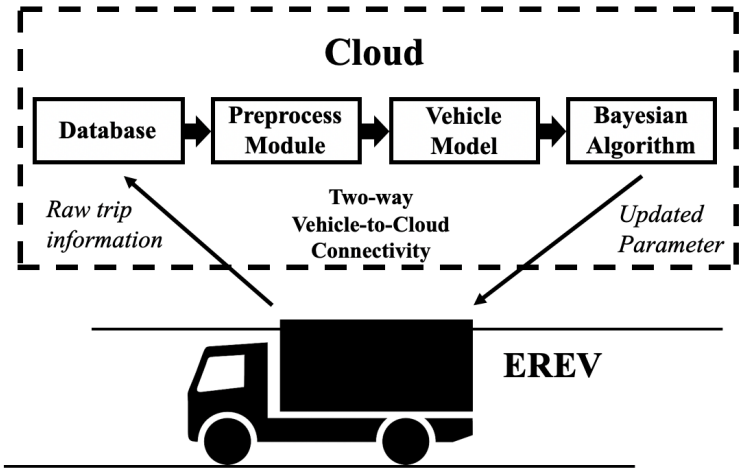


Figure 3.7: Components of the developed Bayesian inference framework.

In Figure 3.8, the actual best L_{set} and Bayesian $\hat{L}_{set}$ is shown for two of the selected demonstration vehicles. It can be observed that the update is conservative. Since the number of data points is small, the margin between Bayesian $\hat{L}_{set}$ and actual best L_{set} is significant at the beginning and gradually decrease as the number of data increases.

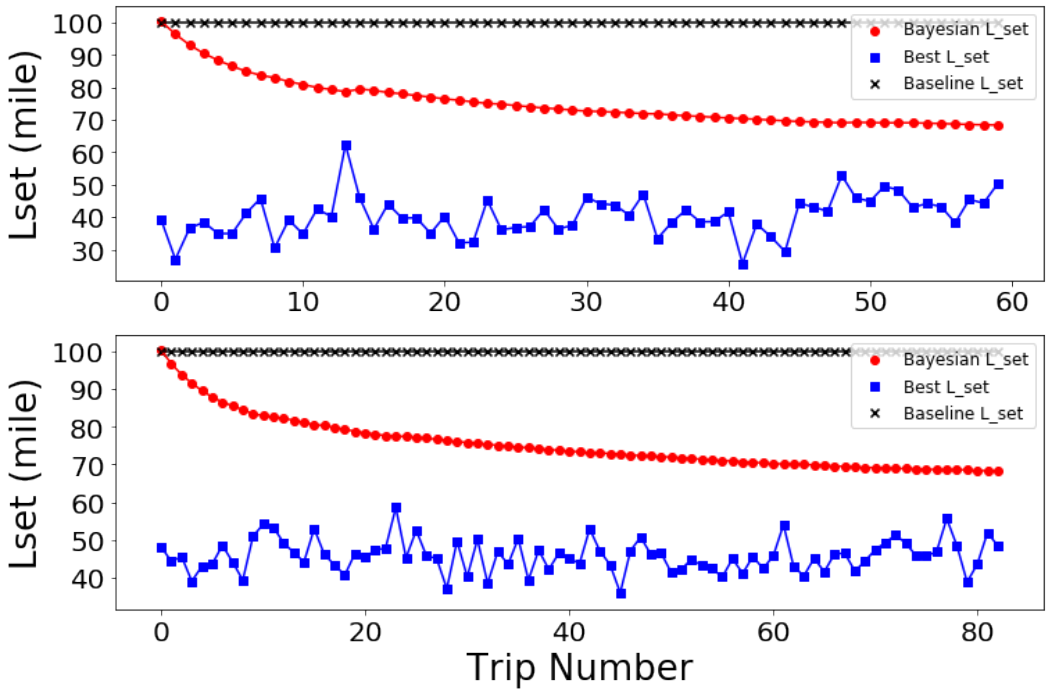


Figure 3.8: Bayesian L_{set}, best L_{set} and baseline L_{set} curves for vehicle T11 (upper) and T12 (lower).

Figure 3.9 illustrates how fuel is saved over the course of a real delivery trip. First, it can be seen that the simulated SOC and the actual SOC are closely aligned. Second, as the $\hat{L}_{set}$ is lowered by the developed method, the actual SOC reference value is lower than the baseline $\hat{L}_{set}$ case (100 miles). Consequently, the actual SOC follows the reference value, consuming less fuel at the last part of the trip compared with the unchanged one. Also, the minimal SOC during the trip is greater than 10%. A comparison of engine operation frequency for the last part of the same trip is shown in Figure 3.10. For clarity, the y-axis has two "on" positions, one for the Bayesian EMS and one for the baseline condition. It is clear from the figure that the REx engine operated much less frequently for the Bayesian EMS case, thus consuming less fuel.

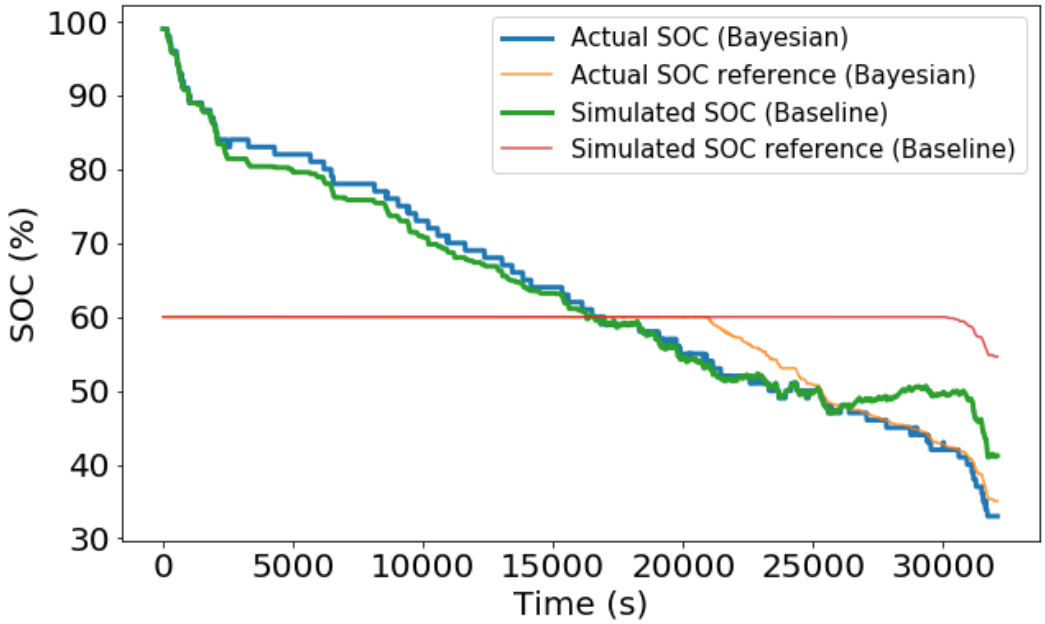


Figure 3.9: Illustration of how fuel is saved in a real delivery trip.

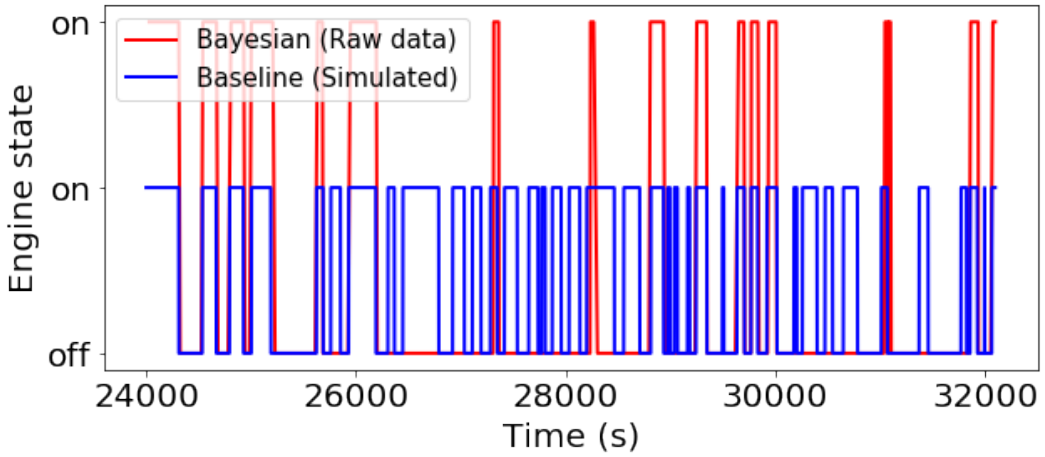


Figure 3.10: Engine operation frequency for Bayesian EMS and baseline case.

Figure 3.12 and Figure 3.13 show the detailed fuel consumption and MPGe of test delivery trips for two vehicles. It can be observed that improvement is not guaranteed when the $\hat{L}_{set}$ is lowered. The reason is that for some of the delivery trips, the $\hat{L}_{set}$ is not low enough to make a difference. Fuel use for a given vehicle in a particular trip is a

function of the L_{set} value. It was observed that fuel use reduction is not guaranteed when the L_{set} value is lowered from the original setting. Also, fuel use will not increase after it is higher than a particular value. For example, Figure 3.11 shows the fuel use under different L_{set} of a trip from vehicle D. It can be found that when the L_{set} is higher than about 90 miles, no matter how high it is, the fuel use is the same. On the other hand, the L_{set} should be lower than 90 miles to achieve some fuel reduction for this trip. The value below which fuel can be saved is different to each trip of one vehicle due to different trip distance and energy intensity.

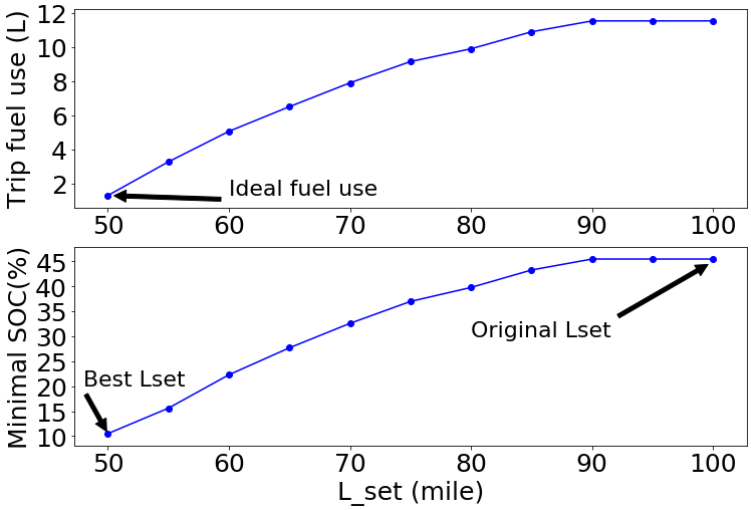


Figure 3.11: Fuel use and minimal SOC under different L_{set} for a trip of vehicle D.

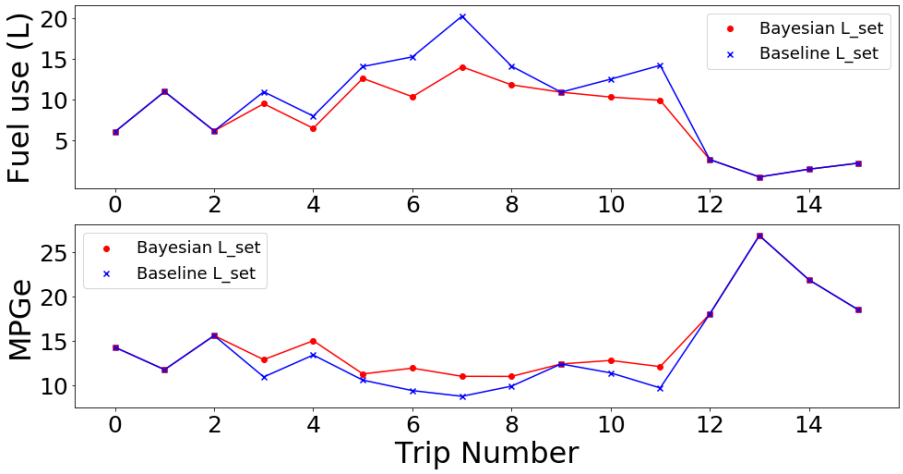


Figure 3.12: Fuel use and MPGe comparison for test trips of vehicle T11.

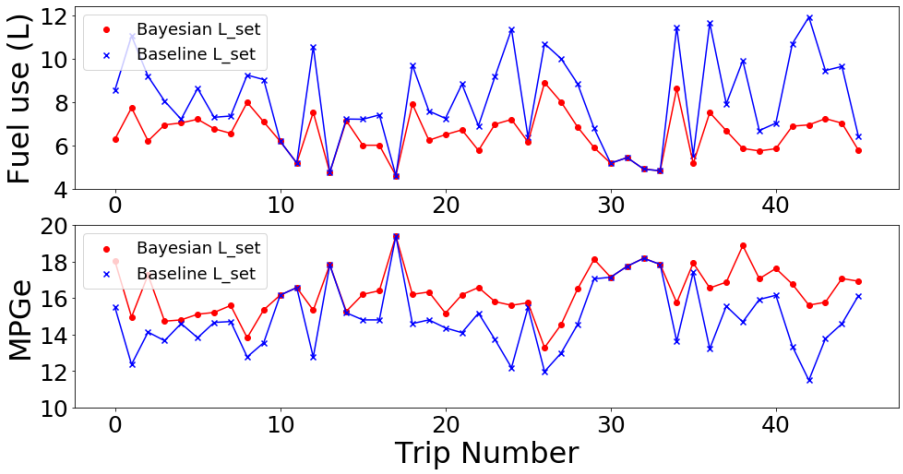


Figure 3.13: Fuel use and MPGe comparison for test trips of vehicle T12.

All fuel reduction data for the 13 demonstration vehicles is summarized in Table 3.3. It can be observed that the average fuel use reduction ranges from 0% to 28.4%. The high fuel use of the original setting is due to the fact that there is no clear method to determine the L_{set} for different delivery vehicles in different delivery areas. The proposed framework can effectively tune the L_{set} for each vehicle in the thermostatic method by updating the estimated distribution associated with each individual vehicle every time there is a new trip data.

This work provides a scalable and systematic framework for the energy management of the EREVs used for package delivery. It can be used with a small amount data and deals with the sequential nature of the problem elegantly. Since the developed framework is conservative and programs the L_{set} only at the beginning of a trip, there is potential to further reduce fuel use if it is set dynamically using real-time information during the trip.

Table 3.3: Fuel efficiency improvement

Vehicle Number	Average MPGe Improvement (%)	Average Fuel Reduction (%)	Number of Trips
T0	0	0	10
T1	2.7	4.2	3
T2	5.8	9.2	3
T3	11.9	15.9	6
T4	3.3	4.9	6
T5	13.8	18.3	7
T6	4.1	6.6	9
T7	5.4	8.7	9
T8	1.2	1.8	10
T9	20.8	28.4	15
T10	8.4	11.9	15
T11	9.3	11.2	16
T12	10.5	16.5	46
Total	**8.9**	**12.8**	**155**

4. DEEP REINFORCEMENT LEARNING BASED ENERGY MANAGEMENT

4.1. *Reinforcement Learning*

Reinforcement learning algorithms are used to solve sequential decision-making problems under uncertainty, which means they are aimed to find good behavioral strategies to achieve some predefined goals under an environment or system that is unknown to the algorithm [83]-[85]. The searching process of good behavioral strategies is driven by the interaction with the unknown environment with trial and error, and this process is also called learning. If the mathematical model of the environment is given, the problem can be solved by planning methods.

Reinforcement learning is closely connected to optimal control [80]-[82]. Optimal control is also used to solve sequential decision-making problems, and the main difference is it requires a known mathematical model of the studied system, for example, dynamic programming (DP) can achieve the theoretical optimal results given the known model. Value-based deep reinforcement learning can be considered to approximate dynamic programming [82] with deep neural networks as function approximators. In the optimal control literature, the goal of a controller is to minimize the cumulated cost in a known system through a sequence of decisions or controls. In the terminology of reinforcement learning, the goal of an agent is to maximize the cumulated reward in an unknown environment through a sequence of actions. The more detailed discussion on the connection and difference between reinforcement learning and optimal control can be found in [81]. In the remaining of this dissertation, our problem is formulated in the RL

setting since the underlying environment that generates the velocity profiles is unknown to us.

In a RL problem setting, there are two roles, the agent and the environment. The boundary between the agent and the environment is determined by the criteria of whether it can be totally controlled. The agent is the part we can control, and the environment is the part that is unknown to us. We can only influence or learn about the environment through interaction with it by controlling the agent. The interaction is enabled by three signals: state (s), action (a), and reward (r). There are different types of RL problems and in this dissertation, we focus on the episodic task with discrete time steps, which means there is a clear terminal state for each task. At the beginning of each episode, the initial state s_0 is provided by the environment. The agent observes s_0 and chooses to execute action a_0 according to its policy π_0, which is a mapping from states to actions: $\pi: s \to a$. The environment responses with the next state s_1 and reward r_o. This interaction process will continue until a terminal state s_T is achieved, and each episode of interaction can be recorded as:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, s_3 \dots s_T. \tag{4.1}$$

The policy π is improved using the recorded interactions and updated during or after each episode according to different RL algorithms. The goal of the policy π is to achieve high discounted cumulated rewards at each time step t, which is defined as:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1}, \tag{4.2}$$

where $\gamma \in [0,1]$ is the discount factor. When it equals 0, the agent only considers the immediate reward r_t, meaning that it is extremely "shortsighted". When it equals to 1, the

agent considers all future rewards equally, meaning that it is very "farsighted". As future rewards are more difficult to earn, a discount factor slightly less than 1 is widely used.

A sequential decision-making problem can be formulated as a RL problem by converting it to a Markov decision process (MDP). An MDP consists of a state space S, an action space A, a reward function $r(r_t|s_{t+1}, s_t, a_t)$, a transition probability model $p(s_{t+1}|s_t, a_t)$ and a discount factor γ. It should be noted that in MDPs, the transition probability to next states s_{t+1} only depends on the current state s_t and action a_t so that it has the Markov Property. It can be considered as the current state s_t consists of all the information that can make a difference for the future, and it is sufficient to make decisions according to it. If a problem can be formulated as an MDP, RL algorithms can be used to solve it, i.e., find satisfying policies. If states are not available and only observations (o_t) can be used, partially observable MDP (POMDP) related algorithms should be considered [74].

One of the key features of RL is the trade-off between exploration and exploitation. The main reason behind it is the delayed rewards. For each time step, the reward received from the environment can not represent how good is the selected action accurately. For example, some states with high rewards may only be achieved by going through some negative or low rewards. In contrast, actions lead to immediate positive rewards may lead to bad state space. Consequently, the agent must try different actions at different states to cumulate knowledge about the unknow environment. At the same time, it should also exploit the experience it gained from previous interaction to get high cumulated rewards. Although these two ideas have the same final goal: to achieve high cumulated reward,

they are contradictory to some degree when solving RL problems, since the best action according to the current estimation is usually not the action with highest uncertainty. Practically, for problems with a large continuous state space, the most widely used methods are based on heuristics like ε-greedy and adding noises to the action directly, which are easy to implement and works well for many problems. More sophisticated exploration methods can be found in [87] and [88], and are an active research area with advancements continually being developed.

4.2. *Feedforward Neural Networks*

Feedforward neural networks (FNNs), which are also called multi-layer perceptron (MLPs), are powerful and flexible function approximators. They are capable of learning complex nonlinear mappings. In fact, it has been proved that neural networks with just two layers of weights are capable of approximating any continuous functional mapping [89][93], detailed discussion about the universal approximation theorem can be found in [90][91]. In this part, NN architectures for regression and classification with corresponding loss functions are introduced.

The structure of a general FNN is shown in Figure 4.1. The inputs have d dimensions. There are M hidden units and c output units. The output z_j of hidden layer can be calculated by transforming the linear combination of inputs using an activation function $g(\cdot)$. The linear combination of the inputs is:

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \,, \tag{4.3}$$

where $w_{ji}^{(1)}$ is the weight in the first layer, going from the ith dimension of the input to the jth hidden unit. Then, it is transformed by an activation function:

$$z_j = g(a_j). \tag{4.4}$$

The output y_k of the FNN can be calculated similarly as z_j:

$$y_k = \tilde{g}\left(\sum_{j=0}^{M} w_{kj}^{(2)} z_j\right). \tag{4.5}$$

By combining (4.3)-(4.5), the function represented by the FNN structure shown in Figure 4.1 can be written as:

$$y_k = \tilde{g}\left[\sum_{j=0}^{M} w_{kj}^{(2)} \cdot g\left(\sum_{i=0}^{d} w_{ji}^{(1)} x_i\right)\right]. \tag{4.6}$$

More layers can be added to the shown general FNN structure which leads to more transformations like (4.5).

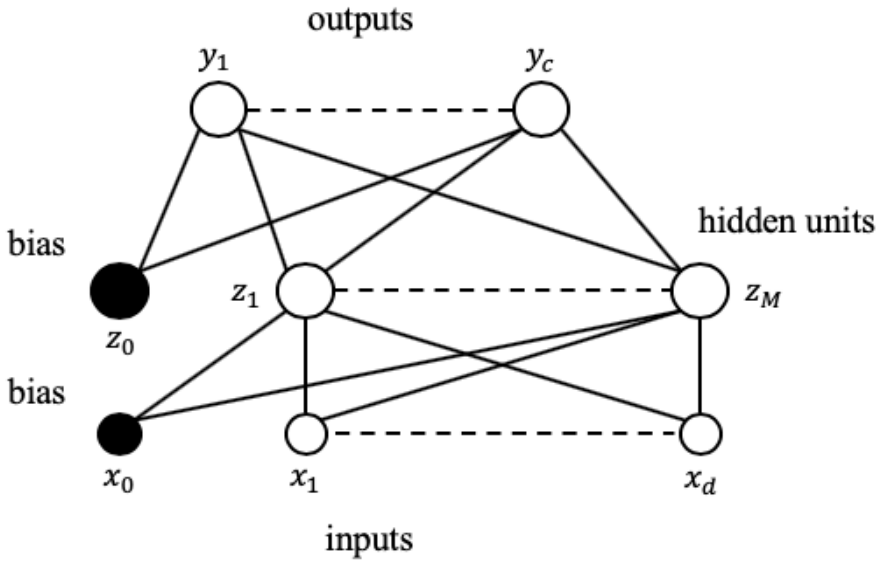


Figure 4.1: An example of a feedforward neural network with one hidden layer. The bias parameters have a fixed value of 1.

For regression tasks, given the known right label t_k^n, the most widely used error function is the sum-of-squares error:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}[y_k(x^n; w) - t_k^n]^2 . \tag{4.7}$$

This error function can be derived from the principle of maximum likelihood. Given a set of training data which is assumed to be independently and identically distributed, the likelihood function can be expressed as:

$$L = \prod_{n=1}^{N} p(x^n, t^n) = \prod_{n=1}^{N} p(t^n|x^n)p(x_n) . \tag{4.8}$$

Usually, instead of maximizing the likelihood, it is more convenient to minimize the negative logarithm of the likelihood:

$$E = -lnL = -\sum_{n=1}^{N} lnp(t^n|x^n) - \sum_{n=1}^{N} ln\, p(x_n). \tag{4.9}$$

Since the second term does not depend on the weights, (4.9) is rewritten as:

$$E = -\sum_{n=1}^{N} lnp(t^n|x^n) . \tag{4.10}$$

By assuming $p(t^n|x^n)$ is Gaussian distributed with fixed unknown variance, the sum-of-squares error can be derived.

For classification tasks, the error function of cross-entropy is widely used with the 1-of-c coding scheme of target data t_k^n , or with $0 \le t_k^n \le 1$ and $\sum_{k=1}^{c} t_k^n$:

$$E = -\sum_{n=1}^{N}\sum_{k=1}^{c} t_k^n lny_k(x^n; w) . \tag{4.11}$$

This function is derived by assuming the conditional probability $p(t^n|x^n)$ in (4.10) can be written as:

$$p(t^n|x^n) = \prod_{k=1}^{c} y_k(x^n; w)^{t_k^n}. \tag{4.12}$$

By using the derived error functions, for regression problems, the output of the NN can be interpreted as the approximation of conditional average of the target data. For classification problems, the output can be interpreted as probabilities of class membership, conditioned on the outputs of the hidden units. With other form of loss functions, the NN can be used to approximate other quantities, e.g., if the mean absolute error is used in a regression task, the conditional median of the target data is modeled.

The activation function $g(\cdot)$ in the FNN structure is used to provide the nonlinearity. The default choice is the rectified linear unit (ReLU) [93][96]. Other useful functions include sigmoid function and tanh function.

With a determined NN structure and a chosen loss function, the optimization or the training process of the weights in the NN is usually performed by gradient based methods and the idea of error back-propagation [92]. The detailed optimization methods beyond the scope of this dissertation and can be found in [93][94]. The implementation of NNs and the training process in this dissertation are all enabled by the popular deep learning package PyTorch [95].

4.3. *Reinforcement Learning Problem Formulation*

The energy management problem studied in this work can be formulated as a RL problem by defining the five components of an MDP. The discount factor γ is set to be 0.99 in this work and the rest of four components are defined as follows.

- State space

The state space is a seven-dimensional space, $S \in \mathbb{R}^7$. Each dimension of the state space represents one type of available information that can be provided by the studied EREV in real-time during the delivery task. Therefore, each state s_t is represented by a vector:

$$s_t = [t_{travel}, d, SOC, f, x, y, L_{set}], \tag{4.13}$$

where t_{travel} is the travelled time, d is the travelled distance, SOC is the current state of charge, f is the current total fuel use, x and y are the GPS coordinates, and L_{set} is the current L_{set} setting. A state s_t is a point in the state space.

- Action space

The action space is a one-dimensional space, $A \in \mathbb{R}^1$. Its value represents the magnitude of L_{set} change at each timestep. In this work, two versions of action space are defined: the discrete version and the continuous version. The discrete version is defined as:

$$a_t \in \{A_{-10}, A_{-5}, A_0, A_{+5}, A_{+10}\}, \tag{4.14}$$

where the subscript represents the magnitude of L_{set} change. The continuous version is defined as:

$$a_t \in [-A_{max}, A_{max}]. \tag{4.15}$$

The a_t can be in the range of $-A_{max}$ to A_{max} ,and A_{max} is set to be 20 in this work. In addition, if the magnitude of the calculated action is smaller than 5, it will be set to 0, which leads to no change of L_{set} at this time step. In other words, the algorithm has a choice of doing nothing by producing an action a_t in the range of $(-5, +5)$. This design is useful to our reward function.

- Reward function

The reward function should reflect our goal of minimizing fuel consumption, with the constraint of battery SOC always higher than 10%. To guide the RL algorithm to find a policy that can achieve this goal, the reward function is designed as:

$$r_t = r_f t_{f,t} + r_{SOC} t_{SOC,t} + r_{a,t} + r_c, \tag{4.16}$$

where the first term penalizes fuel use and its magnitude is proportional to the engine running time $t_{f,t}$ with coefficient r_f equals to -0.001. The second term penalizes the condition of SOC lower than 10% and its magnitude is proportional to the amount of time under that condition $t_{SOC,t}$ with coefficient r_{SOC} equals to -0.060. To guide the algorithm in finding an efficient policy, $r_{a,t}$ is added to penalize actions that change L_{set}. It will cause a reward of -0.020 if the L_{set} is changed.

The first term $r_f t_{f,t}$ penalizes all the fuel use during the trip as the remaining distance and energy intensity is unknown. However, for trips exceeding the vehicle's all-electric range, some amount of fuel use is necessary to keep the SOC larger than 10% during the whole delivery trip. To compensate for this necessary fuel use, a reward term r_c is imposed at the end of the trip. After the trip is finished, the amount of fuel that was necessary to keep the SOC larger than 10% is simulated using the vehicle model and the velocity profile,

which determines the magnitude of r_c. For example, if after a delivery trip it is calculated that 1 gallon of fuel was required to keep the SOC larger than 10% during the trip, and the actual total fuel use was 1.5 gallons, the negative reward caused by the required 1 gallon will be compensated by the r_c term at the end of the trip.

- Transition probability

The transition probability $p(s_{t+1}|s_t, a_t)$ of the environment that generates the velocity profiles is unknown and difficult to model as discussed in the problem statement section. Therefore, it is approximated by the recorded historical delivery trips and the vehicle model. Based on any part of the recorded velocity profiles, the change of SOC and fuel consumption under arbitrary L_{set} values corresponding to that part of trip can be calculated. Since for any segment of recorded velocity profiles, the change of t_{travel}, d and GPS coordinates are fixed, the next state s_{t+1} can be obtained. Combined with the designed reward function, the reward r_t can also be calculated. This approximation provides an interface for the intelligent agent to interact with. The underlying assumption for this approximation method is the recorded trips can represent the future trips well so that the experience learned from interaction with this environment is also useful for future testing. Consequently, to use DRL methods, relatively large amount of historical delivery trips is needed.

4.4. *Value-based Method*

Value-based RL algorithms are based on the concept of action-values, which is also called Q-values [83]. It is defined as:

$$Q_\pi(s, a) = E_\pi[G_t|s_t = s, a_t = a]. \quad (4.17)$$

It represents the expected discounted cumulated reward G_t can be achieved if action a is taken at state s and following policy π thereafter. Intuitively, it quantifies the goodness of taking action a at state s under policy π in the environment. The optimal action-value is defined as:

$$Q^*(s,a) = max_\pi Q_\pi(s,a). \tag{4.18}$$

For each state-action pair, it represents the highest action-value that can be achieved in the predefined environment among all possible policies. Therefore, if the optimal action-values for all possible state-action pairs that is possible to be encountered during the task are known, the optimal control policy can be derived easily by acting greedily with respect to it:

$$\pi^*(s) = argmax_a Q^*(s,a). \tag{4.19}$$

Consequently, the goal of value-based RL methods is to estimate the optimal action-values which is based on the Bellman optimality equation [80][81]:

$$Q^*(s,a) = E_{s'}[r + \gamma max_{a'} Q^*(s',a')|s,a]. \tag{4.20}$$

It describes the relation that the optimal action-values should satisfy for a given MDP. The standard Q-learning algorithm is an off-policy, temporal difference (TD) method which utilizes this relation [83][86]. Instead of trying to evaluate the expectation over all possible next states, which requires the detailed environment model and leads to dynamic programming, it approximates the right hand side with a transition pair (s_t, a_t, r_t, s_{t+1}). This approximation process can be considered as sampling from the underlying environment, i.e., sampling from the right hand side of (4.20), and the sampled result is

called TD target. s_{t+1} is the resulting state and r_t is the immediate reward when action a_t is taken at state s_t:

$$Q_{target} = r_t + \gamma max_{a'} Q(s_{t+1}, a'). \tag{4.21}$$

Traditional RL algorithms use tables to represent the action-values. For tables, the entry for $Q^*(s_t, a_t)$ will be updated by Q_{target} at each iteration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)], \tag{4.22}$$

where α is the step size or learning rate.

For larger scale real-world problems, NNs are widely used as function approximators for the optimal action-value functions, which is expressed as $Q(s, a; \theta)$, and that is why DRL is also called neuro-dynamic programming [80]. It is called Q-Network and θ represents all the parameters in the model. Linear models are also widely investigated in the literature and it can be considered as a special case of Q-Network. The algorithm of Deep Q-Network (DQN) is used in this work [98]. The loss function is the square of the difference between the estimated Q-value for the current state-action pair and its TD target, which is also called TD error:

$$L = \left[Q_{target} - Q(s_t, a_t; \theta)\right]^2. \tag{4.23}$$

The Q-Network can be trained as in the supervised learning case by utilizing (3.23). However, there are mainly two problems encountered in training Q-networks. First, the transition pairs generated during the training process are highly correlated. This is because during one episode of task, the action chosen in the current time step will influence the next state. This makes the Q-Network overfits to some part of the state space, and the data are not independently and identically distributed. Second, when a gradient decent step is

performed to update the parameter θ, both of $Q(s_t, a_t; \theta)$ and the TD target Q_{target} are changed as they are both determined by θ. This means a value is updating toward a changing target, which leads to highly unstable training processes. To solve the first problem, experience replay is used. Every time a transition pair is observed, it is added to a replay buffer. To update θ, a random minibatch of transition pairs is sampled from the buffer, which breaks the temporal correlation. To solve the second problem, two copies of the Q-network are kept. The second Q-network is called the target network. Every time the main Q-network is updated, a soft update is performed on the target network, which means 0.01% of the updated Q-network will be mixed into the target network, providing a more stable TD target. In addition, the technique of double DQN is also highly helpful for both the training process and the final performance of DQN-based algorithms [99]. It solves the problem of maximization bias and the form of TD target is:

$$Q_{target} = r_t + \gamma Q(s_{t+1}, argmax_{a'} Q(s_{t+1}, a'; \theta); \theta^-). \quad (4.24)$$

For each episode of interaction, the initial state s_0 is provided by the environment, which is a fixed vector, since the delivery task starts at the same location, battery SOC as well as the initial L_{set} setting. Then, the action is selected with respect to the current Q-network with the exploration strategy of $\varepsilon-$greedy, i.e., choose actions randomly with probability ε and choose actions with the highest action-value otherwise. The value of ε is initialized as 1.0 and gradually decreased to 0.01. The updated value of L_{set} will be used to calculate the next state s_1 and reward r_0. The transition pair (s_0, a_0, r_0, s_1) will be stored into the experience replay buffer. This process will continue until the current trip is finished. When all the training trips are used once, it is called one epoch of training. The

number of epochs is a hyperparameter. The update step is executed during the training process after the number of stored transition pairs is higher than the batch size. The detailed training algorithm is summarized in Algorithm 1 and Figure 4.2 illustrates one step of interaction which is in the while loop of Algorithm 1.

The main limitation of this method is the algorithm is not guaranteed to converge. The main reason is the combination of the Bellman operator and the squared loss function is not a contraction with respect to any norm, which means the current estimation of Q-values is not guaranteed to be closer to the optimal Q-values after the gradient-based update with respect to any distance metrics, e.g. L1 norm or L2 norm. In addition, due to the use of NN, global optimal solution cannot be achieved. Consequently, the goal of using DQN is to find weights that yield satisfying results and it should be achieved by hyperparameter tuning and multiple times of training with different initial NN weights. This is also true for all other DRL algorithms.

There are many techniques can be added to the DQN framework, such as prioritized experience replay [100], dueling network structure [101] and noisy network structure [102]. In [103], a combination of many techniques mentioned above including double DQN is studied.

Algorithm 1 Double Q-learning

Initialize the action-value function Q-network with random parameters θ
Initialize the target action-value function Q-network as $\theta^- = \theta$
Initialize the replay buffer B to capacity D with no transitions
Initialize $\varepsilon = 1.0$
For epoch = 1, M **do**
. **For** trip = 1, N **do**
. . Get initial state s_0 from current trip data
. . $t = 0$
. . **While** $t < T$, **do**
. . . With probability ε, randomly select an action A_t
. . . Otherwise select $a_t = argmax_a Q(s_t, a; \theta)$
. . . Update current L_{set} according to a_t
. . . Run the vehicle model for 2000s with updated L_{set}
. . . Return state s_{t+1} and r_t
. . . Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer B
. . . Sample random n transitions (s_j, a_j, r_j, s_{j+1}) from B
. . . $y_j = \begin{cases} r_j & \text{if trip terminates at step j + 1} \\ r_j + \gamma Q(s_{j+1}, argmax_{a'} Q(s_{j+1}, a; \theta); \theta^-) & \text{otherwise} \end{cases}$
. . . Perform a gradient descent step on $\left[y_j - Q(s_j, a_j; \theta)\right]^2$
. . . Perform soft update of the target network θ^-
. . . $s_t = s_{t+1}$
. . $\varepsilon = \max(0.01, 0.995 \times \varepsilon)$
. **End For**
End For

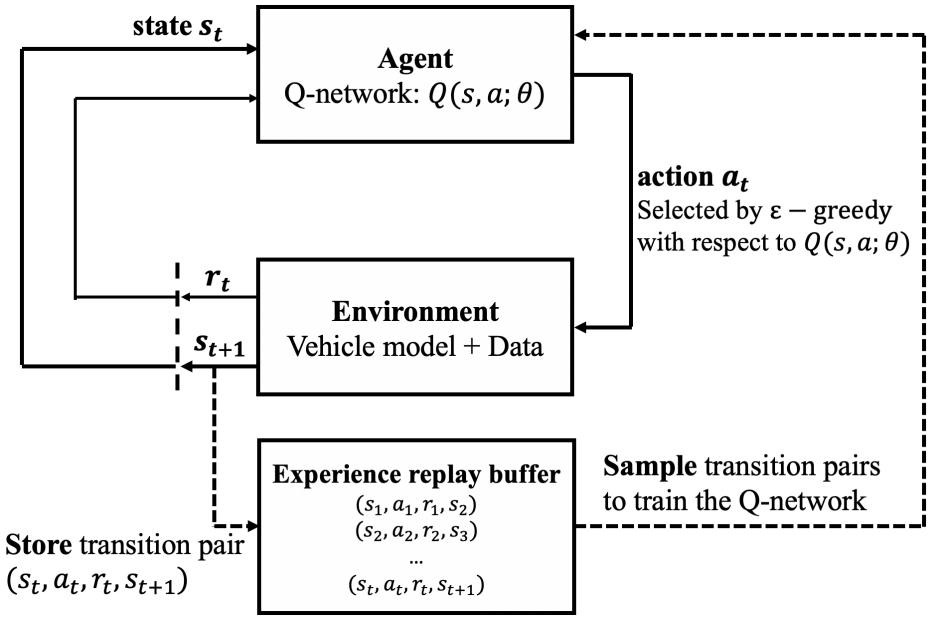


Figure 4.2: Illustration of one update step in Algorithm 1.

4.5. *Actor-Critic-based Method*

Value-based methods make decisions according to the estimated action-values and do not keep explicit policies. For actor-critic-based methods, instead of keeping implicit policies, policies are parametrized directly by function approximators, which take states as inputs and actions as outputs:

$$a_t = \pi_{\theta^\mu}(s_t) = \mu(s_t|\theta^\mu), \tag{4.25}$$

where θ^μ represents all the parameters in the parametrized policy, which is also called actor. The policy can be either stochastic or deterministic. In this work, the algorithm of deep deterministic policy gradient (DDPG) is used [104][105]. Some literature about stochastic policies can be found in [106]-[110]. The goal of the actor is to maximize the expected discounted cumulated reward for each episode of interaction, and it is defined as:

$$J(\theta^\mu) = E_{\tau\sim p_{\theta^\mu}(\tau)}[r(\tau)]. \tag{4.26}$$

τ is a trajectory of interaction which has the form of:

$$s_0, a_0, s_1, a_1, s_2, a_2 \dots s_T. \tag{4.27}$$

$r(\tau)$ is the total discounted reward for a trajectory τ:

$$r(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)\,. \tag{4.28}$$

$p_{\theta^\mu}(\tau)$ is the probability distribution of trajectory τ under policy π_{θ^μ} and the predefined environment:

$$p_{\theta^\mu}(\tau) = p(s_0)\prod_{t=0}^{T} \pi_{\theta^\mu}(s_t)\, p(s_{t+1}|s_t, a_t). \tag{4.29}$$

Therefore, $J(\theta^{\mu})$ represents the expected discounted cumulated reward the policy $\pi_{\theta^{\mu}}$ can achieve under the predefined environment, which quantifies how good is a policy under a specific environment. To improve the quality of the policy $\pi_{\theta^{\mu}}$, the parameters θ^{μ} should be adjusted so that the value of $J(\theta^{\mu})$ is increased. According to the deterministic policy gradient theorem [104], the policy gradient $\nabla_{\theta^{\mu}} J(\theta^{\mu})$ can be estimated from K sampled transition pairs (s_i, a_i, r_i, s_{i+1}) as:

$$\nabla_{\theta^{\mu}} J(\theta^{\mu}) \approx \frac{1}{K} \sum_{i=1}^{K} [\nabla_a Q_{\pi_{\theta^{\mu}}}(s_i, a|\theta^Q)|_{a=\mu(s_i|\theta^{\mu})} \cdot \nabla_{\theta^{\mu}} \mu(s_i|\theta^{\mu})]. \tag{4.30}$$

$Q_{\pi_{\theta^{\mu}}}(s, a|\theta^Q)$ is the action-value defined in (4.17) and it is also called the critic, since it quantifies how good is the action calculated by the actor in the current state. θ^Q represents all the parameters in the critic. If the policy gradient $\nabla_{\theta^{\mu}} J(\theta^{\mu})$ can be evaluated, the parameters in the actor can be improved by the gradient ascent method:

$$\theta^{\mu} \leftarrow \theta^{\mu} + \eta \nabla_{\theta^{\mu}} J(\theta^{\mu}). \tag{4.31}$$

The learning rate, η is a hyperparameter, which means it is not learned by the algorithm but determined by us. The training process is similar to the training process of DQN described and there are two main differences. First, for one update step, both of the actor and the critic are updated. Second, the noise for exploration is added by utilizing a predefined Gaussian distribution. When the action is calculated by the actor, a sample from the Gaussian distribution is added to the calculated action. The detailed algorithm is summarized in Algorithm 2 and the illustration of one update step is shown in Figure 4.3.

Algorithm 2 Deep Deterministic Policy Gradient

Initialize the critic network $Q(s, a|\theta^Q)$ with random parameters θ^Q
Initialize the target critic network $Q'(s, a|\theta^{Q'})$ with $\theta^{Q'} = \theta^Q$
Initialize the actor network $\mu(s|\theta^\mu)$ with random parameters θ^μ
Initialize the target actor network $\mu'(s|\theta^{\mu'})$ with $\theta^{\mu'} = \theta^\mu$
Initialize the replay buffer B to capacity D with no transitions
Initialize $\varepsilon = 0.3$, $\tau = 0.001$, $D = 10^5$, $K = 48$
For epoch = 1, M **do**
. **For** trip = 1, N **do**
. . Get initial state s_0 from current trip data
. . **While** t < T, **do**
. . . Calculate action $a_t = \mu(s_t|\theta^\mu) + \varepsilon \cdot N(0,1)$
. . . Clip a_t to the range of [-1, +1]
. . . Update current L_{set} according to $A_{max} \cdot a_t$
. . . Run the vehicle model for 2000s with updated L_{set}
. . . Return state s_{t+1} and reward r_t
. . . Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer B
. . . Sample K random transitions (s_i, a_i, r_i, s_{i+1}) from B
. . . Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
. . . Update critic network by minimizing the loss:
. . . $L = \frac{1}{K}\sum_{i=1}^{K}(y_i - Q(s_i, a_i|\theta^Q))^2$
. . . Update the actor network using the sampled policy gradient:
. . . $\nabla_{\theta^\mu} J \approx \frac{1}{K}\sum_{i=1}^{K} \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
. . . Perform soft update on target networks:
. . . $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
. . . $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
. . . $s_t = s_{t+1}$
. **End For**
End For

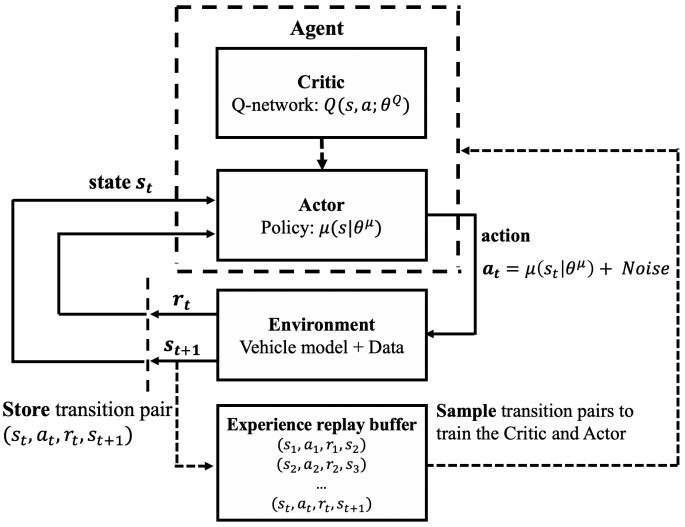


Figure 4.3: Illustration of one update step in Algorithm 2.

4.6. *Results and Discussion*

4.6.1. Deep Q-Network

For model selection and hyperparameter tuning, the Q-network was trained on 52 delivery trips for 500 epochs. The distance and energy intensity of the used training trips ranged from of 38 to 57 miles and 1.03 to 1.20 kWh/miles, respectively. The Q-network used was a feedforward neural network with two hidden layers. The number of hidden units in the two hidden layers was tuned in the range of 16 to 64 with an interval of 16. Learning rate was tuned over the range of 0.00001 to 0.00025 and batch size was tuned over the range of 16 to 64. Part of the tuning results are shown in Figure 4.4. The selected model structure of Q-network is summarized in Table 4.1.

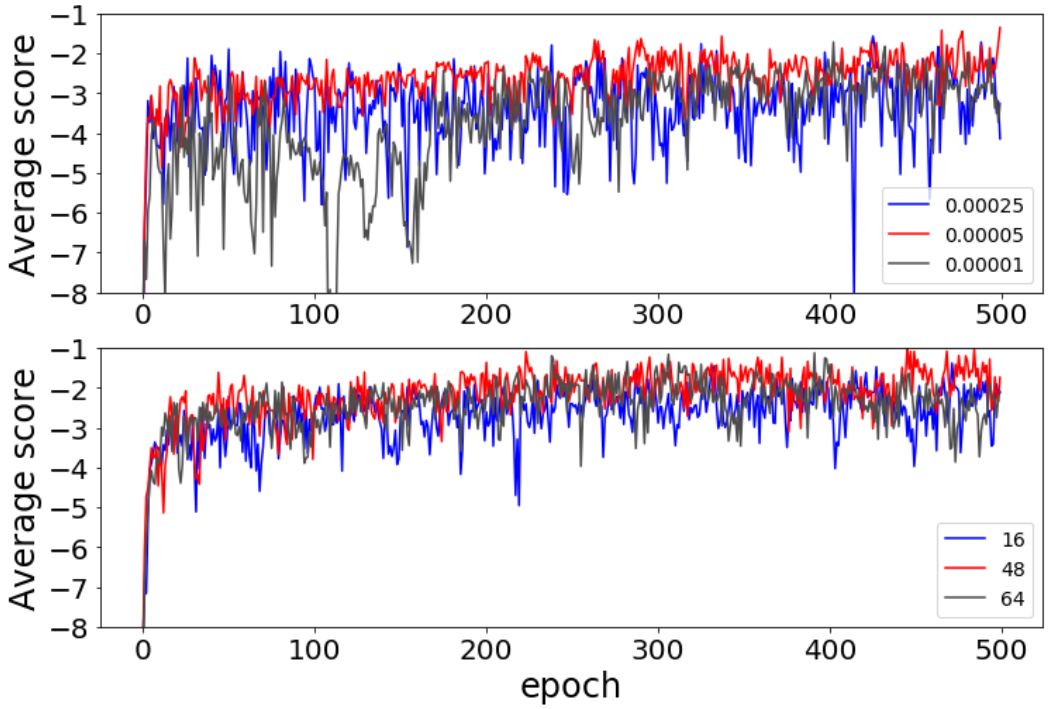


Figure 4.4: Learning curves of 3 learning rates (upper) and batch sizes (lower).

Table 4.1: Neural network structures and hyperparameters for DQN

DQN		
Hidden layer 1	Number of units	64
	Activation function	ReLU [95]
Hidden layer 2	Number of units	32
	Activation function	ReLU
Output layer	Number of units	5
	Activation function	Linear
Hyperparameters	Optimizer	Adam [97]
	Learning rate	5×10^{-5}
	Batch size	48

To evaluate the performance of the trained Q-network, it was tested on 44 unseen trips made by the same vehicle with a distance range of 31 to 54 miles. Figure 4.5 and Figure 4.6 show the detailed performance of the trained DQN algorithm on two of the test trips with different distances. For the trip in Figure 4.5, no fuel was used, so that the vehicle achieved the highest fuel efficiency. For the trip in Figure 4.6, the L_{set} was first decreased and then increased at the last part of the trip, achieving a final SOC very close to 10%.

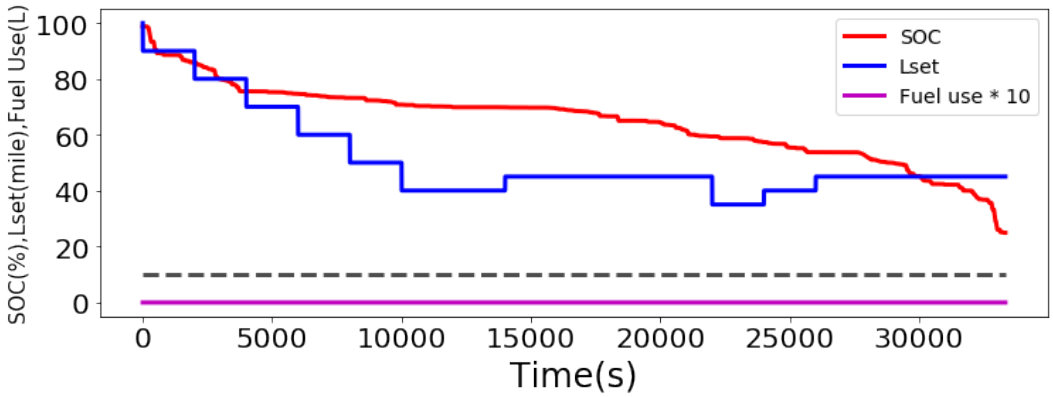


Figure 4.5: Performance of DQN on a test trip with 40 miles.

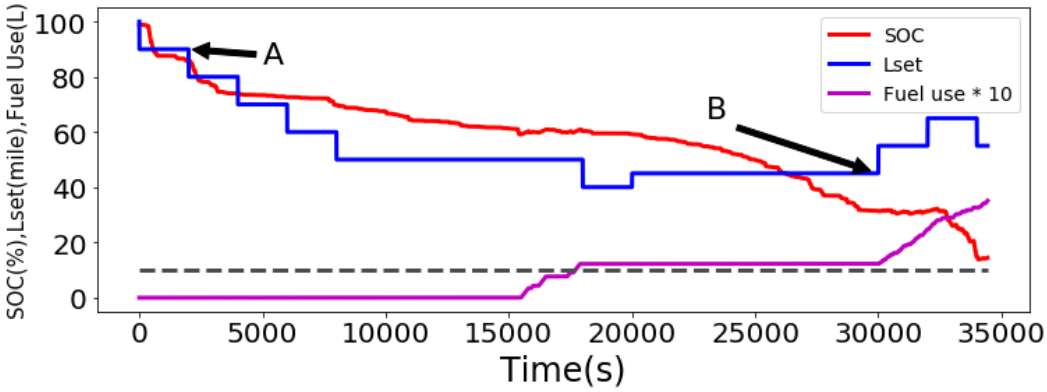


Figure 4.6: Performance of DQN on a test trip with 50 miles.

To better understand how the algorithm makes decisions, the action-values corresponding to state A and B in Figure 4.6 is shown in Figure 4.7. For state A, the action values for different actions do not differ largely as the algorithm knows that no matter what action is chosen in this early time step, there are enough opportunities in the remaining trip to choose the correct L_{set}. Since A_{-10} had the highest action value, L_{set} was decreased by 10. However, for state B, the action-value was high only for A_{10} as the algorithm figured out that unless L_{set} was immediately increased by 10 miles, the SOC would be lower than 10% for some time during the remaining trip, which would lead to a large negative reward.

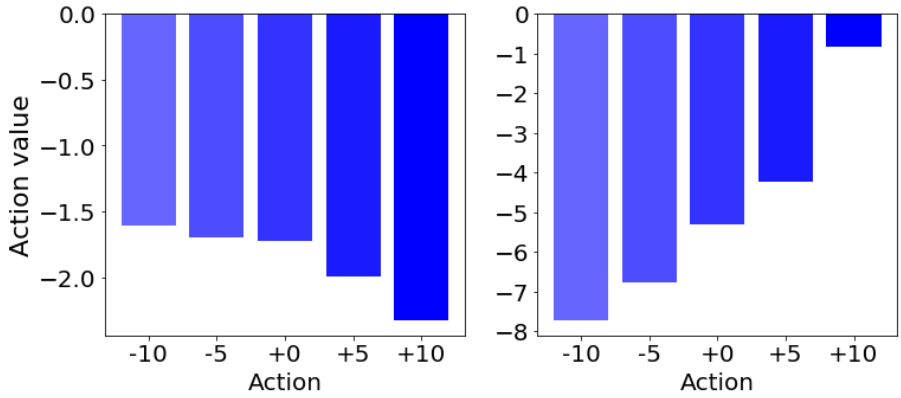


Figure 4.7: Action-values for state A (left) and state B (right).

The developed algorithm can also generalize to unforeseen conditions. During the training, the initial L_{set} was always set to be 100 miles. During the testing, if the initial L_{set} was changed to other values, the algorithm still made stable decisions. For example, in Figure 4.8, the initial L_{set} was set to 20 miles for the trip in Figure 4.6. The algorithm still achieved good performance, even though during training, this condition was never seen by the algorithm.

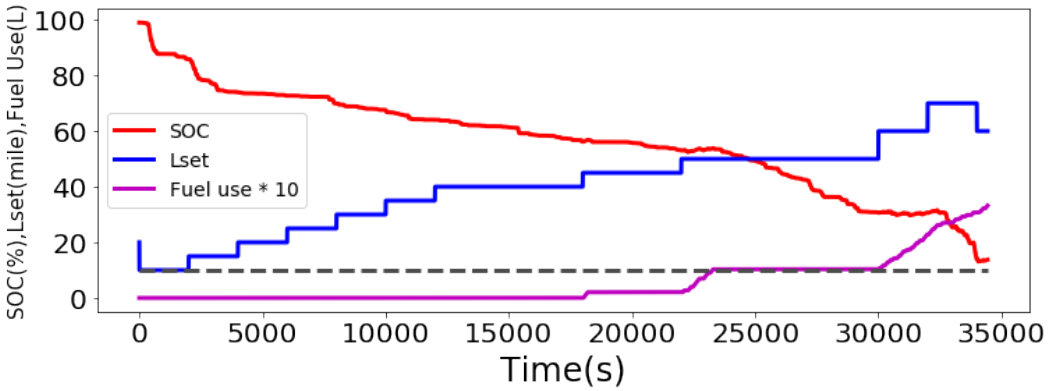


Figure 4.8: Performance of DQN on low initial L_{set} values.

In Figure 4.9, the MPGe (miles per gallon gasoline equivalent) of using DQN and the baseline setting is compared for every test trip. For most of the trips, the DQN method achieved higher fuel efficiency by tuning the L_{set} adaptively using real-time information. The average fuel economy improvement was 19.5% over 44 test trips with a distance range of 31 miles to 54 miles. Also, there was no condition of SOC lower than 10% in all test trips. The results show the trained DQN can generalize to unseen trips with different distance and energy intensity. Consequently, the method can be used in real-world application without information from other sources or an additional predictive model.

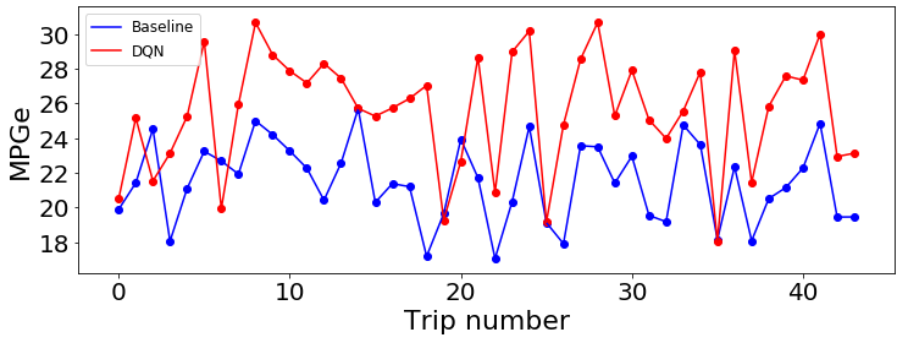


Figure 4.9: Comparison of MPGe by using DQN and baseline settings on test trips.

4.6.2. Deep Deterministic Policy Gradient

The network structure and hyperparameters for the DDPG algorithm is summarized in Table 4.2. The activation function tanh is used in the output layer of the actor since it can provide bounded outputs $(-1,1)$. The training trips are the same with that of DQN.

Table 4.2: Neural network structures and hyperparameters for DDPG

		Actor	Critic
Hidden layer 1	Number of units	64	64
	Activation function	ReLU	ReLU
Hidden layer 2	Number of units	48	48
	Activation function	ReLU	ReLU
Output layer	Number of units	1	1
	Activation function	tanh	Linear
Hyperparameters	Optimizer	Adam	Adam
	Learning rate	10^{-4}	10^{-3}
	Batch size	48	48

To evaluate the performance of the trained DDPG solution, it was tested on 51 delivery trips not used for training made by the same vehicle. The distance range of the test trips was from 31 to 54 miles. Figure 4.10 shows the velocity profiles of two test trips with significantly different distances to demonstrate the performance of the solution. Figure 4.11 and Figure 4.12 show the detailed performance of the solution on the two test trips. For the shorter trip, the vehicle did not use any fuel during the trip, achieving the highest

fuel efficiency under this EMS. For the longer trip, the L_{set} was increased at the final leg of the trip and prevented the SOC from dropping lower than 10% while not using too much unnecessary fuel.

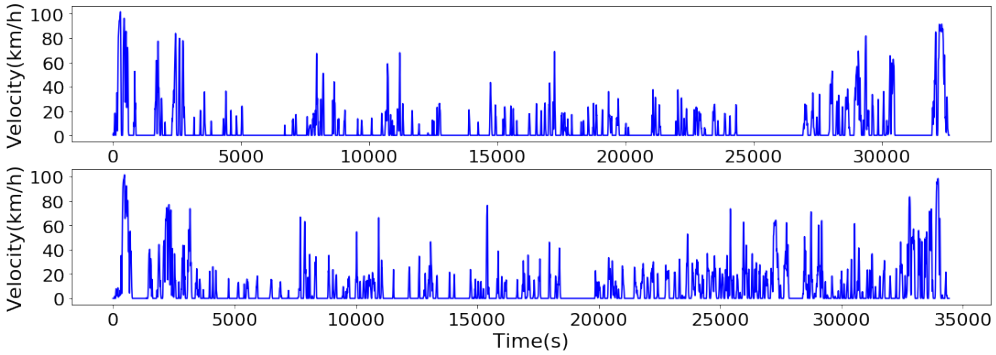


Figure 4.10: Velocity profiles of two example test trips with a distance of 35 miles (upper) and 50 miles (lower).

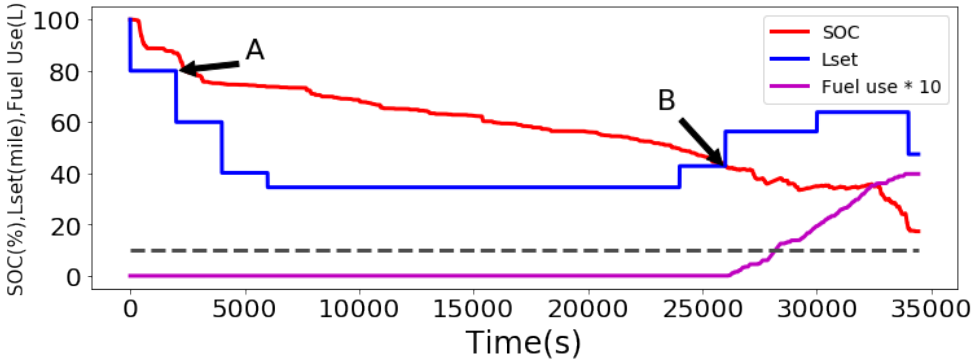


Figure 4.11: Performance of DDPG on a test trip with 50 miles.

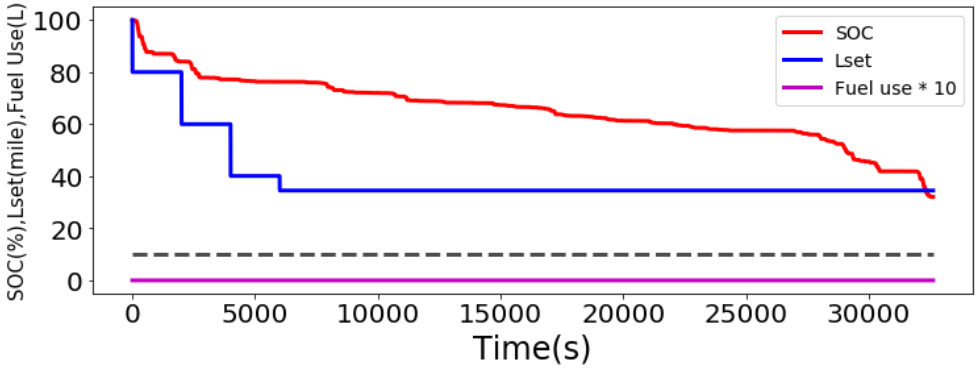


Figure 4.12: Performance of DDPG on a test trip with 35 miles.

A straightforward view of how the algorithm quantifies the value of different actions under different states can be determined by calculating the action-value $Q_{\pi_{\theta^\mu}}(s, a|\theta^Q)$ by the critic for different actions under state A and B shown in Figure 4.11. From Figure 4.13, it can be observed that the actor and critic agree well. The actions calculated by the actor are also quantified with high action-value by the critic compared with other possible actions. On the other hand, for state A, an early stage of the trip, the difference between the action-values are not significant as there is enough remaining time to change the L_{set} no matter what action is taken in this step. However, for state B, there is a gap of action-value when the action is lower than about -10. This means if such actions are used (e.g. decrease the L_{set} by 15), it is highly possible that the SOC will drop below 10%, causing a large negative reward. This is very similar to the interpretation of the action-values in DQN.

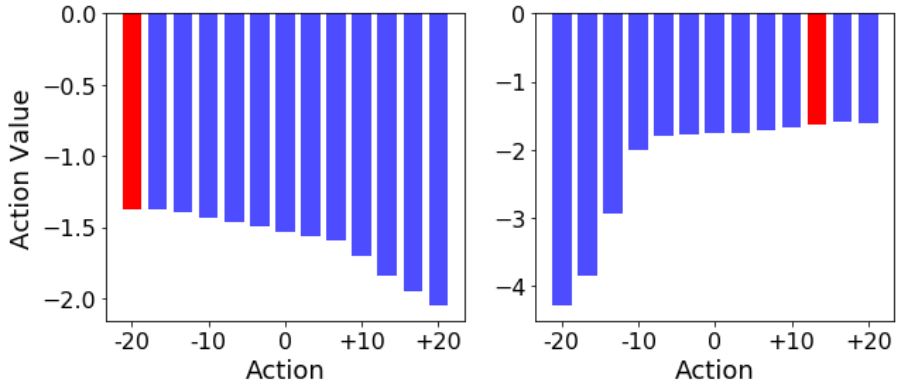


Figure 4.13: Action-values of state A (left) and state B (right) calculated by the critic for 13 example actions in the predefined range of [-20, +20]. The red bar indicates the action that is calculated by the actor.

The presented results also show that the trained DDPG solution is very robust, generalizing to different unseen conditions like different initial SOC and L_{set} setting. The longer test trip shown in Figure 4.11 is used to show this characteristic. During the training process, the initial SOC is 100% and the initial L_{set} is 100 miles. From Figure 4.14 and Figure 4.15, it can be observed that the algorithm can still keep the SOC higher than 10% while minimizing fuel use under unseen conditions. Further investigation on the robustness will be performed in future work by showing more quantitative results from actual optimized trips.

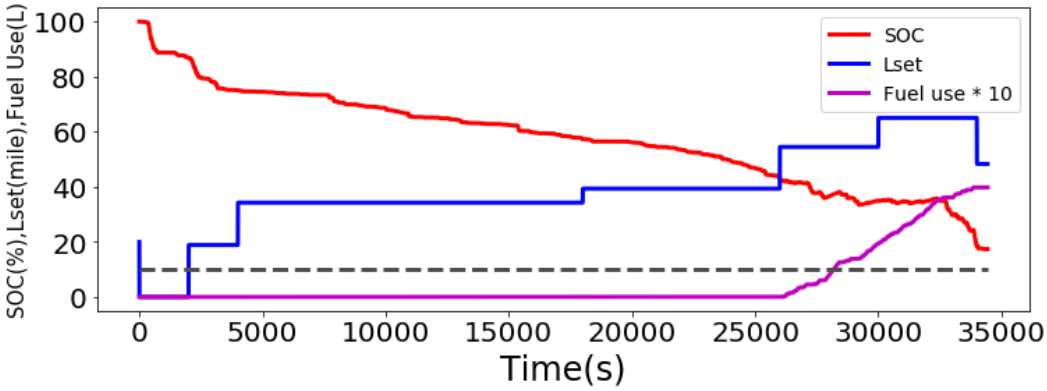


Figure 4.14: Performance of DDPG on low initial L_{set} values.

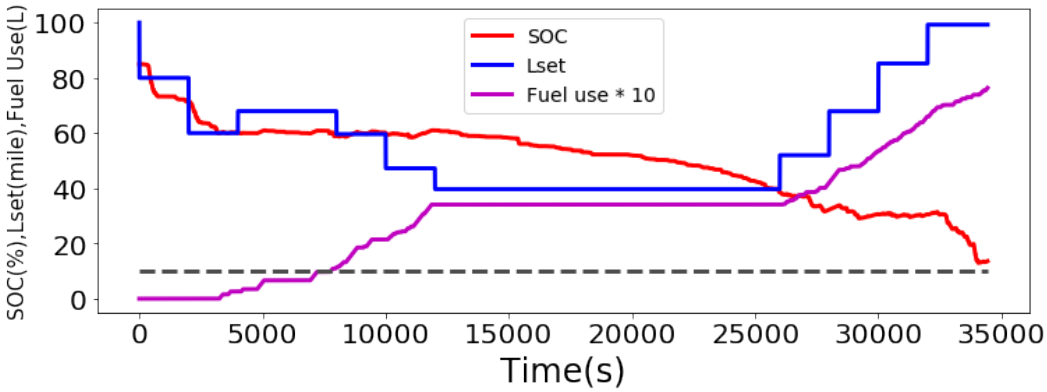


Figure 4.15: Performance of DDPG on low initial SOC values.

The comparison of MPGe by using DDPG and baseline setting for all 51 test trips are summarized in Figure 4.16. In most trips, the DDPG solution that dynamically changes the L_{set} outperforms the static baseline L_{set}. The average MPGe improvement was 21.8% over 51 test trips with a distance range of 31 to 54 miles.

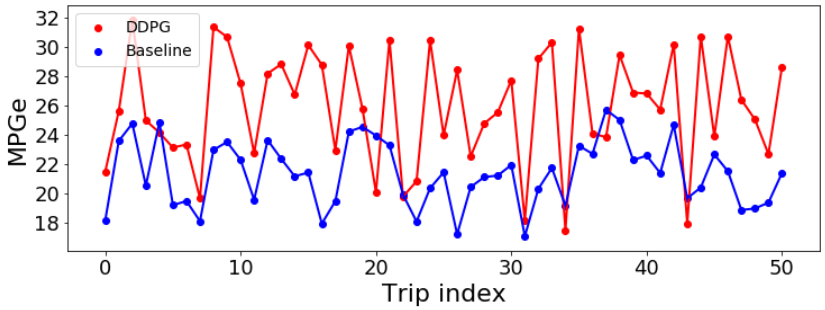


Figure 4.16: Comparison of MPGe by using DDPG and baseline settings on test trips.

4.6.3. Real-world Testing

The DDPG algorithm was tested on 3 in-use delivery vehicles with different characteristics. To quantify the difference and similarity between the driving cycles of the 3 vehicles, characteristic acceleration $\tilde{a}$, aerodynamic speed v_{aero} and kinetic intensity ki are used, which are defined as [111]:

$$\tilde{a} = \frac{\sum_{j=1}^{N-1} positive(\frac{1}{2} \cdot (v_{j+1}^2 - v_j^2))}{D},$$

$$v_{aero}^2 = \frac{\sum_{j=1}^{N-1} \overline{v_{j,j+1}^3} \cdot \Delta t_{j,j+1}}{D},$$

$$ki = \frac{\tilde{a}}{v_{aero}^2}. \tag{4.32}$$

Kinetic intensity is used to quantify the characteristic of driving cycles. Intuitively, urban driving cycles tend to have a high ki as it has lower speed limits and consists of many stop-and-go conditions. On the contrary, highway driving tends to have a low ki

due to its high-speed nature. All the three metrics are closely linked to the energy usage characteristic of a driving cycle independent of vehicle type. Therefore, they can be used to quantify the similarity among driving cycles from an energy perspective [111]. In Figure 4.17, the historical trip of the 3 vehicles are plotted with x-axis represents the $\tilde{a}$ and y-axis represents the v_{aero} of each trip. It can be observed that the data points corresponding to the same vehicle form a cluster in the $\tilde{a}$-v_{aero} space and the main parts of the 3 clusters are clearly separated from each other with some amount of overlapping. Consequently, it can be concluded that first, although the delivery trips corresponding to the same vehicle are different, they share similar characteristics and follow some underlying unknown distributions. Second, the three selected routes are clearly different from each other according to the introduced metrics. The average distance ($\overline{D}$), average kinetic intensity ($\overline{KI}$) and average energy intensity ($\overline{EI}$) of the selected routes are also summarized in Table 4.3.

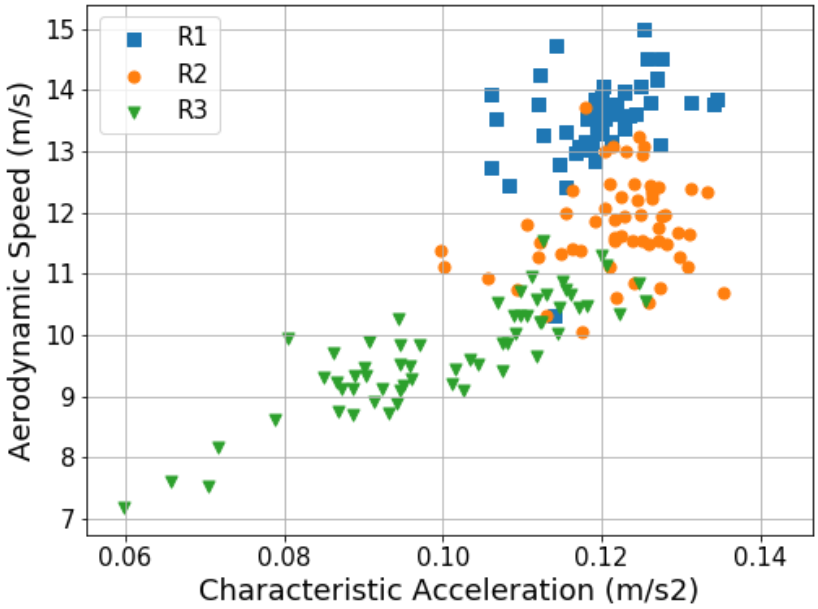


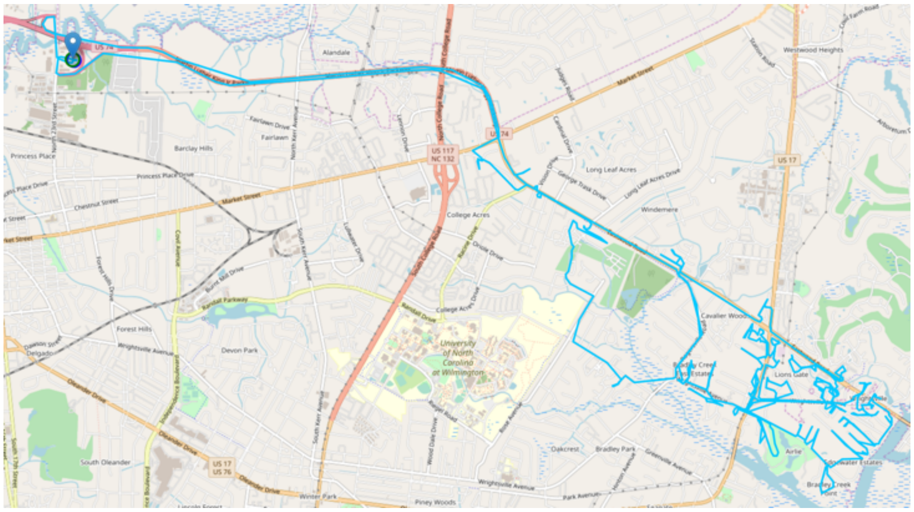
Figure 4.17: Characteristics of the delivery trips of three testing vehicles.

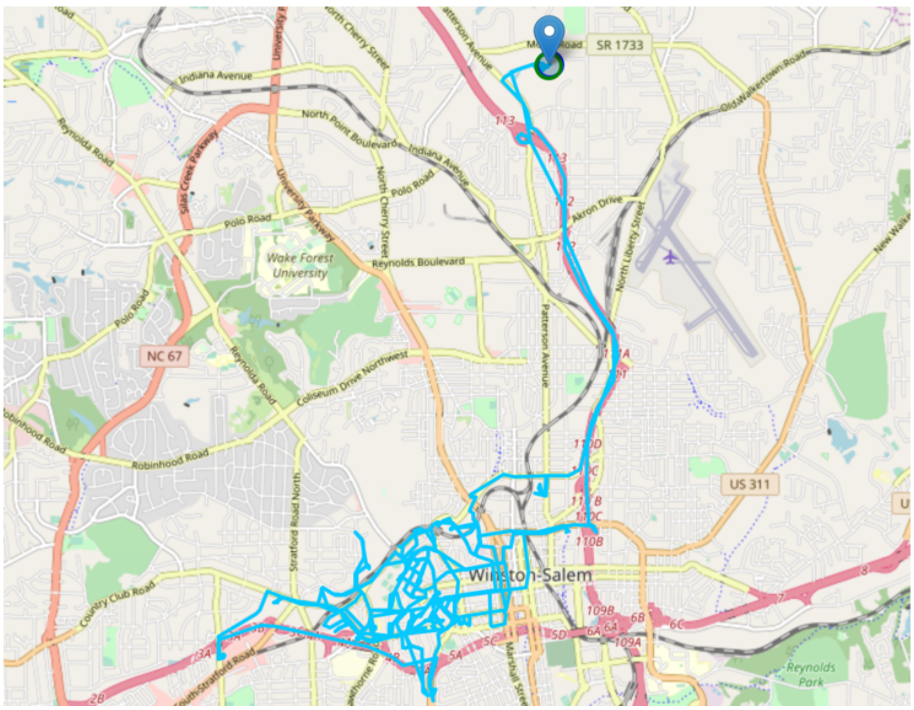
In Figure 4.18, the GPS trajectories of example historical delivery trips of the three test vehicles are shown. For vehicle R1 and R2, the depot is connected with the delivery area by a highway, so the driving condition consists of both highway and urban driving. For vehicle R3, the depot is near to the delivery area so that the driving condition is much like urban driving compared with the other two. These observations closely match the intuition behind kinetic intensity: the average value of kinetic intensity of both R1 and R2 are higher than the R3, which is mostly urban driving.

After training, the trained algorithms were tested on new trips on the same routes. The testing framework is illustrated in Figure 4.19. During each testing trip, the trip information was uploaded from the vehicle every 5s during the driving, which contained the state s_t information. The trained RL algorithm took the s_t as input and calculated the corresponding a_t. Then, the L_{set} value stored in an FTP (file transfer protocol) server was updated according to the action a_t. The EREV requested the value of L_{set} every 30 minutes.

Table 4.3: Statistics of the three selected routes

Vehicle	$\overline{D}$ $(mile)$	$\overline{KI}$ (km^{-1})	$\overline{EI}$ $(kWh/mile)$	Number of trips
R1	46.0	0.66	1.08	52
R2	49.6	0.89	1.03	58
R3	46.3	1.06	0.92	62


Market Street
US 117
NC 132
US 74
US 17
College Acres
University of North Carolina at Wilmington
Barclay Hills
Devon Park
Forest Hills
Kerr Avenue
US 17
US 76


SR 1733
Indiana Avenue
North Point Boulevard
North Cherry Street
Polo Road
Akron Drive
Reynolds Boulevard
Wake Forest University
University Parkway
Silas Creek Parkway
Reynolda Road
NC 67
Patterson Avenue
North Liberty Street
Coliseum Drive Northwest
Robinhood Road
Stratford Road North
Country Club Road
US 311
Winston-Salem
Reynolds Park
Old Walkertown Road
New Walkertown Road

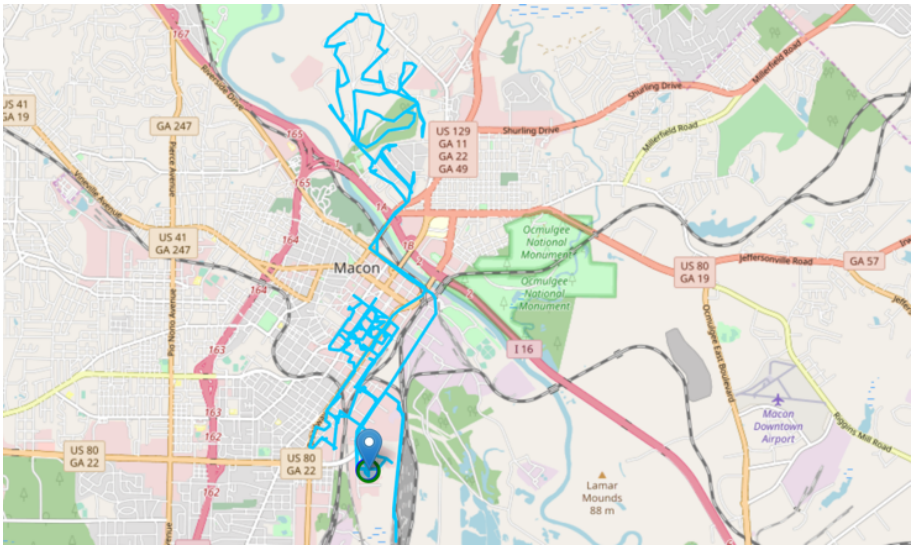


Figure 4.18: Example GPS trajectories of testing vehicle R1, R2 and R3.

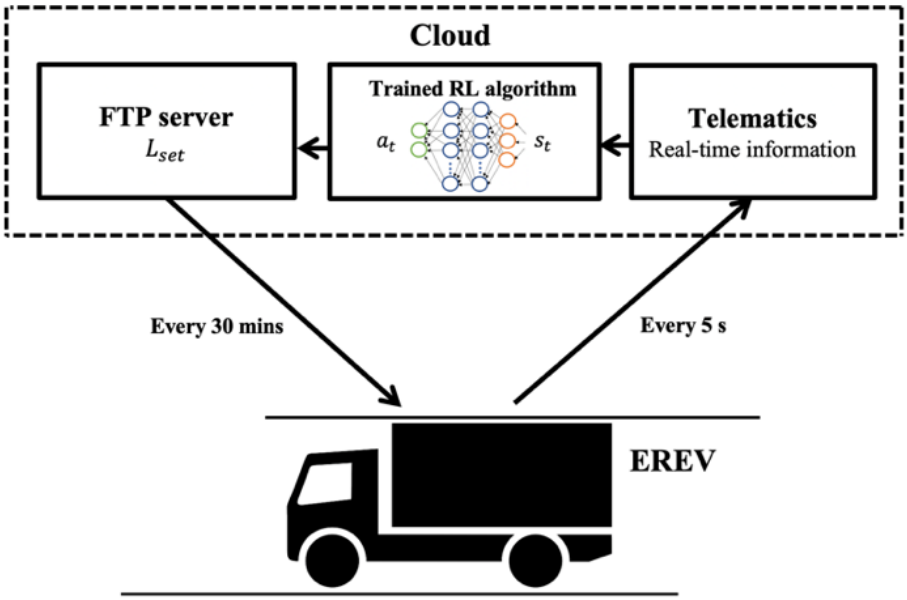


Figure 4.19: Diagram illustrating the testing framework of DDPG.

The velocity profile of one example testing trip of R1 is shown in Figure 4.20, and the L_{set} value along the trip and the corresponding SOC_{ref} is shown in Figure 4.21. It can be shown that on this particular trip, the L_{set} is first decreased to a low value to prevent any fuel use in the first half of the trip. Then, it is increased to make the SOC_{ref} higher than

the real-time SOC to charge the vehicle. After the battery is charged to some value that the RL algorithm considers sufficient according to its interaction with the historical trips, the L_{set} is lowered again and there is no more fuel use in the rest of the trip. The trained RL algorithm can adaptively tune the L_{set} during the trip according to the real-time information of the vehicle and it gains experience from the interaction with the historical trips. The experience is embedded as the weights in the actor network $\mu(s_t|\theta^{\mu})$.

It can also be observed that the L_{set} value is not changed at every 30 minutes; this can be explained by the $r_{a,t}$ term in the defined reward function: to maximize the cumulated future reward, the RL algorithm learned an efficient strategy that requires as little updates as possible.

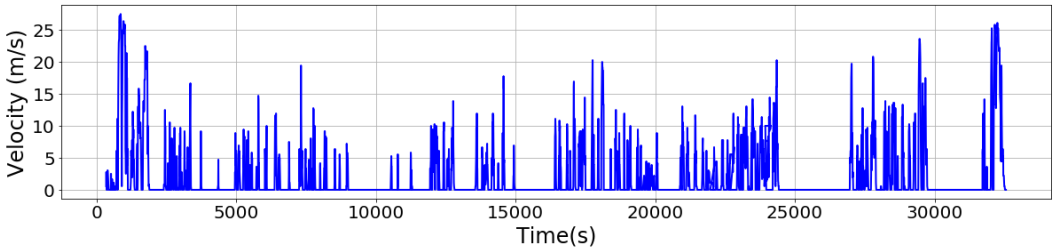


Figure 4.20: Velocity profile of the example test trip.

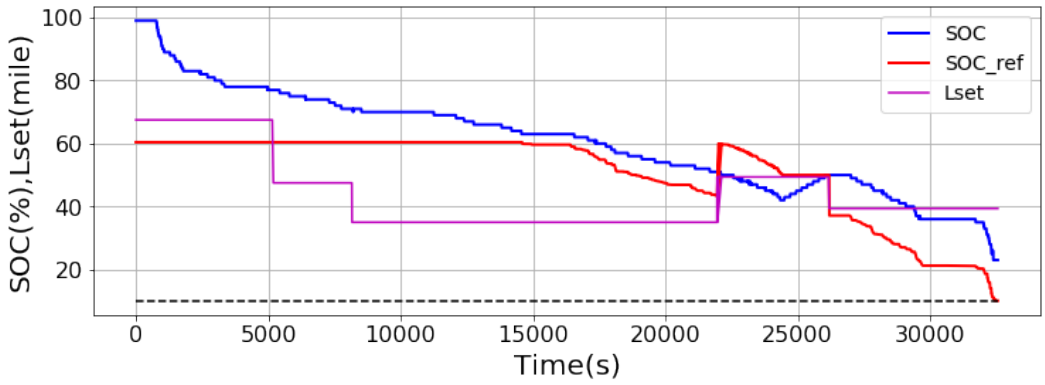


Figure 4.21: SOC, SOC_{ref} and L_{set} profile for the example test trip.

In Figure 4.22, the SOC, fuel use, and engine state under the RL and baseline condition are compared. The data of the RL condition is from the testing trip and the data of the baseline condition is from simulation.

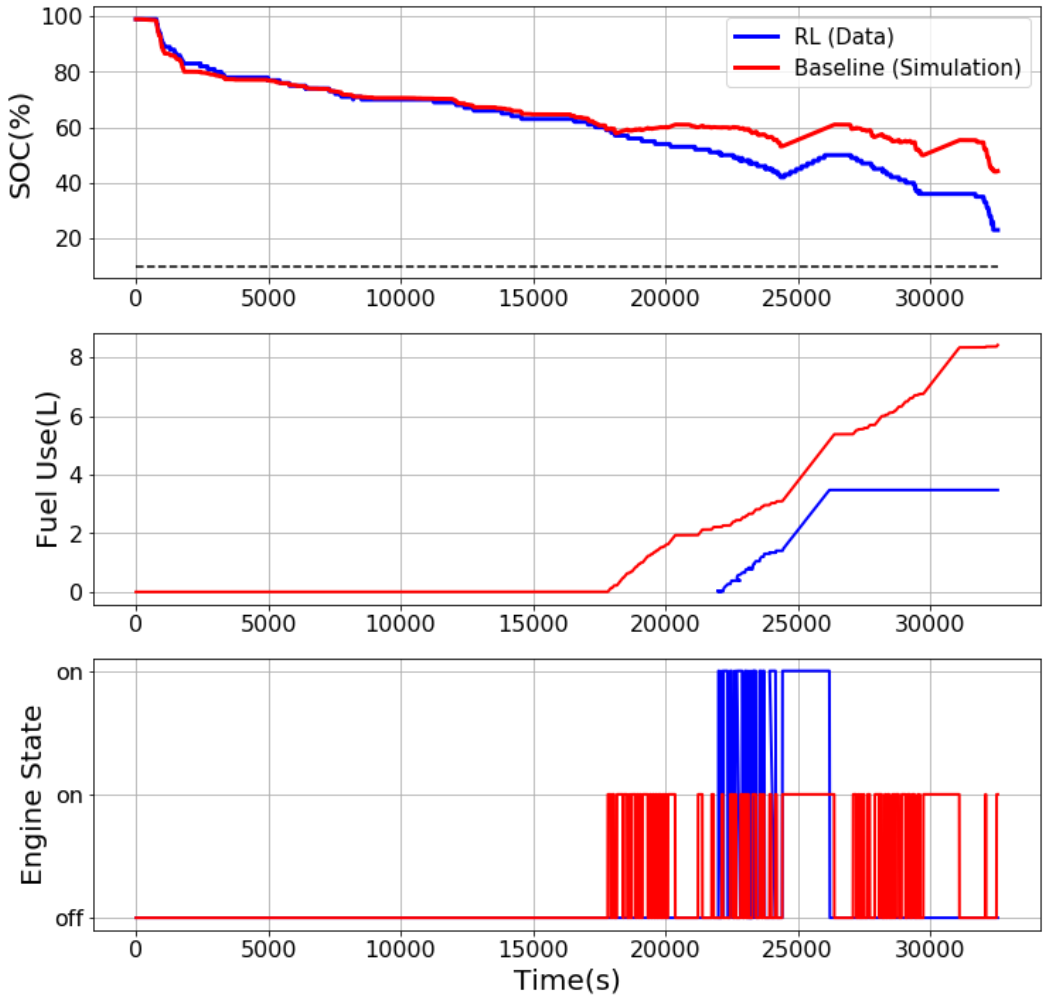


Figure 4.22: SOC, fuel use and engine state comparison between RL method and baseline method. In the last figure, two "on" positions are labeled on the y-axis to differentiate the two conditions.

In Figure 4.23, the MPGe and fuel use for the three test vehicles under RL and baseline conditions are compared. The average MPGe improvement is shown in Figure 4.24, and

the black line on each bar indicates one standard deviation from the mean value. The detailed performance improvements are also summarized in Table 4.4.

In this chapter, under the assumption that the historical delivery trip data can represent future conditions well, the energy management problem can be formulated as a sequential decision-making problem and solved by DRL algorithms.

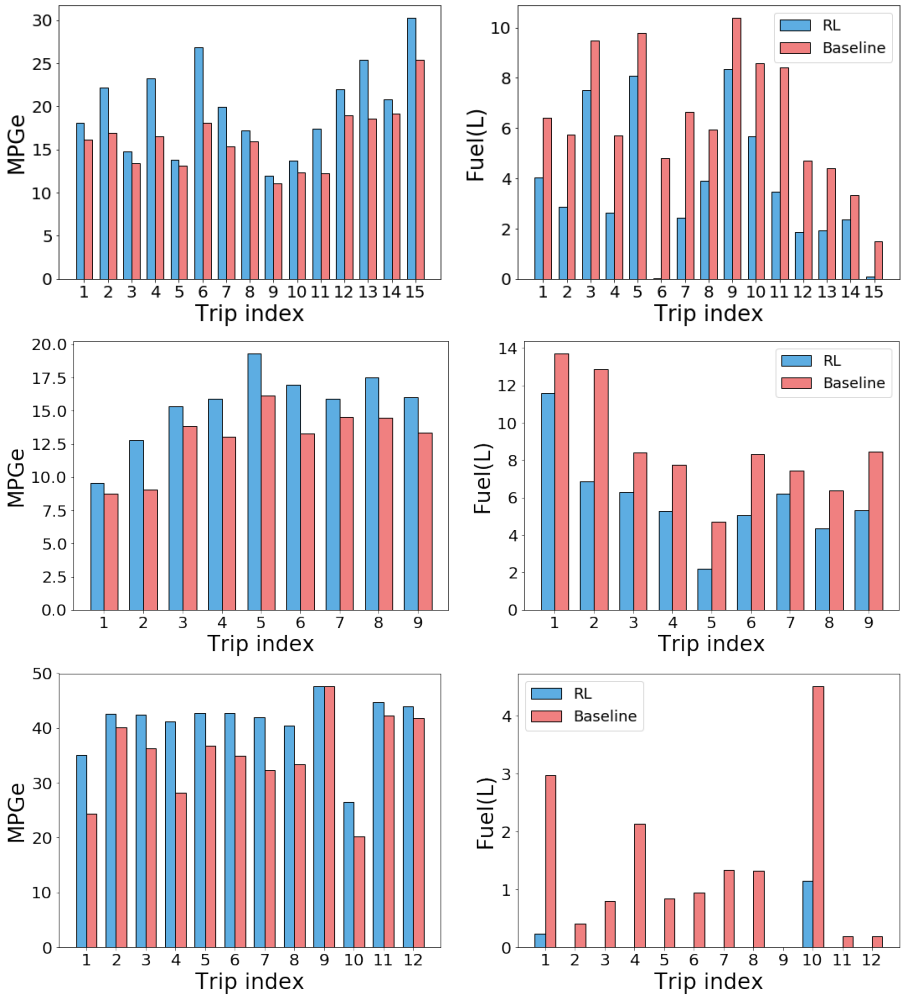


Figure 4.23: MPGe and fuel use comparison of all test trips of R1, R2 and R3.

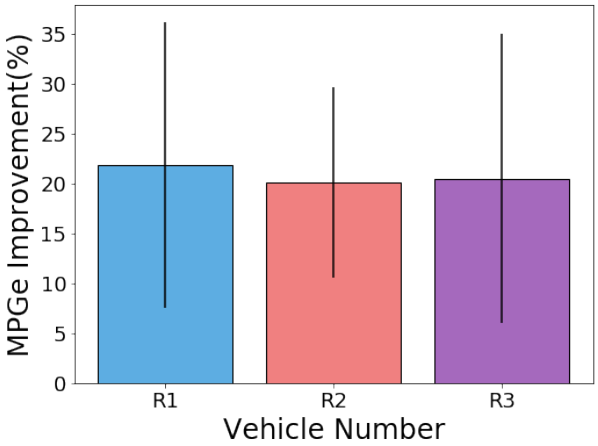


Figure 4.24: Mean and one standard deviation of MPGe improvement of R1, R2 and R3.

Table 4.4: Summary of testing results

Vehicle	Average MPGe improvement (%)	Average fuel use reduction (%)	Number of trips
R1	21.8	48.7	15
R2	20.1	33.1	9
R3	20.5	88.9	12

5. SAFE AND ROBUST ALGORITHMS FOR CONNECTED ENERGY MANAGEMENT SYSTEMS

In recent years, deep reinforcement learning algorithms have been applied to many applications in the area of transportation including traffic light cycle control [112][113], autonomous driving [114][115], bottleneck decongestion [116], online ride order dispatching [117], and energy management of hybrid vehicles [118][119]. It has been a popular research topic especially for applications involving processes of sequential decision making. However, although successes have been achieved with various degrees, due to some characteristics of real-world problems and DRL, it is still not easy to deploy DRL systems in real-world applications, especially for the area of transportation, where robustness and safety are of great importance. This chapter mainly identifies the problems that might be encountered when deploying trained DRL systems into real-world conditions and gives potential solutions and research directions to the challenges. Safety in the area of AI is becoming more and more important in recent years and some related work can be found in [134]-[137].

First, model uncertainty is considered [120][121]. Most prior work about applying DRL algorithms use a simulation environment to train the intelligent agent. This is inevitable due to at least two reasons. First, under simulation environments, the interaction process can be accelerated and paralleled so that it is not restricted by the actual time available. In this way, the amount of data that DRL algorithms need can be generated. Second, training an agent in a real-world environment can be expensive and dangerous. Hardware could be destroyed during the exploration process, when any action could be

chosen by the exploration strategy. Consequently, the problems related to the training in the simulation environment should be considered carefully. For model-based methods, the performance of the agent highly depends on the quality of the model of the environment, and usually, building the model itself is a difficult research problem. Errors will accumulate from the process of building the model and from the process of training the agent using the model. Similarly, although model-free methods do not need an explicit model of the environment, the collected data generated from the interaction between the agent and the simulator may not represent all possible conditions in the real-world scenario. Consequently, this work deals with model uncertainty: training a policy that not only gives actions during a task, but also the confidence level about the action under the current state, so that novel and out-of-distribution states could be captured.

Second, a risk-aware algorithm is developed to deal with environment uncertainty [120]. Under simulation environments, agents are trained to yield high average scores. However, giving an average satisfying results are sometimes not acceptable under real-world problems, especially for safety-critical applications like autonomous driving. Therefore, risk-aware strategies are developed in this work, making decisions using the metric of conditional value at risk (CVaR) [128], which includes the average score as a special case. This is enabled by distributional reinforcement learning algorithms [124]-[127], which models the whole return distribution given state-action pairs instead of just modelling the expected value (Q-value).

Lastly, the effects of adversarial examples [139][140] are investigated. Neural networks are core components of DRL algorithms so that the problems related to them

also need to be considered in DRL systems. In this work, the effects of adversarial examples on DRL-based energy management systems are investigated, which demonstrate NN-based policies are not stable under designed noise.

In the rest of this chapter, model uncertainty, environment uncertainty, and adversarial attacks are discussed respectively, and the corresponding solutions or research directions are followed.

5.1. *Uncertainties in Reinforcement Learning*

5.1.1. Model Uncertainty

There are two kinds of uncertainty in an RL problem setting: model uncertainty and environment uncertainty [120][121]. Model uncertainty (also called epistemic uncertainty) can be explained away with enough data, and it is usually handled under a Bayesian framework [74]. It can be beneficial to RL applications in at least two aspects. First, it helps with exploration during the training process. In [122], a Bayesian ensemble of neural networks (NNs) was used to quantify the predictive uncertainty and facilitate the exploration strategy of Thompson sampling. Second, model uncertainty can help identify novel and out-of-distribution data during testing, providing a safer algorithm. In [115], an autonomous driving system was able to identify novel obstacles during testing and perform safer and more conservative actions. In this work, our goal is to exploit the second benefit.

A simple but inspiring regression problem is shown in Figure 5.1. There is a nonlinear mapping from x to y and the eight red dots represent the available data points. An ensemble of neural networks with imposed diversity are trained to capture the nonlinear

mapping by learning from the available data [122]. It can be observed that, in the region where data exists, different NNs agree well while in the region there is no data, NNs diverge which indicates high model uncertainty. This result is consistent with our intuition: with dense data, our regression model can perform well, and we can be confident about it. However, with spare data or no data, the result of the regression model is not trustworthy, and it is important that we are aware of it.

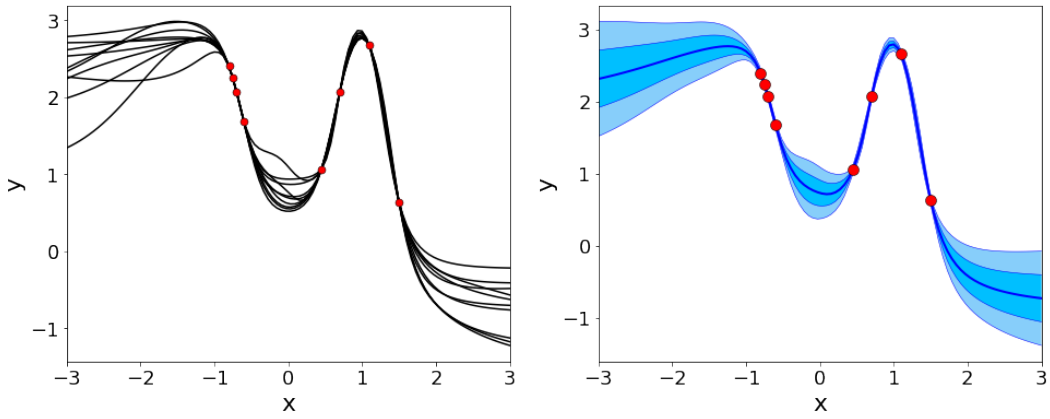


Figure 5.1: Ten NNs are trained. The prediction of each individual NN is shown on the left figure. The mean prediction is shown on the right figure with one and two standard deviation represented by the shaded region.

5.1.2. Environment Uncertainty

Risk, also called environment uncertainty and aleatoric uncertainty [120], is an intrinsic property of a system. For example, no matter how many times we flip a coin, the results are still stochastic. In a standard RL algorithm, the decisions are made based on the expected long-term return, which means if we have many chances to choose an action at a certain state, it will lead to the highest possible return on average. However, for real-world transportation applications, it is not enough to deliver an algorithm that works well

on average, especially for safety-critical applications. Therefore, it is very important to be able to estimate the full return distribution, not just one expected return value. C51 [124][125], the first distributional RL algorithm integrated into the popular DQN algorithm achieved state-of-the-art performance on video games. It calculates the probabilities for a set of predefined returns. Another approach used quantile regression (QR) to calculate a fixed number of quantiles of a return distribution with fixed probabilities [126]. This method achieved better theoretical guarantee in convergence compared with C51. Implicit quantile network (IQN) removes the limitation of a fixed number of quantiles [127] and the quality of the estimated return distributions depends on the network structure.

Figure 5.2 illustrates a one-step decision-making problem which can be considered as the simplest RL problem. At state s_1, actions can be chosen from a_1 and a_2. If the only given is the expected reward for each action, then a_1 would be taken. However, if the underlying return distributions for both actions are given, a_2 might be taken depending on the application scenario as there is no risk in taking a_2. In addition, although the expected reward for taking action a_1 is +3, the reward of +3 will never be achieved as the reward is either +5 or -5 in reality. This also reflects one downside of the expected value: it is sometimes unrealistic to achieve.

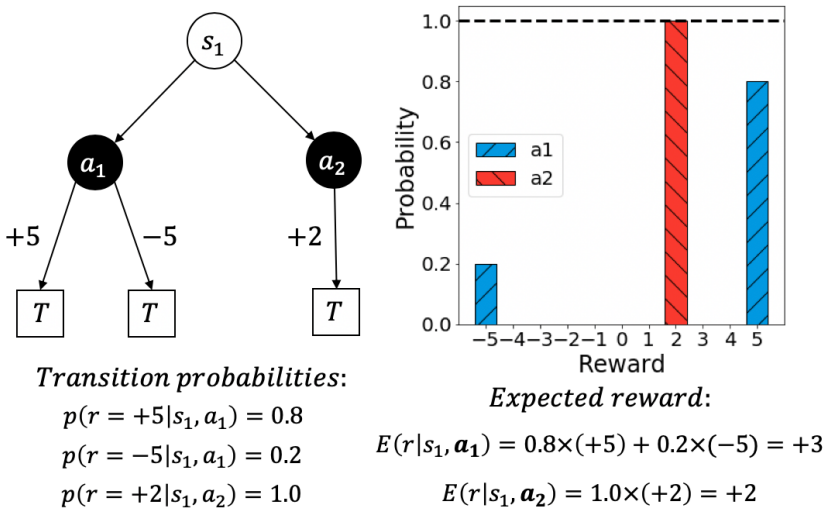


Figure 5.2: A one-step decision-making problem showing the importance of estimating the uncertainty associated with different actions. The T represents terminal states.

Instead of discrete values, the possible return is usually modeled as distributions in complex problems. If the return distribution is multi-modal or has a high variance, only considering the expected value may lead to bad decisions in a real-world scenario. This is because what the agent will receive during one task is just one sample from the distribution. In Figure 5.3, it can be observed that although the expected values of action 2 in both cases are higher than action 1, action 2 may lead to much worse results due to multi-modality (left) and high variance (right).

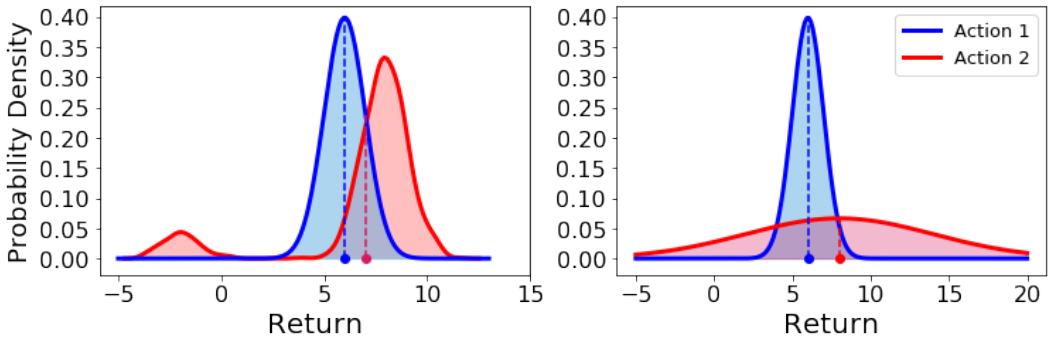


Figure 5.3: Multi-modality (left) and high variance (right) return distributions.

5.2. *Risk-aware Energy Management*

To deal with risk, which is also called the environment uncertainty, decisions should be made according to metrics beyond the expected value, and this needs the full return distribution given state-action pairs. To model the full return distribution instead of just estimating the mean, two distributional RL algorithms are adapted in this work. The first algorithm is C51 [124], whose results are easy to understand and visualize since the return distributions are modeled explicitly. The second is IQN [127], whose results are less straightforward to follow, since the return distributions are implicit and utilized by sampling. Nevertheless, it is very convenient to build risk-aware strategies.

5.2.1. Categorical Distributional Reinforcement Learning

In the C51 algorithm, the Q-network outputs a discrete probability distribution on predefined N atoms: $\{z_i = V_{min} + i\Delta z : 0 \leq i < N\}$ with $\Delta z = (V_{max} - V_{min})/(N-1)$. The set of atoms z_i represents the predefined possible long-term return with $z_0 = V_{min}$ and $z_{N-1} = V_{max}$ as boundaries. Given a state s_t and an action a_t, the distribution is:

$$p(s_t, a_t) = C(s_t, a_t|\theta^Q), \tag{5.1}$$

which has N components and whose ith component $p_i(s_t, a_t)$ is the probability of getting the return z_i if action a_t is taken at state s_t. Figure 5.4 illustrates an example return distribution for a (s_t, a_t) pair with $N = 8$.

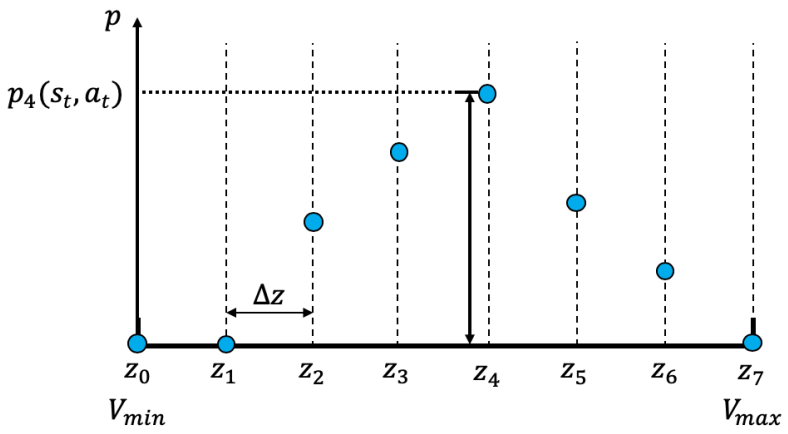


Figure 5.4: An example return distribution for a state-action pair with $N = 8$.

The expected long-term return for a (s_t, a_t) pair is calculated as a weighted sum of the predefined N atoms according to the estimated probability $p(s_t, a_t)$:

$$Q(s_t, a_t) = \sum_{i=0}^{N-1} z_i p_i(s_t, a_t)\,. \tag{5.2}$$

For the training of C51, the error to be minimized is based on transition pairs (s_t, a_t, r_t, s_{t+1}) generated during the interaction process like standard RL algorithms. The difference is, in this case, the distance between two distributions should be minimized. The cross-entropy of the current estimation of return distribution for (s_t, a_t) and its target distribution is minimized by gradient-based method. In this work, the standard Q-network in the DDPG is replaced with C51. For simplicity, the resulting algorithm will be noted as D3PG (Distributional DDPG). The detailed training algorithm is shown in Algorithm 4. The detailed method to calculate the cross-entropy loss is shown in Algorithm 3 and Figure 5.5 illustrates the process in its for loop.

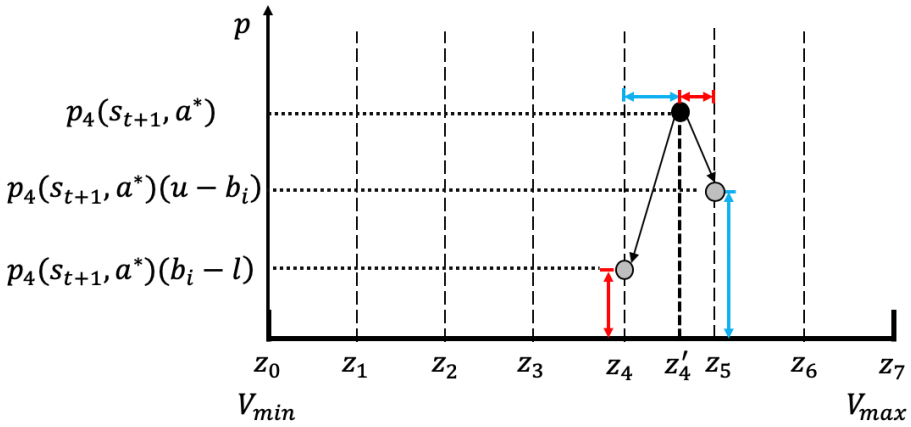


Figure 5.5: One step in the for loop in Algorithm 3. The blue arrow and red arrow indicate the distribution of the probability is according to its distances with the nearest two atoms.

Algorithm 3 Calculate cross-entropy loss

Input A transition pair (s_t, a_t, r_t, s_{t+1})
Initialize $V_{min} = -15$, $V_{max} = +3$, $N = 51$
Initialize the target probabilities: $m_i = 0$, $i \in 0, \dots, N-1$
$p(s_{t+1}, a^*) = C'(s_{t+1}, a^* | \theta^{C'})|_{a^* = \mu'(s_{t+1}|\theta^{\mu'})}$
$p(s_t, a_t) = C(s_t, a_t | \theta^C)$
For i = 0 to $N-1$ **do**
  $z_i' = [r_t + \gamma z_i]_{Vmin}^{Vmax}$
  $b_i = (z_i' - V_{min})/\Delta z$
  $l \leftarrow floor(b_i), u \leftarrow ceil(b_i)$
  $m_l \leftarrow m_l + p_i(s_{t+1}, a^*)(u - b_i)$
  $m_u \leftarrow m_u + p_i(s_{t+1}, a^*)(b_i - l)$
End For
Output $L = -\sum_{i=0}^{N-1} m_i log p_i(s_t, a_t)$ #Cross-entropy loss

Algorithm 4 Distributional DDPG (D3PG)

Initialize the critic network $C(s, a|\theta^C)$ with random parameters θ^C
Initialize the target critic network $C'(s, a|\theta^{C'})$ with $\theta^{C'} = \theta^C$
Initialize the actor network $\mu(s|\theta^\mu)$ with random parameters θ^μ
Initialize the target actor network $\mu'(s|\theta^{\mu'})$ with $\theta^{\mu'} = \theta^\mu$
Initialize the replay buffer B to capacity D with no transitions
Initialize $\varepsilon = 0.3$, $\tau = 0.001$, $D = 10^5$, $K = 48$
For epoch = 1, M **do**
. **For** trip = 1, N **do**
. . Get initial state s_0 from current trip data
. . **While** $t < T$, **do**
. . . Calculate action $a_t = \mu(s_t|\theta^\mu) + \varepsilon \cdot N(0,1)$
. . . Clip a_t to the range of [-1, +1]
. . . Update current L_{set} according to $A_{max} \cdot a_t$
. . . Run the vehicle model for 2000s with updated L_{set}
. . . Return state s_{t+1} and reward r_t
. . . Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer B
. . . Sample K random transitions (s_i, a_i, r_i, s_{i+1}) from B
. . . Call Algorithm II to calculate the loss L_i on the K transitions
. . . Update the critic network by performing gradient descent
. . . on the calculated cross entropy loss:
. . . $L_{mean} = \frac{1}{K}\sum_{i=1}^{K} L_i$
. . . Calculate probabilities associated with z_j, $j \in 0,1 \ldots N-1$
. . . for each sampled transition pair:
. . . $p(s_i) = C(s_i, a|\theta^Q)|_{a=\mu(s_i|\theta_\mu)}$
. . . Calculate Q-value $Q(s, a)$ for each sampled transition pair:
. . . $Q(s_i, a)|_{a=\mu(s_i|\theta_\mu)} = \sum_{j=0}^{N-1} z_j p_j(s_i)$
. . . Update the actor network using the sampled policy gradient:
. . . $\nabla_{\theta^\mu} J \approx \frac{1}{K}\sum_{i=1}^{K} \nabla_a Q(s_i, a)|_{a=\mu(s_i|\theta_\mu)} \nabla_{\theta^\mu} \mu(s_i|\theta^\mu)$
. . . Perform soft update on target networks:
. . . $\theta^{C'} \leftarrow \tau\theta^C + (1-\tau)\theta^{C'}$
. . . $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
. . . $s_t = s_{t+1}$
. **End For**
End For

The NN structure and hyperparameters are summarized in Table 5.1. The activation function softmax is used in the output layer of the critic as it can convert the outputs from the 51 output units to probabilities. The D3PG was trained with the same trips as the DQN and was tested with 66 unseen test trips. The average ideal MPGe for the test trips was 30.8 which was calculated with the full velocity profiles. The average MPGe achieved by

the D3PG was 28.3, only 8.1% less efficient than the condition of knowing the full velocity profiles. Figure 5.6 shows how D3PG performed on three test trips of varying distance. It can be observed that the distributional RL can perform well on these test trips like standard RL algorithms. It should be noted that, although the full return distributions are estimated by the critic, the actions are still based on the Q-values.

Table 5.1: Neural network structures and hyperparameters for D3PG

		Actor	Critic
Hidden layer 1	Number of units	64	64
	Activation function	ReLU	ReLU
Hidden layer 2	Number of units	48	48
	Activation function	ReLU	ReLU
Output layer	Number of units	1	51
	Activation function	tanh	softmax
Hyperparameters	Optimizer	Adam	Adam
	Learning rate	10^{-4}	
	Batch size	48	48

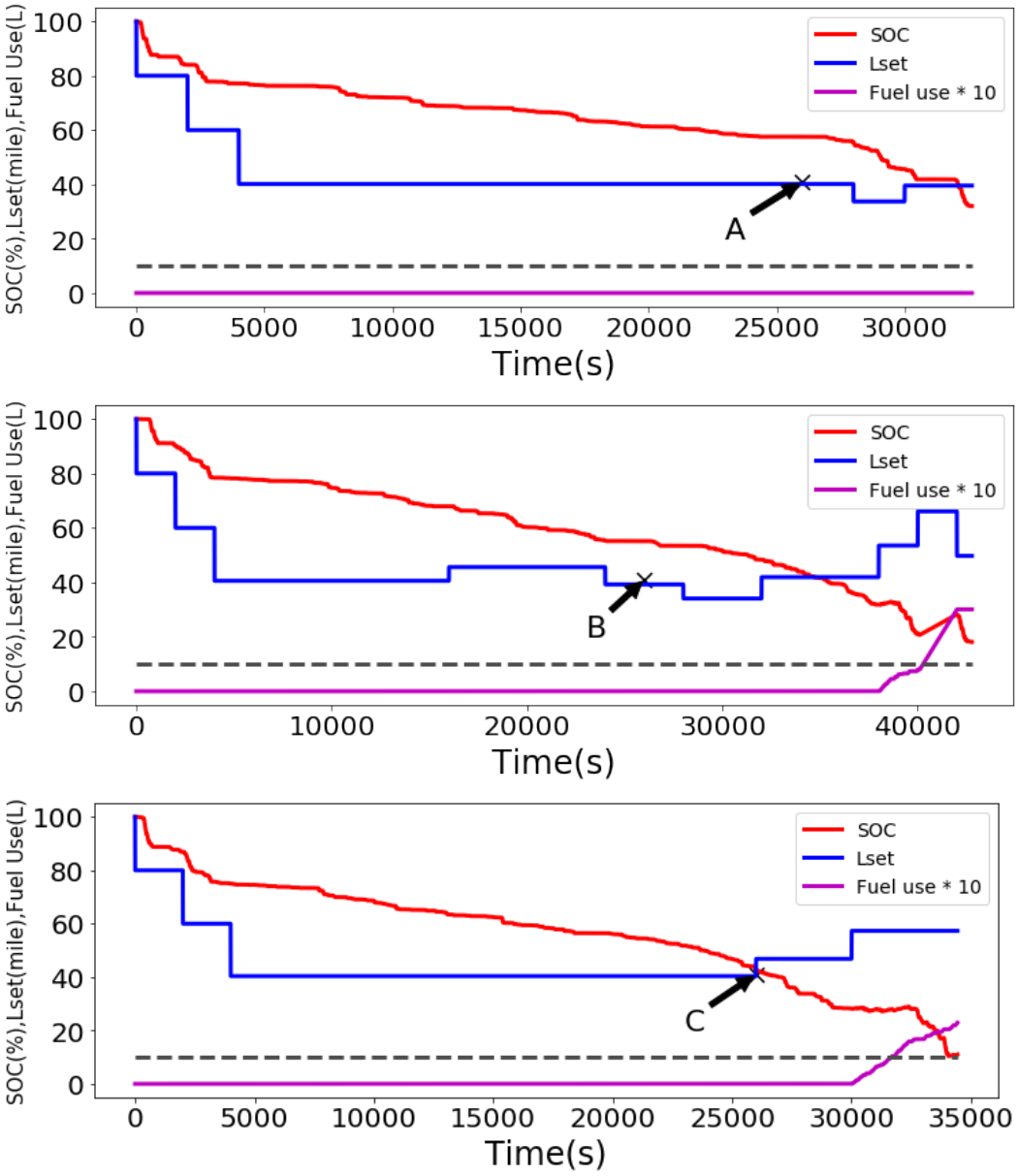


Figure 5.6: Performance of the D3PG on three test trips with distances of 35 miles(upper), 51 miles (middle) and 50 miles (lower). The states associated with A, B and C are all at time step 26000s. The dashed lines help to see how close is the real-time SOC with the value of 10%.

Figure 5.7 shows the Q-values for the actor action and two example actions under states A, B and C (calculated from the return distribution shown in Figure 5.8). It can be seen that in all cases, the actor action has the highest Q-value as expected. Standard value-based RL algorithms model Q-values directly, and actions are chosen with respect to it.

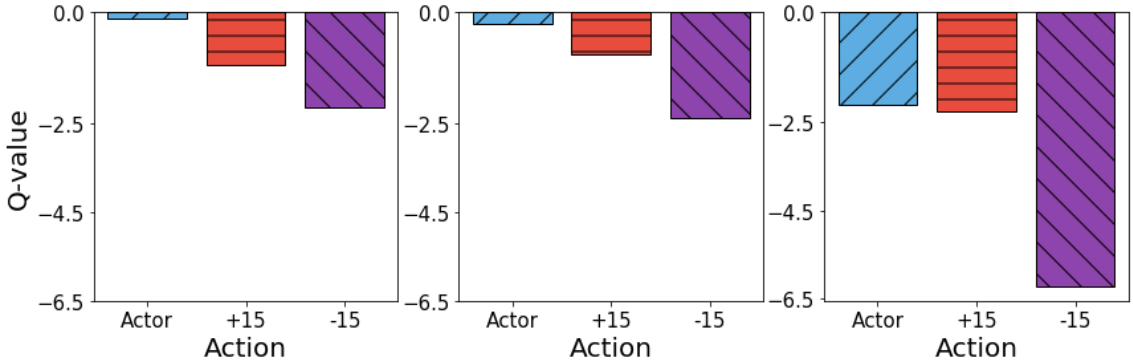


Figure 5.7: The Q-values of three actions under states A (left), B (middle) and C (right). The actor action is 0 for states A and B (do not change the L_{set}) and +6.5 for state C.

We modeled the underlying full return distribution by the D3PG algorithm. Figure 5.8 shows the return distributions for state A, B and C for the three actions. For conditions like state A and B, the distributions are nearly unimodal so that the expected value (Q-value) can represent it well. However, for conditions like state C (distributions are shown separately in Figure 5.9), the return distributions are approximately bimodal, and the long-term reward associated with the two peaks are significantly different.

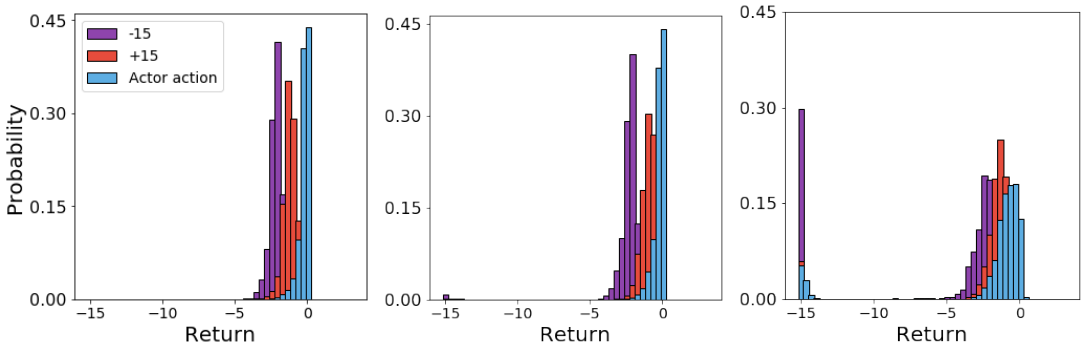


Figure 5.8: Return distributions of an actor action and two example actions for state A (left), state B (middle) and state C (right).

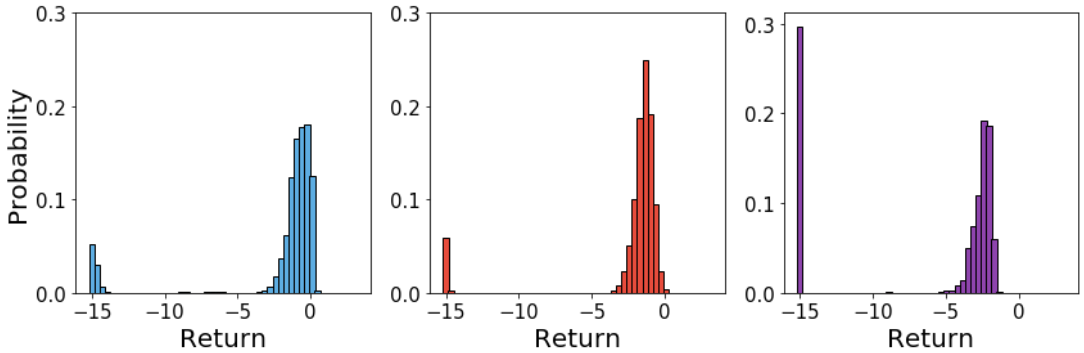


Figure 5.9: Return distribution of actor action (left), +15 (middle) and -15 (right) for state C.

From the reward function, it can be concluded that the distribution density concentrated around -1 is mainly due to excessive fuel use and the density around -15 is due to the condition of $SOC < 10\%$. From this view, taking the actor action had a higher probability of running out of battery compared with the +15 action as less fuel was used. It can be observed that if the L_{set} is increased by 15 at state C, some amount of fuel would be used in the next 2000s while by taking the actor action, no fuel was used. The potential risk of running out of battery for the actor action was offset by the benefits of a higher possibility of not using excessive fuel. The trained D3PG uses the action that has a higher

risk of running out of battery to get a better performance with respect to the expected long-term return. As discussed previously, making decisions with respect to the expected value can yield average higher returns. However, in the real-world condition, more conservative actions might be preferred under some conditions. By modelling and visualizing the return distributions, the necessity of considering the environment uncertainty is demonstrated.

Although the return distributions modeled by C51 is easy to understand, it is not convenient to use to build risk-aware strategies. One naïve method is as follows. By modeling the full return distributions, we can follow the actor action when the uncertainty is low and choose more conservative actions when the uncertainty is high. One simple metric that can represent the magnitude of uncertainty is the variance. The variance of the return distribution of the three actions under states A, B and C are shown in Figure 5.10. As can be seen, the Q-values (expected value) represent the return distribution well when the variance is low as in states A and B. If the variance is 0, i.e., all the probability is concentrated on one value, the Q-value carries all the information about the return distribution. In conditions of high variance, as in state C, the Q-value loses the important spread information which contains the probability of different possible results.

In this work, the importance and benefits of using distributional RL algorithm is shown by visualizing the return distributions under different states. The return distributions can be multi-modal with high variance which shows the probability of different possible results. Under these states, decisions based on the expected value might not be suitable

for real-world applications. Consequently, a risk-aware strategy is develop in the next part of work.

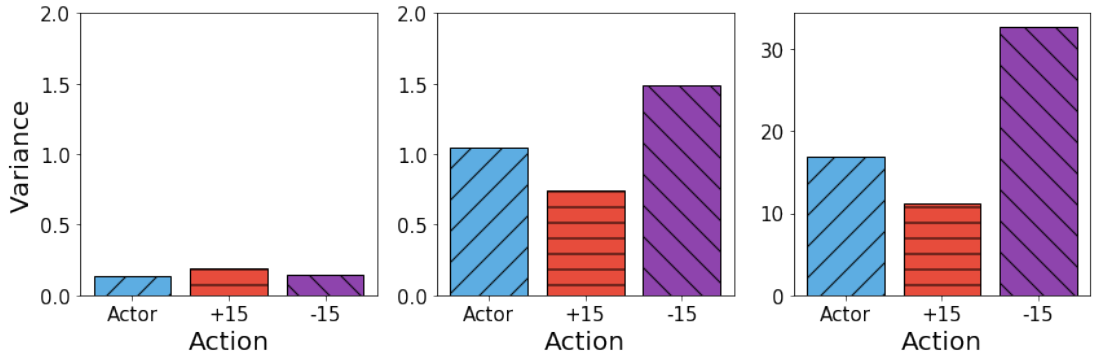


Figure 5.10: The variance of return distributions of three actions under state A(left), state B(middle) and state C(right).

5.2.2. Implicit Quantile Network

The IQN parametrized by θ takes a state-action pair (s, a) and a sample τ from a uniform distribution $U([0,1])$ as inputs and outputs a sample $Z_\tau(s, a; \theta)$ from the implicitly defined return distribution $Z(s, a; \theta)$. The action-value $Q(s, a; \theta)$ can be estimated by multiple samples of $Z_\tau(s, a; \theta) \sim Z(s, a; \theta)$ as:

$$Q(s, a; \theta) = E_{\tau \sim U([0,1])}[Z_\tau(s, a; \theta)] \approx \frac{1}{K} \sum_{i=1}^{K} Z_{\tau_i}(s, a; \theta), \tag{5.3}$$

where index i represents the ith sample from $Z(s, a; \theta)$, and K represents the total number of samples. Figure 5.11 illustrates the difference between a standard DRL algorithm DQN and the IQN.

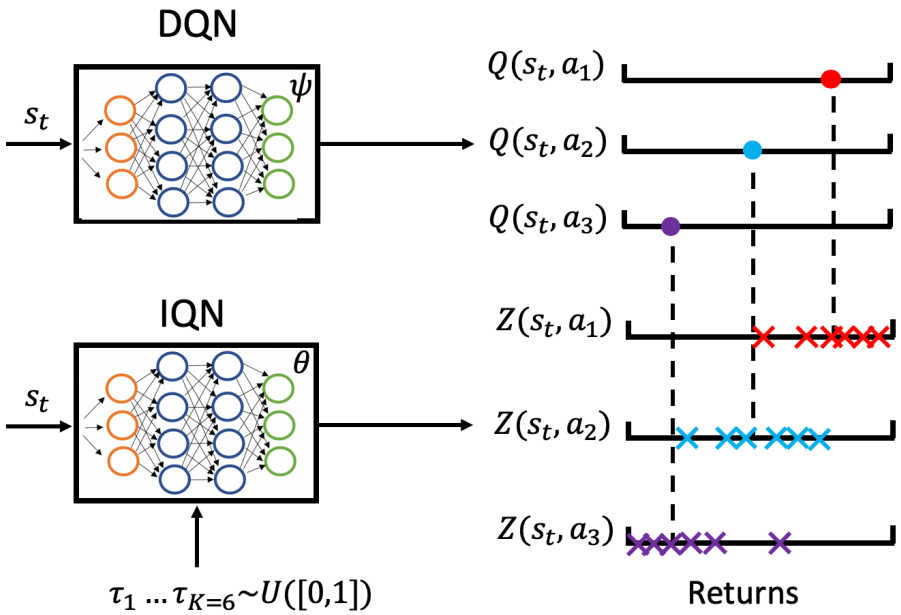


Figure 5.11: Illustration of the differences between DQN and IQN. The DQN outputs the expected value for a given state-action pair while IQN outputs samples from the implicitly modeled return distribution whose mean is the output of DQN.

The return distribution for a given state-action pair is modeled implicitly by estimating the quantile function $QF_{Z(s,a)}(\tau)$, also known as the inverse cumulative distribution function (inverse CDF). Assume the CDF of the return distribution is $F_{Z(s,a)}: Z_\tau(s,a) \rightarrow [0,1]$, then:

$$Z_\tau(s,a) = QF_{Z(s,a)}(\tau) = F^{-1}_{Z(s,a)}(\tau), \tag{5.4}$$

which indicates a mapping from τ sampled from $[0,1]$ to $Z_\tau(s,a)$. A simple example is shown in Figure 5.12. Assume the $QF_{Z(s,a)}$ of a return distribution $Z(s,a)$ for a state-action pair is given, sampling τ from uniform distribution $U([0,1])$ and then feed them into $QF_{Z(s,a)}(\tau)$ equals to sampling returns from the underlying return distribution

according to its probability density function (blue dashed line). The sampled returns (red dots) can be used to calculate action-values and other statistics.

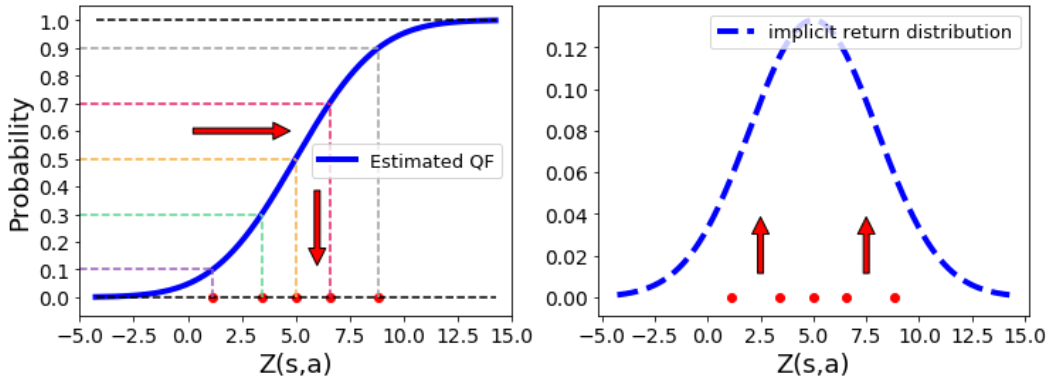


Figure 5.12: A simple example illustrating how to sample returns $Z_\tau(s,a)$ from the implicitly modeled return distribution. Assume five τ values: 0.1, 0.3…0.9 are sampled from the uniform distribution $U([0,1])$, five values of $Z_\tau(s,a)$ are calculated by feeding the five τ values into the estimated $QF_{Z(s,a)}(\tau)$. With enough number of samples, the mean and other statistics of the underlying return distribution can be estimated accurately.

The quantile function $QF_{Z(s,a)}(\tau)$ is estimated by IQN. It is trained by the quantile regression (QR) loss function. In Figure 5.13, a simple linear regression example illustrates the difference between the common mean squared loss (MSE) and QR loss.

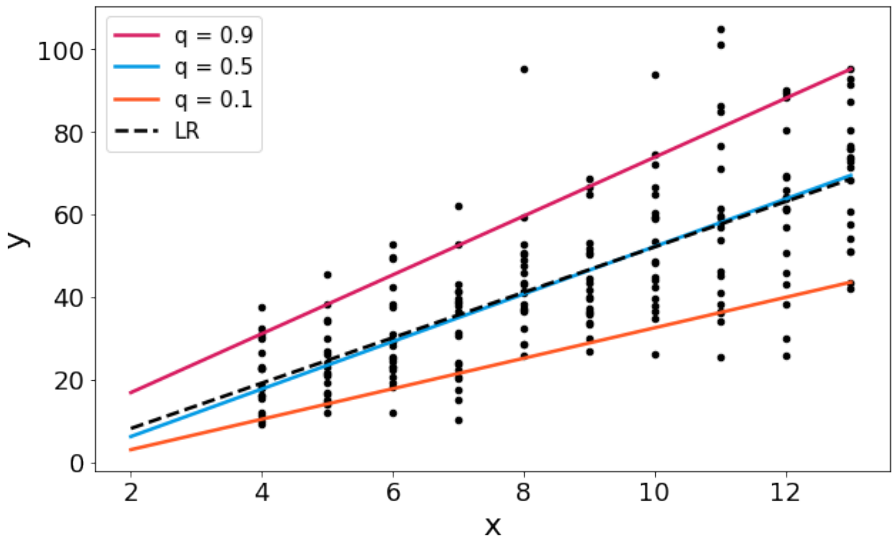


Figure 5.13: Comparison of fitting an artificial dataset with linear regression and quantile regression with 0.9, 0.5 and 0.1 quantiles. For each x, twenty y values are sampled from a gamma distribution. For linear regression (LR), the MSE is minimized, which tries to estimate the conditional mean of the underlying assumed Gaussian distribution for each input x. For quantile regression, there is no assumption about the underlying distribution and the conditional quantile values are estimated.

For standard RL algorithms, the square of the temporal difference (TD) error is minimized where the TD error is:

$$\delta_t = r_t + \gamma max_a Q(s_{t+1}, a; \psi') - Q(s_t, a_t; \psi). \tag{5.5}$$

For IQN, the QR loss of the TD error shown below is minimized:

$$\delta_t^\tau = r_t + \gamma max_a Z_{\tau'}(s_{t+1}, a; \theta') - Z_\tau(s_t, a_t; \theta). \tag{5.6}$$

The detailed QR loss and MSE for the supervised regression problem and DRL problem is summarized in Table 5.2. The detailed algorithm to train the IQN in the EMS problem is shown in Algorithm 5.

Table 5.2: Comparison of QR loss and MSE under two problem settings

	QR loss	MSE
Regression	$(\tau - \mathbb{I}_{y<wx+b})[y-(wx+b)]$	$[y-(wx+b)]^2$
Deep RL	$(\tau - \mathbb{I}_{\delta_t^\tau<0})\delta_t^\tau$	δ_t^2
Goal	Estimate the τth quantile	Estimate the mean

Algorithm 5 Implicit Quantile Network

Initialize IQN and target IQN with same set of random parameters θ, θ'
Initialize the replay buffer B to capacity D with no transitions
Initialize $\varepsilon = 1.0$, $K = 32$, $D = 5e4$, $n = 48$, $M = 800$, $N = 52$
For epoch = 1, M **do**
. **For** trip = 1, N **do**
. . Get initial state s_0 from current trip data
. . $t = 0$
. . **While** $t < T$, **do**
. . . With probability ε, randomly select an action a_t
. . . Otherwise select $a_t = argmax_a \sum_{i=1}^{K} Z_{\tau_i}(s_t, a; \theta)$
. . . Update current L_{set} according to a_t
. . . Run the vehicle model for 2000s with updated L_{set}
. . . Return state s_{t+1} and r_t
. . . Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer B
. . . Sample n transition pairs from B
. . . **For** each transition pair (s_j, a_j, r_j, s_{j+1}), **do**
. . . . $Z_{target,j} = \begin{cases} r_j & \text{if trip terminates at step } j+1 \\ r_j + \gamma max_a Z_{\tau_j'}(s_{j+1}, a; \theta) & \text{otherwise} \end{cases}$
. . . . $\delta_j^{\tau_j} = Z_{target,j} - Z_{\tau_j}(s_j, a_j; \theta)$
. . . **End For**
. . . Perform a gradient descent step on $\sum_{j=1}^{n}(\tau_j - \mathbb{I}_{\delta_j^{\tau_j}<0})\delta_j^{\tau_j}$
. . . Perform soft update on the target network
. . . $s_t = s_{t+1}$
. . $\varepsilon = \max(0.01, 0.995 \times \varepsilon)$
. **End For**
End For

Conditional value at risk is used as the risk measure in this work [128]. It is defined as:

$$CVaR_{\eta}(s,a) = E_{r\sim Z(s,a)}\left[r \middle| r < F^{-1}_{Z(s,a)}(\eta)\right], \tag{5.7}$$

where η is a hyperparameter. If $\eta = 1.0$, the calculated $CVaR_{\eta}(s,a)$ is the expected value of the return distribution, which is equal to the action value $Q(s,a)$ and leads to a standard risk-neutral strategy. If $\eta < 1.0$, it calculates the expected value for the worst η cases, which leads to risk-averse strategies. A simple example is shown in Figure 5.14.

After the IQN is trained, risk-aware strategies can simply sample τ_i from $U([0,\eta])$ and choose actions by:

$$a_t = argmax_a \frac{1}{K}\sum_{i=1}^{K} Z_{\tau_i}(s,a;\theta)\,. \tag{5.8}$$

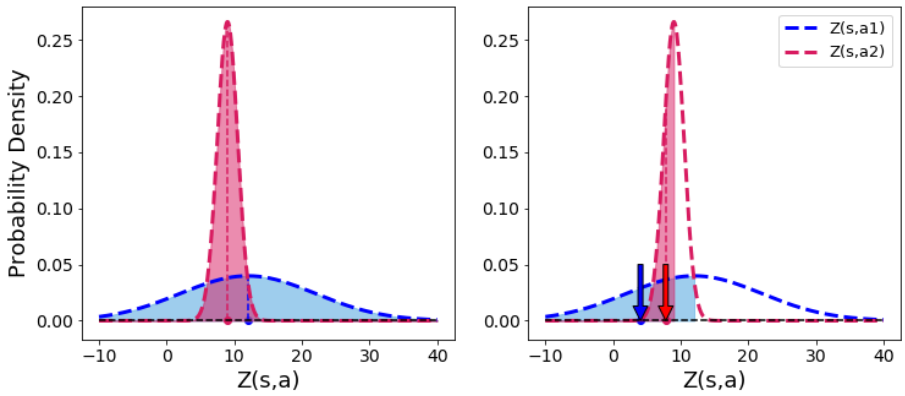


Figure 5.14: Illustration of CVaR with $\eta = 1.0$ (left) and $\eta = 0.5$ (right). It can be observed that although the expected value of return distribution $Z(s,a_1)$ is higher than that of $Z(s,a_2)$ (the position of the vertical dashed line), it does not hold true for $\eta = 0.5$ due to its high variance (annotated by arrows).

A feedforward NN was used to build the IQN which followed the structure of the original paper [127]. There were two hidden layers in the main structure with 64 and 32 units. The embedding layer for the sampled τ had 64 units. The embedding was combined with the features from the first hidden layer in the main structure by element-wise multiplication. The activation function for all units were ReLU [96]. The optimizer used was Adam [97] with an initial learning rate of 0.0005. The IQN was also trained with the same trips as the DQN. The framework is shown in Figure 5.15.

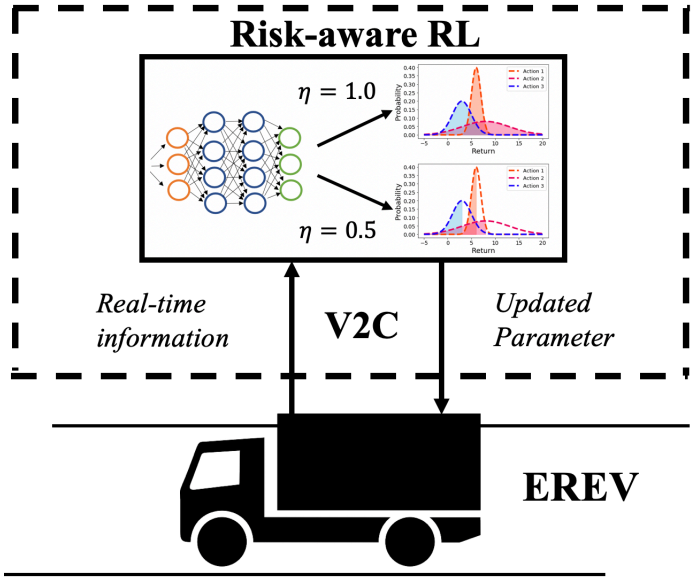


Figure 5.15: Illustration of the proposed risk-aware framework.

Figure 5.16 shows how the trained algorithm performed on a short trip. The L_{set} change at each timestep is very similar under the conditions of $\eta = 1.0$, $\eta = 0.5$ and $\eta = 0.1$, which leads to identical fuel use and SOC curves. This can be explained by the return distribution shown in Figure 5.17, which corresponds to the state *A* annotated in Figure 5.16. For all possible actions, the return distributions are highly concentrated near some value with a low variance and similar range. Therefore, the rank of calculated CVaR for

different actions under $\eta = 0.5$ and $\eta = 0.1$ are still similar to the condition of $\eta = 1.0$. This shows that for short trips, risk-aware strategies do not make a difference.

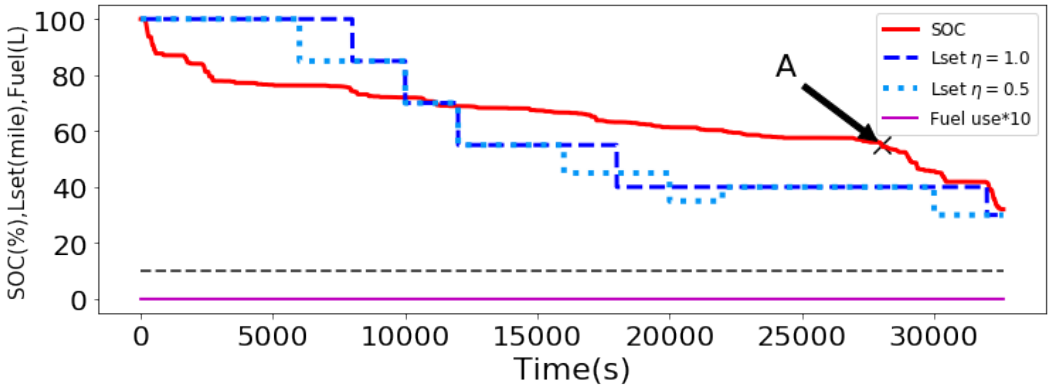


Figure 5.16: Performance of the trained algorithm on a test trip with a distance of 35 miles with $\eta = 1.0$, $\eta = 0.5$ and $\eta = 0.1$. To make the figure clear, the L_{set} corresponding to $\eta = 0.1$ is not shown.

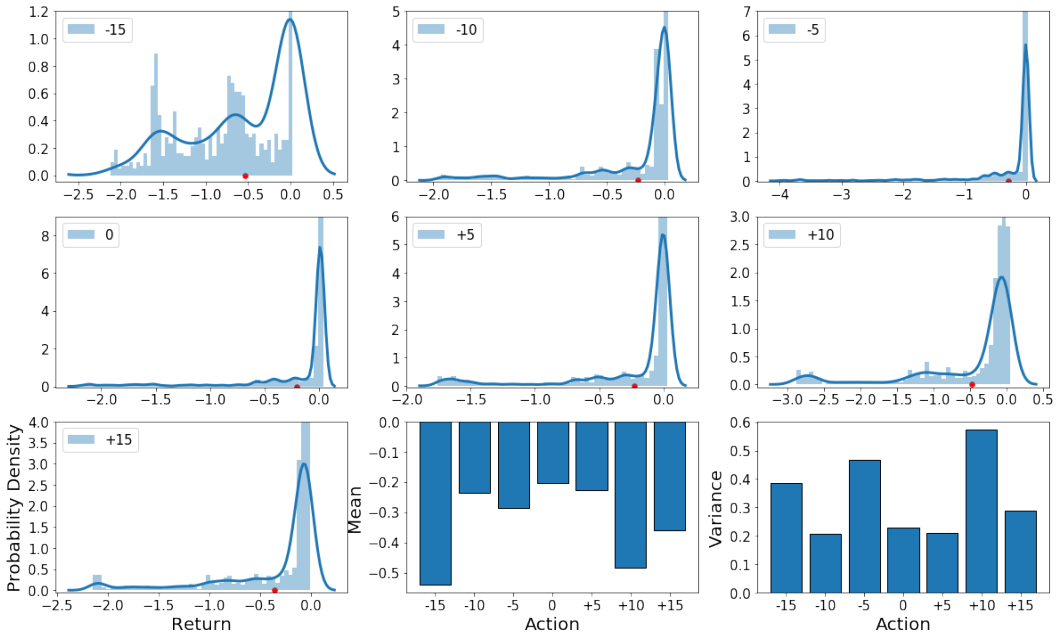


Figure 5.17: Return distributions with mean and variance for all possible actions in state A. The smooth line is fitted with a kernel density estimator for better visualization, and the mean and variance were still calculated with the original data. The y-axis

represents the probability density so that the value could be higher than 1. The red dot represents the mean value of each distribution.

For a longer trip, as shown in Figure 5.18, the L_{set} changes significantly at each timestep under different η values, resulting in different fuel use and SOC curves. This can also be explained by observing the return distributions for some states. Figure 5.19 shows the return distributions for state B. Compared with state A, it can be observed that the return distributions are much less concentrated and the probability densities for the peak are much lower than that of state A. Also, the range of the distributions is much wider on the negative side, indicating the possibilities of running out of battery. Therefore, for these wide and high variance distributions, the ranks of calculated CVaR under the condition of $\eta < 1$ are different from the standard risk-neutral condition. The lower the value of η, the more fuel would be used by the risk-aware algorithm.

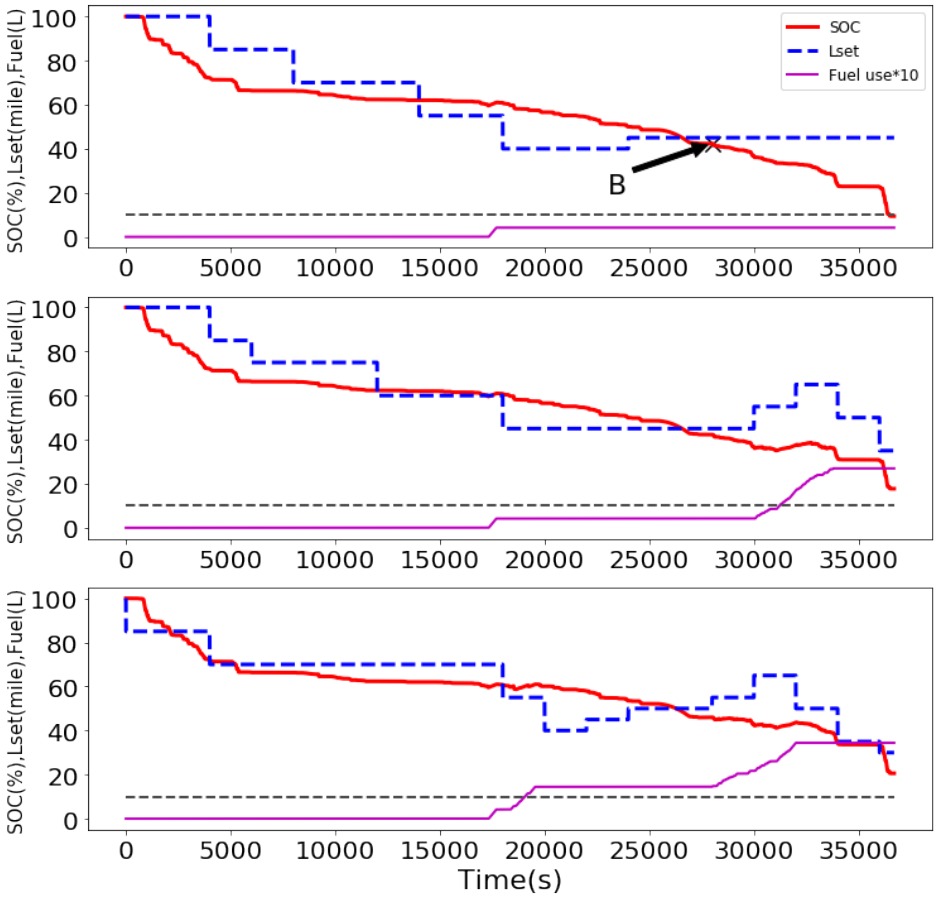


Figure 5.18: Performance of the trained algorithm on a test trip with a distance of 48 miles with $\eta = 1.0$ (upper), $\eta = 0.5$ (middle) and $\eta = 0.1$ (lower).

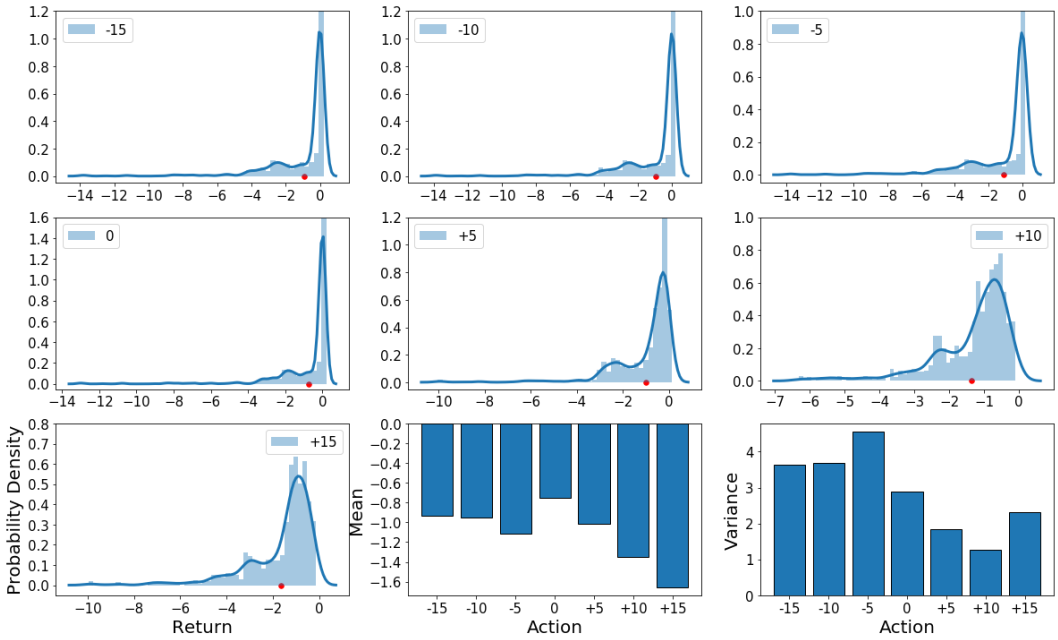


Figure 5.19: Return distributions with mean and variance for all possible actions in state B. The red dots indicate expected values of the estimated return distributions.

The highest CVaR (the values corresponding to the selected action at each state) along the two delivery trips under the three values of η are shown in Figure 5.20. As expected, for each state, the CVaR decreases as the value of η decreases from the upper figure, showing a more and more pessimistic estimation of performance. This can be easily explained by the definition of CVaR. The reason why this seems not true for the last state of the lower figure is because the state information has become significantly different for the three η as different actions were taken during the task. For example, the final SOC and fuel consumption values are quite different for the three conditions. In addition, it can be observed that the values of CVaR are very similar at the beginning of the two trips. This can be explained by the fact that the three trajectories are not clearly distinguishable

at first, but the discrepancy grows as they proceed. The exact value at the initial state reflects the algorithm's estimation of its performance based on the properties of the training trips and without knowing any information about the upcoming trip.

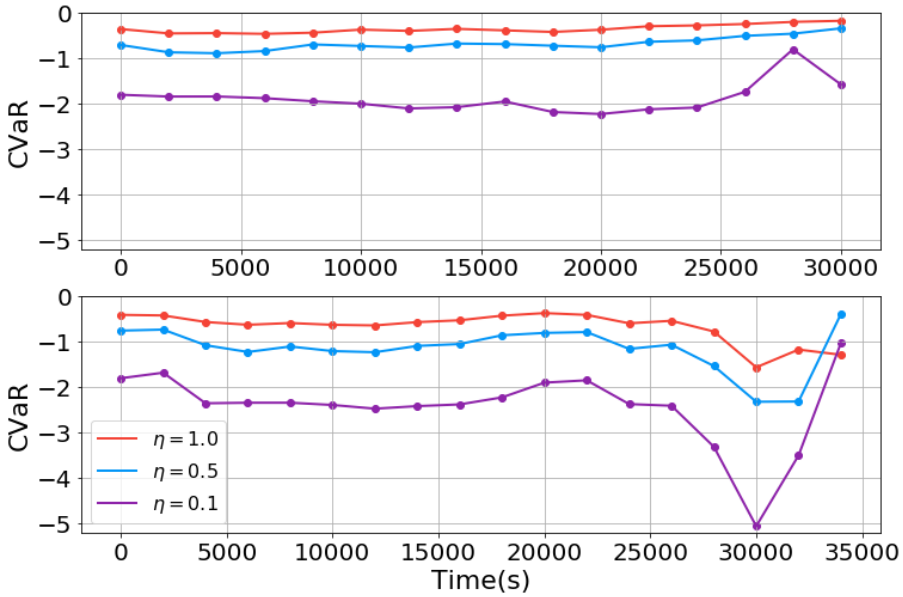


Figure 5.20: Highest CVaR among possible actions along the two test trips.

Table 5.3 summarizes the average fuel use and average final SOC for the 58 test trips under the three η values. The lower the value of η, the more fuel was used to prevent the vehicle from running out of battery during the delivery trip, leading to a higher final SOC value. This is the tradeoff between fuel efficiency and the risk of running out of battery which could lead to delays of delivery tasks and dangerous conditions like stopping on highways. Consequently, after the algorithm is trained, η can be chosen according to different application scenarios and preferences of the users. For example, if the vehicle delivers in an urban area with frequent delivery stops, the η can be set close to 1 as there are enough charging opportunities, and the energy intensities for this kind of driving is relatively low. However, if the delivery area includes long highway driving, a lower η

might be used as the energy intensities for highway driving are usually high, and the charging rate of the range-extender is low compared with the energy consumption rate.

Table 5.3: Tradeoff between fuel use and risk of running out of battery

	Average fuel use (L)	Average final SOC (%)
$\eta = 1.0$	0.72	18.6
$\eta = 0.5$	1.11	20.0
$\eta = 0.1$	1.94	23.1

A simple form of the original loss function is used in this work for faster training. According to the original loss function, for each transition pair (s_t, a_t, r_t, s_{t+1}), N samples of Z_{τ_i} and N' samples of $Z_{\tau'_j}$ should be sampled from the IQN and target IQN respectively. Then, the loss is calculated [127] as:

$$L(s_t, a_t, r_t, s_{t+1}) = \sum_{i=1}^{N} \frac{1}{N'} \sum_{j=1}^{N'} \left(\tau_i - \mathbb{I}_{\delta^{\tau_i,\tau'_j}<0}\right) \delta^{\tau_i,\tau'_j}, \tag{5.9}$$

where:

$$\delta^{\tau_i,\tau'_j} = r_t + \gamma max_a Z_{\tau'_j}(s_{t+1}, a; \theta') - Z_{\tau_i}(s_t, a_t; \theta). \tag{5.10}$$

The loss for every transition pair in a sampled batch should be calculated as in (5.9) and then summed together. In our case, both N and N' were set to 1 so there was only one summation over n sampled transition pairs. A better return distribution estimation is expected to be achieved if more samples (N and N') are used during the training.

In this work, a risk-aware strategy is developed based on IQN. I demonstrate the tradeoff between fuel efficiency and the risk of running out of battery power by changing the value of η in the risk measure of CVaR.

5.3. *Uncertainty-aware Energy Management*

Model uncertainty can be explained away with enough data. If trajectories generated during training process can populate the whole state space that is possible to be encountered in the real-world condition, there is no need to consider the model uncertainty. However, that is usually difficult to achieve as the amount of data needs to fill a space grows exponentially with the dimensionality of the input space [77][89], and as the transition probability of the real-world system cannot be fully captured by the data generated from simulation.

The transition probability $p(s_{t+1}|s_t, a_t)$ of the underlying system that generates the real-world data is unknown, and only can be approximated by existing data with a vehicle model that calculates the SOC curve and fuel use with different L_{set} settings. Consequently, the explore of the state space is constrained within the recorded data, especially for the dimension of distance d and GPS coordinates x and y. In this work, 22 recorded delivery trips were used for training and 30 trips were used for testing. The distance and energy intensity distributions are shown in Figure 5.21. It can be observed that, as data cumulates, the range of the possible trip distance gets larger as well as the overall energy intensity.

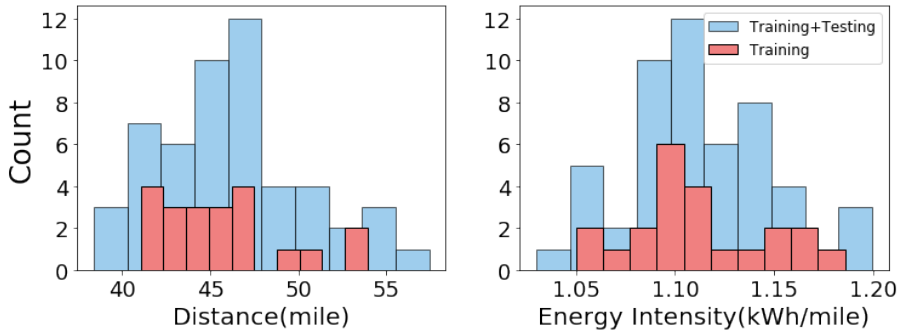


Figure 5.21: Distance and energy intensity distribution of training trips and training-testing trips combined.

Furthermore, even for two delivery trips with similar distance, GPS trajectories of a future trip can be significantly different, which will also lead to unfamiliar states. For example, the GPS trajectories of two delivery trips are shown in Figure 5.22. It can be observed that, although the delivery region is the same, there are differences in the area covered due to factors like delivery demand.

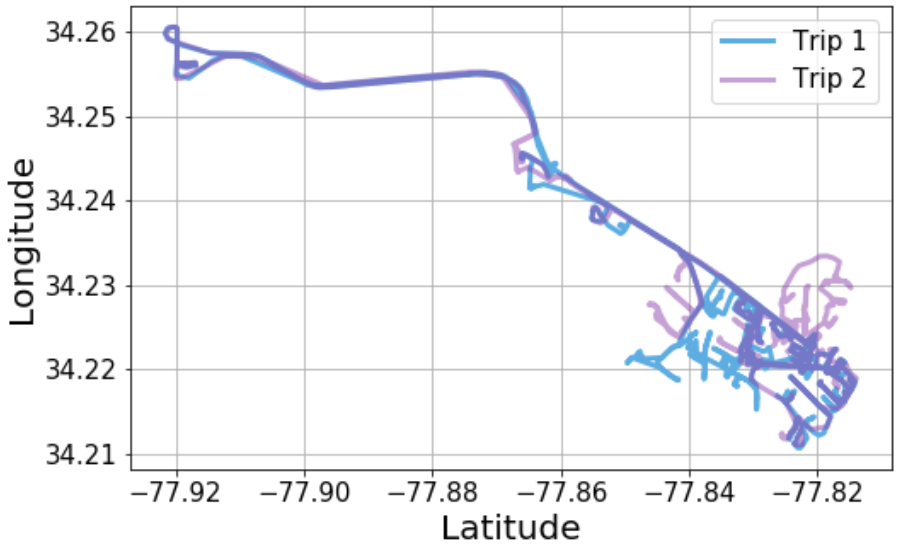


Figure 5.22: Comparison of two GPS trajectories with 43.1 and 44.3 miles.

Besides distance and GPS coordinates that can lead to novel and unseen states, there is another reason corresponding to the dimensionality of the state space, which is the "curse of dimensionality" [80][89]. The amount of data that needs to fill the state space grows exponentially with the dimensionality of the state space. Consequently, for our defined 7-dimensional state space, even for some of the visited states, data and interaction trajectories can be sparse in that region due to relative less frequent visitation during the training process, which is highly dependent on the statistics of the recorded trips. Consequently, uncertainty estimation is of great importance when dealing with real-world problems as it is not feasible to build a model that is well trained on all parts of the possible state space.

To make the above discussion more straightforward, a simplified artificial state space with two dimensions is shown in Figure 5.23. For each episode of interaction, a trajectory from the initial state to the region of possible terminal states will be made. After recording some real-world trajectories, simulations can be performed to train the agent based on the recorded trajectory. During the training process, the region that is reachable from the recorded trajectories by choosing different actions can be visited, and the frequency of visitation depends on the statistics of the recorded trajectories. Region A represents a region which is visited frequently, region B represents a region with sparse data and region C represents an unvisited part of the state space, which is not reachable from changing actions from any of the recorded trajectories. The goal of the uncertainty-aware RL algorithm is to measure whether the encountered state is novel during testing.

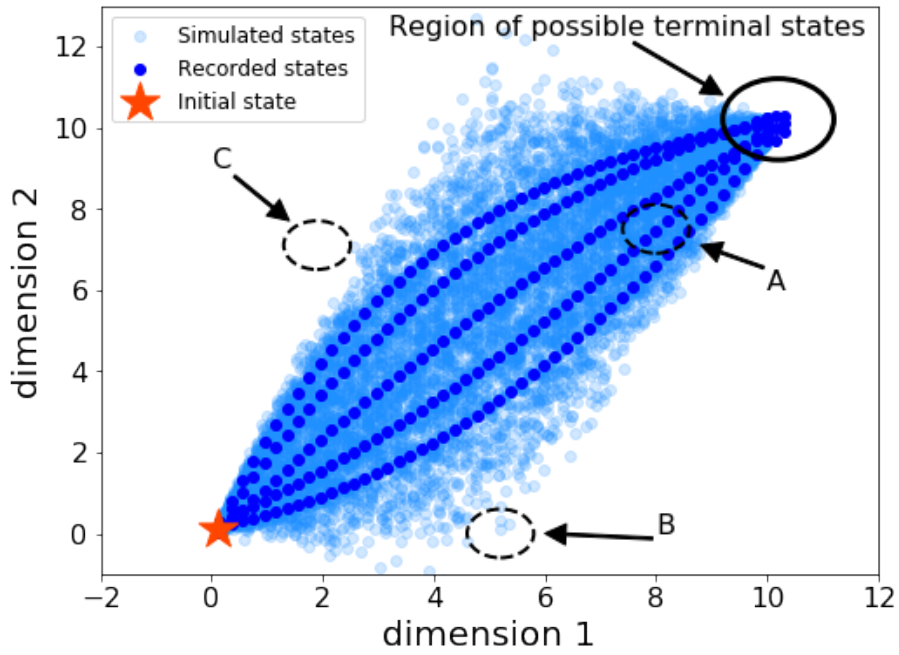


Figure 5.23: Illustration of the concepts of frequently visited states, infrequently visited states and unvisited states in a two-dimensional state space.

Model uncertainty can be handled elegantly with the Bayesian framework, if applicable, and Bayesian NNs provide a systematic way to model uncertainties in NNs [89][121]. It keeps posterior distributions for the parameters calculated from the predefined prior distributions and training data. However, there are mainly two challenges that prevent it from being widely used in real-world problems [121][122][129]. First, exact Bayesian inference is only feasible on small-scale problems [123], and its approximations are also computationally expensive. Second, to implement Bayesian NNs, significant modifications in model building and training procedure are needed compared with standard NNs, which are very easy to use with popular deep learning libraries like Pytorch

and Tensorflow. Therefore, researchers have investigated non-Bayesian methods to model the uncertainty, aiming at developing practical and scalable frameworks.

A simple and scalable method to estimate the predictive uncertainty is developed in [129]. An ensemble of NNs is trained, and for each individual NN, the conditional Gaussian distribution given the input is modeled. The predictive uncertainty is obtained by treating the output as a mixture of Gaussian distributions. In [130], ensemble sampling is developed to approximate the posterior distribution. First, M NNs are drawn from a prior distribution. Then, at each timestep during training, only one model in the ensemble will be sampled to generate data. After the data is generated, all the models in the ensemble will be updated. To bridge the gap between Bayesian NNs and the practical ensemble methods, Bayesian ensemble [122] is developed, which regularizes the parameters in the standard NNs to values drawn from a predefined distribution as the prior information for the NNs, which can produce Bayesian behaviors while keeping the algorithm practical. In [131], the authors investigate why using ensemble of NNs with different initializations can help with the accuracy and uncertainty estimation of deep learning models. In [132], a distribution-free, ensemble method is developed for prediction intervals of neural networks. It is enabled by deriving a loss function that can be optimized by gradient-based methods. A detailed review for prediction intervals can be found in [133].

Energy management problems of hybrid vehicles have been a research focus for a long time. However, there is little work about the model uncertainty in EMS problems. One of the main reasons is, for most hybrid vehicles, the worst condition is unsatisfying energy efficiencies, which does not lead to serious problems. However, for our EREV for delivery,

bad EMS can lead to the condition of running out of battery during the delivery trip, which may cause delays of delivery tasks and can be very dangerous if it happens during highway driving. Consequently, our goal is to build an uncertainty-aware agent that can facilitate a postprocessing module to execute predefined rules when the uncertainty exceeds a predefined threshold. The Bayesian ensemble method is used due to its simplicity in implementation and better theoretical support compared with other ensemble-based nonparametric methods.

For the standard DQN, the parameters θ are trained at each timestep by minimizing the error:

$$L = \left(Q_{target} - Q(s_t, a_t; \theta)\right)^2. \tag{5.11}$$

The visited state s_t is determined by the interaction between the agent and the simulated environment. For the frequently visited region in the state space, the action-values for the corresponding state-action pairs are well estimated. However, for the less visited region where the data is sparse and unvisited region, θ are not well determined so that the output action-value is not reliable. Consequently, having a measure of uncertainty can detect whether the output of Q-network is trustworthy.

The intuition behind ensemble-based uncertainty estimation methods is for different function approximators, the value will agree well on frequently visited states and diverge on the rest of the regions, and the variance of the output values indicates the model uncertainty. The diversity of different Q-networks mainly comes from the following aspects. First, each of them is randomly initialized with different parameters so that the target network that each Q-network is trained on is different. Second, stochastic gradient

descent-based optimization provides another source of randomness. Third, the exploration process like the ε −greedy chooses actions randomly with some probability controlled by the hyperparameter ε. In addition, for the Bayesian ensemble method, the parameters in each Q-network is regularized to some anchored values sampled from a predefined distribution. Therefore, the loss function (5.11) is modified as:

$$L_{anchor} = \left(Q_{target} - Q(s_t, a_t; \theta)\right)^2 + \lambda \|\theta - \theta_0\|^2, \tag{5.12}$$

where λ represents the regularization strength and θ_0 is the anchored value randomly sampled from a distribution which represents the prior information of the Q-network.

Assume M Q-networks $Q(s, a; \theta_j)$ are trained individually with the loss function in (5.12) with corresponding anchored values $\theta_{0,j} \sim Normal(\mu_{prior}, \Sigma_{prior})$, the M outputs of the ensemble given an input state-action pair can be considered as M samples from the posterior distribution [122]. Therefore, the variance of the ensemble's predictions can be interpreted as its uncertainty.

Although this method does not perform exact or approximate Bayesian inference over NN parameters [89], it keeps an explicit notion of prior which does not vanish during training, resulting in Bayesian behavior. Also, it is built on the fact that adding a regularization term to a loss function returns maximum a posterior (MAP) estimates of parameters [138]. Therefore, the Bayesian ensemble method gets better theoretical support compared with other ensemble-based methods without compromising the advantage of being easy to implement.

For each Q-network $Q(s, a; \theta_j)$, both of the model parameters θ_j and the anchored values $\theta_{0,j}$ were sampled from the distribution of $Normal(\mu_{prior}, \Sigma_{prior})$. All the components in μ_{prior} are 0 and Σ_{prior} is a diagonal matrix with equal variance. The variance is set to be 1 and it controls the degree of diversity among the models in the ensemble. In other words, the initial value of each parameter in the Q-networks is sampled from a one-dimensional normal distribution with mean of 0 and variance of 1. It should be noted that although the initial values of θ_j and $\theta_{0,j}$ comes from the same distribution, they were not the same values as recommended in the original paper [122].

The training process of each DQN is similar to the standard DQN, and the NN structures and hyperparameters are the same with our previous work with DQN. The regularization coefficient λ is set to be 0.001. After training, for each s_t encountered during testing, the mean and standard deviation (SD) for the action-values for the state-action pairs are calculated:

$$Q_{mean}(s_t, a) = \frac{1}{M}\sum_{j=1}^{M} Q(s_t, a; \theta_j),$$

$$Q_{SD}(s_t, a) = \sqrt{\frac{\sum_{j=1}^{M}\left[Q(s_t, a; \theta_j) - Q_{mean}(s_t, a)\right]^2}{M-1}}. \quad (5.13)$$

Then the corresponding policy is:

$$\pi(s_t) = argmax_a Q_{mean}(s_t, a). \quad (5.14)$$

The mean is used to select actions and the values of SD indicate the uncertainty corresponding to each action. Next, the detailed performance of the Bayesian ensemble on

three delivery trips are shown, demonstrating the three kinds of conditions discussed in Figure 5.23 respectively.

The frequently visited states where the model parameters are well-determined and the infrequently visited states where the model parameters are less well-determined are shown on two training trips with 43.1 and 52.9 miles from Figure 5.24-Figure 5.27. For the shorter trip, it can be observed from Figure 5.24 that for all states encountered during the trip, the SD of action-values for all the 5 possible actions are lower than 1, which indicates low model uncertainties. Taking state *A* annotated in Figure 5.24 as an example, the mean and one standard deviation of action-values are plotted on the left part of Figure 5.25 and the individual predictions from each agent are plotted on the right. Although there are imposed diversities among the agents, it is obvious that all the agents agree well on the action-values for each possible action. All the states encountered in this trip can be considered as locating in the frequently visited region in the state space. This can be partly explained by looking at Figure 5.21. The distance of most of the 22 training trips are concentrated near 43 miles, and energy use is highly correlated with distance so that states with similar distance, SOC, fuel use as well as the L_{set} setting will be generated frequently when the simulation is run on any of these trips with similar distance, which leads to a frequently visited region near trajectories generated from these training trips in the state space, like region *A* in Figure 5.23.

The simulation results for the longer trip which is also the second longest trip in the training trips are shown in Figure 5.26. It requires the highest fuel use among the 22 training trips, the states encountered at the last part of this trip can only be generated when the

simulation is run on the longest two trips, which leads to relative less frequent visitations, corresponding to region *B* in Figure 5.23. Consequently, the uncertainties for part of the action-values are higher than 1, indicating the parameters in the agents are not well-determined for the corresponding state-action pairs.

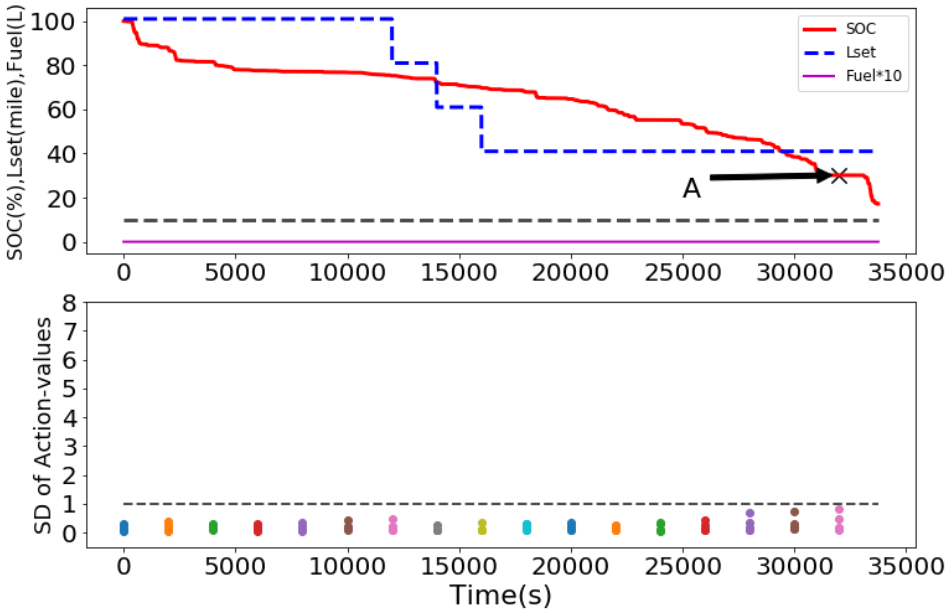


Figure 5.24: Performance of the trained algorithm on a training trip with a distance of 43.1 miles (upper) and the estimated uncertainties along the trip (lower).

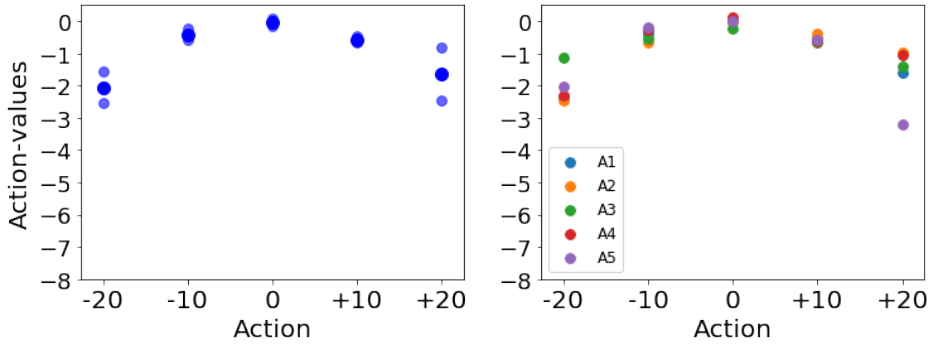


Figure 5.25: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state *A*.

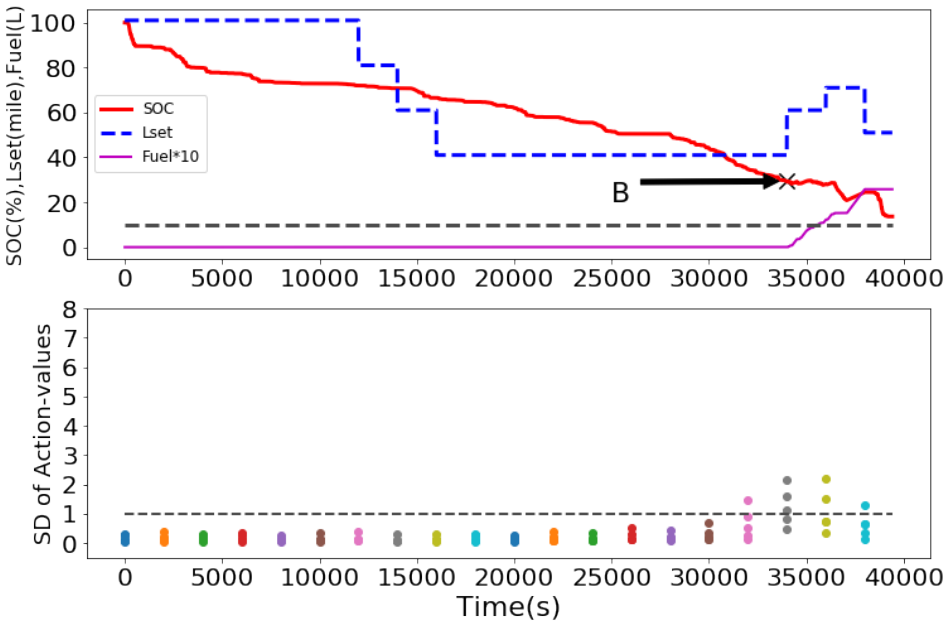


Figure 5.26: Performance of the trained algorithm on a training trip with a distance of 52.9 miles (upper) and the estimated uncertainties along the trip (lower).

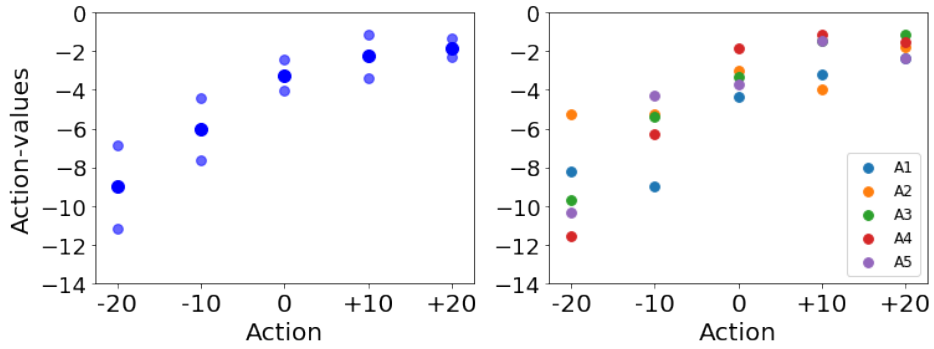


Figure 5.27: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state *B*.

For a test trip with a distance longer than any of the training trips, it is obvious from Figure 5.28 that the algorithm does not perform well as the battery SOC drops below 10% for a significant amount of time. Nevertheless, the model outputs very high uncertainties

for all the actions from 34000s, indicating the states are novel and significantly different from the states that it experienced during the training process. It can be observed from Figure 5.29 that although the ranking of the action-values for state C is correct, the agents cannot agree on the long-term return after this state, i.e., the agents are not sure whether the EMS can prevent the EREV from running out of battery. The states that encountered at the last part of this trip can be considered as locating in the region that is never visited during training, just like region C in Figure 5.23. Therefore, the predictions from different agents are significant different, just like the NNs diverge on the region there is no data in Figure 5.1.

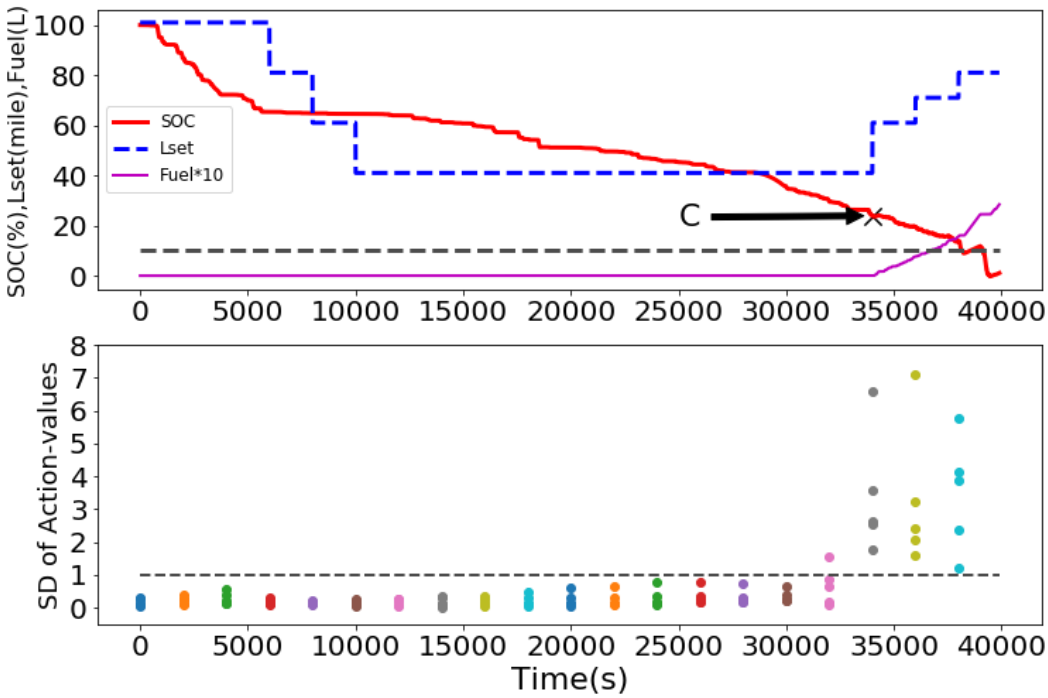


Figure 5.28: Performance of the trained algorithm on a test trip with a distance of 57.4 miles (upper) and the estimated uncertainties along the trip (lower).

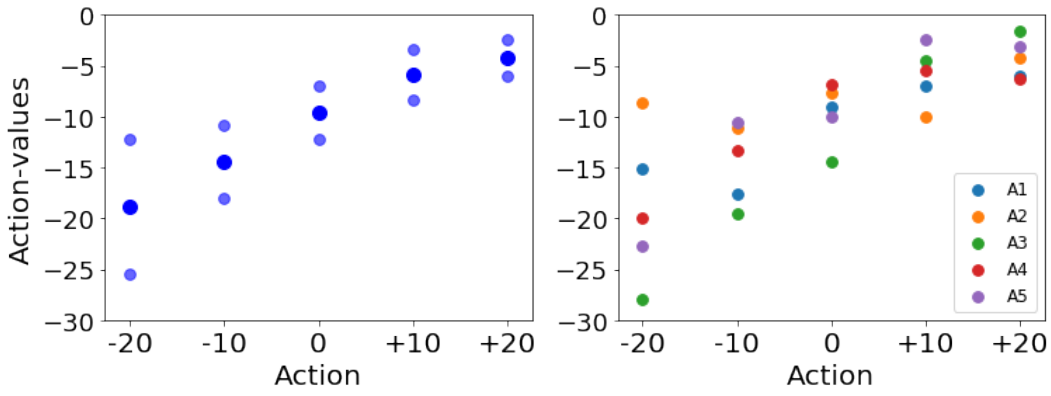


Figure 5.29: The mean action-values for each action with one standard deviation (left) and the predictions from each agent (right) for state C.

With the uncertainty-aware RL algorithm, an interpretable white-box postprocessing module can be integrated into the black-box RL system to include predefined explicit rules for unfamiliar or novel states as shown in Figure 5.30. The threshold to define unfamiliar or novel states can also be controlled. If the threshold is set to be 0, then the RL-based EMS becomes the widely used interpretable rule-based EMS. For example, for our EREV application, if the model uncertainty is higher than the predefined threshold value, the postprocessing module can overwrite the L_{set} with a high value, or it can notify the driver to stop the EREV until the SOC is charged back to some value. The focus of this work is to develop the uncertainty-aware RL algorithm that can identify the novel states so that the design of postprocessing module is not discussed here.

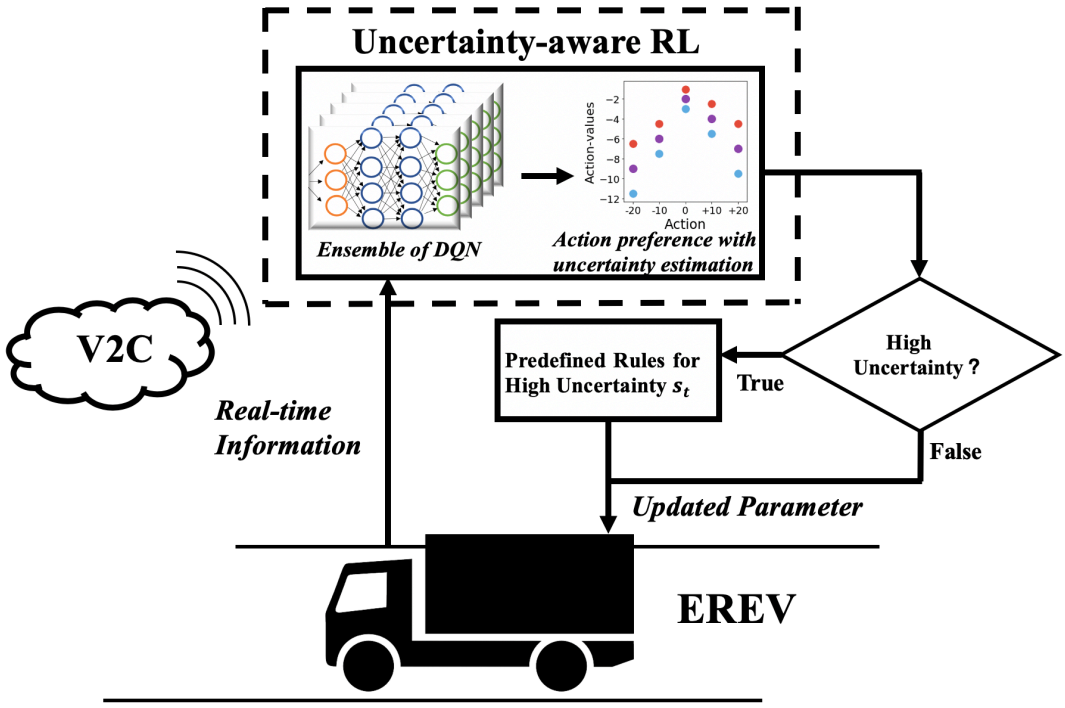


Figure 5.30: Illustration of the proposed uncertainty-aware framework.

5.4. *Adversarial Attacks on Deep RL-based Energy Management*

Robustness is one of the most important factors that needs to be considered when implementing a new system into the transportation area. However, it was found for a well-trained NN-based classifier, it is possible to add a small crafted perturbation to the original image so that the classifier would misclassify the perturbed image, even the difference are not discernable by human [139]. In [140], a method called Fast Gradient Sign Method (FGSM) was introduced to create such adversarial examples. It requires small computation resources and achieves very good results. In addition, they found one adversarial example created for a specific model can also be misclassified by other models with different architectures for the same task. Further, it was shown that the adversarial examples are still effective even they are perceived through cameras in a real-world problem setting [141].

As NNs are also the core components of DRL algorithms, researchers have investigated the impacts of adversarial examples under several simulation settings. In [142], FGSM and its two variations with different norm constraints were applied to degrade the performance of a DRL agent on playing video games using image inputs. Also, adversarial examples were compared with random noise under the same simulation environment in [143]. It was shown that adversarial examples were much more effective in misleading the agent. In [144], a more complex and computational expensive approach involving a generative model and a planning algorithm was introduced to lure the agent to designated target states. In [145], a systematic characterization of adversarial attacks was shown. They also performed comprehensive experiments on video game playing as well as physical system control which used low-dimensional state representations. The same physical system control simulation environment was also used in [146]. They developed an adversarial robust policy learning algorithm to help the agent perform better at test time under the gap between simulation domain and physical domain caused by random noise or adversarial noise.

In this work, I investigate the impacts of adversarial examples generated by FGSM related methods on a well-behaved DRL-based EMS. The methods are categorized as white-box or black-box methods according to the information known to the attacker. This study investigates the adversarial examples on DRL algorithms in the area of ITS, where robustness and safety are of great importance. Figure 5.31 illustrates the attack process. It is shown that adversarial attacks can degrade the performance of the DRL-based EMS

significantly, either causing the EREV using too much fuel, or leading it running out of battery during the delivery trip.

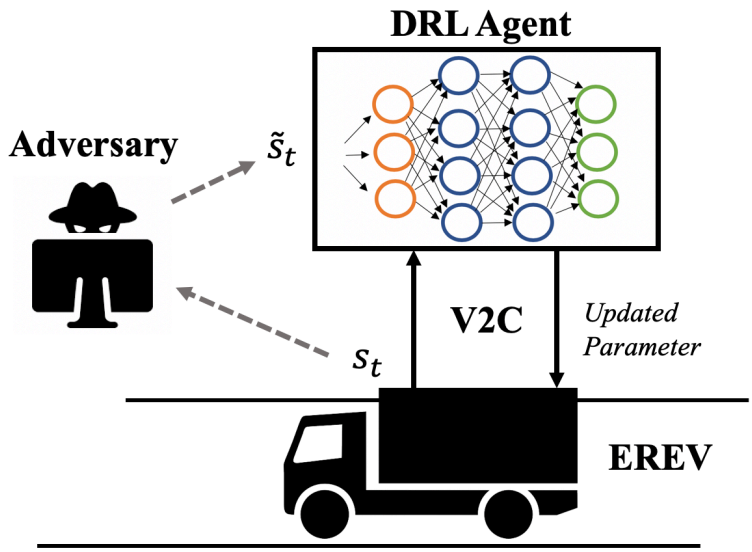


Figure 5.31: Illustration of the adversarial attacks. The low-dimensional state representation s_t provided by the EREV is processed by the adversary before sending to the DRL agent on the cloud.

5.4.1. White-box Methods

Under a white-box condition, it is assumed that the attacker has full access to the target agent including network structure, parameter weights as well as all the training details. Our white-box method is FGSM [140] with its two variations [142].

The main idea of the original FGSM is straightforward. Given a loss function $J(\theta, x, y)$, where θ represents the weights in the NN-based classifier, x represents the input image with d dimensions and y is the true label corresponding to x, a small perturbation is added

to the original input x so that the loss function $J(\theta, \tilde{x}, y)$ on the perturbed image $\tilde{x}$ is increased as much as possible. This is achieved by：

$$\tilde{x} = x + \varepsilon sign[\nabla_x J(\theta, x, y)], \tag{5.15}$$

which is derived by linearizing the cost function around the current parameters θ, and solving a L_∞ −constraint optimization problem. A more detailed analysis of the algorithm can be found in the original papers [140][141]. In addition, the L_1 and L_2 norm constrained form are also included [142]. For L_1, we maximally perturb the dimension i that has the highest gradient magnitude:

$$\tilde{x} = x + \varepsilon d \cdot e_i, \tag{5.16}$$

where e_i is an d dimensional vector whose ith component is 1 and 0 otherwise. For the L_2 constraint, the perturbed input is:

$$\tilde{x} = x + \varepsilon\sqrt{d}\frac{\nabla_x J(\theta, x, y)}{\|\nabla_x J(\theta, x, y)\|_2}. \tag{5.17}$$

In the RL setting, to get the perturbed $\tilde{s}_t$ from s_t, the action-value vector is first calculated from the DRL agent. Then, it is transformed into a set of probabilities y_p with a softmax function. The label y is obtained by replacing the position of the highest entry in y_p with 1 and replacing other entries by 0. The loss $J(\theta, s_t, y)$ is then calculated as the cross-entropy between y_p and y. Cross-entropy loss is used for classification and in the RL setting, it is assumed the target agent performs well and what we want is to make it choose another action instead of the action based on the original input (“misclassify actions”).

5.4.2. Black-box Methods

Finite Difference Method

Under a black-box condition, the attacker does not know the NN structure or the parameter weights, so the gradient $\nabla_{s_t} J(\theta, s_t, y)$ cannot be calculated directly. One method is to estimate it with the finite difference (FD) method [145]. To apply FD on an input with d dimensions requires querying the target agent $2d$ times. The ith component of the estimated gradient $\nabla_{s_t} \tilde{J}(\theta, s_t, y)$ is:

$$\frac{J(\theta, s_t + \delta e_i, y) - J(\theta, s_t - \delta e_i, y)}{2\delta}, \tag{5.18}$$

where δ is a hyperparameter that controls the numerical accuracy. After $\nabla_{s_t} \tilde{J}(\theta, s_t, y)$ is calculated, it can be used as $\nabla_{s_t} J(\theta, s_t, y)$. This method can be computationally expensive if the input dimension is high, as with an image, for example. In this work, the inputs are low-dimensional state representations, so the computation burden is insignificant.

Transferability Property

If the attacker is not allowed to query the target agent, but has access to the training environment, the transferability property of adversarial examples can be utilized. Adversarial examples generated for one NN have been found to often work for another NN even if the parameter weights and structures are different [140][142]. Therefore, the attacker can train an agent for the same task using the training environment and generate $\tilde{s}_t$ using its own agent for the unknown target agent.

I investigate two cases: transfer across policies and transfer across algorithms. For the first case, it is assumed that the attacker knows the RL algorithm, NN structures, and

training hyperparameters used by the target agent. For the second case, the attacker only has access to the training environment. There are plenty of variations of experiments that can be done. For example, for the transfer across policies, the attacker may only know the RL algorithm and nothing else. Although I only tested the two special cases described above, it is enough to show initial results about the transferability property for this problem.

5.4.3. Results and Discussion

I applied the three adversarial methods on a DQN agent on 52 recorded delivery trips. Two types of random noise were used as baselines. The first was random sign noise. A value with magnitude ε was added to each dimension and the sign was randomly chosen from positive and negative with equal probabilities. The second was uniform random noise. For each dimension, a value sampled from a uniform distribution $U([-\varepsilon, \varepsilon])$ was added to the original input.

Figure 5.32 shows how the DQN performs under these two types of noise. For each value of ε, I ran 10 experiments and got 10 average scores.

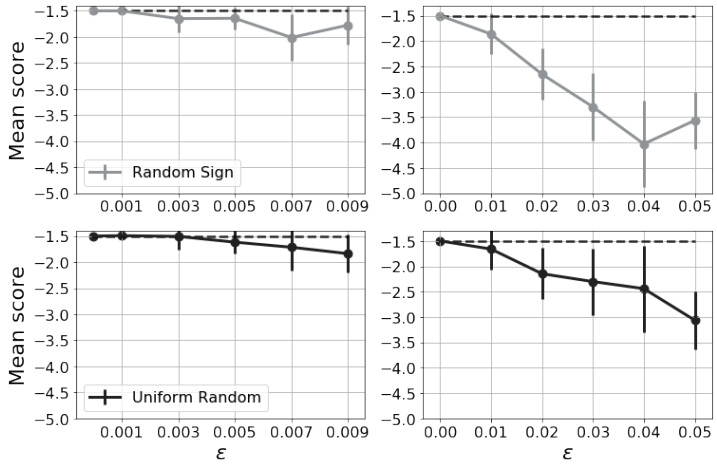


Figure 5.32: Performance of the DQN under two kinds of random noise with different values of ε. The error bars indicate one standard deviation.

The performance of the DQN under FGSM and its two variations are summarized in Figure 5.33. First, it can be observed that all three versions are more effective at degrading the performance of the deep RL-based EMS than random noise. For the same magnitude of ε, the gradient-based adversarial examples can degrade the agent to much worse scores. On the other hand, to misguide the agent to a certain low level of performance, the adversarial examples can be much more similar to the original inputs. Second, it is obvious the $L_1 -$ constrained method performs best for most cases. However, one major downside is that it uses the entire εd budget to perturb one dimension, making it much easier to detect compared with other methods.

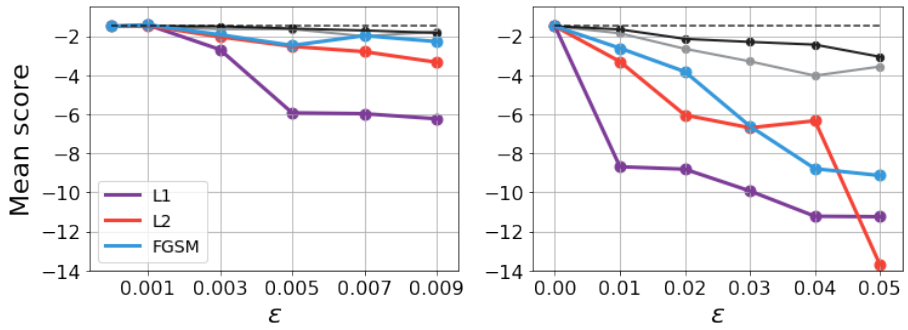


Figure 5.33: Performance of the DQN under FGSM and its two variations.

Figure 5.34 compares the performance of FGSM under the white-box condition and its estimation with the FD method under the black-box condition. δ is set to 0.0001. It can be observed that the performance is nearly the same, which indicates the FD method can estimate the gradient with high accuracy under low-dimensional states.

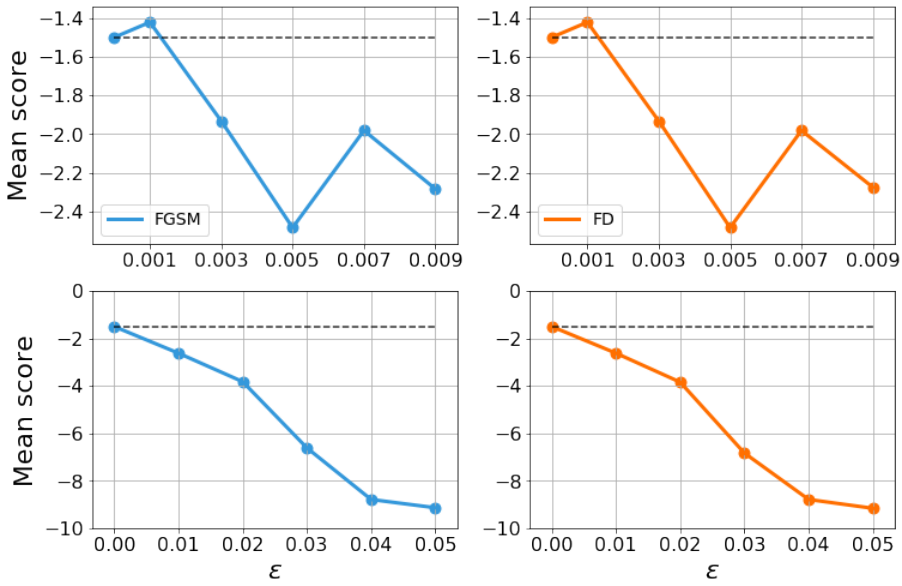


Figure 5.34: Performance of the DQN under the FGSM and FD methods.

Figure 5.35 and Figure 5.36 show how well the adversarial examples exploit the transferability property when the attacker cannot query the target agent freely. For the transfer across policies case, adversarial examples were generated by a DQN with different parameter weights. For the transfer across algorithms case, adversarial examples were generated by an IQN with different NN structure, hyperparameters and training process. Moreover, the IQN is a distributional RL algorithm so the process of calculating the gradient $\nabla_{s_t} J(\theta, s_t, y)$ is also different from the standard DQN. It can be observed that although all three methods under these two cases are less effective compared with the white-box FGSM method, they all outperform random noise in most cases. This indicates the transferability property holds true in this EMS problem. Also, the results match our expectation: the more we know about the target agent, the better the adversarial examples that can be generated by another agent. The adversarial examples generated by the IQN are less effective than those by DQN, because DQN is more similar to the target agent.

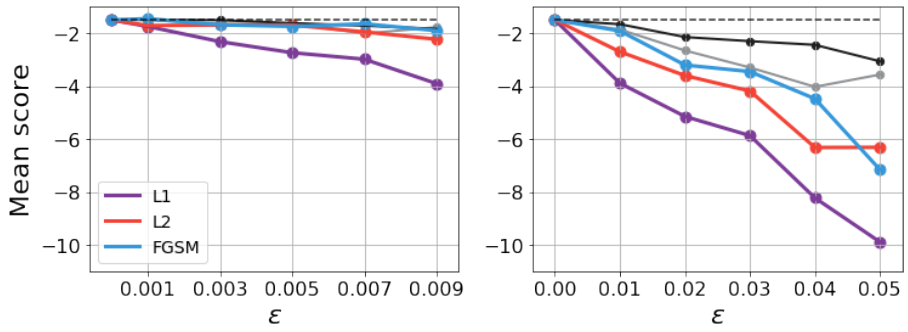


Figure 5.35: Performance of the DQN under the strategy of transfer across policies.

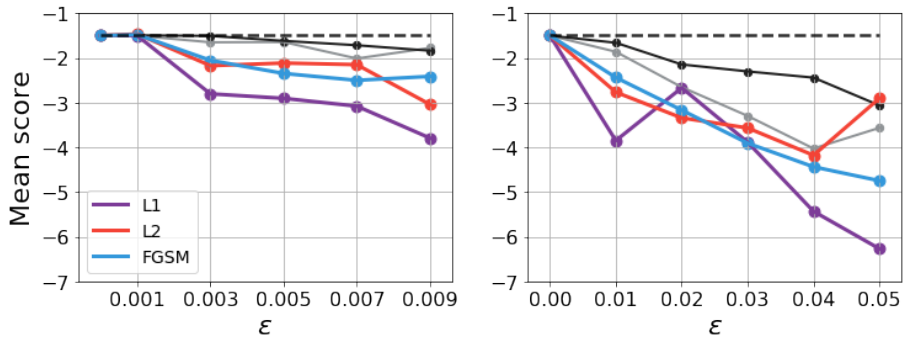


Figure 5.36: Performance of the DQN under the strategy of transfer across algorithms.

Figure 5.37 and Figure 5.38 provide some concrete examples of how the DQN agent behaves under adversarial attacks. The figures show the detailed trip information obtained under the attack scenario for the two previously shown trips in chapter 3 (Figure 4.5 and Figure 4.6). For the shorter trip, although no fuel was needed, 2.65 L fuel was consumed under the adversarial attacks. For the longer trip, some amount of fuel should have been used at the end of the trip to prevent the SOC from falling lower than 10%. However, under the perturbed states, the DQN agent chose to decrease rather than increase the value of L_{set}.

To further understand the impact of adversarial examples on the DQN agent, we chose state *A* and state *B* from the two trips to compare the action-values under normal states

and perturbed states. In Figure 5.39. it is clear that the agent would choose the action of decreasing the L_{set} by 10 if the state information was correct. The action preference among all possible actions is also easy to recognize according to the ranking of action-values. However, the differences between the action-values became unclear under the perturbed state. For state B in Figure 5.40, it can be observed that if the state information was transmitted correctly, the agent would consider that all actions could lead to bad results since all the action-values were very low, and the action of $+10$ was the best it could do. By contrast, when the state was perturbed, the magnitude of the action-values all changed significantly, and the ranking seemed random.

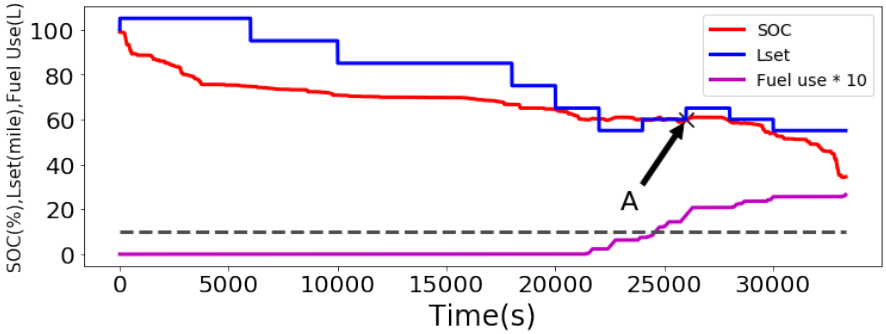


Figure 5.37: Performance of DQN on the recorded trip shown in Figure 4.5 under adversarial examples generated by L_2 method with $\varepsilon = 0.05$.

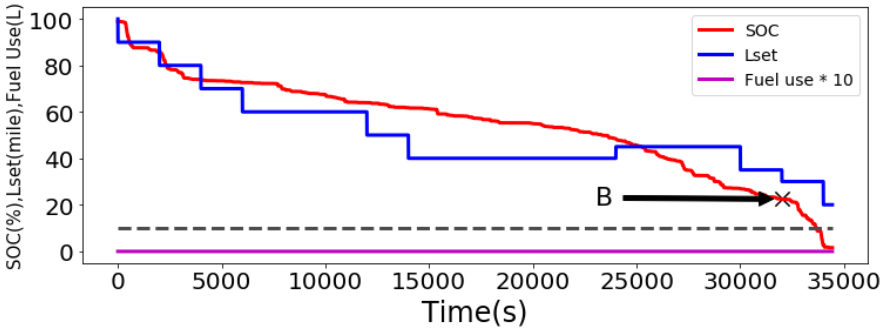


Figure 5.38: Performance of DQN on the recorded trip shown in Figure 4.6 under adversarial examples generated by FGSM method with $\varepsilon = 0.02$.

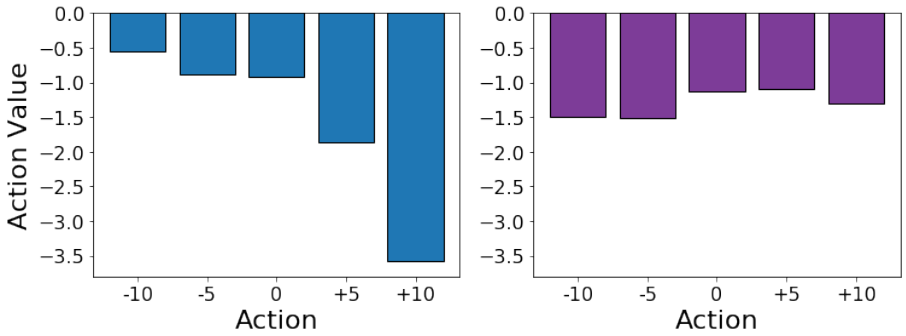


Figure 5.39: Action-values comparison for state A. The left is for the original state and the right is for the perturbed state.

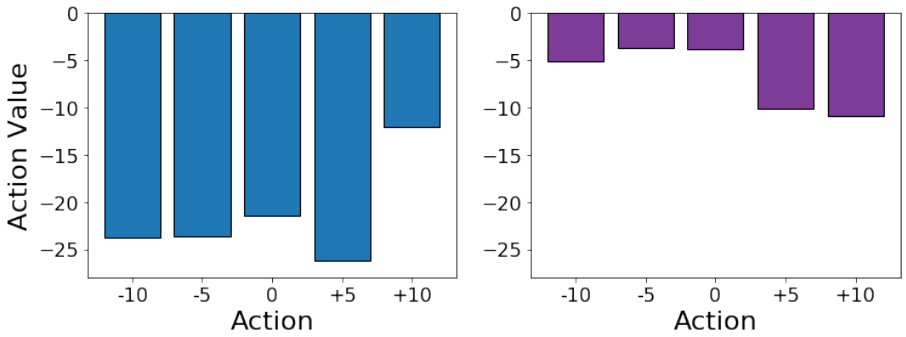


Figure 5.40: Action-values comparison for state B. The left is for the original state and the right is for the perturbed state.

It should be emphasized that the results reported from Figure 5.32- Figure 5.36 are all mean scores over all the 52 trips. Consequently, the methods used here to generate adversarial examples are not guaranteed to degrade the performance of all trips. Addressing two limitations of the methods can help improve the effectiveness of the adversarial examples.

First, all the methods used in this work come from the area of computer vision. They ignore a key notion from reinforcement learning which is that the inputs at each timestep are correlated temporally. More efficient attacks could be achieved if the temporal

correlation is utilized. One way to exploit this property is to train an adversarial agent to generate adversarial examples. This can be formulated as a new RL problem which considers the environment and the target agent combined as a new environment. The goal of the adversarial agent is to decrease the cumulated reward of the target agent in the original environment. By designing the reward function of the new problem, the norm and frequency of attacks can be controlled. It is expected that with a well-trained adversarial agent, the same level of performance drop could be reached with even smaller perturbations and less frequent attacks. Work by [143] and [144] discuss two heuristic and easy to implement methods to reduce attack frequency using value functions .

Another limitation of the current methods is that no target action or target state is specified. The goal of a designed attack is to change an action so that it is different from what the target agent intends to do. This could be partly addressed by more sophisticated and time-consuming methods like combining a generative model and planning algorithm [144] or an iterative least-likely class method [141].

There are at least two directions that could be taken to build a more robust agent against possible adversarial attacks. First, detection methods could be utilized to monitor the inputs. Second, adversarial training could be applied [146][147].

Robustness is one of the most important considerations when building technologies to solve real-world problems. In this work, the impact of adversarial examples generated by several fast methods under white-box and black-box conditions are investigated on a deep RL-based vehicle energy management system. This work shows that on average adversarial examples can significantly degrade the performance of a target agent in an

EMS compared with random noise. As deep RL becomes more and more popular in the area of intelligent transportation systems, researchers will need to understand and prepare for a range of adversarial attacks. Threats to safety-critical systems will need especially careful consideration.

6. CONCLUSION

Development of energy management systems for HEVs and PHEVs has advanced significantly in the last two decades. Yet, there are still gaps between simulation and real-world applications, for example, future driving cycles are usually unknown, and detailed trip information is difficult to predict due to complex and stochastic factors in real-world transportation systems. In this dissertation, data-driven methods were presented to bridge the gaps, which includes both traditional machine learning techniques like Bayesian inference and emerging algorithms like deep reinforcement learning to improve the fuel efficiency of in-use EREVs used in package delivery applications.

The Bayesian inference-based method developed here is a very conservative approach for energy management in vehicles and is especially useful when the number of historical trips associated with the target vehicle is low. Also, it is easy to implement and requires low computational resources. Consequently, it is relatively easy to deploy in a delivery fleet. The downside of this approach is that the fuel efficiency improvement is moderate. The two main reasons are: first, the L_{set} is only updated before the trip so that the real-time information is not used. Second, even there are large amount of data available, the average fuel efficiency improvement will still be moderate due to the conservative nature of the algorithm.

The DRL-based approaches in this work have some of the opposite characteristics to the Bayesian method. This approach assumes that the number of historical delivery trip data is large enough to represent future trips. Therefore, the intelligent agent trained based on the simulator which is built with historical data and vehicle model can adaptively

update the value of L_{set} in future trips. This approach can achieve high fuel efficiencies, however, the training of each intelligent agent for each vehicle needs careful tuning and validation, which may prevent it from being used on a large scale. There are always trade-offs in real-world applications. If an algorithm is easy to use, the performance might be compromised. If an algorithm can achieve near optimal performance, the feasibility might be low. The advantages and disadvantages of the two proposed approaches can be explained by this trade-off.

Some challenges may be encountered when applying the developed DRL approach in real-world applications. Three types of problems are identified and presented in this work, along with possible solutions and future research directions. First, the assumption of the developed DRL-based approach is that the historical trips used to build the simulation environment can accurately represent future trips; however, this might not hold true, meaning that some future states are unfamiliar or novel to the intelligent agent. Under this condition, the action calculated by the agent is not guaranteed to work and may lead to poor performance, or even dangerous conditions like running out of battery when driving on the highway. To address these challenges, an uncertainty-aware strategy was built that can capture the out-of-distribution inputs and inform the system so that the EMS can switch to predefined rules or notify the driver. Secondly, standard RL algorithms make decisions based on expected return. This works well if the underlying return distributions are unimodal with low variance. However, it is shown in this work that the underlying return distributions are not always unimodal and may have high variance, which means the expected value losses important information of the future return. To solve this problem,

a risk-aware agent was developed, which makes decisions based on CVaR, leading risk-averse strategies. Lastly, influences of adversarial attacks on DRL agents are investigated. It is shown that the adversarial examples can significantly disturb the agent and need to be protected against.

For future work in this problem area, many directions can be further studied. Some directions are listed as of them are:

- Initialize the NN-based agent: One of the disadvantages of RL-based algorithms is that it requires carefully tuning for each individual vehicle since the characteristics of each trip data are diverse. This leads to training each NN-based agent from random initialization, which will explore many obviously bad policies for all vehicles at the beginning of the interaction. However, it is intuitive that although the policies should be different for different vehicles, some basic strategies should be similar, for example, when the battery SOC is low and close to 10%, the engine should be turned on, and there is no benefit to increase the L_{set} if it is already very high. Consequently, for a new vehicle, the NN-based agent can be initialized to some existing NNs so that the training time can be reduced as it is not from random. This idea is called transfer learning in deep learning as well as in deep RL [148].
- Improve the Vehicle Model: Further improvements can be achieved by improving the environment, which consists of the historical trip data and a simplified vehicle model in this work. The main idea is to bridge the gap between simulation and reality. Some related work in other application domains can be found in [149]-

[151]. Also, it is possible to build a model of the environment, which itself is a research topic. This requires large amount of data and more advanced transportation infrastructures. Some model-based RL algorithms are listed here [152]-[158]. A straightforward and relatively easy next step for this work is to manually or automatically build new trips using the available data; i.e., combining different trips into one trip. The velocity profiles and GPS trajectories of the combined new trips are all realistic, but the combination is novel. Generative models [159]-[163] can be potentially used in this process.

- Build an Interpretable Agent: Although the RL-based agent can achieve good performance as demonstrated in this work, it is a black-box method so that the decisions are not interpretable. Therefore, a potential research direction is to extract explicit rules from the trained agent or build an interpretable agent. This direction is not only useful in our EMS application, but is also highly beneficial to nearly all RL applications and data-driven methods involving NNs [164][165].
- Defend Against Adversarial Attacks: It is demonstrated that the performance of NN-based agents can be influenced by adversarial attacks. Therefore, agents that can resist adversarial methods are of great importance for real-world applications [166]-[169]. More robust strategies for mitigating the impacts of adversarial attacks are advised in future research.

This dissertation not only presents a systematic framework for the energy management problem of in-use EREVs for package delivery, but also provides useful directions for solving many other problems in the area of intelligent transportation systems. The data-

driven methods used in this work can be applied to other vehicle applications so long as the nature of the problems is similar. For example, they here have already been applied to powertrain mode switching in power-split plug-in hybrid vehicles [170]. The three challenges identified in this work also exist in many ITS-related applications like autonomous driving [114][115]. Methods and discussion contained in this dissertation can provide some initial insights and guidance to deal with other domain-specific problems.

BIBLIOGRAPHY

[1] U.S. EIA, “Monthly Energy Review”, tables 1.1, 1.2, 1.3, 1.4a, 1.4b and 2.1, April 2019.

[2] “Inventory of U.S. greenhouse gas emissions and sinks: 1990–2009,” U.S. Environmental Protection Agency (EPA), Washington, DC, USA, Final Rep., EPA 430-R-11-005, Apr. 2011.

[3] U.S. EIA, "Annual Energy Outlook 2011 with Projections to 2035," U.S. Energy Information Administration,2010.

[4] Salisa, Abdul Rahman, Nong Zhang, and J. G. Zhu. "A comparative analysis of fuel economy and emissions between a conventional HEV and the UTS PHEV." *IEEE Transactions on Vehicular Technology* 60, no. 1 (2010): 44-54.

[5] Martinez, Clara Marina, Xiaosong Hu, Dongpu Cao, Efstathios Velenis, Bo Gao, and Matthias Wellers. "Energy management in plug-in hybrid electric vehicles: Recent progress and a connected vehicles perspective." *IEEE Transactions on Vehicular Technology* 66, no. 6 (2016): 4534-4549.

[6] Guzzella, Lino, and Antonio Sciarretta. *Vehicle propulsion systems*. Vol. 1. Springer-Verlag Berlin Heidelberg, 2007.

[7] Michel, Pierre, Alain Charlet, Guillaume Colin, Yann Chamaillard, Gérard Bloch, and Cédric Nouillant. Pollution constrained optimal energy management of a gasoline-HEV. No. 2013-24-0083. SAE Technical Paper, 2013.

[8] Akar, Furkan, Yakup Tavlasoglu, and Bulent Vural. "An energy management strategy for a concept battery/ultracapacitor electric vehicle with improved battery life." IEEE Transactions on Transportation Electrification 3, no. 1 (2016): 191-200.

[9] Tang, Li, Giorgio Rizzoni, and Simona Onori. "Energy management strategy for HEVs including battery life optimization." IEEE Transactions on Transportation Electrification 1, no. 3 (2015): 211-222.

[10] Zhu, Li, Fei Richard Yu, Yige Wang, Bin Ning, and Tao Tang. "Big data analytics in intelligent transportation systems: A survey." IEEE Transactions on Intelligent Transportation Systems 20, no. 1 (2018): 383-398.

[11] Zhang, Junping, Fei-Yue Wang, Kunfeng Wang, Wei-Hua Lin, Xin Xu, and Cheng Chen. "Data-driven intelligent transportation systems: A survey." IEEE Transactions on Intelligent Transportation Systems 12, no. 4 (2011): 1624-1639.

[12] Rodrigues, Maradona, Steve King, Dan Scott, and Ducai Wang. Advanced energy management strategies for range extended electric vehicle. No. 2015-26-0121. SAE Technical Paper, 2015.

[13] Pozzato, Gabriele, Simone Formentin, Giulio Panzani, and Sergio M. Savaresi. "Least costly energy management for extended range electric vehicles with start-up characterization." In 2018 IEEE Conference on Control Technology and Applications (CCTA), pp. 1020-1025. IEEE, 2018.

[14] Kalia, Aman V., and Brian C. Fabien. "On Implementing Optimal Energy Management for EREV using Distance Constrained Adaptive Real-Time Dynamic Programming." Electronics 9, no. 2 (2020): 228.

[15] Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." *Nature methods* 17, no. 3 (2020): 261-272.

[16] Huang, Bufu, Xi Shi, and Yangsheng Xu. "Parameter optimization of power control strategy for series hybrid electric vehicle." In *2006 IEEE International Conference on Evolutionary Computation*, pp. 1989-1994. IEEE, 2006.

[17] Tulpule, P., Vincenzo Marano, and Giorgio Rizzoni. "Effects of different PHEV control strategies on vehicle performance." In *2009 American Control Conference*, pp. 3950-3955. IEEE, 2009.

[18] Wirasingha, Sanjaka G., and Ali Emadi. "Classification and review of control strategies for plug-in hybrid electric vehicles." *IEEE Transactions on vehicular technology* 60, no. 1 (2010): 111-122.

[19] Zhang, Fengqi, Xiaosong Hu, Reza Langari, and Dongpu Cao. "Energy management strategies of connected HEVs and PHEVs: Recent progress and outlook." *Progress in Energy and Combustion Science* 73 (2019): 235-256.

[20] Zeng, Xiangrui, and Junmin Wang. "A parallel hybrid electric vehicle energy management strategy using stochastic model predictive control with road grade preview." *IEEE Transactions on Control Systems Technology* 23, no. 6 (2015): 2416-2423.

[21] Martinez, Clara Marina, Mira Heucke, Fei-Yue Wang, Bo Gao, and Dongpu Cao. "Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey." *IEEE Transactions on Intelligent Transportation Systems* 19, no. 3 (2017): 666-676.

[22] Ripaccioli, Giulio, Daniele Bernardini, Stefano Di Cairano, Alberto Bemporad, and I. V. Kolmanovsky. "A stochastic model predictive control approach for series hybrid electric vehicle power management." In *Proceedings of the 2010 American Control Conference*, pp. 5844-5849. IEEE, 2010.

[23] Chen, Zheng, Chunting Chris Mi, Jun Xu, Xianzhi Gong, and Chenwen You. "Energy management for a power-split plug-in hybrid electric vehicle based on dynamic programming and neural networks." *IEEE Transactions on Vehicular Technology* 63, no. 4 (2013): 1567-1580.

[24] Nguyen, Bao-Huy, Ronan German, João Pedro F. Trovão, and Alain Bouscayrol. "Real-time energy management of battery/supercapacitor electric vehicles based on an adaptation of Pontryagin's minimum principle." *IEEE Transactions on Vehicular Technology* 68, no. 1 (2018): 203-212.

[25] Tianheng, Feng, Yang Lin, Gu Qing, Hu Yanqing, Yan Ting, and Yan Bin. "A supervisory control strategy for plug-in hybrid electric vehicles based on energy demand prediction and route preview." *IEEE Transactions on Vehicular Technology* 64, no. 5 (2014): 1691-1700.

[26] Musardo, Cristian, Giorgio Rizzoni, Yann Guezennec, and Benedetto Staccia. "A-ECMS: An adaptive algorithm for hybrid electric vehicle energy management." *European Journal of Control* 11, no. 4-5 (2005): 509-524.

[27] Zeng, Xiangrui, and Junmin Wang. "Optimizing the energy management strategy for plug-in hybrid electric vehicles with multiple frequent routes." *IEEE Transactions on Control Systems Technology* 27, no. 1 (2017): 394-400.

[28] Zhang, Shuwei, Yugong Luo, Junmin Wang, Xiao Wang, and Keqiang Li. "Predictive energy management strategy for fully electric vehicles based on preceding vehicle movement." *IEEE Transactions on Intelligent Transportation Systems* 18, no. 11 (2017): 3049-3060.

[29] Xie, Shaobo, Xiaosong Hu, Zongke Xin, and Liang Li. "Time-efficient stochastic model predictive energy management for a plug-in hybrid electric bus with an adaptive reference state-of-charge advisory." *IEEE Transactions on Vehicular Technology* 67, no. 7 (2018): 5671-5682.

[30] Anselma, Pier Giuseppe, Yi Huo, Joel Roeleveld, Giovanni Belingardi et al., "Integration of On-Line Control in Optimal Design of Multimode Power-Split Hybrid Electric Vehicle Powertrains." *IEEE Transactions on Vehicular Technology* 68, no. 4 (2019): 3436-3445.

[31] Stroe, Nicoleta, Sorin Olaru, Guillaume Colin, Karim Ben-Cherif et al., "Predictive Control Framework for HEV: Energy Management and Free-Wheeling Analysis." *IEEE Transactions on Intelligent Vehicles* 4, no. 2 (2019): 220-231.

[32] Zhang, Chen, Ardalan Vahidi, Pierluigi Pisu, Xiaopeng Li, and Keith Tennant. "Role of terrain preview in energy management of hybrid electric vehicles." *IEEE transactions on Vehicular Technology* 59, no. 3 (2009): 1139-1147.

[33] Kermani, Saida, Rochdi Trigui, Sebastien Delprat, Bruno Jeanneret, and Thierry Marie Guerra. "PHIL implementation of energy management optimization for a parallel HEV on a predefined route." *IEEE Transactions on Vehicular Technology* 60, no. 3 (2011): 782-792.

[34] Wu, Guoyuan, Kanok Boriboonsomsin, and Matthew J. Barth. "Development and evaluation of an intelligent energy-management strategy for plug-in hybrid electric vehicles." *IEEE Transactions on Intelligent Transportation Systems* 15, no. 3 (2014): 1091-1100.

[35] Sun, Chao, Scott Jason Moura, Xiaosong Hu, J. Karl Hedrick, and Fengchun Sun. "Dynamic traffic feedback data enabled energy management in plug-in hybrid electric vehicles." *IEEE Transactions on Control Systems Technology* 23, no. 3 (2014): 1075-1086.

[36] Zhang, Fengqi, Junqiang Xi, and Reza Langari. "Real-time energy management strategy based on velocity forecasts using V2V and V2I communications." *IEEE Transactions on Intelligent Transportation Systems* 18, no. 2 (2016): 416-430.

[37] Qi, Xuewei, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J. Barth. "Development and evaluation of an evolutionary algorithm-based online energy management system for plug-in hybrid electric vehicles." *IEEE Transactions on Intelligent Transportation Systems* 18, no. 8 (2016): 2181-2191.

[38] Chen, Di, Youngki Kim, and Anna G. Stefanopoulou. "State of charge node planning with segmented traffic information." In *2018 Annual American Control Conference (ACC)*, pp. 4969-4974. IEEE, 2018.

[39] Rama, Neeraj, Huanqing Wang, Joshua Orlando, Darrell Robinette, and Bo Chen. *Route-Optimized Energy Management of Connected and Automated Multi-Mode Plug-In Hybrid Electric Vehicle Using Dynamic Programming*. No. 2019-01-1209. SAE Technical Paper, 2019.

[40] McNew, John-Michael. "Predicting cruising speed through data-driven driver modeling." In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1789-1796. IEEE, 2012.

[41] Bender, Frank A., Martin Kaszynski, and Oliver Sawodny. "Drive cycle prediction and energy management optimization for hybrid hydraulic vehicles." *IEEE Transactions on vehicular technology* 62, no. 8 (2013): 3581-3592.

[42] Platho, Matthias, Horst-Michael Groß, and Julian Eggert. "Predicting velocity profiles of road users at intersections using configurations." In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 945-951. IEEE, 2013.

[43] Di Cairano, Stefano, Daniele Bernardini, Alberto Bemporad et al., "Stochastic MPC with learning for driver-predictive vehicle control and its application to HEV energy

management." *IEEE Transactions on Control Systems Technology* 22, no. 3 (2013): 1018-1031.

[44] Lefèvre, Stéphanie, Chao Sun, Ruzena Bajcsy et al., "Comparison of parametric and non-parametric approaches for vehicle speed prediction." In *2014 American Control Conference*, pp. 3494-3499. IEEE, 2014.

[45] Park, Jungme, Yi Lu Murphey, Ryan McGee et al., "Intelligent trip modeling for the prediction of an origin–destination traveling speed profile." *IEEE Transactions on Intelligent Transportation Systems* 15, no. 3 (2014): 1039-1053.

[46] Ziegmann, Johannes, Jieqing Shi, Tobias Schnörer, et al., "Analysis of individual driver velocity prediction using data-driven driver models with environmental features." In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 517-522. IEEE, 2017.

[47] Jiang, Bingnan, and Yunsi Fei. "Vehicle speed prediction by two-level data driven models in vehicular networks." *IEEE Transactions on Intelligent Transportation Systems* 18, no. 7 (2016): 1793-1801.

[48] Simmons, Reid, Brett Browning, Yilu Zhang, and Varsha Sadekar. "Learning to predict driver route and destination intent." In *2006 IEEE Intelligent Transportation Systems Conference*, pp. 127-132. IEEE, 2006.

[49] Froehlich, Jon, and John Krumm. *Route prediction from trip observations*. No. 2008-01-0201. SAE Technical Paper, 2008.

[50] Larsson, Viktor, Lars Johannesson Mårdh, Bo Egardt, and Sten Karlsson. "Commuter route optimized energy management of hybrid electric vehicles." *IEEE transactions on intelligent transportation systems* 15, no. 3 (2014): 1145-1154.

[51] Xu, Bin, Farzam Malmir, Dhruvang Rathod, and Zoran Filipi. *Real-time reinforcement learning optimized energy management for a 48V mild hybrid electric vehicle*. No. 2019-01-1208. SAE Technical Paper, 2019.

[52] Lin, Xue, Paul Bogdan, Naehyuck Chang, and Massoud Pedram. "Machine learning-based energy management in a hybrid electric vehicle to minimize total operating cost." In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 627-634. IEEE, 2015.

[53] Liu, Chang, and Yi Lu Murphey. "Power management for Plug-in Hybrid Electric Vehicles using Reinforcement Learning with trip information." In *2014 IEEE Transportation Electrification Conference and Expo (ITEC)*, pp. 1-6. IEEE, 2014.

[54] Liu, Teng, Yuan Zou, Dexing Liu, and Fengchun Sun. "Reinforcement learning of adaptive energy management with transition probability for a hybrid electric tracked vehicle." *IEEE Transactions on Industrial Electronics* 62, no. 12 (2015): 7837-7846.

[55] Xu, Bin, Dhruvang Rathod, Darui Zhang, Adamu Yebi, Xueyu Zhang, Xiaoya Li, and Zoran Filipi. "Parametric study on reinforcement learning optimized energy management strategy for a hybrid electric vehicle." *Applied Energy* (2019): 114200.

[56] Zhao, Pu, Yanzhi Wang, Naehyuck Chang, Qi Zhu, and Xue Lin. "A deep reinforcement learning framework for optimizing fuel economy of hybrid electric vehicles." In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 196-202. IEEE, 2018.

[57] Liu, Teng, Xiaosong Hu, Shengbo Eben Li, and Dongpu Cao. "Reinforcement learning optimized look-ahead energy management of a parallel hybrid electric vehicle." *IEEE/ASME Transactions on Mechatronics* 22, no. 4 (2017): 1497-1507.

[58] Wu, Jingda, Hongwen He, Jiankun Peng, Yuecheng Li, and Zhanjiang Li. "Continuous reinforcement learning of energy management with deep Q network for a power split hybrid electric bus." *Applied energy* 222 (2018): 799-811.

[59] Han, Xuefeng, Hongwen He, Jingda Wu, Jiankun Peng, and Yuecheng Li. "Energy management based on reinforcement learning with double deep Q-learning for a hybrid electric tracked vehicle." *Applied Energy* 254 (2019): 113708.

[60] Li, Yuecheng, Hongwen He, Jiankun Peng, and Hong Wang. "Deep Reinforcement Learning-Based Energy Management for a Series Hybrid Electric Vehicle Enabled by History Cumulative Trip Information." *IEEE Transactions on Vehicular Technology* 68, no. 8 (2019): 7416-7430.

[61] Hu, Yue, Weimin Li, Kun Xu, Taimoor Zahid, Feiyan Qin, and Chenming Li. "Energy management strategy for a hybrid electric vehicle based on deep reinforcement learning." *Applied Sciences* 8, no. 2 (2018): 187.

[62] Zou, Yuan, Teng Liu, Dexing Liu, and Fengchun Sun. "Reinforcement learning-based real-time energy management for a hybrid tracked vehicle." *Applied energy* 171 (2016): 372-382.

[63] Xiong, Rui, Jiayi Cao, and Quanqing Yu. "Reinforcement learning-based real-time power management for hybrid energy storage system in the plug-in hybrid electric vehicle." *Applied energy* 211 (2018): 538-548.

[64] Qi, Xuewei, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J. Barth. "A novel blended real-time energy management strategy for plug-in hybrid electric vehicle commute trips." In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pp. 1002-1007. IEEE, 2015.

[65] Chen, Zheng, Liang Li, Xiaosong Hu, Bingjie Yan, and Chao Yang. "Temporal-Difference Learning-Based Stochastic Energy Management for Plug-in Hybrid Electric Buses." *IEEE Transactions on Intelligent Transportation Systems* 20, no. 6 (2018): 2378-2388.

[66] Hu, Xiaosong, Teng Liu, Xuewei Qi, and Matthew Barth. "Reinforcement Learning for Hybrid and Plug-In Hybrid Electric Vehicle Energy Management: Recent Advances and Prospects." *IEEE Industrial Electronics Magazine* 13, no. 3 (2019): 16-25.

[67] Rousseau, Aymeric, Sylvain Pagerit, and David Wenzhong Gao. "Plug-in hybrid electric vehicle control strategy parameter optimization." *Journal of Asian Electric Vehicles* 6, no. 2 (2008): 1125-1133.

[68] Taherzadeh, Erfan, Morteza Dabbaghjamanesh, Mohsen Gitizadeh, and Akbar Rahideh. "A new efficient fuel optimization in blended charge depletion/charge sustenance control strategy for plug-in hybrid electric vehicles." *IEEE Transactions on Intelligent Vehicles* 3, no. 3 (2018): 374-383.

[69] Overington, Shane, and Sumedha Rajakaruna. "High-efficiency control of internal combustion engines in blended charge depletion/charge sustenance strategies for plug-in hybrid electric vehicles." *IEEE Transactions on Vehicular Technology* 64, no. 1 (2014): 48-61.

[70] Lin, Chan-Chiao, Huei Peng, Jessy W. Grizzle, and Jun-Mo Kang. "Power management strategy for a parallel hybrid electric truck." *IEEE transactions on control systems technology* 11, no. 6 (2003): 839-849.

[71] Kum, Dongsuk, Huei Peng, and Norman K. Bucknor. "Optimal energy and catalyst temperature management of plug-in hybrid electric vehicles for minimum fuel consumption and tail-pipe emissions." *IEEE Transactions on Control Systems Technology* 21, no. 1 (2011): 14-26.

[72] Pam, Abdoulaye, Alain Bouscayrol, Philippe Fiani, and Frederic Noth. "Rule-Based Energy Management Strategy for a Parallel Hybrid Electric Vehicle Deduced from Dynamic Programming." In *2017 IEEE Vehicle Power and Propulsion Conference (VPPC)*, pp. 1-6. IEEE, 2017.

[73] Padmarajan, Brahmadevan V., Andrew McGordon, and Paul A. Jennings. "Blended rule-based energy management for PHEV: System structure and strategy." *IEEE Transactions on Vehicular Technology* 65, no. 10 (2015): 8757-8762.

[74] Kochenderfer, Mykel J. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[75] Alpaydin, Ethem. *Introduction to machine learning*. MIT press, 2020.

[76] Murphy, Kevin P. "Conjugate Bayesian analysis of the Gaussian distribution." *def* 1, no. 2σ2 (2007): 16.

[77] Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006.

[78] Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian data analysis*. CRC press, 2013.

[79] Freyermuth, Vincent, Eric Fallas, and Aymeric Rousseau. "Comparison of powertrain configuration for plug-in HEVs from a fuel economy perspective." *SAE International Journal of Engines* 1, no. 1 (2009): 392-398.

[80] Bertsekas, Dimitri P., and John N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[81] Bertsekas, Dimitri P. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[82] BERTSEKAS, DIMITRI P. "Approximate Dynamic Programming." (2012).

[83] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[84] Russell, Stuart, and Peter Norvig. "Artificial intelligence: a modern approach." (2002).

[85] Sugiyama, Masashi. *Statistical reinforcement learning: modern machine learning approaches*. CRC Press, 2015.

[86] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." *Machine learning* 8, no. 3-4 (1992): 279-292.

[87] Bellemare, Marc, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. "Unifying count-based exploration and intrinsic motivation." In *Advances in neural information processing systems*, pp. 1471-1479. 2016.

[88] Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN." In *Advances in neural information processing systems*, pp. 4026-4034. 2016.

[89] Bishop, Christopher M. *Neural networks for pattern recognition*. Oxford university press, 1995.

[90] Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2, no. 4 (1989): 303-314.

[91] Funahashi, KenIchi. "On the approximate realization of continuous mappings by neural networks." Neural networks 2, no. 3 (1989): 183-192.

[92] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." nature 323, no. 6088 (1986): 533-536.

[93] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436-444.

[94] Bertsekas, Dimitri P. "Nonlinear programming." Journal of the Operational Research Society 48, no. 3 (1997): 334-334.

[95] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "PyTorch: An imperative style, high-performance deep learning library." In *Advances in Neural Information Processing Systems*, pp. 8024-8035. 2019.

[96] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807-814. 2010.

[97] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[98] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." *Nature* 518, no. 7540 (2015): 529-533.

[99] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In *Thirtieth AAAI conference on artificial intelligence*. 2016.

[100] Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." *arXiv preprint arXiv:1511.05952* (2015).

[101] Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. "Dueling network architectures for deep reinforcement learning." *arXiv preprint arXiv:1511.06581* (2015).

[102] Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih et al. "Noisy networks for exploration." *arXiv preprint arXiv:1706.10295* (2017).

[103] Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. "Rainbow: Combining improvements in deep reinforcement learning." In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[104] Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms." 2014.

[105] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[106] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In *International conference on machine learning*, pp. 1889-1897. 2015.

[107] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

[108] Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation." *arXiv preprint arXiv:1506.02438* (2015).

[109] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." *arXiv preprint arXiv:1801.01290* (2018).

[110] Wang, Ziyu, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay." *arXiv preprint arXiv:1611.01224* (2016).

[111] O'Keefe, Michael P., Andrew Simpson, Kenneth J. Kelly, and Daniel S. Pedersen. *Duty cycle characterization and evaluation towards heavy hybrid vehicle applications*. No. 2007-01-0302. SAE Technical Paper, 2007.

[112] Du, Yu, Wei ShangGuan, Dingchao Rong, and Linguo Chai. "RA-TSC: Learning Adaptive Traffic Signal Control Strategy via Deep Reinforcement Learning." In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3275-3280. IEEE, 2019.

[113] Liang, Xiaoyuan, Xunsheng Du, Guiling Wang, and Zhu Han. "A deep reinforcement learning network for traffic light cycle control." *IEEE Transactions on Vehicular Technology* 68, no. 2 (2019): 1243-1253.

[114] Min, Kyushik, Hayoung Kim, and Kunsoo Huh. "Deep Distributional Reinforcement Learning Based High-Level Driving Policy Determination." *IEEE Transactions on Intelligent Vehicles* 4, no. 3 (2019): 416-424.

[115] Lötjens, Björn, Michael Everett, and Jonathan P. How. "Safe reinforcement learning with model uncertainty estimates." In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8662-8668. IEEE, 2019.

[116] Vinitsky, Eugene, Kanaad Parvate, Aboudy Kreidieh, Cathy Wu, and Alexandre Bayen. "Lagrangian control through deep-rl: Applications to bottleneck decongestion." In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 759-765. IEEE, 2018.

[117] Wang, Zhaodong, Zhiwei Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. "Deep reinforcement learning with knowledge transfer for online rides order dispatching." In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 617-626. IEEE, 2018.

[118] Wang, Pengyue, Yan Li, Shashi Shekhar, and William F. Northrop. "A deep reinforcement learning framework for energy management of extended range electric delivery vehicles." In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1837-1842. IEEE, 2019.

[119] Wang, Pengyue, Yan Li, Shashi Shekhar, and William F. Northrop. "Actor-Critic based Deep Reinforcement Learning Framework for Energy Management of Extended Range Electric Delivery Vehicles." In *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1379-1384. IEEE, 2019.

[120] Osband, Ian. "Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout." In *NIPS Workshop on Bayesian Deep Learning*, vol. 192. 2016.

[121] Gal, Yarin. "Uncertainty in deep learning." *University of Cambridge* 1 (2016): 3.

[122] Pearce, Tim, Nicolas Anastassacos, Mohamed Zaki, and Andy Neely. "Bayesian Inference with Anchored Ensembles of Neural Networks, and Application to Exploration in Reinforcement Learning." *arXiv preprint arXiv:1805.11324* (2018).

[123] Dearden, Richard, Nir Friedman, and Stuart Russell. "Bayesian Q-learning." In *Aaai/iaai*, pp. 761-768. 1998.

[124] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449-458. JMLR. org, 2017.

[125] Rowland, Mark, Marc G. Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. "An analysis of categorical distributional reinforcement learning." *arXiv preprint arXiv:1802.08163* (2018).

[126] Dabney, Will, Mark Rowland, Marc G. Bellemare, and Rémi Munos. "Distributional reinforcement learning with quantile regression." In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[127] Dabney, Will, Georg Ostrovski, David Silver, and Rémi Munos. "Implicit quantile networks for distributional reinforcement learning." *arXiv preprint arXiv:1806.06923* (2018).

[128] Majumdar, Anirudha, and Marco Pavone. "How should a robot assess risk? Towards an axiomatic theory of risk in robotics." In *Robotics Research*, pp. 75-84. Springer, Cham, 2020.

[129] Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles." In *Advances in Neural Information Processing Systems*, pp. 6402-6413. 2017.

[130] Lu, Xiuyuan, and Benjamin Van Roy. "Ensemble sampling." In *Advances in neural information processing systems*, pp. 3258-3266. 2017.

[131] Fort, Stanislav, Huiyi Hu, and Balaji Lakshminarayanan. "Deep Ensembles: A Loss Landscape Perspective." *arXiv preprint arXiv:1912.02757* (2019).

[132] Pearce, Tim, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. "High-quality prediction intervals for deep learning: A distribution-free, ensembled approach." *arXiv preprint arXiv:1802.07167* (2018).

[133] Khosravi, Abbas, Saeid Nahavandi, Doug Creighton, and Amir F. Atiya. "Comprehensive review of neural network-based prediction intervals and new advances." *IEEE Transactions on neural networks* 22, no. 9 (2011): 1341-1356.

[134] Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete problems in AI safety." *arXiv preprint arXiv:1606.06565* (2016).

[135] McAllister, Rowan, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning." International Joint Conferences on Artificial Intelligence, Inc., 2017.

[136] Alshiekh, Mohammed, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. "Safe reinforcement learning via shielding." In *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[137] Berkenkamp, Felix, Matteo Turchetta, Angela Schoellig, and Andreas Krause. "Safe model-based reinforcement learning with stability guarantees." In *Advances in neural information processing systems*, pp. 908-918. 2017.

[138] MacKay, David JC, and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[139] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *Proceedings of the 2014 International Conference on Learning Representations*. Computational and Biological Learning Society, 2014.

[140] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *Proceedings of the 2015 International Conference on Learning Representations*. Computational and Biological Learning Society, 2015.

[141] Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." *arXiv preprint arXiv:1607.02533* (2016).

[142] Huang, Sandy, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. "Adversarial attacks on neural network policies." *arXiv preprint arXiv:1702.02284* (2017).

[143] Kos, Jernej, and Dawn Song. "Delving into adversarial attacks on deep policies." *arXiv preprint arXiv:1705.06452* (2017).

[144] Lin, Yen-Chen, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. "Tactics of adversarial attack on deep reinforcement learning agents." *arXiv preprint arXiv:1703.06748* (2017).

[145] Xiao, Chaowei, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Bo Li, and Dawn Song. "Characterizing attacks on deep reinforcement learning." *arXiv preprint arXiv:1907.09470* (2019).

[146] Mandlekar, Ajay, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. "Adversarially robust policy learning: Active construction of physically-plausible perturbations." In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3932-3939. IEEE, 2017.

[147] Pinto, Lerrel, James Davidson, Rahul Sukthankar, and Abhinav Gupta. "Robust adversarial reinforcement learning." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817-2826. JMLR. org, 2017.

[148] Taylor, Matthew E., and Peter Stone. "Transfer learning for reinforcement learning domains: A survey." *Journal of Machine Learning Research* 10, no. Jul (2009): 1633-1685.

[149] Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world." In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23-30. IEEE, 2017.

[150] Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-real transfer of robotic control with dynamics randomization." *In 2018 IEEE international conference on robotics and automation (ICRA),* pp. 1-8. IEEE, 2018.

[151] Bousmalis, Konstantinos, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping." *In 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4243-4250. IEEE, 2018.

[152] Abbeel, Pieter, Morgan Quigley, and Andrew Y. Ng. "Using inaccurate models in reinforcement learning." *In Proceedings of the 23rd international conference on Machine learning*, pp. 1-8. 2006.

[153] Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models." *In Advances in Neural Information Processing Systems*, pp. 4754-4765. 2018.

[154] Nagabandi, Anusha, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning." *In 2018 IEEE International Conference on Robotics and Automation (ICRA),* pp. 7559-7566. IEEE, 2018.

[155] Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous deep q-learning with model-based acceleration." *In International Conference on Machine Learning,* pp. 2829-2838. 2016.

[156] Watter, Manuel, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images*." In Advances in neural information processing systems,* pp. 2746-2754. 2015.

[157] Deisenroth, Marc Peter, Carl Edward Rasmussen, and Dieter Fox. "Learning to control a low-cost manipulator using data-efficient reinforcement learning." *Robotics: Science and Systems VII* (2011): 57-64.

[158] Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465-472. 2011.

[159] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." *In Advances in neural information processing systems*, pp. 2672-2680. 2014.

[160] Kingma, Durk P., and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions." *In Advances in Neural Information Processing Systems*, pp. 10215-10224. 2018.

[161] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114* (2013).

[162] Yu, Lantao, Weinan Zhang, Jun Wang, and Yong Yu. "Seqgan: Sequence generative adversarial nets with policy gradient." *In Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[163] Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. "Wavenet: A generative model for raw audio." *arXiv preprint arXiv:1609.03499* (2016).

[164] Doshi-Velez, Finale, and Been Kim. "Towards a rigorous science of interpretable machine learning." *arXiv preprint arXiv:1702.08608* (2017).

[165] Gunning, David. "Explainable artificial intelligence (xai)." *Defense Advanced Research Projects Agency (DARPA), nd Web* 2 (2017).

[166] Cohen, Jeremy M., Elan Rosenfeld, and J. Zico Kolter. "Certified adversarial robustness via randomized smoothing." *arXiv preprint arXiv:1902.02918* (2019).

[167] Weng, Tsui-Wei, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S. Dhillon, and Luca Daniel. "Towards fast computation of certified robustness for relu networks." *arXiv preprint arXiv:1804.09699* (2018).

[168] Lütjens, Björn, Michael Everett, and Jonathan P. How. "Certified Adversarial Robustness for Deep Reinforcement Learning." *arXiv preprint arXiv:1910.12908* (2019).

[169] Lecuyer, Mathias, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. "Certified robustness to adversarial examples with differential privacy." *In 2019 IEEE Symposium on Security and Privacy (SP),* pp. 656-672. IEEE, 2019.

[170] Wang, Pengyue, and William Northrop. Reinforcement Learning based Energy Management of Multi-Mode Plug-in Hybrid Electric Vehicles for Commuter Route. No. 2020-01-1189. SAE Technical Paper, 2020.