Deep Learned Multi-Modal Traffic Agent Predictions for Truck Platooning Cut-Ins

by

Samuel Paul Douglass Jr.

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 2, 2020

Keywords: Time Series Forecasting, Truck Platoon, LSTM, Cut-in Detection, Trajectory Prediction

Approved by

Scott Martin, Chair, Assistant Research Professor of Mechanical Engineering
David Bevly, Professor of Mechanical Engineering
Anh Nguyen, Assistant Professor of Computer Science and Software Engineering

Abstract

Recent advances in Driver-Assisted Truck Platooning (DATP) have shown success in linking multiple trucks in leader-follower platoons using Cooperative Adaptive Cruise Control (CACC). Such setups allow for closer spacing between trucks which leads to fuel savings. Given that frontal collisions are the most common type of highway accident for heavy trucks, one key issue to truck platooning is handling situations in which vehicles cut-in between platooning trucks. Having more accurate and quicker predictions of cut-in behavior would improve the safety and efficiency of truck platooning by prompting the control system to react to the intruder sooner and allow for proper spacing before the cut-in occurs.

This thesis implements a deep neural network that generates multimodal predictions of traffic agents around a truck platoon in a simulated environment and culminates in testing on data obtained from the Auburn truck platoon. The method uses Long Short-Term Memory networks in an ensemble architecture to predict multiple possible future positions of vehicles passing by a truck platoon over a 5 second prediction horizon and classifies the potential vehicle behavior as 'passing' or 'cut-in' with prescribed certainties. The network performance is compared to a baseline of common state-based predictors including the Constant Velocity Predictor, the Constant Acceleration Predictor, and the Constant Turn Radius Predictor.

The Ensemble LSTM network is shown to be a promising predictor, outperforming state-based predictors over a 5 second prediction horizon with lower average and standard deviation of root mean squared error across 1000 test trajectories. The network is also shown to provide good predictions for a cut-in detector, which is able to accurately detect cut-in behavior on test trajectories with a balanced accuracy of 87.6 percent. Finally, the network is run on data collected from the Auburn truck platoon to demonstrate the viability of adapting the system to real world testing and development.

Acknowledgments

I cannot be more thankful for the opportunities and quality of education that has been provided to me by Auburn and the GAVLAB. I am truly lucky to have ended up here.

There are many who have helped me along through grad school and the writing of this thesis. Thanks to Drew Jennings and David Bell for being great partners to work with on class and research projects. Thanks to Amy Strong for editing all my reports, papers, and this thesis. Thanks to James Pool for encouraging me through the troughs of my research. Thanks to the guys in the truck team who have supported my work including Jacob Ward, Patrick Smith, and Dan Pierce. You have saved me months of work. Thanks to Josh Wood for always being available to answer homework questions. Thanks to Alec Letsinger who volunteered his time to aid me in processing data. Thanks to Matt Boler for providing encouragement each time I generated results. Thanks to Howard Chen who encouraged me to begin writing early, and thanks to Tanner Watts who gave me the inspiration for this work.

Thank you to my wife Michelle for sustaining me so that I could work longer hours. Thank you to my dad, who taught me how to read, write, and do math. Thank you to my mom, who has supported me in everything. Thanks to my grandparents for housing me at the beginning. And thanks to my Uncle Vito and Aunt Kristi for gifting me and my wife opportunities to travel that we would only dream of having.

Thanks to my professors at the University of Georgia who have given me research opportunities and encouraged me to attend graduate school incluing Dr. Knox, Dr. McCord, Dr. Lawrence, Dr. Rotavera, and Dr. Wagner. I wouldn't be here without you. Go Dawgs.

Lastly, thank you to Dr. Bevly and Dr. Martin for setting me on a career path to perform research in the areas I am passionate about.

Matthew 5:33-45

1 Corinthians 15:58

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background and Motivation

In 2018, the transportation industry in the United States reached $1.6 trillion, or 8% of Gross Domestic Product. Of this, revenues in the trucking industry totalled $700 million, with 11 billion tons of freight moved throughout the country [1]. The average class 8 truck travels 6 times as many miles as the average passenger vehicle and consumes 26 times as much fuel per year [25] [24]. This makes fuel consumption a large cost in the trucking industry, and even small improvements can lead to large savings. For example, if the FedEx truck fleet, which is comprised of roughly 25,000 trucks, improved gas mileage by 1% it would lead to $20 million in savings per year. [37]

One of the promising areas of fuel savings in the trucking industry is truck platooning. Truck platooning relies on Cooperative Adaptive Cruise Control (CACC) to allow trucks to follow one another at distances as close as 50 feet. By having close spacings between vehicles, the trucks experience fuel savings through drafting. CACC makes this close spacing possible by removing human reaction time from the driving process. [77] [68].

In the research and development of CACC for platooning trucks at Auburn University, there is a recurring problem of vehicles merging in between platooning trucks, otherwise known as a ”cut-in”. When a cut-in is detected, it causes the CACC to switch references from the lead truck to the cut-in vehicle in order to create more spacing between the follower truck and the cut-in vehicle. This process leads to reduced fuel savings as it requires more work from the longitudinal controller and can be dangerous if the gap is not wide enough by the time the

vehicle cuts in, and given that an 80,000 lb trailer requires between 400 and 500 ft of stopping distance on the highway [37], there isn't much margin for error. One potential solution to this problem is to begin creating the gap sooner, provided the platoon has knowledge of the future pose of the cut-in vehicle. This thesis explores various prediction methods for vehicles surrounding a simulated truck platoon, which is modeled to reflect scenarios encountered on the Auburn University truck platoon during highway testing.

The Auburn University truck platoon is comprised of two Peterbilt 579 and two military Freightliner M915 trucks. Highway testing involves the two Peterbilt trucks platooning at varying distances, of which the truck leading the platoon is referred to as the "lead" truck and the truck following the lead truck is referred to as the "follower" truck.

On board the follower truck in the Auburn University truck platoon is a 64 channel delphi radar which, coupled with DRTK relative positon vectors, is used to provide accurate range estimates between the follower and lead truck for the CACC as well as to detect vehicle cut-ins. The current method for detecting cut-ins is by estimating the intruding vehicle's local position relative to the follower truck via radar measurements, estimating lane lines using the relative position vector between the lead and follower trucks using radar measurements and DRTK, and signaling a cut-in if the vehicle crosses over an estimated lane line between the follower and lead truck. The lane lines are drawn by estimating a circular path between the lead and follower trucks using Pure Pursuit and then extrapolating the drawn path laterally on each side by a nominal lane width [77]. This holds well assuming the lead and follower trucks keep to the center of their lanes and that each truck remains in the same respective lane.

Predicting the future pose of vehicles is an emerging field with many new innovations coming in recent years. Currently, predictors most commonly used are state-based [32] and stochastic prediction methods such as Kalman Filtering with a Nearly Constant Acceleration Model [81], but the use of Machine Learning for prediction is gaining traction. Among the methods studied are Hidden Markov Models [71] [16], CNNs [21], and LSTMs [82] [16]. Two papers particularly influential to the network design in this thesis come from Cara et al. [16] at TNO Helmond and Cui et al. at Uber [21].

Previous work by Cara et al. has shown that using a constant velocity predictor (CV) provides a good model of cut-in behavior, with Support Vector Regression and K-Nearest Neighbor providing even better results [16]. In the study, these methods outperform Long Short-Term Memory Recurrent Neural Networks (LSTM), a common neural network architecture used in time series forecasting. However, the study used a small sample of 140 trajectories to train the LSTM, which they acknowledge is likely too small to achieve a good generalized fit due to their low number of cut-ins. Additionally, the predictions were produced in a unimodal nature which likely causes the LSTM to average the output behavior of potential future trajectories, leading to predictions that are less accurate. The average root mean squared error of the LSTM trained in their study compared to a constant velocity predictor is shown in Figure 1.1.

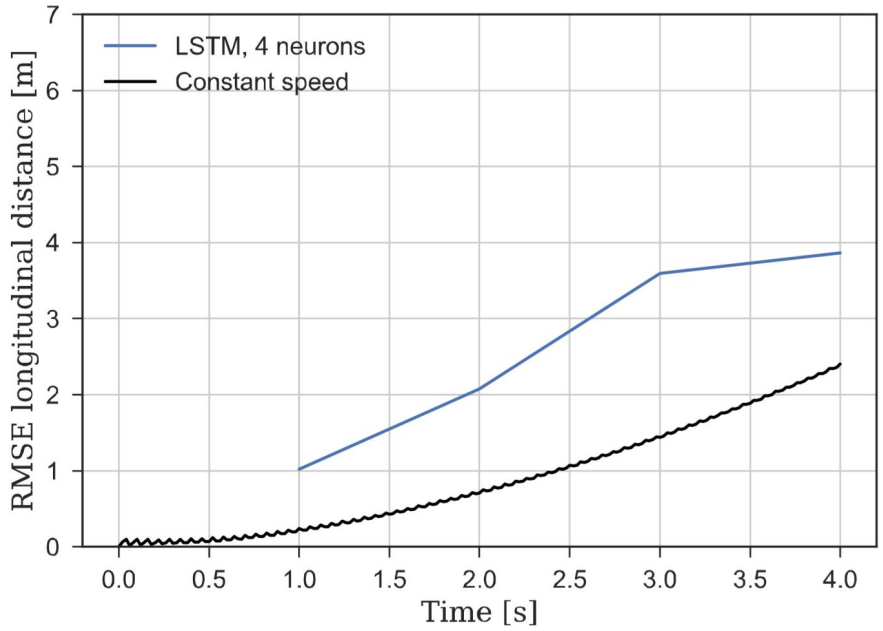
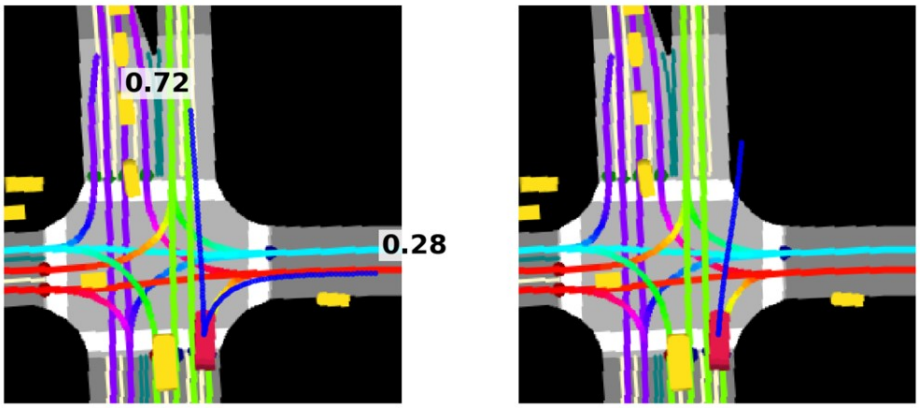


Figure 1.1: The RMSE in the longitudinal distance as function of time for prediction using an LSTM neural network consisting of 4 neurons. [16]

In other related work, Cui et al. used a Convolutional Neural Network (CNN) in conjunction with a Multilayer Perceptron Model (MLP) to generate multi-modal vehicle trajectory predictions of vehicle behavior in intersections [21]. The paper showed that Deep Learning networks have the ability to learn multiple behavior modes of traffic agents in an intersection and can accurately predict the correct mode of behavior that a vehicle approaching an intersection will perform. This is useful, as on a highway a vehicle neighboring the platoon has primarily

two decisions, whether to continue straight or to merge right. An example of the averaging prediction behavior is shown in Figure 1.2.



(c) Two-modal, time $T + 3$ (d) Single-modal, time $T + 3$

Figure 1.2: Multimodal vs Single Modal Predictions of a Vehicle at an Intersection [21]

Figure 1.2 shows a vehicle represented by a red rectangle approaching an intersection. The predictions of the vehicle behavior are shown as the blue lines extending from the vehicle. The plot on the right shows the result of a unimodal predictor, which averages the possible predicted outcomes into a single prediction that predicts the car to drive off the road. The image on the left uses two modes, one predicting a right hand turn, and the other predicting the vehicle to drive straight through the intersection. The network assigns a percentage confidence value to each prediction that sums to one to indicate which mode is more likely to occur.

In addition to multimodal predictions, the paper incorporates contextual road information by passing rasterized top down images of the road and traffic scenario to the network. This gives the network the ability to observe where the road and other cars are, allowing it to rule out improbable trajectories. The downside of using CNN's, however, is the large amount of memory it requires during training. Loading images for each time epoch requires a significant amount of video memory, requiring the designer to use small batch sizes. Thus training networks using this method requires methods to combat over-fitting such as early stopping, which leads to long training times. On the other hand LSTMs can be trained very quickly to predict sequences with good generalization provided they have enough training data.

This thesis aims to give LSTMs another shot at this problem by incorporating the principles developed by Cui et al. into the LSTM design process. The network has been designed to output two trajectory predictions, one optimized to track and predict "passing" behavior and the other to track and predict "cut-in" behavior. The two predictions are assigned probabilities by the network that reflects the network's confidence in the predicted trajectories.

Training data for the network was created by simulating a bicycle model commanded to follow merging or passing waypoints using a Pure Pursuit controller. Longitudinal dynamics of the following vehicle are modeled using a second order transfer function that tracks a noisy reference. This controller was used to drive the vehicle to a set longitudinal distance behind the lead truck if it decided to cut into the platoon. The trucks were simulated as points traveling at a constant velocity and separation distance. The simulation parameters, including look ahead distance and longitudinal velocity, were randomly varied during training and testing to allow the network to train off of a wide range of random trajectories. The scenarios are simplified to consider to single vehicles neighboring the truck platoon.

The Network performance is benchmarked against several state-based predictors which use knowledge of the vehicles past and present states to predict future states [32]. These predictors include the Constant Velocity Predictor, the Constant Acceleration Predictor (CA), and the Constant Turn Radius Predictor (CTR).

1.2 Contributions

The academic contributions of this thesis are listed below.

- Provided a thorough background on Time Series Forecasting and how it relates to Machine Learning and Ground Vehicle Navigation.
- Designed a Deep Neural Network to predict the future pose of vehicles around a truck platoon.
- Compared the performance of the network to traditional state-based prediction methods.
- Designed and Evaluated a cut-in detection algorithm using network and state-based predictions.

- Evaluated the ability of a simulation trained neural network to predict behavior on experimentally collected data.

1.3 Thesis Outline

This thesis consists of 4 remaining chapters. Chapter 2 provides an introduction to Machine Learning and introduces methods used in Deep Learning. Chapter 3 discuss background in Time Series Forecasting. Chapter 4 discusses the Modeling and Simulation environment used to generate training and test data sets for this thesis, the design of the network architecture, training methodology, and results in the simulated environment. Chapter 5 discusses the process of collecting empirical data from the Auburn University truck platoon, and provides prediction results on that data. The final chapter contains a summary, conclusions, and future work.

2

Deep Learning Background

2.1 Introduction

This chapter will introduce principles in Deep Learning that are used in this thesis. Section 2.2 gives a topical presentation of general concepts in Machine Learning, describing Tasks, Experiences, and Performance measures, and presenting the Universal Approximation Theorem and Active Learning Pipeline. Sections 2.3, 2.4, and 2.5 present three commonly used types of Neural Networks: Multi-layer Perceptron Models or Feedforward Neural Networks, Covolutional Neural Networks, and Recurrent Neural Networks.

2.2 General Concepts in Machine Learning

In his 1997 text on Machine Learning, Tom Mitchell gives a definition of Machine Learning: "A computer is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [54]. In the case herein the task T is to predict the future pose of traffic agents around a truck platoon and classify their behavior as either "passing" or "cut-in". This is accomplished with the experience E of observing vehicle positions taken from radar measurements over time, and with performance methods P being Mean Squared Error Loss and Cross-Entropy Loss. The first objective of this chapter is to discuss the Machining Learning concepts of Task, Experience, and Performance further, then to hone in on Deep Learning, a subset of Machine Learning. Therein general equations used in Neural Networks will be developed, and Recurrent Neural Networks will be introduced.

2.2.1 Task

Machine Learning is beneficial in that it provides solutions to tasks that could otherwise be too time consuming or difficult to be traditionally programmed. A task is the way a machine learning algorithm processes an example, where an example is a collection of features that have been quantitatively measured by an event or object that we would like the machine learning algorithm to process. Features are typically represented as a vector

$$x \in \mathbb{R}^n \tag{2.1}$$

where each value in the vector x is a feature value. Commonly in Deep Learning, the type of Machine Learning used herein, features are pixels of images which are used in image processing. In this thesis, the features used in training are the time-ordered positions of traffic agents around the truck platoon.

Common tasks of Machine Learning include but are not limited to classification, classification with missing inputs, regression, transcription, machine translation, structured output, anomaly detection, synthesis and sampling, imputation of missing values, denoising, and density estimation or probability mass function estimation [29].

Probably the most common of these tasks to a beginning Machine Learning practitioner is classification. Classification involves training the algorithm to specify which category an input belongs to. An example of this is tasking a machine learning algorithm with classifying an image as either a dog or a cat.

Classification with missing inputs is commonly used in the medical field. Classifying objects with missing inputs complicates the task because the learning algorithm needs to learn more functions in order to make the proper classification. Usually probability density functions are fitted to measure the confidence of classifications. Such algorithms allow doctors to make more accurate diagnoses without needing to perform more invasive or expensive procedures [29].

Regression is the task of training the network to predict a numerical value given a sequence of preceding numerical values. An example would be to feed the algorithm a sequence of

values such as $[0, 1, 2, 3]$ and expect the algorithm to output the number $p = 4$ or another sequence of specified length such as $[4, 5, 6, \ldots, p_n]$. This task is also commonly known as time series forecasting and is used in stock market forecasting, weather forecasting, and product recommendation [12]. The network discussed in Section 4.3 to provide obstacle pose prediction solutions to the truck platoon is designed to perform this task. The task of Regression, or Time Series Forecasting, is further elaborated upon in Chapter 3.

Transcription is the task of training an algorithm to describe an image or speech recording with written text. Such tasks typically employ Recurrent Neural Networks, which will form the basis of the Learning Algorithm designed herein. Machine Translation performs the task of translating words from one language to another. A neural network architecture known as the Encoder-Decoder model was designed to better perform this task than traditional methods. Previously, one large network would be trained to both understand the first language and speak the second. The Encoder-Decoder architecture divides this task into two independent tasks: one to learn the first language and the other to speak the second language [3]. This method has been shown to reduce overall network size and training time. Given that transcription is a task closely related to Time Series Forecasting, it is likely that the Encoder-Decoder architecture will improve network performance for Time Series Forecasting as well. Thus the Encoder-Decoder architecture is included in the network design in Section 4.3. Encoder-Decoder models are elaborated upon in Section 3.6.3.

Structured Output is a broad category of tasks and involves any task in which the output is a vector with interrelationships between elements. Such tasks include mapping grammatical structure to a sentence, identifying roads from an aerial view camera, and describing an image with a sentence [45].

Anomaly Detection is commonly used in cyber security and tasks the algorithm with identifying events or objects that don't normally belong. Such algorithms perform tasks such as fraud detection and spam email filtering.

Synthesis and Sampling is the task of generating new samples of data similar to data trained upon. One example is generating new landscapes in a video game [52] or using a Generative Adversarial Network to generate new images [28].

Imputation of Missing Values is the task of predicting the values that are missing in a given input. An example of this task is filling in the missing hole in an image or filling in the blanks in a sentence.

Denoising is the task of modeling the conditional probability distribution $p(x|x)$ or more specifically predicting a filtered sequence from a noisy one. Filtering properties of Deep Neural Networks tend to be inherent and will be studied further herein [66].

Density estimation or Probability Mass Function Estimation is the task of capturing the structure of the data given to the algorithm and defining a probability mass function for discrete data or a probability density function for continuous data [29].

The tasks of this work incorporate both classification and regression. The first task of the network is to perform time series forecasting, of which a sequence of neighbor vehicle positions are used to predict future positions. The second task is known as time series classification, in which the network is tasked to predict which of its output trajectories will have a lower root mean squared error to the truth trajectory.

2.2.2 Performance Measure

The performance measure is a quantitative measure of performance of the learning algorithm. In Deep Neural Networks, this performance measure is defined as the loss, which is the value to be optimized in the training process of gradient descent. Performance measures must be carefully chosen for each task. In image classification a common performance measure is Cross-Entropy Loss [29]. In sequence prediction the commonly used performance measure is Mean Squared Error, otherwise known as L2 Loss, or Mean Absolute Error, otherwise known as L1 Loss. This work employs Cross-Entropy to quantify the performance of the classifier and Mean Squared Error loss to characterize the regression performance. These performance measures are further described in Section 4.3.5.

2.2.3 Experience

Machine Learning algorithms can be generally divided into several categories of experience methods including supervised learning, unsupervised learning, and reinforcement learning. These categories loosely organize Machine Learning algorithms by the way they're given and process data sets.

Supervised learning is the experience method used in this work, and involves comparing the network output to the truth output. This is one of the key processes in back-propagation, which is described in Section 2.3.2. In this case, the predicted trajectories output from the network are compared to the true trajectories, and the predicted confidence probabilities assigned to the predicted trajectories are calculated as a result of comparing the predicted trajectory predicted to have the lowest root mean squared error to the truth trajectory to the actual trajectory with the lowest root mean squared error to the truth.

Unsupervised learning is much more difficult to implement and requires the algorithm to learn without access to correct data points for comparison. Unsupervised learning is an area of growing research due to its resemblance to the way humans learn and its difficulty in implementation. Tasks in unsupervised learning include density estimation and denoising [29].

Reinforcement Learning is a method of learning in which the data set used is not fixed and employs a feedback loop between the learning algorithm and new experiences. A few examples of this include a network designed to play Atari through trial and error [55], and a robot that continually trains on new data to avoid obstacles [15]. Recently in April of 2019, OpenAI sucessfully trained a network to beat professional teams in the video game DOTA [57]. The network was trained by playing itself repetitively since June 2018.

Given that reinforcement learning has shown to be able to beat the best human players in strategic video games, there is reason to believe that reinforcement learning can be used extensively in the autonomous vehicle space. However, networks likely need to reach a certain level of safety before they practice driving on the road [76]. For this reason, it is likely that efforts like the one being made in this thesis will contribute significantly toward this goal in two ways. First, developing a high fidelity simulation close to reality can allow for significant

reinforcement learning before real world deployment, and second, algorithms trained and tested manually in the "Active Learning Pipeline" as described in Section 2.2.5 can provide ample ground from which reinforcement algorithms can begin training from through transfer learning. By this, in future work reinforcement learning be be used to give the platoon the ability to learn as it runs.

2.2.4 The Universal Approximation Theorem

The Universal Approximation Theorem given by Hornik et al. in 1989 [36] and by Cybenko in 1989 [22] states that any continuous Borel measureable function can be approximated with any specified non-zero amount of error by a Multilayer Perceptron Model with any "squashing" activation function such as Sigmoid given the network has enough layers. In other words, MLP models can be theoretically used to represent any function. However, successful training to achieve satisfactory representation is not guaranteed. First, the network might not be able to converge to the correct weights to approximate the function, second, the network might over-fit the function desired to be approximated. This points to the "No Free Lunch" theorem [78], in which there is no universally superior Machine Learning algorithm at learning all possible tasks. To quote from Goodfellow, Bengio, and Courville [29]:

> Fortunately, [this] holds only when we average over all possible data-generating distributions. If we make assumptions about the kinds of probability distributions we encounter in real-world applications, then we can design learning algorithms that perform well on these distributions. This means that the goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm. Instead, our goal is to understand what kinds of distributions are relevant to the "real world" that an AI agent experiences, and what kinds of machine learning algorithms perform well on data drawn from the kinds of data-generating distributions we care about.

In other words, this thesis aims to 1) define the desired tasks for the machine learning algorithm to perform, 2) choose and design an algorithm well suited to learn and perform the

desired task, and 3) design relevant experience distributions for the network to train off of. Hence, the efforts of the subsequent chapters below are set upon thoughtful design of each of these facets.

2.2.5 Active Learning Pipeline

> *"Start out dumb, become brilliant over time."*
>
> – Lex Fridman, *MIT Deep Learning State of the Art 2020* [27]

The Active Learning Pipeline [27], also dubbed the Data Engine by Telsa's AI director Dr. Andrej Karpathy [41], is the iterative process of network design used by companies like Waymo and Tesla to design their networks [27]. The process begins with designing a network to perform a task given the steps listed above in Section 2.2.4. Then the network performance is evaluated and failure modes are searched for. Once failure modes are found, the experience distribution and network design is revised to account for the problems. Finally the network is retrained and reevaluated, and so on.

This thesis aims to make one pass at the Active Learning Pipeline by 1) designing an experience distribution in Section 4.2, 2) designing and training the network to perform the task of predicting vehicle trajectories in Section 4.3, 3) evaluating the performance of the network in Section 4.4 and Section 5.5, and 4) suggesting avenues for future work in future passes.

2.3 Multi-Layer Perceptron (MLP)

Multi-layer Perceptron models, also referred to as Feed Forward Neural Networks or "Vanilla" Neural Networks, are the simplest kind of neural network. MLPs can be intuitively thought of as a series of linear regressions, with nonlinearities added in between each regression layer.

Linear Regression is a basic machine learning algorithm in which the parameters of a line or polynomial are used to perform the task of classification. The process involves data which is input into a series of linear regressions that then calculate a classification. This is given in vector form by

$$f(x, W, b) = w^T x + b \tag{2.2}$$

Where x is the input array, w is a vector of weights, and b is a scalar commonly referred to as the bias.

Multi-layer Perceptron Models contain at least three layers of linear regressions, usually with activation functions at each node except the ones on the first layer. The architecture of neural networks is defined as the width (number of nodes per layer) and the depth (total number of layers). Stacking many layers on top of one another yields a "Deep Neural Network", of which the term "Deep Learning" is derived.

A diagram representation of a typical MLP architecture is shown in Figure 2.1.

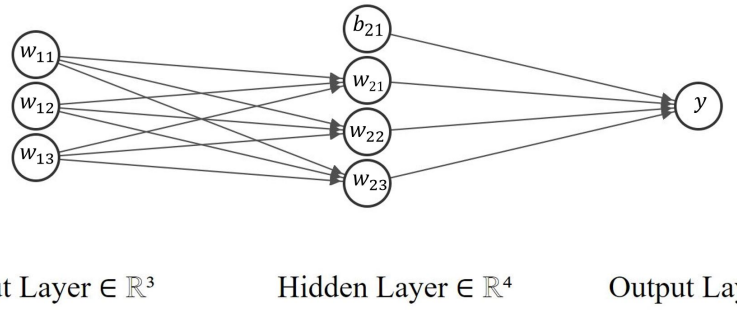


Figure 2.1: Multilayer Perceptron Network with one hidden layer of width 3

This network has three layers total one hidden layer with three nodes. Algebraically this network can be rewritten as:

$$h = g(W^T x + b) \tag{2.3}$$

Where W is the vector of weights at each layer, b is the bias scalar at each layer, g is the nonlinear activation function applied to the outputs of each layer, and x is the state passed through each layer. Through a process known as back propagation, the network may be trained to learn the weight vectors W_j and biases b_j to approximate a function, as given by the Universal Approximation Theorem.

2.3.1 Activation Functions

The power of neural networks rests largely in the design and use of activation functions. Designing a neural network without activation functions will yield a linear function like equation 2.2. Applying activation functions within the hidden layers of the network will provide the

network with nonlinearities, and it is these nonlinearities that allow neural networks to learn complex problems well. Commonly used activation functions include Sigmoid, tanh, ReLU, and ReLU variants such as leaky ReLU and GELU. These functions can be visualized in figure 2.2 and will be discussed in brief below.

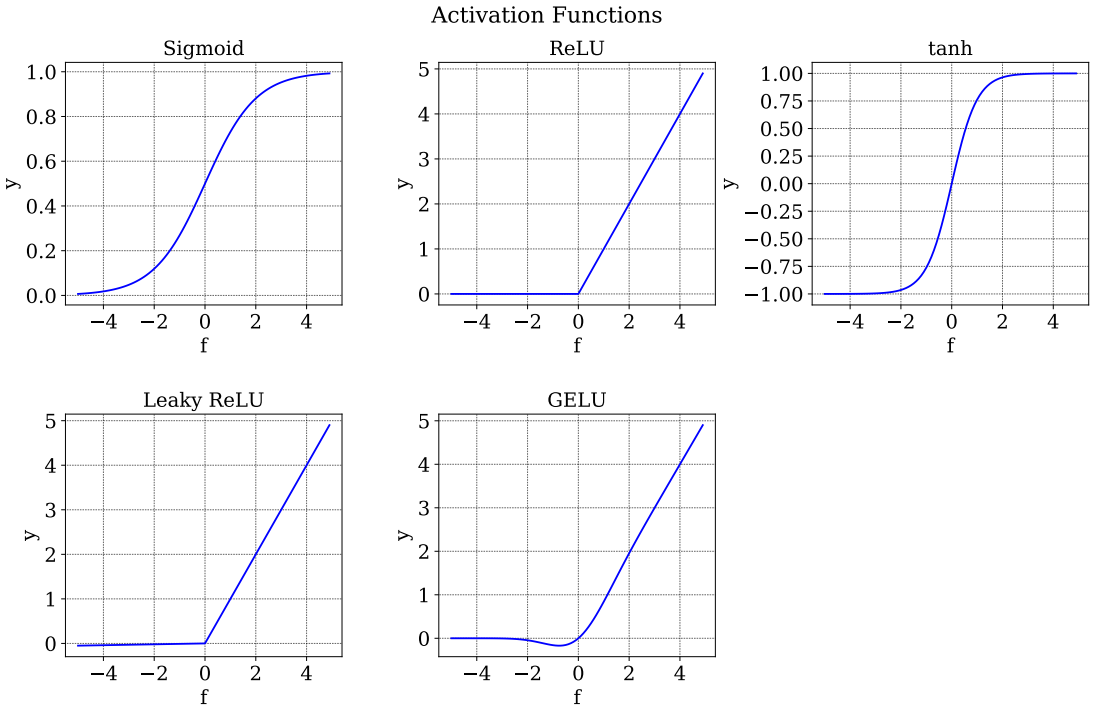


Figure 2.2: Commonly used Activation Functions

2.3.1.1 Rectified Linear Unit (ReLU)

The rectified linear unit or ReLU is one of the most common activation functions. It is given by

$$g(f) = max(0, f). \tag{2.4}$$

The benefits of ReLU are that it is computationally inexpensive compared to other activation functions and that it has been found to increase the rate of loss convergence with stochastic gradient descent [48]. The downside of ReLU is that it can cause nodes in the network to get permanently stuck outputting zeros or "die" [42].

2.3.1.2 Variants of ReLU

Several variants of ReLU have been invented with the intention of fixing the dying node problem. One of these is the Leaky ReLU, which instead of setting negative inputs to zero, it multiplies them with a linear function with a small negative slope of around 0.01. Success of the Leaky ReLU function is mixed [42].

Another variant is the Gaussian Error Linear Unit (GELU) function [33], which appears to outperform ReLU in some studies. GELU also addresses the problem of not allowing negative values to pass through the network. The GELU function is given by

$$g = 0.5f\left(1 + \frac{2}{\sqrt{\pi}}\int_0^{\frac{x}{\sqrt{2}}} e^{-t^-2}dt\right) \tag{2.5}$$

which can be approximated with

$$g = .5f(1 + tanh(0.797885f + 0.035677f^3)) \tag{2.6}$$

2.3.1.3 Sigmoid Activation Function

The sigmoid activation function is given by

$$\sigma(f) = 1/(1 + e^{-f}) \tag{2.7}$$

which ”squashes” signals passing through the function to values between 0 and 1. The sigmoid function is meant to model the firing of a neuron, with an output signal of 0 indicating the neuron not firing at all up to a maximum output signal of 1 indicating full firing.

The sigmoid activation function suffers two main drawbacks. First it has a tendency to saturate and kill gradients. If large positive or negative inputs enter the sigmoid activation they will saturate very close to either 0 or 1. As such, the gradients calculated during back propagation will be very close to zero. This has an effect of significantly attenuating the signal flowing through that node in the network. Secondly the sigmoid function is not zero centered. This causes the output of the activation function to always be positive, which means the weights

of the network have no way to change sign. Due to these setbacks the sigmoid activation is rarely used in modern neural network design, although they can still be found in the architecture of more complex networks like the Long-Short Term Memory network [42].

2.3.1.4 tanh Activation Function

The tanh activation function "squashes" signals passing through the function to a range between -1 and 1.

$$tanh(f) = 2\sigma(2f) - 1 \tag{2.8}$$

Like the sigmoid function it suffers from issues with saturating, however its benefit is that it is zero mean. Tanh is used more commonly than sigmoid today [43].

2.3.2 Back Propagation

Back Propagation is the fundamental supervised learning method of which neural networks are trained. This method has existed for decades [64], and since its inception, many methods have been published with newer and faster ways to perform the process. Back propagation works by first computing the calculating the end values given the inputs to the network, which is the forward propagation of the network. Then the network outputs are run backwards through the network, recursively applying the chain rule at each layer to compute the gradients at each step. These gradients are then used to update the weights and bias of each layer in a manner dependent upon the type of optimizer chosen, with the ultimate aim of minimizing the loss of the output to a global minima. Two popular optimizers, among others, will be discussed briefly below.

2.3.2.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent performs the operation of gradient descent over minibatches of data. The optimizer draws a sample or "minibatch" of data from the training data set, and takes the average gradient across the minibatch of m examples. The gradient estimate is calculated, multiplied by the learning rate, and subtracted off the original weights to create the new weights. The use of minibatches is beneficial in that it allows for faster computation time

with larger datasets, since small amounts can be passed through the computer at a time. Using Stochastic Gradient Descent introduces two hyperparameters: learning rate and batch size. These parameters are tuned to improve learning performance. Adding a momentum term to the SGD algorithm can speed up training by reducing the amount that SGD tends to zig-zag within a bowl to get to the bottom. Momentum aims to solve poor conditioning of the Hessian matrix and variance in the stochastic gradient [29]. There are several different methods to include momentum in the optimizer model, with the most successful being the Adam optimizer.

2.3.2.2 Adam Optimizer

The name for the Adam Optimizer is derived from "Adaptive Moments". It is an adaptive learning rate optimizer that utilizes momentum which helps carry the minimization process across local minima to make convergence to the global minimum more likely [29]. Adam is currently held as one of the most efficient optimizers and it is therefore the optimizer used for network training in this thesis. More information on the inner workings of Adam can be found in [46].

2.4 Convolutional Neural Networks

Convolutional Neural Networks, originally composed in the late eighties by LeCun [50], today make up the backbone of most image recognition networks. Although not used in this thesis, these are worth being briefly touched upon due to their significant success in pushing the state of the art in deep learning applications, particularly to applications in image classification.

Convolutional Neural Networks have an inherent structure very similar to that of vanilla neural networks or MLPs with the key difference being the replacement of conventional matrix multiplication between layers with the convolutional operator. The convolutional operator typically assumes the input to be shaped as an image with a length, width, and depth. In image processing, the length and width make up the pixels of an image, while the depth is usually comprised of three values representing the red, green, and blue color components. The convolutional operator uses at least one "filter" which is a window that is slid across the image, performing convolutions at each step. Filters can be designed to look for different features in

the image such as color and edges. The depth of the next layer in the convolutional neural network is equal to the number of filters parsing through the preceding layer. A typical summing filter of size 2x2 will be a 2x2 matrix of weights that are multiplied to the values in the image that the filter is panning over and then summed, essentially performing a dot product of the weights and the values in the image. The new values outputted from the filter make up that next block in the next layer image, which is completed as the filter pans across the preceding image. The stride is the number of pixels the filter pans over at a time. This is a tune-able hyperparamter. Padding is a processed used to increase size of images in order to make geometry flow better. Padding is usually set to zeros [42].

The use of CNNs may find itself in the truck platoon at some point or another. The obvious use for them is for applications regarding image data received from cameras. Given images from a camera on the follower truck, CNNs can be used to place bounding boxes on the lead truck trailer, and can bound and classify other traffic agents such as cars, trucks, and road signs. These can be used to aid in visual odometry algorithms as well, such as determining the location of lane lines or the relative location of other traffic agents. Another application of CNNs to this work could be using raster images of surrounding traffic and road boundaries such as was done in [21] in order to produce results similar to the results of this thesis, but perhaps more robust to interactions between traffic agents.

2.5 Recurrent Neural Networks (RNN)

Closing the loop on the networks mentioned above allows a network to relay information to itself over training sequences and yields another class of neural networks known as Recurrent Neural Networks (RNNs). Recurrent Neural Networks specialize in processing sequential information and due to their recurrent nature are able to learn more complex and longer term sequential patterns. Recurrent Neural Networks can come in several different orientations: many-to-many, one-to-many, and many-to-one. Many-to-many takes a sequence and outputs a sequence. Similarly one-to-many takes one input (such as a photo) and outputs a sequence (a description of the photo). Many-to-one takes in a sequence and outputs a single output [44]. These orientations are elaborated upon in Section 3.6.2.

A computational graph of a vanilla RNN is shown below.

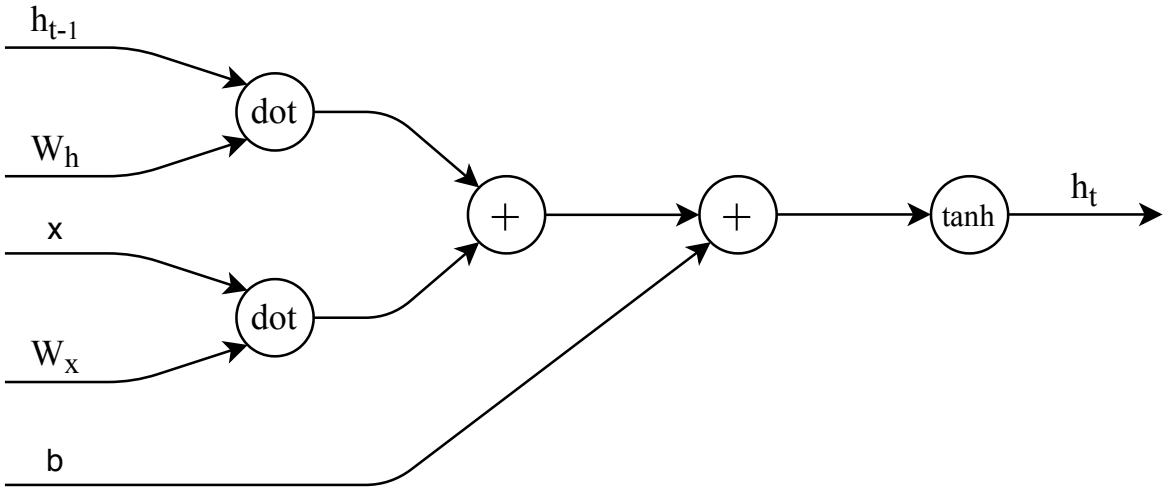


Figure 2.3: Diagram of RNN Forward Propagation

The variable x is the vector input to the network, and the variable h is the hidden state vector that is carried through from one RNN cell to the next. This variable is passed through to the next RNN layer to share information about the past. As shown in the diagram, both the input and hidden state are multiplied by their own respective weight vectors. Forward propagation through a vanilla RNN is shown algebraically in equation 2.9.

$$h_t^l = tanh\left(W^l \begin{bmatrix} h_t^{l-1} \\ h_{t-1}^l \end{bmatrix}\right) \tag{2.9}$$

where t denotes the time step and l denotes each RNN cell.

There are two well known fundamental issues with vanilla RNNs. First is the issue of overload of information. Flooded with data from the hidden state, the RNN has no way to decide which information is needed and what is not, leading to loss of generalization. Second, discovered originally by Bengio in 1994 [5] and Hochreiter back in 1991 [34], is the issue of vanishing (or exploding) gradients during training. The vanishing gradient problem is caused when small gradients less than zero pass from the end of a deep neural network back to the front during back propagation. As the chain rule is applied at each layer, the gradients can shrink exponentially. Similarly, when large gradients are passed back through back propagation in deep networks, continual multiplications through the chain rule in back propagation can lead to

exploding gradients. One solution to the exploding gradients of RNNs is by gradient clipping [58], however, to solve the problem of vanishing gradients, changing the architecture to the Long-Short Term Memory Recurrent Neural Network is usually the route taken.

2.5.1 Long-Short Term Memory (LSTM)

The Long-Short Term Memory Recurrent Neural Network (LSTM) solves many of the problems associated with the vanilla Recurrent Neural Network [35]. The LSTM solves the vanishing gradient issue by carrying over the states in an additive manner between each layer through the cell state c, and better learns long term dependencies through the use of a forget gate f, which decides which information is useful for remembering. This allows the network to only remember information that is important, leading to better use of context.

A computational graph of the LSTM Forward Propagation is shown below.

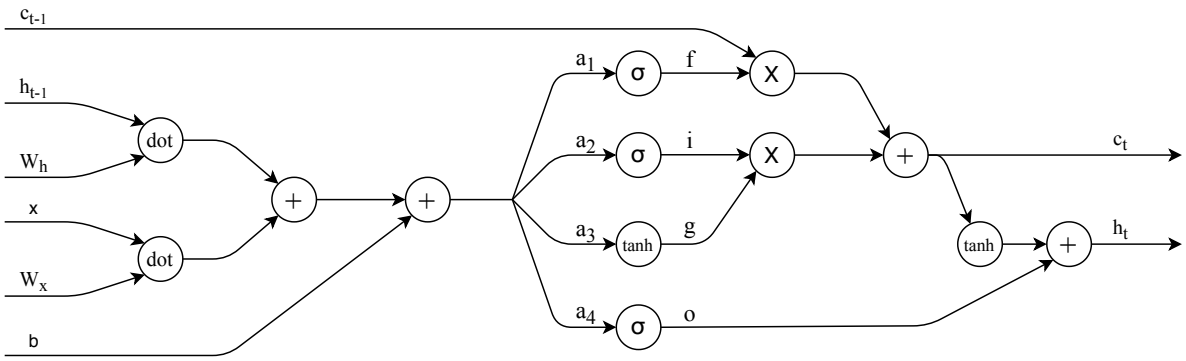


Figure 2.4: Diagram of LSTM Forward Propagation

LSTM Forward Propagation is given algebraically as:

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \tag{2.10}$$

$$c_t = f \circ c_{t-1} + i \circ g \tag{2.11}$$

$$h_t = o \circ tanh(c_t) \tag{2.12}$$

Where W_h and W_x are the weights multiplied to the training data x and the hidden states h respectively. The vector c is the cell state. The cell state c and hidden state h both serve to conserve information from one step in the network to the next, allowing the network to learn time series dependencies. The vectors a_1, a_2, a_3, a_4 serve as inputs to the forget gate, input gate, "gate" gate, and output gates respectively. These gates collectively determine how information is retained (or forgotten) as the network is trained. Back propagation equations for RNNs and LSTMs are given in the Appendix. A detailed history of the LSTM can be found in [31].

2.5.2 Gated Recurrent Unit (GRU)

Gated Recurrent Units were devised by Cho et al. in 2014 as another architecture of Recurrent Neural Network [18]. According to the findings of Karpathy et al. both LSTM and GRU architectures achieve comparable performance while both outperforming traditional RNNs when trained for character-level language modeling. The Gated Recurrent Unit is simpler than the LSTM with only two gates instead of four. Algebraically GRUs take the form

$$\begin{bmatrix} r \\ z \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \end{bmatrix} W_r^l \begin{bmatrix} h_t^l \\ h_{t-1}^l \end{bmatrix} \tag{2.13}$$

$$\tilde{h}_t^l = tanh(W_x^l h_t^{l-1} + W_g^l (r \circ h_{t-1}^l)) \tag{2.14}$$

$$h_t^l = (1 - z) \circ h_{t-1}^l + z \circ \tilde{h}_t^l \tag{2.15}$$

Where W_r^l is a matrix of shape [2n x 2n], and W_g^l and W_x^l are of shape [n x n].

Despite the successes of GRUs in recent studies, the LSTM network is still by far the most popular Recurrent Neural Network used in application. For these reasons, the LSTM was chosen for this work. Future studies may incorporate comparisons between LSTMs and GRUs for this application if desired.

2.5.3 Convolutional LSTM (ConvLSTM)

When dealing with large arrays of input data or time-distributed image input data, one may choose to use Convolutional LSTMs, which were originally developed by researchers from the Hong Kong University of Science and Technology [79]. They proposed using a convolutional LSTM (ConvLSTM) to predict future rainfall magnitude given radar images. Their findings show that ConvLSTM layers are able to fuse the strengths of Convolutional Neural Networks with LSTMs into a layer that can capture both time dependencies and spatial dependencies.

The Convolutional LSTM works by performing convolutions within the state-to-state and state-to-input transitions of the LSTM. This is not to be confused with stacking a CNN layer in front of an LSTM layer. Instead, the CNN is fused inside of the LSTM. The downside of the ConvLSTM compared to the LSTM is longer training time. Therefore, one may try using an LSTM for their problem first and see if the results are good enough before attempting to train a ConvLSTM. For reference, the ConvLSTM forward propagation equations are given below:

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} W * \begin{bmatrix} H_{t-1} \\ X_t \end{bmatrix} \tag{2.16}$$

$$C_t = f \circ C_{t-1} + i \circ g \tag{2.17}$$

$$H_t = o \circ tanh(C_t) \tag{2.18}$$

where the formulation is the same as forward propagation for LSTMs, but the convolutional operator $*$ is applied between the weights W and the hidden states H and inputs X. Additionally, the cell state C, hidden state H, and input X become 3 dimensional tensors with the width, length, and depth that an image would have.

2.6 Conclusion

This concludes the discussion on machine learning concepts and the fundamentals of Deep Neural Networks. To wrap up, Deep Neural Networks, per the Universal Approximation Theorem, are theoretically able to approximate any nonlinear function or task. Some network configurations are better at certain tasks than others. In this thesis, the task of the designed network is to approximate the future positions of vehicles surrounding a truck platoon on a highway, which falls within the general task of Time Series Forecasting. Recurrent Neural Networks, particularly Long-Short Term Memory networks have been shown in literature to perform particularly well at the task of time series forecasting, hence LSTMs are the chosen algorithm in this thesis. From this we gather some tools with which to approach the overall task of time series forecasting from a machine learning perspective, which will be explored in the next chapter.

3

Time Series Forecasting

3.1 Introduction

Typical data used in Deep Learning applications, such as images used to train a neural network to recognize traffic agents, lacks a temporal dependency. Once the observations contain a time dimension, they become a time series. To quote George Box, "A time series is a sequence of observations taken sequentially in time" [7]. When working with time series data, one may choose to understand it, or one may use it to make predictions of the future. The former is known as Time Series Analysis and aims at modeling the mechanism that give rise to an observed series. The latter is known as Time Series Forecasting. While Time Series Analysis can aid in Time Series Forecasting, it is not necessarily required to perform the task.

This chapter aims to survey various Time Series Forecasting methods used to forecast vehicle behavior, and categorizes these methods into "State-Based" and Stochastic prediction methods, "Goal-Based" prediction methods, and Machine Learning methods. Particular attention will be given to methods used later in this thesis including the Constant Velocity, Constant Acceleration, and Constant Turn Radius predictors, and the Long-Short Term Memory Network.

3.2 State-Based Predictors

In a report released by the Naval Surface Warefare Center in March 1994, several methods are detailed to generate the predicted future positions of incoming torpedo threats of US Navy ships. In their work, future threat position prediction methods were split into two main

categories: State-Based predictors and Goal-Based predictors. State-based predictions involve using the current pose of the vehicle and predicting out future pose via a state propagation method. Such methods include Constant Velocity (CV), Constant Acceleration (CA), Constant Turning Rate (CTR), Exponentially Deceasing Turning Rate (EDTR), and Helical predictors [32]. Among these used in this thesis are the CA, CV, and CTR predictors. These predictors will be described in the sections below.

3.2.1 Constant Velocity

The Constant Velocity Predictor operates by differentiating the last two trajectory observations of the tracked vehicle with respect to time such that

$$V_{est} = \frac{p_k - p_{k-1}}{dt}. \tag{3.1}$$

Where p is the position vector of the tracked vehicle, dt is the time step, and V_{est} is the estimated velocity vector. Then future positions are estimated such that

$$p_{k+1} = p_k + V_{est}dt. \tag{3.2}$$

3.2.2 Constant Acceleration

Likewise the Constant Acceleration Model first estimates the vehicle's acceleration at the current time instant such that

$$a_{est} = \frac{p_k - p_{k-1}}{dt^2} - \frac{p_{k-1} - p_{k-2}}{dt^2}. \tag{3.3}$$

Then future accelerations are estimated such that

$$p_{k+1} = p_k + V_{est}dt + \frac{1}{2}a_{est}dt^2. \tag{3.4}$$

3.2.3 Constant Turn Radius

The Constant Turn Radius Predictor holds the velocity magnitude constant over time while allowing for lateral accelerations. The Constant Turn Radius predictions are calculated to be

$$p_{k+1} = p_k + \alpha V_{est} + \beta a_{est}. \tag{3.5}$$

Where α and β are defined as

$$\alpha = \frac{\sin \Omega_0 t}{\Omega_0}, \tag{3.6}$$

and

$$\beta = \frac{1 - \cos \Omega_0 t}{{\Omega_0}^2}, \tag{3.7}$$

and Ω is given to be

$$\Omega = \frac{norm(a_{est})}{norm(V_{est})}. \tag{3.8}$$

3.3 Stochastic Estimators for Tracking and Prediction

Kalman Filtering [40] and other stochastic track filters are commonly used to track obstacles for collision avoidance using dynamic models derived from the state-based models mentioned above. The key addition to the state-based models in the filter dynamics is the modelling of noise in the dynamic model. The models are derived in two main methods, differing in the treatment of noise: First is by driving the continuous dynamic model with continuous-time white noise acceleration (for CV) or jerk (for CA) and discretizing for a given sampling period. This is referred to as discretized continuous white noise and is given in Section 3.3.1. Second is by defining the process noise in discrete time as a piecewise constant white noise sequence [4], relying off the assumption that the process noise is constant and independent between sampling periods. This is referred to as discrete white noise and is given in Section 3.3.2 [63]. The track filter using these models can be modified to be a predictor by skipping the measurement update and recursively running the process update out in time.

These models are widely used in applications related to this thesis. Cosgun et al. use a Nearly Constant Acceleration (NCA) model in a Kalman Filter to provide obstacle detection for their autonomous vehicle implementation [20]. Engineers at Mercedes-Benz use an Extended Kalman Filter with a Nearly Constant Velocity (NCV) model to provide tracking solutions of obstacle vehicles [81].

Similar to these works, the Nearly Constant Velocity model using discrete white noise acceleration is used for the dynamic model of a Kalman Filter to track neighboring vehicles in Chapter 5. The testing in Chapter 4 of this thesis focuses on ideal observations of neighboring vehicles, of which radar measurements perfectly capture vehicle position. In this case, the stochastic state-based models presented below will conform to the ideal models listed above. For example, the Nearly Constant Velocity model will conform to the Constant Velocity model in a non-noisy environment. In future work, once a sizeable data set is empirically collected from the Auburn truck platoon, the stochastic models below can replace their ideal versions in analysis.

The stochastic models will be presented given the two methods of modeling noise in the subsections below.

3.3.1 Discretized Continuous-Time State-Based Models

In this method, the discrete kinematic motion models are derived by first defining the continuous models driven by continuous time white noise and then discretizing. The models resulting from this method are better suited to model systems with variable sampling intervals [4].

3.3.1.1 Nearly Constant Velocity (NCV)

An object in motion with just slight changes in velocity over time can be modeled with a continuous time zero-mean white noise $v(t)$ acceleration such that

$$\dot{x}(t) = Ax(t) + Bv(t) \tag{3.9}$$

where

$$v(t) = \ddot{r}(t) \tag{3.10}$$

and where

$$E[v(t)] = 0 \tag{3.11}$$

and

$$E[v(t)v(t)] = \tilde{q}(t)\delta(t-\tau) \tag{3.12}$$

Equation 3.9 is expanded such that

$$\underbrace{\begin{bmatrix} \dot{r}(t) \\ \ddot{r}(t) \end{bmatrix}}_{\dot{x}(t)} = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} r(t) \\ \dot{r}(t) \end{bmatrix}}_{x(t)} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} v(t) \tag{3.13}$$

Then discretizing gives

$$\underbrace{\begin{bmatrix} r_{k+1} \\ \dot{r}_{k+1} \end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} r_k \\ \dot{r}_k \end{bmatrix}}_{x_k} + v_k \tag{3.14}$$

where dt is the time step, r_k is the position, $\dot{r}_k$ is velocity, and v_k is related to $v(t)$ such that

$$v_k = \int_0^{dt} e^{A(dt-\tau)} B v(kdt+\tau) d\tau \tag{3.15}$$

The covariance is then given as

$$Q_k = E[v(k)v(k)'] = \begin{bmatrix} \frac{1}{3}dt^3 & \frac{1}{2}dt^2 \\ \frac{1}{2}dt^2 & dt \end{bmatrix} \tag{3.16}$$

where $\tilde{q}$ is the power spectral density for a time-invariant system [63]. Using a small q value gives the Nearly Constant Velocity model, as changes in velocity of the track are small compared to the modelled velocity [4][63].

3.3.1.2 Nearly Constant Acceleration (NCA)

If the object being tracked is frequently maneuvering, the white noise can be modeled as zero-mean white noise jerk such that

$$v(t) = \dddot{r}(t) \tag{3.17}$$

Then the dynamics are given as

$$\underbrace{\begin{bmatrix} \dot{r}(t) \\ \ddot{r}(t) \\ \dddot{r}(t) \end{bmatrix}}_{\dot{x}(t)} = \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} r(t) \\ \dot{r}(t) \\ \ddot{r}(t) \end{bmatrix}}_{x(t)} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{B} v(t) \tag{3.18}$$

This is then discretized to give

$$\underbrace{\begin{bmatrix} r_{k+1} \\ \dot{r}_{k+1} \\ \ddot{r}_{k+1} \end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix} 1 & dt & \frac{1}{2}dt^2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} r_k \\ \dot{r}_k \\ \ddot{r}_k \end{bmatrix}}_{x_k} + \upsilon_k \tag{3.19}$$

where dt is the time step. r_k, $\dot{r}_k$, and $\ddot{r}_k$ are position, velocity, and acceleration. υ_k is white, zero mean process Wiener process acceleration error. The covariance is given as

$$Q_k = E[\upsilon(k)\upsilon(k)^T] = \begin{bmatrix} \frac{1}{20}dt_k^5 & \frac{1}{8}dt_k^4 & \frac{1}{6}dt_k^3 \\ \frac{1}{8}dt_k^4 & \frac{1}{3}dt_k^3 & \frac{1}{2}dt^2 \\ \frac{1}{6}dt_k^3 & \frac{1}{2}dt^2 & dt \end{bmatrix} \tilde{q} \tag{3.20}$$

Where $\tilde{q}$ is the power spectral density of $v(t)$. Setting a small $\tilde{q}$ gives low jerk relative to acceleration levels, leading to the Nearly Constant Acceleration Model [4].

This model works well to track highly maneuvering targets, so long as the measurement noise is relatively low and there is a high enough sampling rate to capture each maneuver well.

If there are only 2 or 3 measurements taking during each maneuver, acceleration of the target cannot be estimated well and the Nearly Constant Velocity model should be used instead [63].

3.3.2 Direct Discrete-Time Kinematic Models

The second method of defining the kinematic models used in tracking filters is by defining them directly in discrete time. This method is more commonly used [4], and is the method used in defining the NCV model used in the tracking filter currently on board the truck platoon as described in Section 5.3.3.

3.3.2.1 Nearly Constant Velocity (NCV)

The Nearly Constant Velocity model is

$$\underbrace{\begin{bmatrix} r_{k+1} \\ \dot{r}_{k+1} \end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} r_k \\ \dot{r}_k \end{bmatrix}}_{x_k} + \underbrace{\begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix}}_{B_k} v_k \tag{3.21}$$

where

$$v_k \sim N(0, \sigma_{vk}) \tag{3.22}$$

where, again, dt is the time step, r and $\dot{r}$ are position and velocity, and acceleration, and v_k is discrete Wiener process noise that is assumed to be constant in between time steps. The process noise covariance is related to the system such that

$$Q = E[Bv_k v_k B^T] = B_k \sigma_{vk}^2 B_k^T = \sigma_{vk}^2 \begin{bmatrix} \frac{1}{4}dt_k^4 & \frac{1}{2}dt_k^3 \\ \frac{1}{2}dt_k^3 & dt^2 \end{bmatrix}. \tag{3.23}$$

Where σ_{vk} is a design parameter. Setting a small σ_{vk} gives the NCV model.

3.3.2.2 Nearly Constant Acceleration (NCA)

The NCA model as derived directly within discrete time is given as

$$\underbrace{\begin{bmatrix} r_{k+1} \\ \dot{r}_{k+1} \\ \ddot{r}_{k+1} \end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix} 1 & dt & \frac{1}{2}dt^2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} r_k \\ \dot{r}_k \\ \ddot{r}_k \end{bmatrix}}_{x_k} + \frac{1}{2}dt^2 dt1 \tag{3.24}$$

and the covariance matrix is given as

$$Q = B_k \sigma_{vk} B_k^T = \sigma_{vk}^2 \begin{bmatrix} \frac{1}{4}dt_k^4 & \frac{1}{2}dt_k^3 & \frac{1}{2}dt_k^2 \\ \frac{1}{2}dt_k^2 & dt_k^2 & dt \\ \frac{1}{2}dt_k^2 & dt & 1 \end{bmatrix} \tag{3.25}$$

Again, setting a small σ_v gives low jerk relative to acclerations and leads to the Nearly Constant Acceleration model.

3.3.2.3 Nearly Constant Speed (NCS)

The Nearly Constant Speed (NCS) model derived directly in discrete time can be used to characterize movement of targets who maintain constant speed as they maneuver. This model is analogous to the CTR predictor described in Section 3.2.3. From [63], the NCS model is given as

$$A_k = \begin{bmatrix} 1 & \frac{sin(\Omega_k dt_k)}{\Omega_k} & \frac{1-cos(\Omega_k dt_k)}{\Omega_k^2} \\ 0 & cos(\Omega_k dt_k) & \frac{sin(\Omega_k dt_k)}{\Omega_k} \\ 0 & -\Omega_k sin(\Omega_k dt_k) & cos(\Omega_k dt_k) \end{bmatrix} \tag{3.26}$$

Where

$$\Omega_k = \frac{\|\ddot{r}_k\|}{\|\dot{r}_k\|}. \tag{3.27}$$

3.3.3 Kalman Filter

The Kalman Filter [40] is a recursive filter that utilizes a dynamic model of a system to predict the system's states through a time update, which it then corrects with measurements during a measurement update. The Kalman Filter performs this process assuming linear dynamics and stochastic evolution of the model. Because the filter is linear, the state estimates are also stochastic. Thus the state estimation error can be characterized by a mean and covariance. The steps of the Kalman Filter are given as

$$\text{Time Update} \begin{cases} \hat{x}_{k+1}^{-} = A_k \hat{x}_k^{+} \\ P_{k+1}^{-} = A_k P_k^{+} A_k^T + B_k Q_k B_k^T \\ P_k^{+} = [P_k^{-^{-1}} + C^T R_k^{-} C]^{-1} \end{cases} \tag{3.28}$$

$$\text{Measurement Update} \begin{cases} L_k = P_k^{+} C^T R_k^{-1} \\ \hat{x}_k^{+} = \hat{x}_k^{-} + L_k (Y_k - C\hat{x}_k^{-}) \end{cases}$$

Where A_k and B_k are given by the CV or CA model, L_k is the Kalman Gain, Y_k is the measurement update, P_k is the state error covariance, and R_k is the measurement covariance matrix. More information on the Kalman Filter can be found in [60] [63] [70].

3.4 Goal-Based Predictors

Goal-based predictors propagate the future states of the tracked vehicle forward in time by assuming knowledge of 1) the vehicle's dynamics, 2) the vehicle's goal point, and 3) the vehicle's guidance and control law [32]. The accuracy of the prediction heavily relies on the accuracy on the assumptions made, therefore, a potential area of research would be to find good vehicle models, goal points, and guidance and control laws for cut-in prediction. This, however, is beyond the scope of this work, as the goal of this thesis is to train a network to learn these assumptions.

3.5 Machine Learning Predictors

Many Machine Learning predictors have been applied to time series forecasting. A brief summary of prior work for several predictors will be given below. This list is not comprehensive. There are likely other methods other than these that currently exist or are yet to be invented that may perform well at time series forecasting. The algorithms chosen to be listed below have been have been picked due to their application in literature to problems related to traffic behavior prediction.

3.5.1 Hidden Markov Model

In [71], Hidden Markov Models (HMMs) were used to predict driver behavior at intersections. The study found that HMMs could use average velocities of vehicles approaching an intersection to predict whether the vehicle would turn right, left, or stay straight with an accuracy of 90 percent with mean prediction times of 7 seconds before the car reached the intersection.

3.5.2 Bayesian Networks

Schreier et al. in [65] used Bayesian Networks to create maneuver based probabilistic models to predict vehicle motion forward in time. The study incorporated random scenarios and driver decisions, including "irrational" driver decisions, using Monte Carlo simulations. The study divides predictions into different classes of maneuvers such as "Follow Road", "Follow Vehicle", "Lane Change", and "Target Brake". Each maneuver relies on a model for predictions, including the Constant Acceleration and Constant Turn models.

3.5.3 Gaussian Processes

Wang et al. in [75] use Gaussian process dynamical models to learn nonlinear models of human pose from motion capture data. The paper demonstrates the ability of the Gaussian Processes to effectively learn nonlinear behavior, even with small sample sizes.

3.5.4 Inverse Reinforcement Learning

Reinforcement learning operates by training an agent to perform an action based off some loss function or reward policy. Inverse Reinforcement Learning, also known as Inverse Optimal Control, attempts to calculate the loss function used by the observed agent via expert observations of the actions of that agent. Work by Andrew Ng and Stuart Russell in 2000 explored using Inverse Reinforcement Learning to predict Monte Carlo simulated trajectories [56], and work by Kitani et al. explored Inverse Optimal Control to predict future motion of pedestrians [47].

3.5.5 Convolutional Neural Networks

Engineers from Uber used Convolutional Neural Networks (CNNs) to predict the future positions of vehicles around intersections [21]. The algorithm used top-down rasterized maps of the road, lanes, vehicles, and vehicle orientations as input to a CNN. The time series output from the CNN was concatenated onto a vector of the tracked vehicle states and fed through an MLP network. The network outputted multiple trajectories with associated probabilities. This paper inspired the multi-modal trajectory prediction method used in this thesis.

3.5.6 Mixture Density Network

Mixture Density Networks (MDN) [6] were used by Shah and Romijdners in conjunction with LSTMs to predict the future position and success rate of NBA three point basketball trajectories [66]. The Mixture Density Network and LSTM combination allowed for a network output of Gaussian means and covariances for future basketball positions. The study found that the combination could predict success rate of three point shots with an accuracy of 72 percent at 8 feet from the basket. The prediction accuracy increased as balls moved closer to the basket, maxing out at 94 percent accuracy at 2 feet from the basket.

3.6 LSTM for Time Series Forecasting

LSTMs have been widely studied and applied to many Time Series Forecasting problems including predicting basketball trajectories [66], handwriting [30], image captioning and Shakespear play writing [44], and increasing reading efficiency by learning to read from left to right [2]. According to Jason Brownlee of Machine Learning Mastery, LSTMs provide state of the art results on difficult sequence problems [14][31]. Contrary to ordinary RNNS, LSTM networks have the advantage of being able to remember context over large amounts of data due to the workings of the forget gate as described in the previous chapter. For example, in the task of sentence completion, giving a network the sentence fragment "I live in France so I speak" and expecting the output to be "French", requires knowledge of the context attached to living in France. LSTMs have been shown to excel at learning these problems. This thesis set outs to show their viability in forecasting traffic behavior as well. But first, it is worth detailing a few more concepts surrounding the practical implementation of LSTMs in Time Series Forecasting.

3.6.1 Data Augmentation

> *"The success of all Machine Learning algorithms depends on how you present the data."*
>
> – Mohammad Pezeshki, *Mila* [59]

Training a network on raw data is typically a bad idea, as networks have trouble training off of data that has large variations in magnitude, non-normal distributions, or is non-stationary. To help with training, the raw trajectory data must be prepared before being pulled into the network. The most popular transforms to time series data include coordinate transform, power transform, differencing, standardization, and normalization, usually performed in that respective order to the data. Once the network is trained, predictions are detransformed in reverse order to arrive to the forecasted solution. In addition to having properties that enhance training, data augmentation methods can also be used to increase the robustness of the learned solution. This is further described in Section 3.6.1.6.

3.6.1.1 Stationarity

One of the goals of data augmentation is to ensure the time series is stationary. A time series is considered stationary if its statistical parameters (mean and variance) are not changing over time. In other words, a stationary time series does not have a trend or seasonality. As it turns out, having a stable mean and variance can make machine learning models much easier to train, and thus we explore ways of making time series models stationary. A useful method to detrend a time series is by differencing consecutive values. Methods for deseasoning a time series include differencing consecutive seasons in the data or by applying a log transform [11].

3.6.1.2 Differencing

Differencing is performed by subtracting the previous values from the current values at each time-step, essentially performing a discrete differentiation of the data, estimating velocity from position measurements, acceleration from velocity measurements, jerk from acceleration measurements, etc. An example of differencing is given in Equation 3.29.

$$X_{train} = \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} \longrightarrow X_{train} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ z_k \\ t_k \end{bmatrix} - \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \\ t_{k-1} \end{bmatrix} \tag{3.29}$$

This detrending of the data serves two main purposes. The first property of differencing is that it can make data stationary by stabilizing the mean of the time series. It does this by removing the changes of the time series data, helping to remove trends or seasonality [38].

Second, differencing allows the neural network to learn the trends of a sequence rather than the original values, ensuring that the predictions can be made even if the data is shifted in space or taken at different time intervals. This property of differencing is rather powerful in spatial forecasting, as it can significantly reduce the amount of data needed to train a network.

For example in [66], given a list of sequential positions taken from a basketball following a 3 point shot trajectory, a network can learn to predict the next position in the sequence, so long as the shot lies in the same area as shots that were trained upon and is headed toward the

origin located at the basket. In order to predict shots headed toward the basket on the other side of the court, all that needs to be done is a rotation on the shots to orient them toward the basket in the same way the training data was oriented. However, if the network tried to predict shots taken at another point in space, such as shots not directed toward the basket or translated away from the training data, the predictions would fail. Thus, in order to successfully learn the dynamics of a basketball shot, the network would need to see examples of shots taken in all points in space. If the basketball trajectories used for training were differenced beforehand, the network would be able to predict the rest of the trajectory at any point in space and at any point in time.

3.6.1.3 Power Transform

If the time series has a quadratic or exponential trend, applying a square root or log transform can remove the respective power trend and reduce the trend to a linear one. These transforms are given in equations 3.30 and 3.31.

$$X_{train} = log(X) \tag{3.30}$$

Or

$$X_{train} = \sqrt{X} \tag{3.31}$$

Once the trend is linear, it can be detrended by consecutive differencing. Moveover, applying a power transform may aid in deseasoning the data if the variances grow with time [10]. Determining which power transform to use is typically a process determined visually with trial and error until the variances and means appear to be of first or zeroth order.

3.6.1.4 Standardization

Standardization is a process of making the data zero mean with a standard deviation of 1. Standardization is commonly used when training neural networks as it aids in optimization in the process of gradient descent. This is because of the way gradients are calculated during the optimization process. If the features in the network are scaled differently, the gradients

for each feature will vary. Given that the step size toward the minima is the learning rate multiplied by the gradient, training a network on unstandardized data will cause each feature to have differing optimal learning rates. Standardization solves this problem by ensuring the features have similar gradients, thus one learning rate can be chosen for all features and an optimal solution can be reached quicker [61] [62].

Standardizing data is rather straight forward, simply subtract the mean value of the time series and then divide by the standard deviation as is shown in Equation 3.32

$$X_{train} = \frac{X - \bar{X}}{\sigma_X} \tag{3.32}$$

It is important to accurately estimate the mean and standard deviations for standardization. Using estimates given by the training set should suffice, given the training set is well representative of the general time series problem at hand [13].

The standardization should be performed for each feature independently. Additionally, it is important that standardization parameters are calculated from random samples of the training data prior to training as opposed to using means and standard deviations of each respective training segment. This ensures the network predictions on new data be destandardized accurately. Standardization also will not be accurate if the data is not yet stationary, thus differencing and/or power transforms should be performed prior to standardization if the data is not already stationary.

3.6.1.5 Normalization

Rescaling or normalizing the time series before training can be beneficial when training neural networks. This is because the activation functions within the networks can attenuate signals that lie beyond certain ranges. For Long-Short Term Memory networks, the typical activation functions used include sigmoid and tanh functions. Sigmoid functions will attenuate data outside the bounds of 0 and 1, while tanh will attenuate signals outside -1 and 1. This was seen in Figure 2.2. As such, it is prudent to normalize the data in between one of these two ranges before training [13].

To normalize each feature in the time series tensor to values between 0 and 1, each feature is first differenced by the global minimum, and then divided by the global maximum minus the global minimum, as seen in Equation 3.33. To normalize between -1 and 1, perform the procedure above and then multiply by 2 and subtract 1, as shown in Equation 3.34.

$$X_{train} = \frac{X - min(X)}{max(X) - min(X)} \tag{3.33}$$

Or

$$X_{train} = \frac{2(X - min(X))}{max(X) - min(X)} - 1 \tag{3.34}$$

Like standardization, normalization requires accurate estimates of the statistical parameters used in the transform. Particularly important is that normalization uses global parameters; the use of local parameters would require knowledge of future minimum and maximum values. A potential difficulty with normalization is that any errors in estimate maximums and minimums will cause errors in the predictions after denormalization.

Additionally using global minimum and maximum values alone can cause large discrepancies in the orders of magnitudes between features. Ideally, networks train off of time series with order of magnitude differences within 10^2. Any larger or smaller may harm the training process. Thus, it is desirable to normalize only after differencing and standardizing the time series data. This allows for accurate estimation of global minimums and maximums within a predictable range, ensuring that the output values are in a desirable range. More information on Normalization can be found in [49] and [9].

3.6.1.6 Feature Engineering

Perhaps some of the most practical transforms made to the data are transforms that aim to reduce the degrees of freedom that the machine learning model is required to learn. In this step lies much of the thought and creativity when a machine learning practitioner is attempting to successfully train accurate and robust networks. Several examples of this have been discussed above, particularly pertaining to transforming coordinate frames and differencing. First, rotating coordinate frames can significantly reduce the amount of data needed to train a network as

well as the size of the network needed to learn a particular relationship. An example of this can be seen in Figure 3.1.

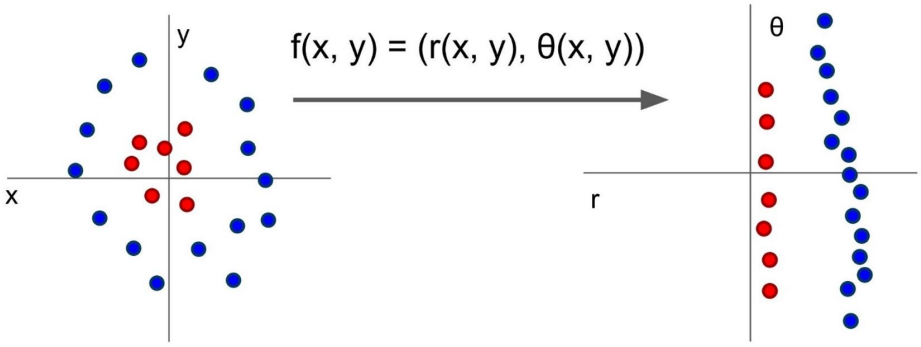


Figure 3.1: Coordinate Transform of data from Cartesian to polar coordinates [51]

In order to design a network to classify the red dots from the blue dots in the plot on the left, a nonlinear model would need to be learned. However, by transforming the coordinates from cartesian to polar coordinates, the classification problem is reduced to a first order linear regression [51]. Such is the importance of carefully and thoughtfully transforming data before it is chewed on by the network.

Another example of this was described in the subsection pertaining to differencing. By performing consecutive differencing and training off changes in the states as opposed to the states themselves, the network can predict future states within any area in space and time.

Finally, combining properties of the rotation transform and the difference transform can be beneficial to both the robustness of the trained network as well as reduction in network complexity. The implementation discussed in Chapter 4 performs all predictions in a local coordinate frame referenced to the radar located on the follower truck in the truck platoon as seen in Figure 4.9. This way, the network only needs to learn trajectories moving in the same general direction, thereby reducing the required complexity of the network. Similarly in Chapter 5, tracks of neighboring vehicles are converted to Normal and Tangential Coordinates referenced to the estimated center of the lane the trucks are platooning in. This transformation removes the complexities of road curvature from the data the network is tasked with learning. The effects of this transform is further explained in Sections 5.3.1 and 5.3.2.

3.6.2 Sequence Prediction Input to Output Mapping

Most neural network architectures are constrained to having a single input map to a single output, known as "One-To-One", however Recurrent Neural Networks can be configured to receive inputs of varying sizes, adding a level of flexibility when designing time series prediction networks. These input sizes, or input-output mapping, can be an input sequence of size one and an output sequence of size one (one to one), input sequences of varying size and output sequences of size one (many to one), or receive inputs of varying size and output sequences of varying size (many to many) [44].

3.6.2.1 One-To-One

One-To-One mapping, as shown in Figure 3.2, is commonly used with non-recurrent neural networks such as Convolutional Neural Networks and Multi-layer Perceptron Models. It maps fixed size inputs to fixed size outputs. An example is feeding a network images and outputting classifications (image classification). One-To-One modeling is not used in Recurrent Neural Networks as it cannot learn over time steps.

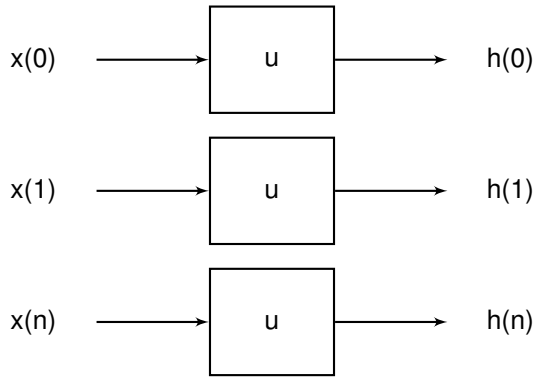


Figure 3.2: One-To-One Sequence Prediction Model

3.6.2.2 One-To-Many

One-To-Many mapping, shown in Figure 3.3, involves mapping one fixed size input to a sequence output. One common usage for this mapping is image captioning. The input to the Recurrent Neural Network is an image, and the output is a caption describing the image.

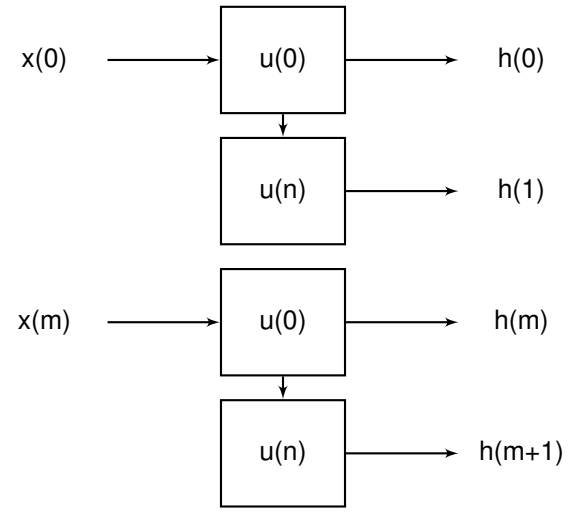


Figure 3.3: One-To-Many Sequence Prediction Model

3.6.2.3 Many-To-One

The Many-to-One configuration uses a vector of sequences to predict a future value, as is depicted in Figure 3.4. An example would be given the list of numbers (1,2,3,4), the network could predict the next number in the list to be 5. The Many-To-One configuration can be used in time series forecasting if all that is required is a single point in the future. If a vector of future sequences is required the network can be run recursively after each prediction by appending the new prediction onto the input vector and dropping the last value. The weakness in this method is that it introduces compounding errors as the algorithm calculates new predictions using the old predictions.

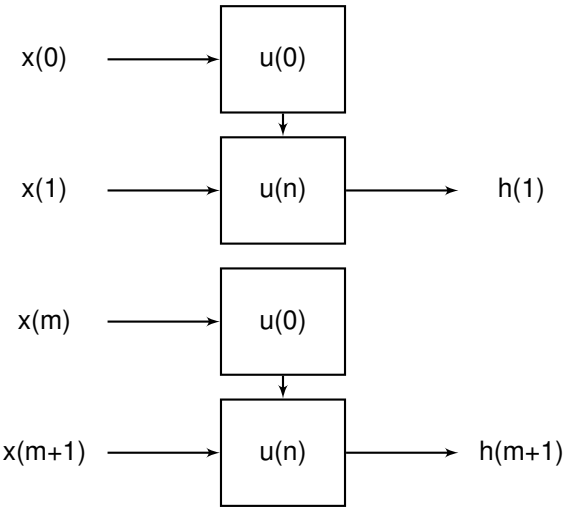


Figure 3.4: Many-To-One Sequence Prediction Model

A better application of Many-To-One is for time series classification. Given a sequence of values, the network can be trained to classify the sequence between different categories. An example of this is predicting the success of NBA 3 point basketball shots given prior ball positions [66]. This thesis employs this configuration to predict probabilities attached to predicted trajectories.

A potential applications of the Many-To-One configuration in Recurrent Neural Networks in the field of vehicle control could be to train a network to perform the duties of a guidance and control algorithm. A network could be trained to receive inputs of the vehicle's states, positions of obstacles, location of waypoints on the road, etc. and output actuator inputs.

Another potential application of Many-To-One in Navigation could be to use a Recurrent Neural Network as a nonlinear filter. A network can be trained to provide navigation solutions in difficult environments in which traditional methods such as Kalman Filters have trouble performing. In pedestrian navigation, a network can be trained to operate in GPS denied environments using a map and imu measurements such as in [60]. It would be interesting to see RNNs applied to other state of the art issues in the field of Guidance, Control, and Navigation such as what is being done in this thesis.

3.6.2.4 Many-To-Many

The Many-To-Many configuration removes the issue of compounding errors by allowing the network to predict future values over a desired amount of steps into the future. A diagram of this can be seen in Figure 3.5. The future sequence length is not fixed to the length of the input sequence. For example, an input sequence of size 5 can be used to predict a future sequence of size 10. This adds flexibility over other prediction methods that are limited to predicting fixed sequence lengths. This is the chosen mapping to be used to predict the future pose of vehicles neighboring the truck platoon given the vehicles' previous states, as a shorter time length can be mapped to a longer prediction horizon.

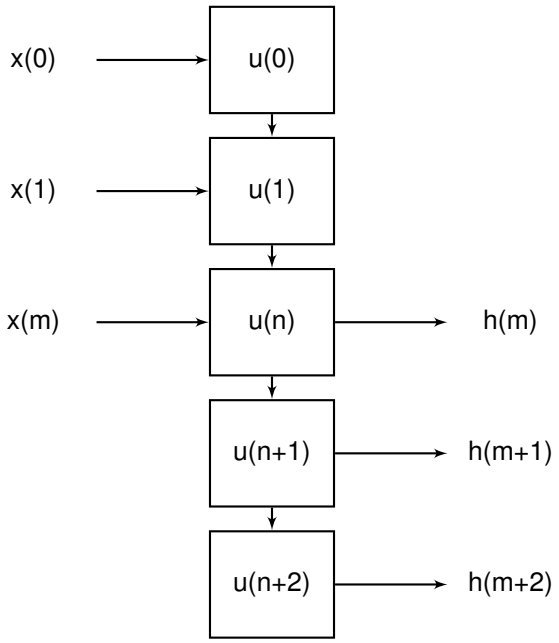


Figure 3.5: Many-To-Many Sequence Prediction Model

Another form of Many-To-Many maps synced sequence inputs to outputs, and can be seen in Figure 3.6. An example of this would be labeling frames of video one frame at a time. This method is used for CNN labeling of objects over time such as labeling vehicles in each epoch of camera or radar data. This mapping could be useful in aiding radar detection algorithms. For example, a forward facing camera could be used to validate detected radar objects like cars, trucks, or clutter.

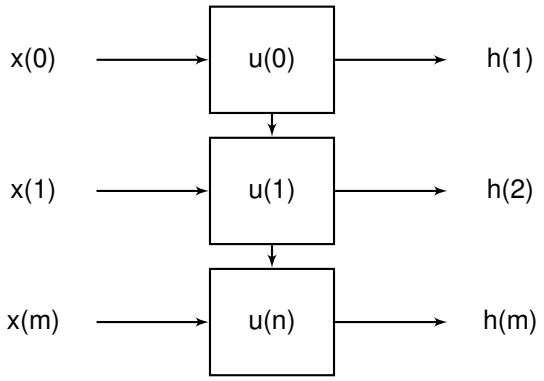


Figure 3.6: Synced Many-To-Many Sequence Prediction Model

3.6.3 Encoder Decoder Models

Encoder Decoder Models divide the work of a network into two discrete tasks between an Encoder and a Decoder network. The Encoder Network is tasked with receiving input data

and “encoding” meaning from the data into a vector. The Decoder Network is then tasked to “decode” or interpret the vector outputted from the Encoder network and to generate predictions from it. An example of this would be a network that is trained to translate a sentence written in French to English. By using an Encoder-Decoder structure, one network can essentially be trained to understand French, and the other to speak English. Together they work to complete the entire task more efficiently than a single network of the same size. This allows for smaller, more efficient network sizes [73][39][17].

3.7 Conclusion

Many methods of tracking and time series forecasting have been presented with aim to predict the future states of systems. These methods include Kalman Filtering, state-based and goal-based predictors, as well as machine learned and deep learned methods, including the Long-Short Term Memory Recurrent Neural Network. Recurrent Neural Networks are a type of deep neural network specifically designed to predict sequential behavior and have seen large success in many areas including natural language processing, economics, and weather. This thesis utilizes on the Long Short-Term Memory Network to take advantage of its ability to learn complex nonlinear problems such as truck platoon cut-ins.

4

LSTM Network for Cut-in Prediction and Detection in Simulated Environment

"Essentially, all models are wrong, but some are useful."

– George Box, *Empirical Model-Building and Response Surfaces* [22]

4.1 Introduction

A simplified approach is given to provide proof of concept for the use of an LSTM predictor for track platoon cut-ins. A truck platoon is simulated to drive due north at a constant velocity and constant spacing while a neighboring vehicle is randomly commanded to either cut into the platoon or to drive past. The simulation is performed for straight, flat roads, and vehicle pose estimates are calculated without radar noise. The results of this initial study serve to guide the direction of the implementation in Chapter 5, in which these principles will be applied to empirical data obtained from highway runs on the track platoon.

4.2 Training Data Design

One weakness of Deep Neural Networks is the requirement of large swaths of data needed for training. If the training set for a network is too small, the network may fail to generalize to new data sets after training, a phenomena well known as over-training or over-fitting. There are several solutions to over-training including early stopping, decreasing the network size, and dropout. However, none of these solutions plays to the advantages of Deep Neural Networks as much as increasing the size of the data-set [72].

This creates difficulties when using a tool like LSTMs to solve a problem with a dataset of limited size. This thesis aims to predict cut-ins for a truck platoon, but there is no existing database centered on this issue. Moreover, the available experimental data does not provide enough instances of cut-ins to create a training set, thus this thesis relies heavily off of modeling and simulation to provide enough data to train and test the LSTM networks designed herein. Chapter 4 focuses on the theoretical feasibility of LSTMs to predict cut-ins given a simulated environment. Chapter 5 explores the concept of using modeling and simulation to generate supplementary data for training networks to predict behavior of real world systems, this case being radar tracked vehicles driving near the truck platoon. This section focuses on the Modeling and Simulation environments created to supply training and test sets for these studies.

4.2.1 Lateral Vehicle Dynamics

The simulation environment for this study uses a bicycle model shown in Figure 4.2 to model the lateral dynamics of a vehicle merging in between two trucks. The car parameters originate from a generic SUV vehicle model in the high fidelity Anvel car simulation. This model was chosen in the case this work would be moved into a higher fidelity simulation environment such as Anvel. A comparison between the Anvel simulation environment and other environments such as Gazebo and Carsim can be found in [8]. A rendering of the vehicle is given in Figure 4.1.

Figure 4.1: Vehicle Rendering of Anvel Generic SUV

The vehicle parameters are given as:

Table 4.1: Anvel Generic SUV Vehicle and Tire Parameters

Vehicle Parameters		Tire Parameters	
l_f	1.05 *m*	Ca_f	20000 $\frac{N}{deg}$
l_r	1.61 *m*	Ca_r	20000 $\frac{N}{deg}$
I_{zz}	2059.2 kgm^2		
m	1430 *kg*		

A visual diagram of the bicycle model used herein is given in Figure 4.2.

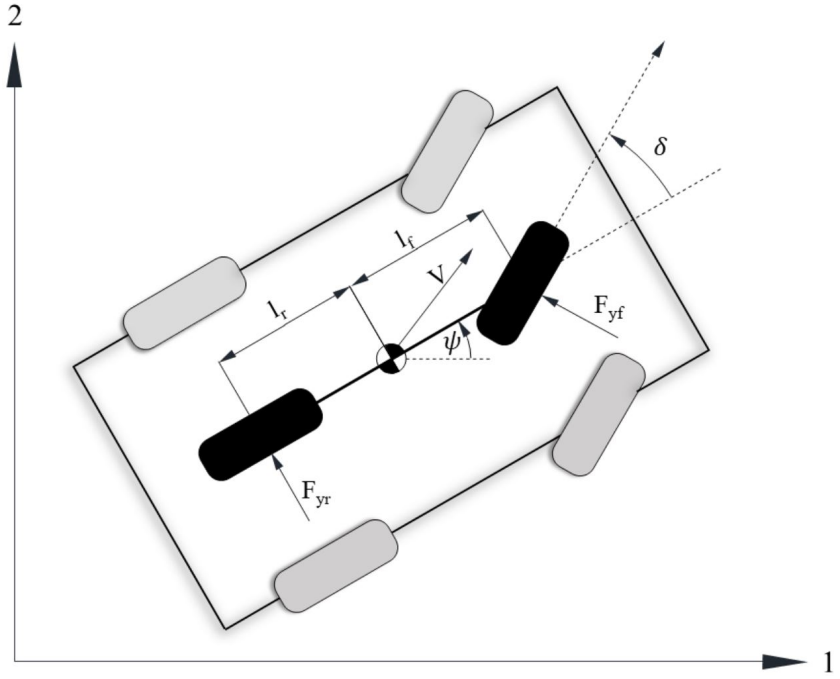


Figure 4.2: Bicycle Model used in Simulation Environment to Represent the Merging Vehicle

Assuming two degrees of freedom (lateral velocity and heading), no lateral or longitudinal load transfer, and no rolling or pitching motion, the dynamic model of the system can be written as

$$m\ddot{y} = -m\dot{\psi}V_x + 2F_{yf} + 2F_{yr}, \tag{4.1}$$

$$I_z\ddot{\psi} = 2l_fF_{yf} - 2l_rF_{yr}, \tag{4.2}$$

where l_r and l_f are the distances from the center of gravity to the rear and front axles respectively, F_{yf} and F_{yr} are the front and rear tire forces, and m and I_z are the mass and inertia of the vehicle.

The front and rear tire slip angles can be expressed as:

$$\alpha_f = \delta - \frac{\dot{y} + l_f\dot{\psi}}{V_x}, \tag{4.3}$$

$$\alpha_r = -\frac{\dot{y} + l_r\dot{\psi}}{V_x}. \tag{4.4}$$

Where δ is steer angle and V_x is the longitudinal velocity. These equations can be arranged into State Space format yielding the system

$$A = \begin{bmatrix} -\frac{(l_f^2 C_{af} + l_r^2 C_{ar}}{V_x I_z} & \frac{-l_f C_{af} - l_r C_{ar}}{V_x I_z} \\ \frac{-l_f C_{af} + l_r C_{ar} - m V_x^2}{m V_x} & \frac{-C_{af} + C_{ar}}{m V_x} \end{bmatrix}, \tag{4.5}$$

$$B = \begin{bmatrix} \frac{l_f * C_{af}}{I_z} \\ \frac{C_{af}}{m} \end{bmatrix}. \tag{4.6}$$

Where

$$\begin{bmatrix} \ddot{\psi} \\ \dot{V}_y \end{bmatrix} = A \begin{bmatrix} \dot{\psi} \\ V_y \end{bmatrix} + B\delta(t). \tag{4.7}$$

The local lateral velocities obtained from the bicycle model can then be integrated and rotated about the heading ψ to yield the vehicle's global position [67]. Using Euler Integration, the global position is described as

$$\begin{bmatrix} X \\ Y \end{bmatrix}_{k+1} = \begin{bmatrix} X \\ Y \end{bmatrix}_k + \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix}_k dt. \tag{4.8}$$

4.2.2 Longitudinal Vehicle Dynamics

Given that this chapter is intended to provide proof of concept, varying longitudinal velocity in the following manner may not be perfectly representative of real driver behavior, but it should provide the training data with comparable complexity, as it introduces nonlinearities into the simulated vehicle dynamics. To approximate varying longitudinal velocities in passing vehicles, the ground vehicle's longitudinal velocity is controlled via a second order transfer function that follows a sine reference with random magnitude and frequency and with zero mean random added noise σ. An integrator was added to remove steady state errors [26]. The transfer function is given as

$$\frac{V(s)}{e(s)} = \frac{1}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)}. \tag{4.9}$$

Where the error between the reference velocity V_{ref} and the vehicle velocity V_k is given as

$$e_k = V_{ref} - V_k + \sigma. \tag{4.10}$$

The transfer function may be inverse Laplace Transformed to yield

$$\dddot{V}_{k+1} = -2\zeta\omega_n\ddot{V}_k - \omega_n^2\dot{V}_k + e_k. \tag{4.11}$$

Euler integrating gives

$$\ddot{V}_{k+1} = \ddot{V}_k + \dddot{V}_{k+1}dt, \tag{4.12}$$

which may be integrated once more to yield the longitudinal acceleration command

$$\dot{V}_{k+1} = \dot{V}_k + \ddot{V}_k dt. \tag{4.13}$$

This is then used to calculate the new longitudinal velocity such that

$$V_{xk+1} = V_{xk} + \dot{V}_{xk+1}dt. \tag{4.14}$$

Lastly, longitudinal velocity in the A matrix of the bicycle model is updated with V_{xk+1}.

A separate controller is used to drive the vehicle to a reference distance behind the lead truck if the vehicle decides to merge into the platoon. In similar fashion as above, the merging vehicle's longitudinal dynamics are modeled as a second order transfer function of which an integrator has been added to remove steady state error.

$$\frac{d(s)}{e(s)} = \frac{1}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)} \tag{4.15}$$

Where e_k is the error between the current distance from the lead, d_k, and the reference distance d_{ref} is given as

$$e_k = d_{ref} - d_k. \tag{4.16}$$

d_k is calculated as the longitudinal distance between the lead truck and the vehicle and is given as

$$d_k = Y_{LeadTruck} - Y_{vehicle}. \tag{4.17}$$

Applying the inverse Laplace Transform yields

$$\dddot{d}_{k+1} = -2\zeta\omega_n\ddot{d}_k - \omega_n^2\dot{d}_k + e_k. \tag{4.18}$$

Which may be Euler Integrated twice to give

$$\dot{d}_{k+1} = \dot{d}_k + \ddot{d}_{k+1}dt. \tag{4.19}$$

Given that

$$\ddot{d}_{k+1} = a_{LeadTruck_{k+1}} - a_{x_{k+1}}, \tag{4.20}$$

then (4.20) can rearranged and substituted into (4.19) to yield

$$V_{xk+1} = V_{xk} + a_{xk+1}dt. \tag{4.21}$$

The longitudinal dynamics may be tuned with the parameters ζ and ω_n which are the respective damping ratio and the natural frequency of the system.

4.2.3 Pure Pursuit Controller

The ground vehicle is controlled along a two lane highway to either make a lane change in between two platooning trucks moving at constant velocity or to pass the platoon. A Pure Pursuit controller is used to provide lateral control for the vehicle. The controller can be defined as

$$\delta(t) = \tan^{-1}\left(\frac{2(l_f+l_r)\sin\alpha(t)}{L_d}\right) \tag{4.22}$$

where L_d is the distance between the vehicle and the next waypoint, known as the look ahead distance, and $\alpha(t)$ is the angle between the vehicle's heading and the desired waypoint. The pure pursuit controller drives the vehicle along a circular arc to its desired waypoint, which it chooses using a set look-ahead distance L_d. This distance acts as a tuneable parameter with shorter look-ahead distances providing quicker responses and longer distances providing slower responses [69].

4.2.4 Radar

The Delphi radar on the follower truck has a wide span of 45 degrees with a range of 60 meters, and a narrow span of 15 degrees with a range of 120 meters. The radar has an update frequency of 20 Hz. The radar is modeled in simulation using a 45 degree span with a range of 120 meters extending from the center of the follower truck. A vehicle is detected and tracked once it first enters this region, and is tracked at a frequency of 20 Hz. The three measurement outputs from the radar are azimuth, range, and range rate. These are determined as follows:

$$\theta = atan2\left(\frac{y}{x}\right) \tag{4.23}$$

$$R = \sqrt{y^2 + x^2} \tag{4.24}$$

$$\dot{R} = \sqrt{\dot{y}^2 + \dot{x}^2} \tag{4.25}$$

where x and y are the vehicle position coordinates local to the follower truck. The radar outputs are converted to global North and East coordinates to be used as features in the network input and to determine a cut-in once the tracked vehicle crosses the lane line.

4.2.5 Monte Carlo Trajectory Generation

Monte Carlo runs are performed with randomized lateral and longitudinal control parameters between each simulated trajectory. The aim of this is to provide randomness and unpredictability that may be roughly representative of driver behavior, or at least as difficult to predict. The vehicle waypoints are given zero mean uniform white noise such that the controller response may capture some disturbances and a steady state bias to represent drivers that may have tendencies to drive within different fractions of the lane. The waypoints are also added to a sinusoid with random magnitude and frequency such that the vehicle may sway from the left lane into the right lane but may not run off the road.

The initial vehicle speed is uniformly randomized to be between

$$V_{Truck} + 1 < V_{car} < V_{truck} + 17, mph. \tag{4.26}$$

This allows the network to view more interesting behavior such as a vehicle slowing down quickly to merge.

The Pure Pursuit initial look ahead distance is initialized randomly between

$$20 < L_{dk=0} < 100, m. \tag{4.27}$$

Look ahead distances of 20 meters provide aggressive cut-in behavior by driving the lateral response to a settling time of roughly a second. On the other end of the spectrum, look ahead distance of 100 meters provides a slower cut-in response with a settling time of roughly 5 seconds.

To make the merge response more unpredictable, subsequent look ahead distances are calculated as

$$L_{dk+1} = L_{dk} + \rho \sin \omega t_k + \sigma. \tag{4.28}$$

Where ρ and σ are given as

$$20 < \rho < 50, m, \tag{4.29}$$

and

$$20 < \sigma < 50, m. \tag{4.30}$$

ω is set to be

$$0 < \omega < 0.5, \frac{rad}{s}. \tag{4.31}$$

This can cause a slow merging vehicle to suddenly merge quickly, and vice-versa.

The longitudinal control transfer function parameters are randomized uniformly between

$$0.7 < \zeta < 1.4, \tag{4.32}$$

and

$$0.5 < \omega_n < 4, \frac{rad}{s}. \tag{4.33}$$

This causes some longitudinal velocity responses to behave with quick response times, representing more aggressive drivers, or more slowly representing more conservative drivers.

The longitudinal velocity reference is added with a sinusoid of random magnitude and frequency, as well as white noise given by

$$V_{ref_{k+1}} = V_{ref_k} + \rho \sin \omega t_k + \sigma \tag{4.34}$$

where

$$0 < \rho < 6, mph, \tag{4.35}$$

and

$$0 < \omega < 0.5, \frac{rad}{s}. \tag{4.36}$$

The random noise disturbance, σ, added to the velocity response is set as $\sigma = 3\frac{m}{s}$. This parameter allows the vehicle to remain close to a reference velocity but may waver about the reference depending on the values of ζ and w_n.

The positions of the vehicle are uniformly randomized between

$$Y_{FollowerTruck} - 40 < Y_{car} < Y_{FollowerTruck} - 250, m. \tag{4.37}$$

If the passing vehicle decides to cut into the platoon, the reference longitudinal cut-in position is set as a uniformly random distance behind the lead truck in order to mimic driver's differing preferences of following distance. The possible range of following distances is given by

$$4 < Y_{LeadTruck} - Y_{car} < 12, m. \tag{4.38}$$

4.2.6 Sample Trajectories

Sampled trajectories for the straight road simulation are presented below to demonstrate the dynamics of passing and cut-in vehicles near the platoon. Whether a trajectory is a cut-in or passing is randomly predetermined at the start of the simulation. If the simulation is of a cut-in, the vehicle will begin to merge between the trucks once it has calculated that it can merge without a collision. Presented in detail are one passing and one cut-in trajectory, then 1000 training trajectories are plotted together for comparison.

4.2.6.1 Passing Trajectory

A passing trajectory was sampled with a Look Ahead Distance of 90 meters and initial velocity of 66.8 miles per hour and is shown in Figure 4.3. The waypoints are shown in black, the vehicle trajectory is in red, and the lead and follower trucks are in green and blue respectively.

It can be seen from Figure 4.3 the effect of the 90 meter Look Ahead distance on the lateral dynamics of the passing vehicle, as the vehicle captures the sine reference with roughly half π of phase delay. This is useful in modeling drivers that may tend to sway back in forth in their lane. The vehicle Steer Input and Heading can be viewed in Figure 4.4.

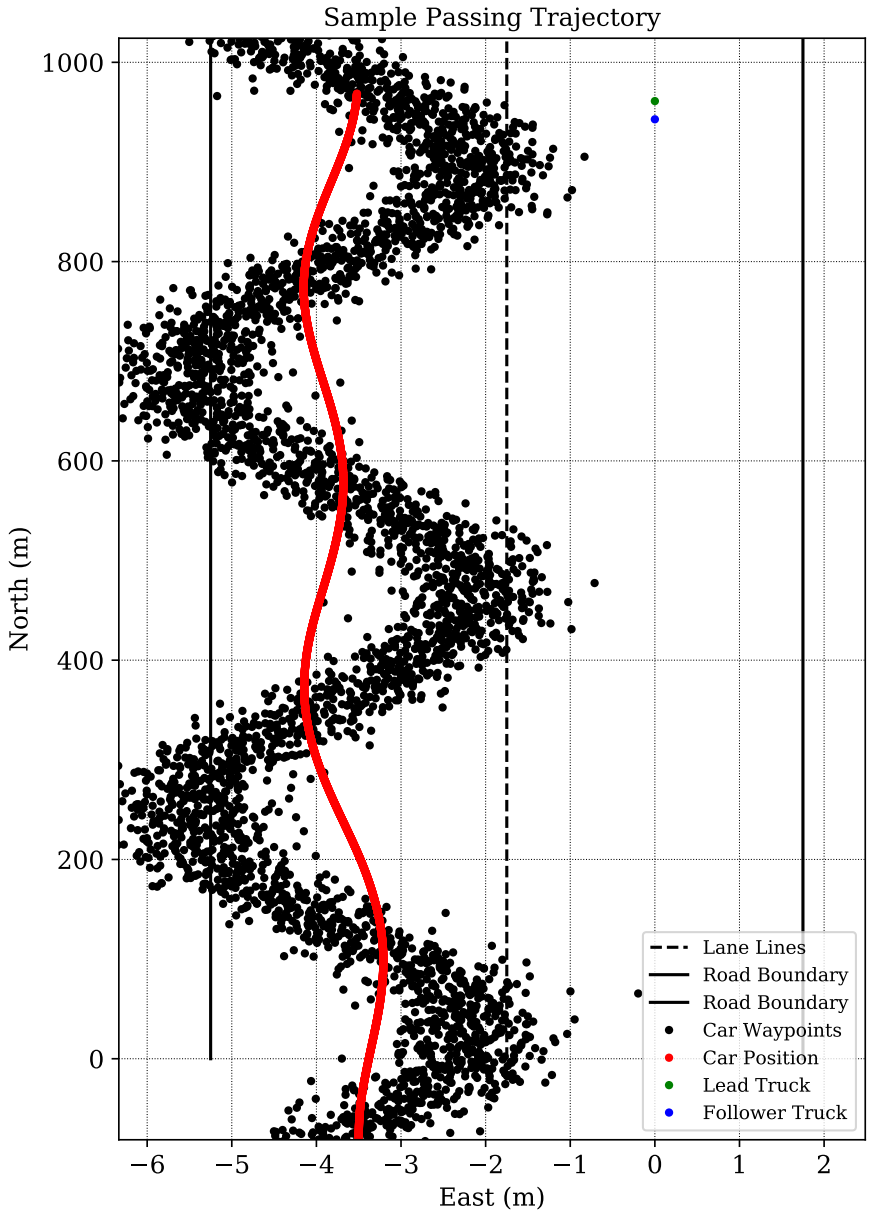


Figure 4.3: Sample Passing Trajectory with Look Ahead Distance of 90 meters

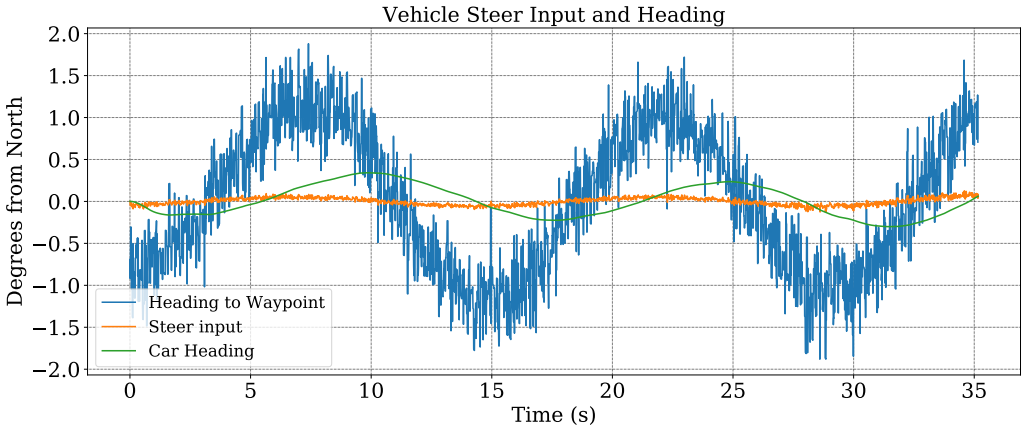


Figure 4.4: Heading for a Sample Passing Trajectory with Look Ahead Distance of 90 meters

Figure 4.4 shows the slight phase delay in heading that the vehicle undergoes that occurs with a period of roughly 15 seconds. Figure 4.5 shows the longitudinal velocity of the passing vehicle, which oscillates with a period of roughly 35 seconds with an amplitude of 2.2 miles per hour.

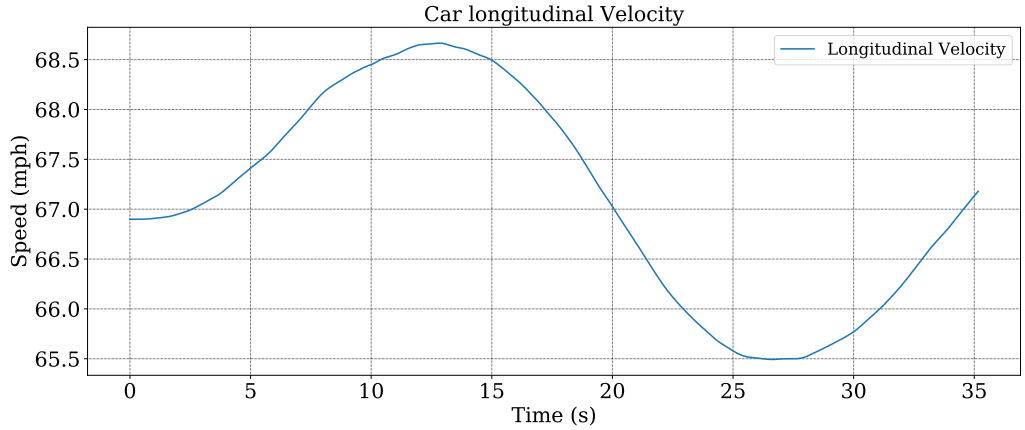


Figure 4.5: Longitudinal Velocity for a Sample Passing Trajectory with Look Ahead Distance of 90 meters

Given the input to the network is 1.25 seconds or less, the random noise and sinusoids applied to the vehicle lateral and longitudinal responses significantly increase the needed learning space for the designed network.

4.2.6.2 Cut-in Trajectory

Figure 4.6 shows a sample cut-in trajectory with a look ahead distance of 50 meters. Like the passing trajectory sampled above, the cut-in trajectory follows a sine reference with a random amplitude, frequency, and bias. The waypoints snap to the left lane when the vehicle decides to cut-in. For this sample, the cut-in decision is made at t = 23.14 seconds and the subsequent cut-in occurs at t = 27.85 seconds, giving a time to cut in of 4.71 seconds.

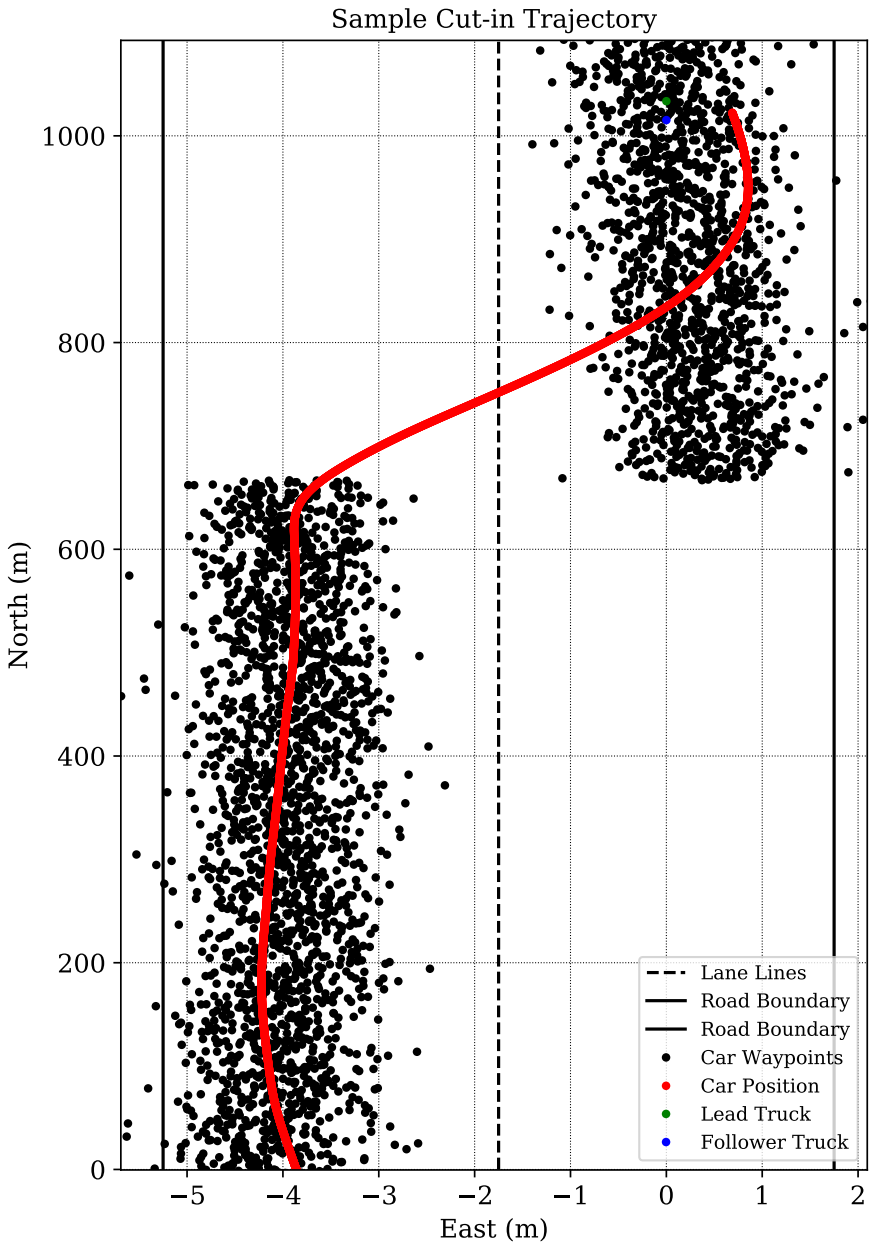


Figure 4.6: Sample Cut-in Trajectory with Look Ahead Distance of 50 meters

Figure 4.7 shows the the steer angle input from the Pure Pursuit controller and the heading response from the bicycle model. The heading to the next waypoint can be seen snapping from 0 degrees to roughly -4.8 degrees when t = 23.14s, marking the moment when the vehicle begins to track waypoints that are now in the right lane.

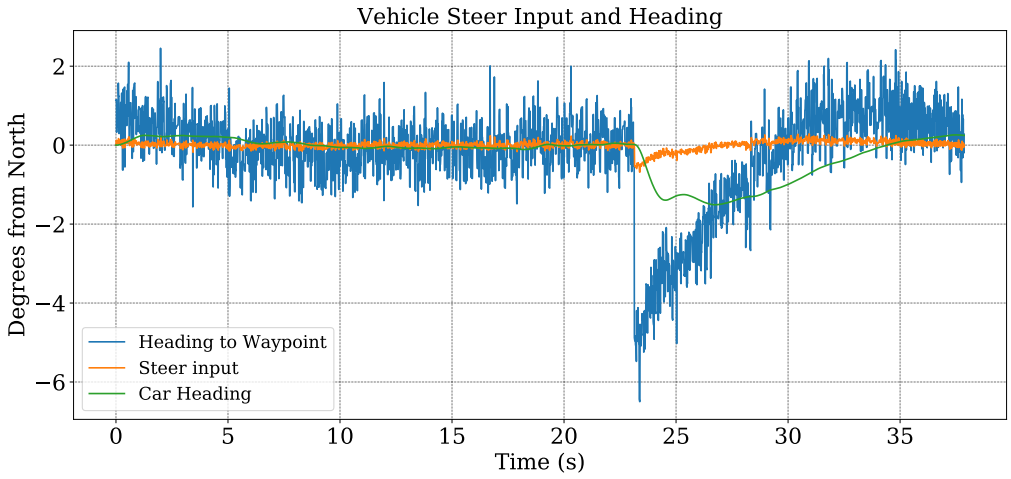


Figure 4.7: Heading for a Sample Cut-in Trajectory with Look Ahead Distance of 50 meters

As shown in Figure 4.8 the vehicle brakes at t = 23.78 seconds in order to align itself longitudinally behind the lead truck. This process occurs once the vehicle has determined it can start braking without merging into the follower truck.

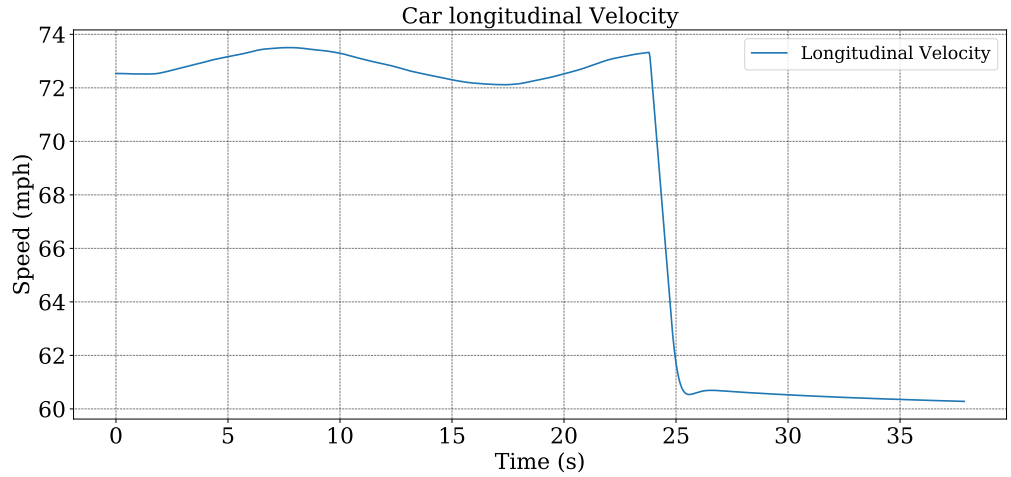


Figure 4.8: Longitudinal Velocity for a Sample Cut-in Trajectory with Look Ahead Distance of 50 meters

This sampled trajectory demonstrates the nonlinear and random varying behavior of generated cut-in trajectories that will make training the network non-trivial.

4.2.6.3 Training Data

Running the simulation over 100 iterations yields the sample trajectories seen in Figure 4.9. The sample trajectories plotted are the north and east components calculated from the radar measurements taken from the front of the follower truck, which points due north in the center of the right lane. The passing modes are colored blue while the cut-in modes are colored red.

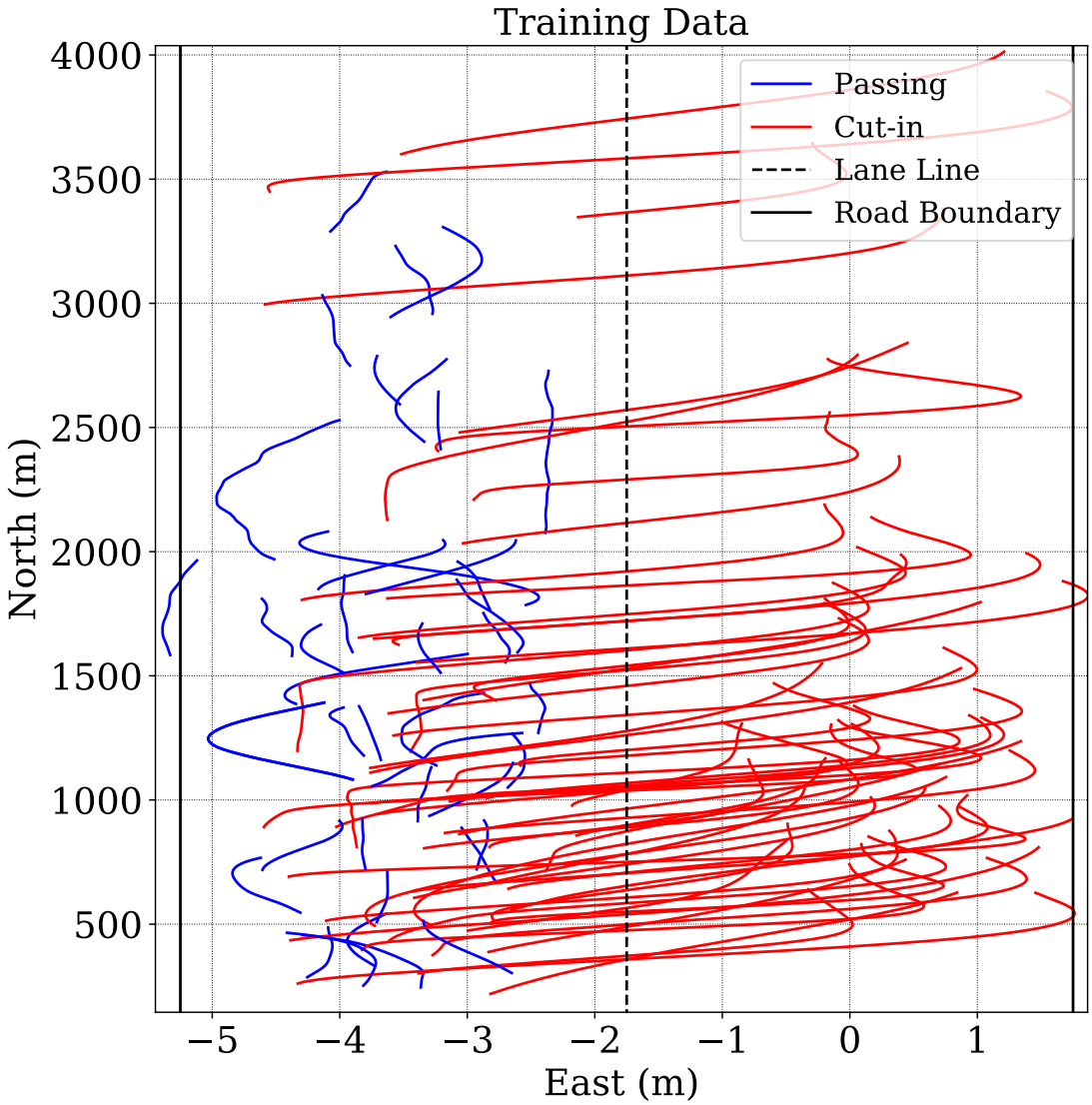


Figure 4.9: 100 Sampled Trajectories of Straight and Merging Vehicles

Translating the sample trajectories to the same starting point gives a better view of the training data set. This is shown in Figure 4.10.

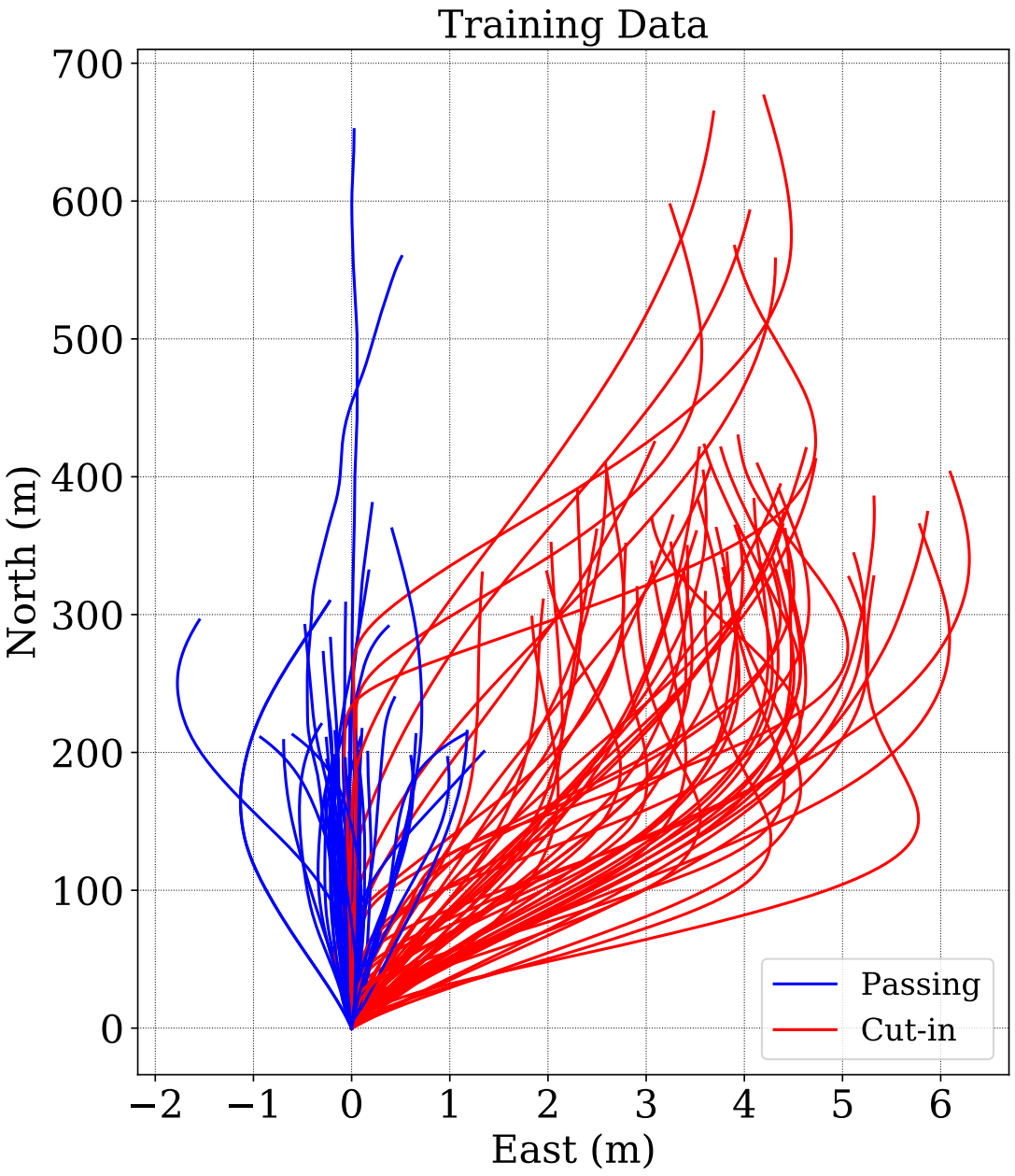


Figure 4.10: 100 Sampled Trajectories of Straight and Merging Vehicles Translated to the Same Starting Point

There were threee training sets in total: one consisting entirely of cut-ins, one consisting of passings, and one evenly divided between both modes. For the first two sets simulations were run until 5000 trajectories of each mode were generated. These sets were used to train the cut-in and passing networks. The third training set of 2000 trajectories mixed equally between both modes was generated to train the probability piece of the ensemble network. In addition to the training set, comprised of 11000 trajectories in total, validation sets were generated to test the progress of network training throughout training. The validation sets consisted roughly of 1000 cut-ins, 1000 passings, and 1000 randomly mixed modes. Finally, a test set of 1000 trajectories, randomly generated evenly between cut-in trajectories and passing trajectories was generated to test the performance of the trained network.

4.3 Neural Network Design

This section describes the algorithms and procedures used in the simplified implementation, divided into subsections describing the design of the Neural Network and the design of the simulation used to generate training and testing trajectories.

4.3.1 Inputs and Outputs

The input to the network is the north and east position components of the tracked vehicle as measured by the radar at each time step referenced to the navigation frame of the follower truck. These inputs were chosen because they are sensor nonspecific, and can be used given other methods of traffic agent position estimation. This leads to an input array shape of (batch size, sequence length, features). The batch size is the number of sampled trajectories, sequence length is the length of each trajectory, and the number of features is summed to two, where the two features are the north and east positions of the tracked vehicle.

4.3.2 Data Augmentation

Before being fed into the network, the radar input is preprocessed to improve training. The data is first differenced and then standardized about the mean and standard deviation of the

differenced values. The differencing of the input data is given as

$$X_{diff} = \begin{bmatrix} \Delta N \\ \Delta E \end{bmatrix} = \begin{bmatrix} N_k \\ E_k \end{bmatrix} - \begin{bmatrix} N_{k-1} \\ E_{k-1} \end{bmatrix}. \tag{4.39}$$

Then the data is standardized such that

$$X_{train} = \frac{X_{diff} - \bar{X}_{diff}}{\sigma_{X_{diff}}}. \tag{4.40}$$

Standardization is performed using means and standard deviations of a differenced sample set of the training data and is performed for each feature independently. The standardization is performed after differencing because the standardization will not be accurate if the data is not yet stationary.

After training, the network output is converted back to Cartesian coordinates by destandardizing, using the same sampled mean and standard deviation for each feature, and then reverse differencing using the original initial values of each segment.

4.3.3 Network Architecture

The network is comprised of an ensemble of two separately trained networks, one trained to recognize cut-in behavior hereby known as the Cut-in Network, and the other trained to recognize passing behavior hereby known as the Passing Network. Each network in the ensemble is comprised of an encoder-decoder structure, which divides the task of time series prediction into two discrete tasks. The encoder network interprets the input data and derives objective meaning from it in the form of a vector. The decoder observes the objective meaning of the output vector from the encoder and infers predictions from it. The encoder and decoder are both composed of two LSTM layers with 128 hidden units.

The network trajectory predictions are taken from the output of the respective final LSTM layers in the Cut-in and Passing Networks. The probabilistic confidence values attached to each trajectory output are generated by passing the respective outputs through another LSTM layer to yield a single output array with two features, which is then passed through an MLP

(dense) layer to remove the time dependency, and finally the output is passed through a Softmax activation function so that the final output is two scalars with a sum equal to one. The entire network architecture can be viewed in Figure 4.11. The network and custom loss functions were coded using the Keras Functional API [19] with TensorFlow [53] as the back-end.

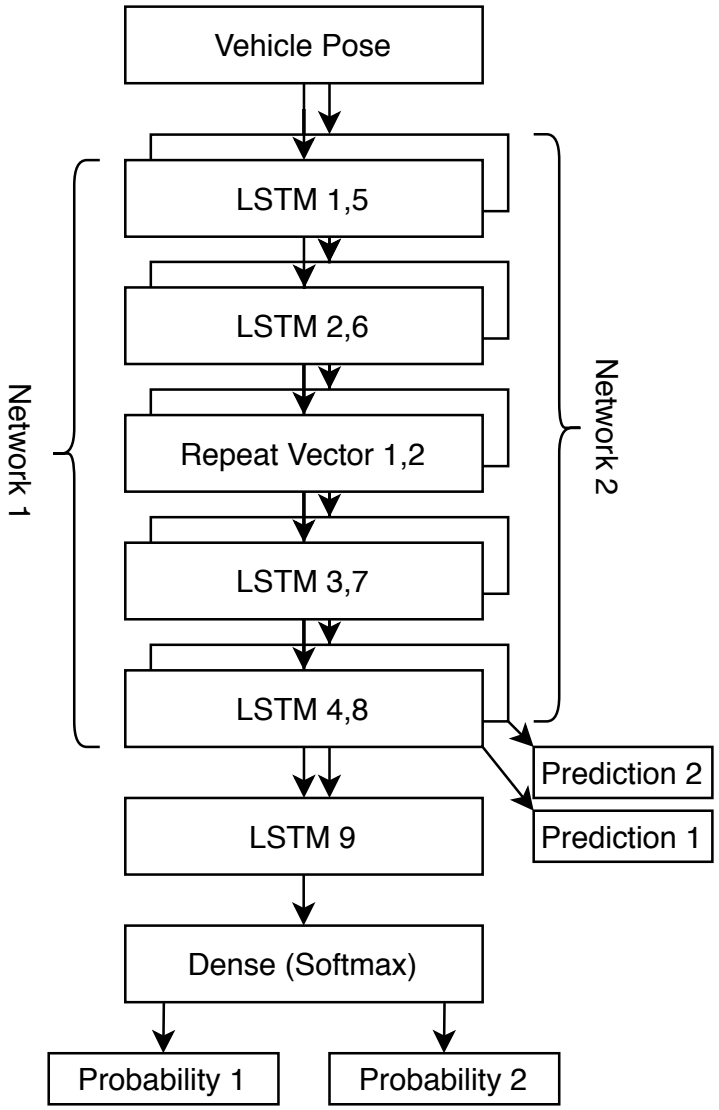


Figure 4.11: Network Architecture

4.3.4 Network Training

The network is trained using Adam optimizer with a learning rate of 0.001, input sequence lengths of 25 neighboring vehicle position estimations over 1.25 seconds, output sequence lengths of 100 future neighboring vehicle positions over 5 seconds, and batch sizes of 1000 trajectory segments, each randomly sampled from a unique trajectory in the training set.

Dropout of 0.5 was used in the final LSTM and dense layers of the Ensemble Network. The entire Ensemble Network is trained in two steps. First, the Cut-in and Passing networks are trained on data sets consisting solely of their respective vehicle behavior mode. In other words, the Cut-in Network is trained and validated on cut-ins only, and the Passing Network is trained and validated on passings only. These networks are trained using Mean Squared Error (MSE) as the loss function. The second step in training was to train the probability outputs assigned to the trajectories predicted by the trained Cut-in and Passing Networks.

4.3.5 Loss Function for Ensemble Probability Output

To train the network to predict probabilistic certainty values for each prediction mode, a custom loss function was created that takes into account the two modal trajectory predictions as well as the predicted probabilities. The network outputs from LSTM layers 1 and 4 as described by Figure 4.11 are passed through an MSE distance loss given by (4.41).

$$L_{Distance j} = \frac{1}{sl} \sum_{i=1}^{sl} (y_{pred_{j,i}} - y_{true_i})^2 \tag{4.41}$$

Where sl is the output sequence length, y_{pred_j}, i is the j^{th} predicted trajectory at the i^{th} time instance and y_{true} is the truth trajectory.

A slightly modified Cross-Entropy loss, given in (4.42), is applied to the probability output corresponding to the prediction with the lowest distance loss to penalize low probabilities attributed to winning trajectories.

$$L_{Classification} = mask \left[-\alpha \log(p)^{\beta} \right] \tag{4.42}$$

where

$$mask_j = \begin{cases} 1 & \text{if minimum index of } L_{Distance} = j \\ 0 & \text{otherwise} \end{cases} \tag{4.43}$$

This procedure drives the probability of the winning predicted trajectory to be closer to one and the loser closer to zero. α and β are used as tuning parameters to aid in training.

Increasing α punishes predicted probabilities of winning predicted trajectories that are further from 1, encouraging the network to provide more confident predictions. Increasing β punishes probabilities of winning trajectories that are close to zero, encouraging the network to provide more conservative predictions, or predictions closer to the probability of a coin flip. The distance loss is then discarded and the classification loss is passed back through the network for back propagation to train the probability outputs.

4.4 Results

The results of the network performance on a simulated test data set are presented below. Several different metrics are used to measure performance of different pieces of the network. To evaluate the accuracy of predicted trajectories, Root Mean Squared Error to the truth trajectory is calculated at each time epoch. This is given in Section 4.4.1. The accuracy of the probabilities attached to the trajectories are evaluated by plotting the Mode Probability Calibration, which describes how well the predicted probabilities correspond to actual rate of occurrence. This, as well as the overall Root Mean Squared Error performance of the Ensemble Network, is presented in Section 4.4.1.3. In section 4.4.2 the root mean squared error of predicted time until cut-ins were calculated for each test cut-in trajectory at each time step. A naive cut-in detection algorithm is developed in Section 4.4.3 and results are compared between predictors. Finally in section 4.4.4 plots of sampled predictions are presented.

4.4.1 Root Mean Squared Error

The network is tested on 1000 test trajectories comprised of 488 cut-ins and 512 passing trajectories. The results are divided into sections regarding the Cut-in Network, the Passing Network, and the Ensemble Network.

4.4.1.1 Cut-In Network

The network was tested on 488 test cut-in trajectories and compared to state based predictors. The resulting average error and standard deviations over a prediction horizon of 5 seconds

can be seen in Figure 4.12. The network begins to out-predict the other predictors at the 2 second mark. This is likely due to the network's ability to capture the entire lane change maneuver in its predictions, whereas the other predictors are constrained to linear behavior.

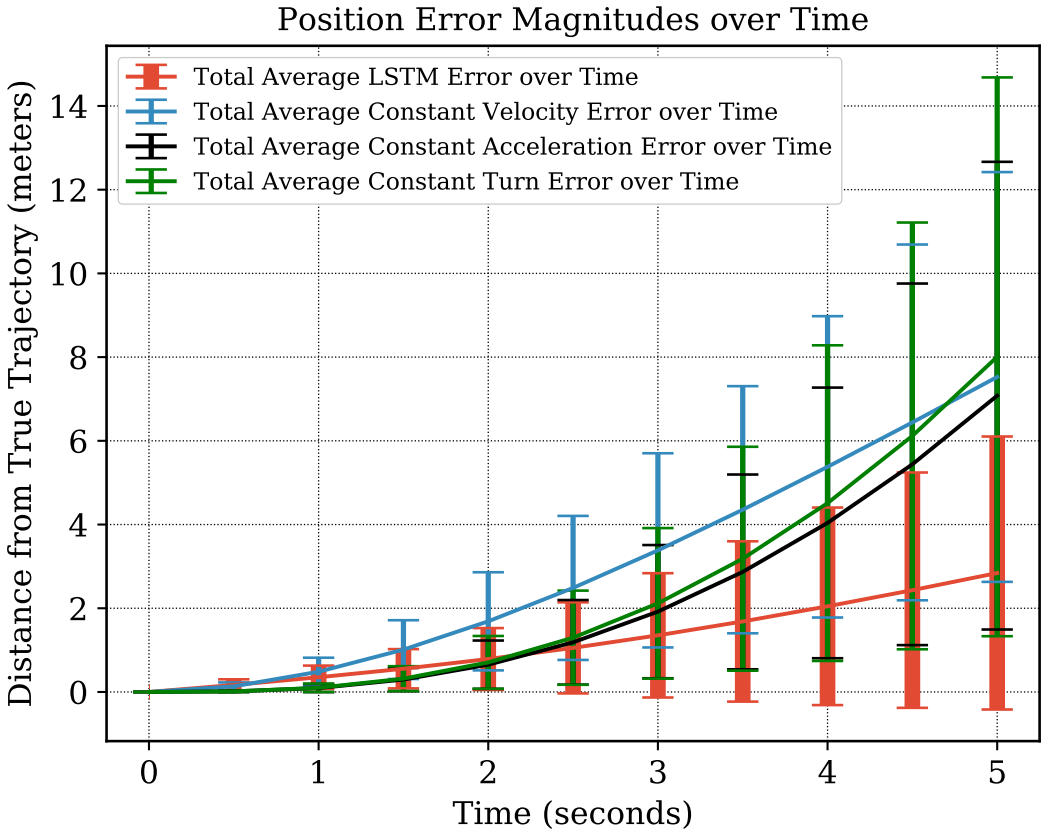


Figure 4.12: Cut-in Network Error on 488 test cut-in trajectories over 5 second prediction horizon

As seen in Table 4.2, the LSTM network significantly outperforms the state-based predictors over the 5 second prediction horizon with a final mean distance to truth of less than 3 meters and an average RMSE close to 1.5 meters. The next best predictor was the Constant Acceleration predictor which had an average final distance to truth of roughly 7 meters and an average RMSE of 2.7 meters.

Table 4.2: Cut-in Predictor Performance over 488 Test Trajectories

Predictor Performance for Cut-Ins Only				
Predictor	***Mean RMSE (m)***	***RMSE Standard Deviation (m)***	***Mean Error at t = 5s (m)***	***Error Standard Deviation at t = 5s (m)***
LSTM	1.496	1.590	2.842	3.261
Constant Velocity	3.739	2.483	7.525	4.895
Constant Accel.	2.914	2.307	7.079	5.587
Constant Turn	3.268	2.711	8.007	6.675

4.4.1.2 Passing Network

The network was tested on 512 test passing trajectories and is compared to state based predictors. The resulting average error and standard deviations over a prediction horizon of 5 seconds can be seen in Figure 4.13.

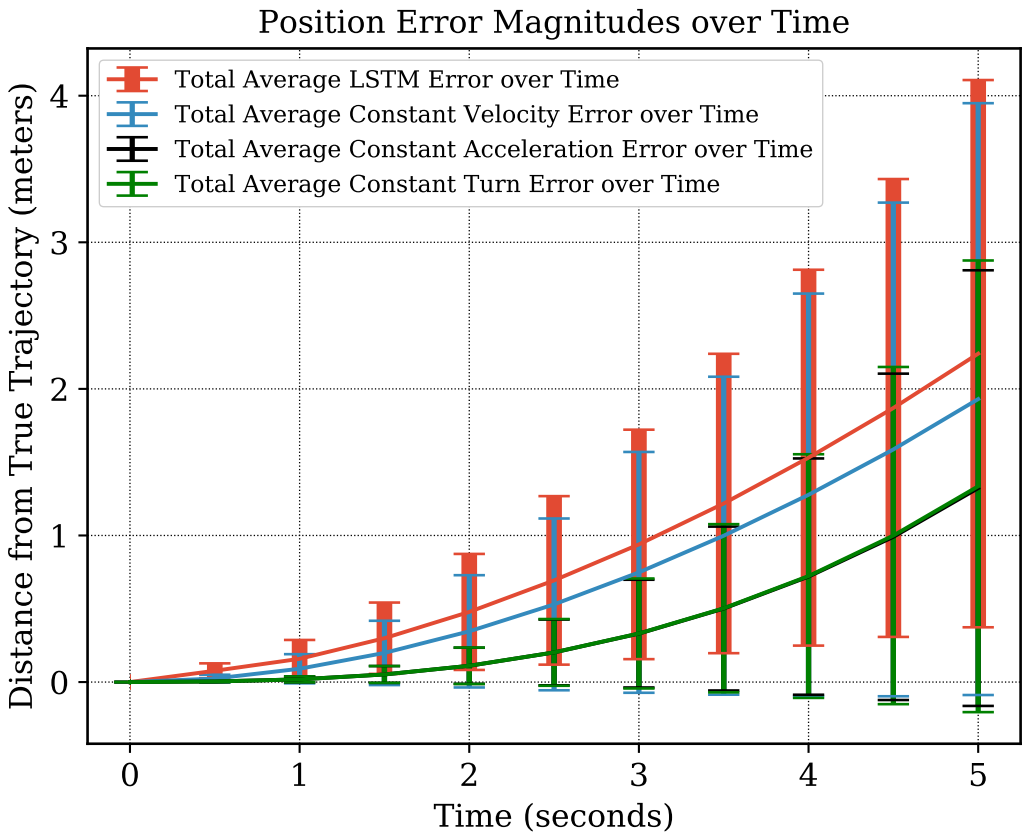


Figure 4.13: Passing Network Error on 512 Test Passing trajectories over 5 second prediction horizon

The LSTM network under performs the state based predictors when predicting straight driving behavior. This can be seen again in Table 4.3.

Table 4.3: Passing Predictor Performance over 512 Test Trajectories

Predictor Performance for Passing Only				
Predictor	***Mean RMSE (m)***	***RMSE Standard Deviation (m)***	***Mean Error at t = 5s (m)***	***Error Standard Deviation at t = 5s (m)***
LSTM	1.090	0.876	2.240	1.867
Constant Velocity	0.902	0.949	1.930	2.019
Constant Accel.	0.530	0.587	1.323	1.486
Constant Turn	0.534	0.606	1.335	1.541

The under-performance of the LSTM in this case can likely be attributed to it over-fitting to unmeaningful maneuvers that the vehicle may be making while passing the trucks. It is worth noting that the overall performance of the Passing Network has better metrics than the Cut-in Network with an average final distance to truth of 2.24 meters and an average RMSE of 1.09 meters as opposed to the Cut-in Network performance of 2.85 meters and 1.5 meters respectively. This suggests that within the constraints of the simplified model of perfectly straight roads and scenarios with only one passing vehicle, the state-based predictors are likely sufficient to predict future behavior of passing vehicles. However, as in the case of the cut-in mode, added complexity to the model such as curved roads or added traffic may produce an environment in which the LSTM may shine.

4.4.1.3 Ensemble Network

The Ensemble Network was tested on the entire test data set of 1000 trajectories. A mode probability calibration chart, shown in Figure 4.14 was used to evaluate the performance of the Ensemble Network probability predictions. The red line in Figure 4.14 corresponds to

the likelihood of predicting the correct trajectory mode versus the network's predicted mode confidence probability. The dashed grey line along the diagonal indicates perfect calibration, meaning that the predicted confidence probabilities match perfectly with their corresponding likelihood of occurrence or accuracy. For example, in perfect calibration a predicted confidence probability of 85 percent attached to a trajectory mode would correspond to an 85 percent likelihood the predicted mode is correct.

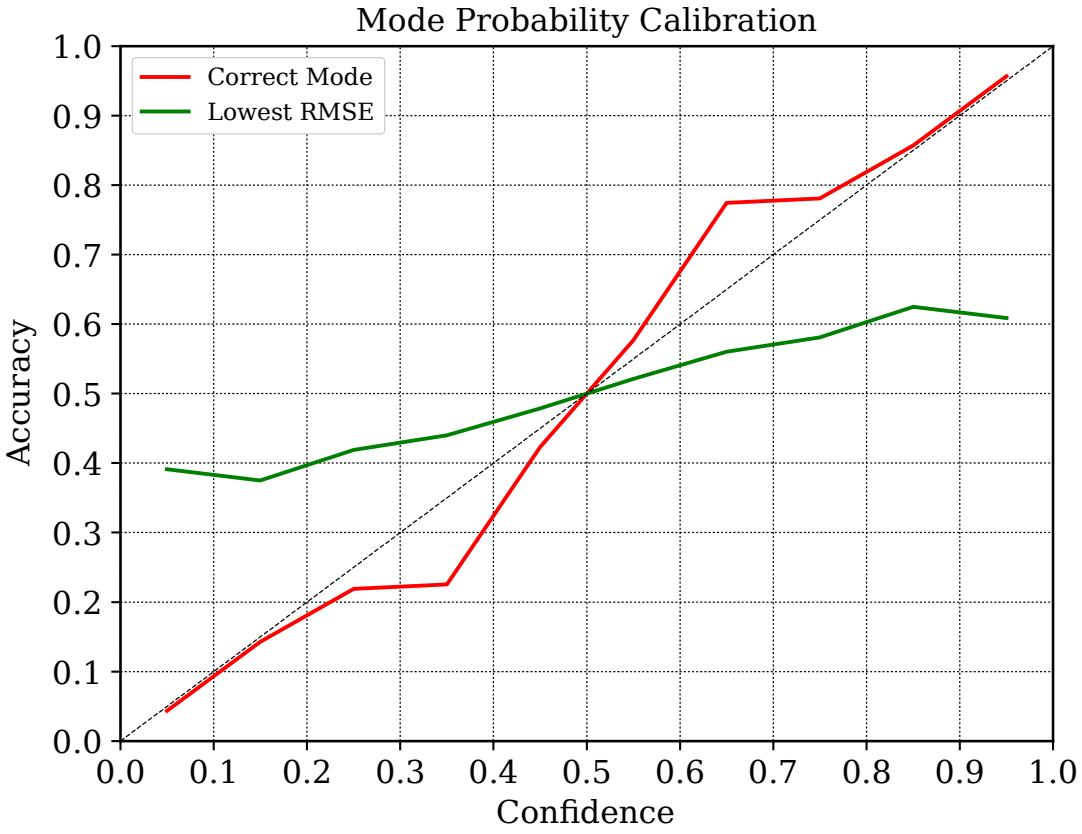


Figure 4.14: Mode Probability Calibration taken from 1000 Test Trajectories

As shown in Figure 4.14, the predicted mode probabilities are trained to be well calibrated, but a little under-confident in the 60% to 70% bucket. Figure 4.15 shows the distribution of predictions.

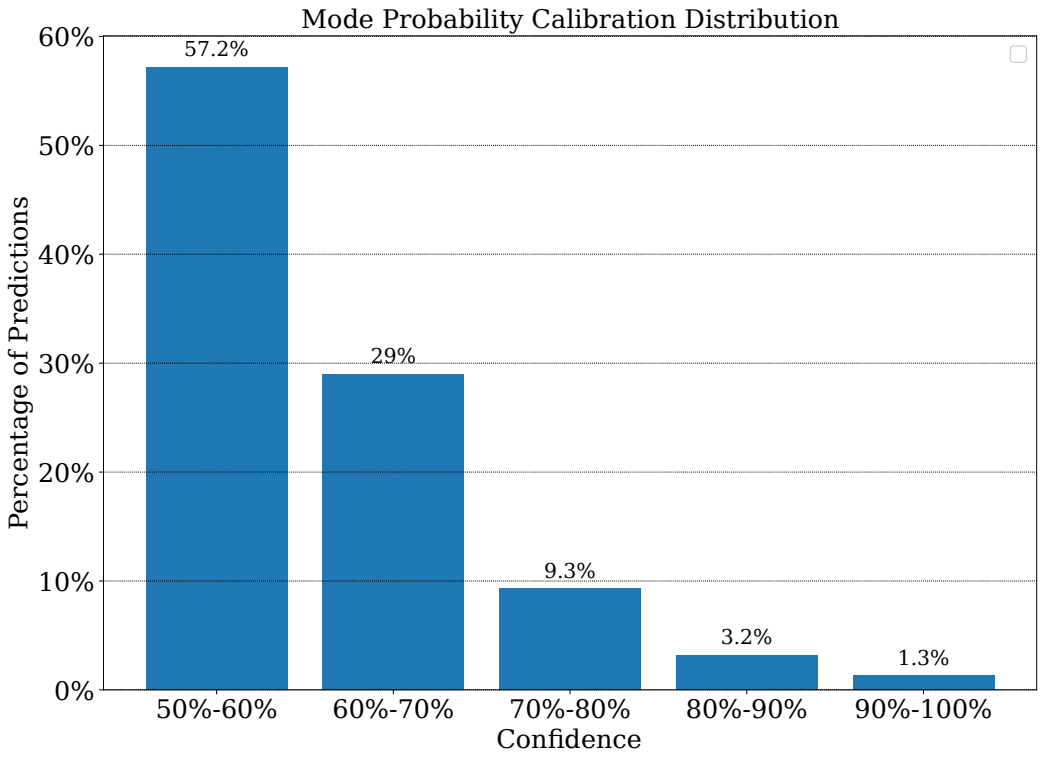


Figure 4.15: Distribution of Predictions by Predicted Probability

The majority of predictions are between 50% and 60%, indicating that the predictions are cautious. This is fine performance, given that the primary design goal of training the ensemble network was to generate well calibrated predictions, even if it came at the cost of confident predictions. In other words, the goal in this work was to ensure that if the network predicts behavior with 95% confidence that it will be correct 95% of the time. In future design iterations more care and attention should be given toward boosting the confidence of the network predictions while maintaining good calibration.

Training the ensemble predictions necessitated adding the tuning parameters α and β in the Cross Entropy Loss function designed in Section 4.3.5 in Equation 4.42. Tweaking these parameters in the loss function during training allowed the trained distribution to be shaped in the desired way, which in this work was to be as confident as possible while maintaining a close to perfect calibration.

The correct mode prediction did not always coincide with the trajectory prediction with the lowest RSME. This is indicated by the green line in Figure 4.14. Although better than a coin

flip, a 95 percent confidence corresponds to only 61 percent likelihood the chosen trajectory will have the lower RMSE. This is because the Cut-in Network may sometimes better predict a passing trajectory and vice-versa, which causes the Ensemble Network RSME to be sub-optimal. One solution to this could be to retrain the network without differencing. This way the networks can better incorporate lane positions into predictions. It is seen in Figure 4.16 and in Table 4.4, however, that the Ensemble Network still out performs the state-based predictors.

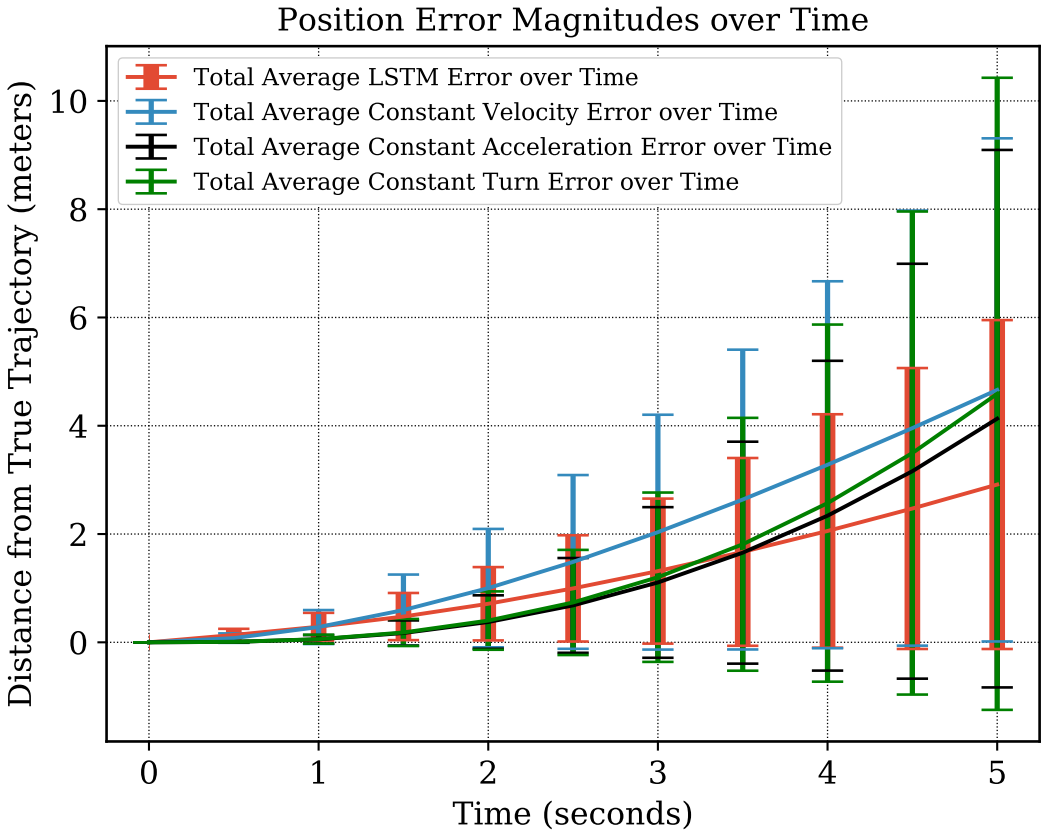


Figure 4.16: Ensemble Network Error on 1000 Test Trajectories over 5 Second Prediction Horizon

Figure 4.16 shows the network outperforms the state-based predictors in both average RMSE and average final distance with lower variance.

Table 4.4: Ensemble Predictor Performance over 1000 Test Trajectories

Predictor Performance for All Trajectories				
Predictor	***Mean RMSE (m)***	***RMSE Standard Deviation (m)***	***Mean Error at t = 5s (m)***	***Error Standard Deviation at t = 5s (m)***
LSTM	1.461	1.439	2.803	2.879
Constant Velocity	2.331	2.393	4.655	4.667
Constant Accel.	1.679	2.051	3.995	4.850
Constant Turn	1.853	2.377	4.444	5.697

4.4.2 Predicted Time to Cut-in

Estimated time to cut-in was calculated for each predictor by propagating the predictions forward and recording the time that the prediction crosses the lane line. The Root Mean Squared Errors were taken from the estimated cut-in times for each predictor at each time step of prediction. This is shown in Figure 4.17. Trajectories that cut in before the 1.25 seconds of data needed to be run through the neural network is collected are padded with zeros and then fed through the network anyway, generating sub-optimal predictions.

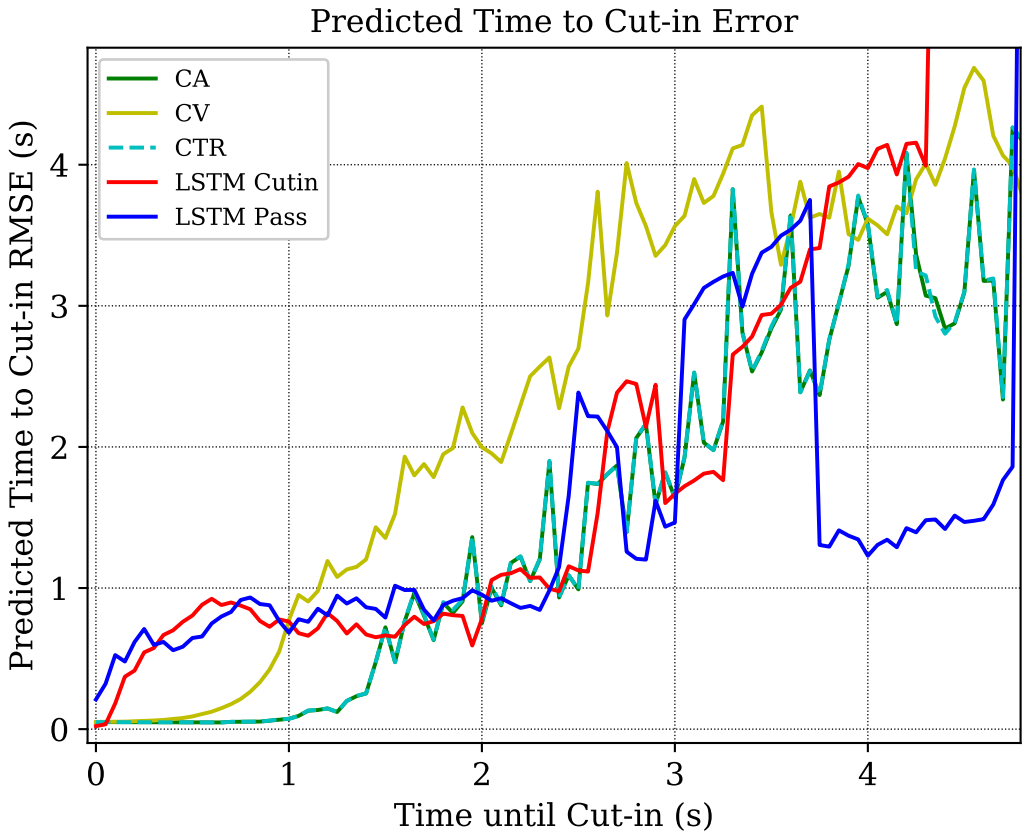


Figure 4.17: RMSE of Predicted Time to Cut In over 241 Cut-in Trajectories

From Figure 4.17 it appears that the network predictions under perform the state-based predictions, especially once the time until cut-in is less than 1 second. This is due to the inaccuracy in the outputs given by the network when it receives incomplete inputs. A solution to this could be to train another network optimized for smaller input vectors, however the performance of the CA and CTR predictors suggest that at best this would provide redundant results. However, as the time horizon increases, the network begins to outperform other predictors. This is shown in Figure 4.18.

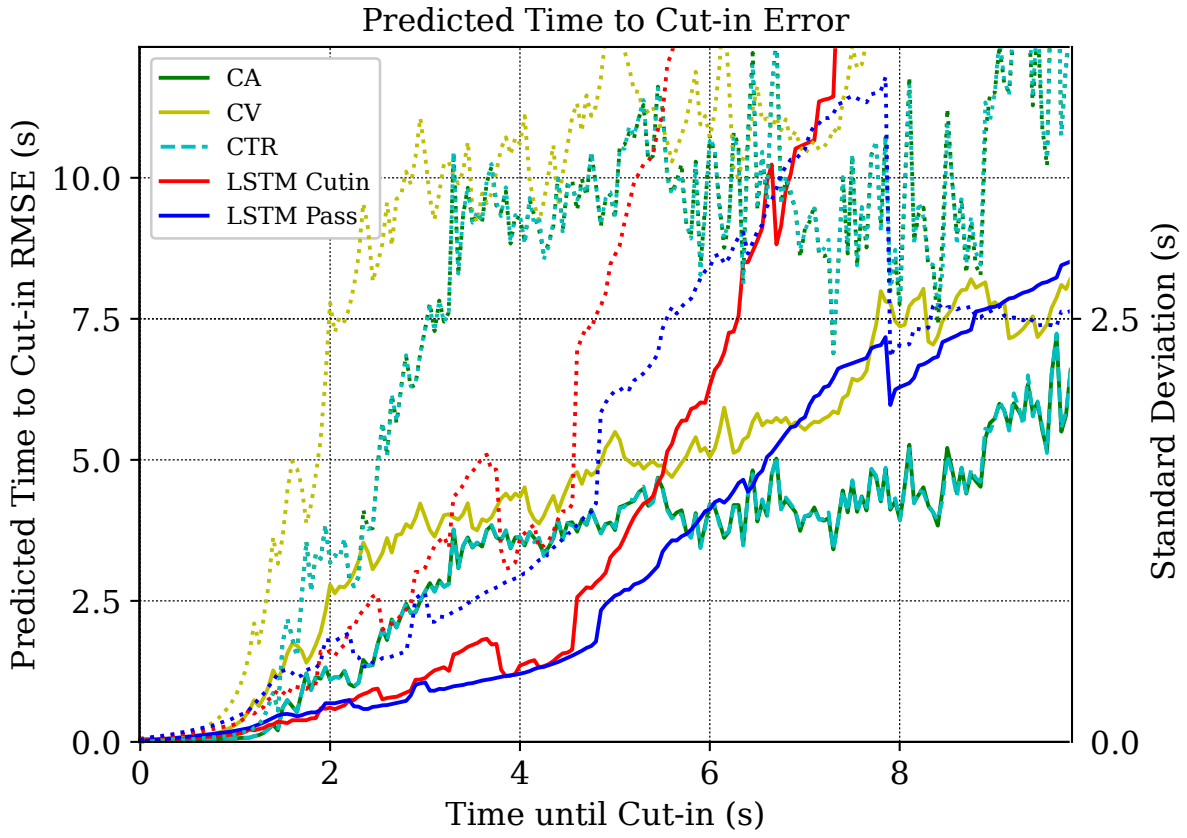


Figure 4.18: RMSE of Predicted Time to Cut In over 409 Cut-in Trajectories with a Minimum 1.25 Seconds Before Cut-in

Figure 4.18 shows that when functioning with trajectories of the full designed input sample size of 1.25 seconds, the network predictions outperform the state-based predictors, particularly between the range of 2 to 5 seconds. Within this time window, the LSTM predictors are able to provide second-level accuracy of predicting cut-in times, being roughly twice as accurate as the state-based predictors while having much lower variances. The LSTM predictors begin to

under perform again around 6 seconds to cut-in, which is likely due to the LSTM predictors only predicting over 5 second horizons. Given that the maximum time the LSTM predictors are designed to predict is 5 seconds, the under performance after this horizon is expected.

Oddly, the Passing Network outperforms the Cut-in Predictor in predicting cut-in times within this window as well. This could be due to the Passing Network being more patient with cut-in predictions, as it has been trained to keep the vehicles within their lane. Given that the cut-in trajectories are filtered to be instances in which the vehicles deliberate in the left lane before merging, having more patience would increasingly benefit time to cut-in predictions as the true time to cut-in increases. Conversely, the eagerness to predict cut-ins benefits the Cut-in Predictor as the true time to cut-in decreases below 2 seconds. This implies that the modal predictions are in fact working as intended, and that training the network to assign confidence values to its predictions can lead to more accurate predictions overall.

Filtering the trajectories to only trajectories with at least 1.25 seconds of data, however, cuts the sample size by about 80 percent. Of 2087 generated cut-in trajectories, only 409 trajectories were long enough for the network to operate to its full potential. One reason for this is due to the narrowness of the radar span. Given that the radar on the follower truck can only view vehicles within a span of 45 degrees in front of the truck, a vehicle has to be roughly 3.5 meters ahead of the follower truck longitudinally before it can be detected and tracked. As such, cut-in vehicles may have very short trajectories before cutting into the platoon.

These results motivate several avenues for practical implementation. 1) Increasing the vision range of the follower truck can provide more time for both the state-based predictors and LSTM predictors to make earlier and more accurate predictions. 2) Given that the state-based predictors perform best in short time horizons, and the LSTM over longer horizons, state-based predictors can be used for the immediate case and then the prediction responsibilities can be switched to the LSTM after a tracking time threshold between 1 to 2 seconds.

4.4.3 Cut-in Detection

A cut-in detection algorithm has been designed to signal to the truck platoon when a cut-in is predicted to occur. The detection algorithm operates by first choosing the predicted trajectory

from the ensemble model with the higher probability, then calculating if the predicted trajectory crosses the Pure Pursuit lane line. If the prediction crosses into the lane of the platoon, the cut-in detector signals a cut-in. A threshold is introduced to prevent chattering and decrease the frequency of false positives (FP) such that a cut-in will not be signalled by the detector until the predictor predicts enough cut-ins in succession to satisfy the set threshold.

To evaluate the detector, network and state-based predictions were run over a test set of 2350 trajectories of which 13% are cut-ins and 87% are passing. Like in Section 4.4.2, trajectories were limited to those long enough to employ the network predictions. Cut-in signals were recorded for each predictor at every time epoch in the tested trajectories. Similarly, the truth mode is recorded at each time stamp using a horizon of 5 seconds into the future. If the true trajectory cuts into the platoon within 5 seconds, the time epochs are labeled at cut-ins. Otherwise they are labeled as passings. This is to reduce the reduction in performance due to the unpredictable randomness of cut-ins that are beyond 5 seconds into the future.

The truth mode of each time stamp in the trajectory is compared to the predicted mode signalled by the detector. Each signal yields either a false positive (FP), a false negative (FN), a true positive (TP), or a true negative (TN). A true positive is recorded if the detector signals a cut-in and the truth signal is a cut-in. Similarly a true negative is recorded if the detector signals a pass and the truth is a pass. A false positive occurs if the detector signals a cut-in and the truth is a pass, and a false negative occurs if the detector signals a pass but the truth is a cut-in. Results for varying thresholds for each predictor are presented below in Tables 4.5 and 4.6.

The metrics used to evaluate the performance of the detector include Balanced Accuracy (BACC), False Positive Rate (FPR), False Negative Rate (FNR), mean and standard deviation of the time the detector signals a cut-in before the vehicle crosses the lane line. Balanced Accuracy is the average between the True Positive Rate (TPR) and the True Negative Rate (TNR). This is given in Equation 4.44.

$$BACC = \frac{TPR + TNR}{2} \tag{4.44}$$

where the True Positive Rate, TPR, is the ratio of correctly classified cut-in signals to all true cut-in signals, and the True Negative Rate, TNR, is the ratio of correctly classified passing signals to all true passing signals. These are given in equations 4.45 and 4.46 as

$$TPR = \frac{TP}{TP + FN} \tag{4.45}$$

and

$$TNR = \frac{TN}{TN + FP}. \tag{4.46}$$

where TP is the total number of true positive signals, TN is the total number of true negative signals, FP is the total number of false positive signals, and FN is the total number of false negative signals.

The False Positive Rate FPR, or false alarm rate, is given as the rate in which the detector predicts a cut-in on a passing trajectory. This is given in equation 4.47.

$$FPR = \frac{FP}{FP + TN} \tag{4.47}$$

The False Negative Rate FNR, or missed detections, are instances in which the detector predicts a vehicle to pass the platoon when it is bound to cut into the platoon. This is given in equation 4.48

$$FNR = \frac{FN}{FN + TP} \tag{4.48}$$

The time until cut-in at detection is determined by calculating the time difference between the first cut-in signal and the instance the vehicle crosses the lane line. The mean and standard deviation are taken of the aggregate of times until cut-in since detection across all true positive detections.

Table 4.5 below shows the detector performance given a truth horizon of 5 seconds and thresholds ranging between 1, representing 0.02 seconds, to 50, representing 1 second.

Table 4.5: Cut-in Detection Performance for Different Thresholds with 5 Second Horizon

Predictor	**Threshold**	***Balanced Accuracy (%)***	***FPR (%)***	***FNR (%)***	***Mean Time until Cut-in at Detection (s)***	***Standard Deviation of Time until Cut-in at Detection (s)***
CA	1	90.34	3.88	15.44	4.40	1.40
	10	89.88	2.96	17.28	4.04	1.37
	20	89.04	2.70	19.23	3.47	1.32
	30	87.97	2.50	21.57	2.94	1.24
	40	86.90	2.33	23.86	2.46	1.11
	50	85.42	2.39	26.77	1.95	1.00
CV	1	83.39	2.05	31.16	3.47	1.27
	10	81.91	1.99	34.20	3.02	1.19
	20	79.92	1.90	38.25	2.53	1.16
	30	77.72	1.83	42.72	2.11	1.15
	40	75.77	1.62	46.84	1.78	1.22
	50	73.34	1.49	51.82	1.41	0.89
CTR	1	90.33	3.86	15.48	4.38	1.38
	10	89.87	2.93	17.34	4.02	1.36
	20	89.02	2.67	19.28	3.46	1.30
	30	87.95	2.48	21.63	2.93	1.23
	40	86.88	2.32	23.93	2.45	1.11
	50	85.40	2.38	26.83	1.95	1.00
LSTM	1	87.61	7.06	17.71	4.29	1.98
	10	87.10	6.37	19.44	3.98	2.13
	20	86.38	6.04	21.19	3.36	2.12
	30	85.50	6.29	22.72	2.88	2.10
	40	84.45	8.35	22.76	2.47	2.65
	50	82.12	9.51	26.26	1.84	2.54

In table 4.5 it can be seen that increasing the threshold tends to decrease the rate of false positives or false alarms for each predictor, but increases the rate of false negatives or missed detections. Intuitively, increasing the threshold also reduces the mean time until cut-in since detection.

Overall the CA predictor performed best, with the CTR and LSTM predictors in second and the CV predictor in last. The LSTM predictor had similar missed detection rates and mean time until cut-in at detection to the CTR and CA predictors, but came up short on the Balanced Accuracy score largely due to its under-performance in its rate in false alarms. This could be due to the composition of the data set the network was trained on. The LSTM Ensemble network was trained on a data set that was roughly comprised evenly between modes, whereas in this test and in real testing the ratio is likely to be heavily skewed in favor of the passing mode. This suggests that the LSTM has learned to expect cut-in behavior more often than it should.

Given that the NCV model is the dynamic model currently being used by the truck platoon in the tracking filter as described in section 5.3.3, running the CV predictor will likely be the easiest to implement. Implementing the CA or CTR predictors instead requires replacing the NCV dynamic model in the track filter with the NCA and NCS dynamic models as given in sections 3.24 and 3.3.2.3. Given the noisy nature of radar measurements, it may be difficult to implement the CA model due to the amplification of noise in range acceleration estimates, but this has yet to be tested. For these reasons, the CV predictor is the recommended predictor for immediate term cut-in detection implementation onto the platoon, but developing the CA and CTR predictors for use in cut-in detection is also recommended. It is also worth noting that the CV predictor comes with the lowest rates of false alarms, which however comes at the cost of a higher rate of missed detections than the other predictors.

Table 4.6 below shows the same test performed with a 2 second truth horizon. In other words, the ”truth” signal on cut-in trajectories is set to be a pass until 2 seconds before the vehicle crosses the lane line.

Table 4.6: Cut-in Detection Performance for Different Thresholds with 2 Second Search Horizon

Predictor	**Threshold**	***Balanced Accuracy (%)***	***FPR (%)***	***FNR (%)***	***Mean Time until Cut-in at Detection (s)***	***Standard Deviation of Time until Cut-in at Detection (s)***
CA	1	95.41	8.29	0.88	4.40	1.40
	10	95.38	8.01	1.22	4.04	1.37
	20	94.81	8.70	1.67	3.47	1.32
	30	93.62	9.83	2.93	2.94	1.24
CV	1	96.20	5.04	2.57	3.47	1.27
	10	95.66	5.39	3.29	3.02	1.19
	20	94.78	5.94	4.50	2.53	1.16
	30	93.21	6.81	6.77	2.11	1.15
CTR	1	95.42	8.27	0.88	4.38	1.38
	10	95.40	7.98	1.22	4.02	1.36
	20	94.83	8.67	1.67	3.46	1.30
	30	93.63	9.81	2.93	2.93	1.23
LSTM	1	93.36	11.85	1.44	4.29	1.98
	10	93.19	11.93	1.70	3.98	2.13
	20	92.37	12.77	2.49	3.36	2.12
	30	90.85	14.73	3.57	2.88	2.10

Once the horizon is reduced to 2 seconds, the rate of missed detections drops significantly for all predictors. This is intuitive, as the predictors should perform better at accurately predicting cut-ins if the time frame is shorter. Perhaps unexpectedly, however, the rate of false alarms rises across detectors. This is likely due to the mean time until cut-in since detection being greater than 2 seconds for all predictors. This means the the predictor may be correctly classifying a cut-in beyond 2 seconds out, but it is showing up as a false positive because the truth signal isn't set to 'cut-in' yet.

It looks from these results that the LSTM is inferior to the state-based predictors. After all, why train a sophisticated learning algorithm if a simple CV or CA model can do the job? There are two answers to this. First, as is seen in Section 4.4.4, the LSTM has the ability to predict the full dynamics of a lane change maneuver whereas the state-based predictors do not. This is useful in that the learning algorithm may theoretically perform better in a more dynamic setting with more modal behavior, such as introducing more lanes and off ramps. Additionally, as in [21], deep neural networks can take advantage of additional information such as road maps and visual indicators such as detecting blinkers [80], although this additional information is not used in this implementation.

Second, keeping in mind the Active Learning Pipleine outlined in section 2.2.5, this LSTM design is just a first pass design and there are potentially many other network architectures, network experience sets, and performance measures that may result in a better performing network. In other words, the LSTM has room to improve from here, whereas the performance of the state-based predictors is fixed. This goes without saying the performance of the state-based predictors is surprisingly good, and that they are significantly easier to implement and understand than an LSTM model. The result in real world implementation could be to start by using the state-based predictors, and if improvement in prediction is desired a network can be designed to fit the job.

4.4.4 Sampled Predictions

This section gives a look at several simulation results. Figure 4.19 shows a sample cut-in trajectory and its predictions. It can be seen that while the state-based predictors capture

the vehicle crossing the lane, they also predict the vehicle running off the road. The LSTM predictions, however, predict the rest of the lane change maneuver.

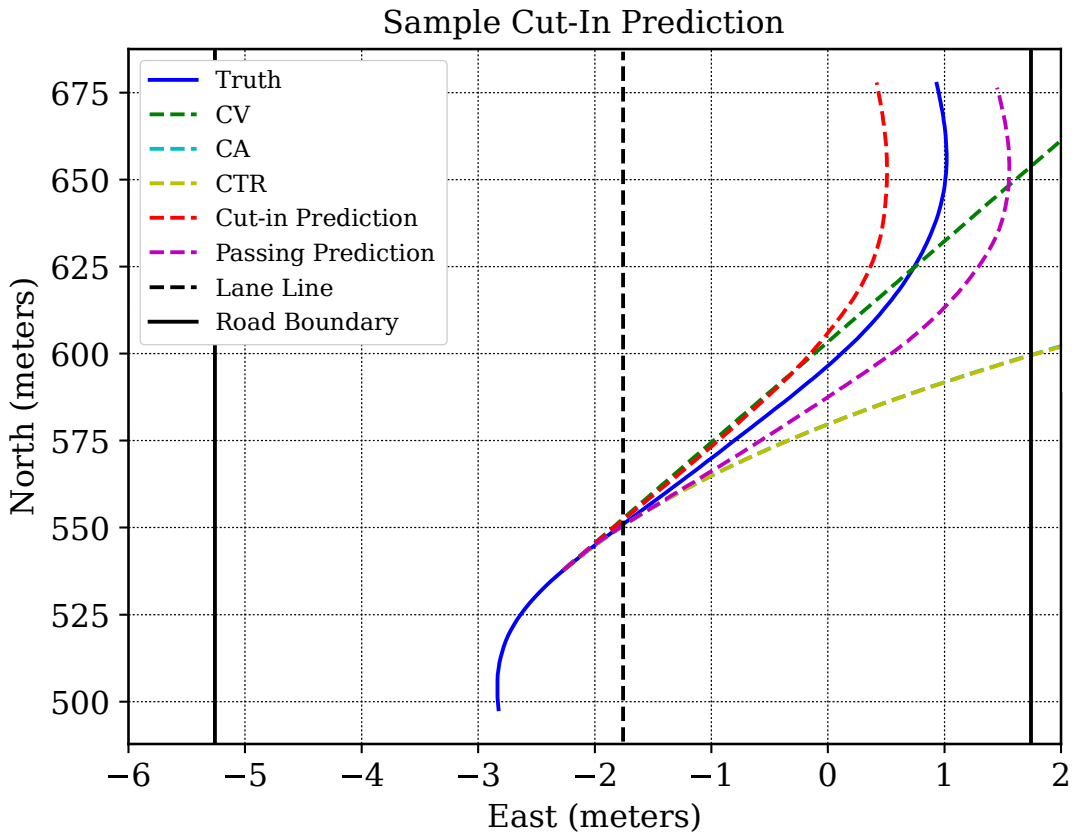


Figure 4.19: Sample Cut-in Mode Trajectory Prediction with 5 Second Horizon

Table 4.7 shows the performance metrics of the predictions in Figure 4.19. It can be seen that the network correctly predicts the behavior to be a cut-in. Additionally, the cut-in prediction has a final distance to the truth of less than a meter after the 5 second prediction horizon.

It is also interesting that the Passing Network captures the typical cut-in behavior without being trained to predict cut-ins. This is likely due to the training trajectories always being oriented north, so the network learns to steer trajectories to head north. This property could be useful when implementing the LSTM predictions on empirical data in Chapter 5, as all empirical trajectories can be rotated to be oriented in the same cardinal direction, acting to give the LSTM an indication of where the road is headed.

Table 4.7: Sample Cut-in Mode Prediction Performance

Predictor Performance			
Predictor	***Probability***	***RMSE (m)***	***Error at t = 5s (m)***
Cut-in Network	0.628	7.524	0.612
Passing Network	0.372	15.52	1.243
Constant Velocity	–	67.29	13.52
Constant Accel.	–	66.43	15.93
Constant Turn	–	73.80	17.89

Figure 4.20 shows a sample passing trajectory and its predictions, and Table 4.8 shows the corresponding performance metrics. Although the state-based predictors have lower RMSE and final distance values, they run the vehicle off the road.

The Ensemble Network correctly chose the passing mode prediction, which has a final distance to the truth of 2 meters after the 5 second prediction horizon and ends with the vehicle remaining on the road. The passing mode also ends with the correct heading, indicating that it may be a more useful prediction than those provided by the state-based predictors even if it has a higher RMSE.

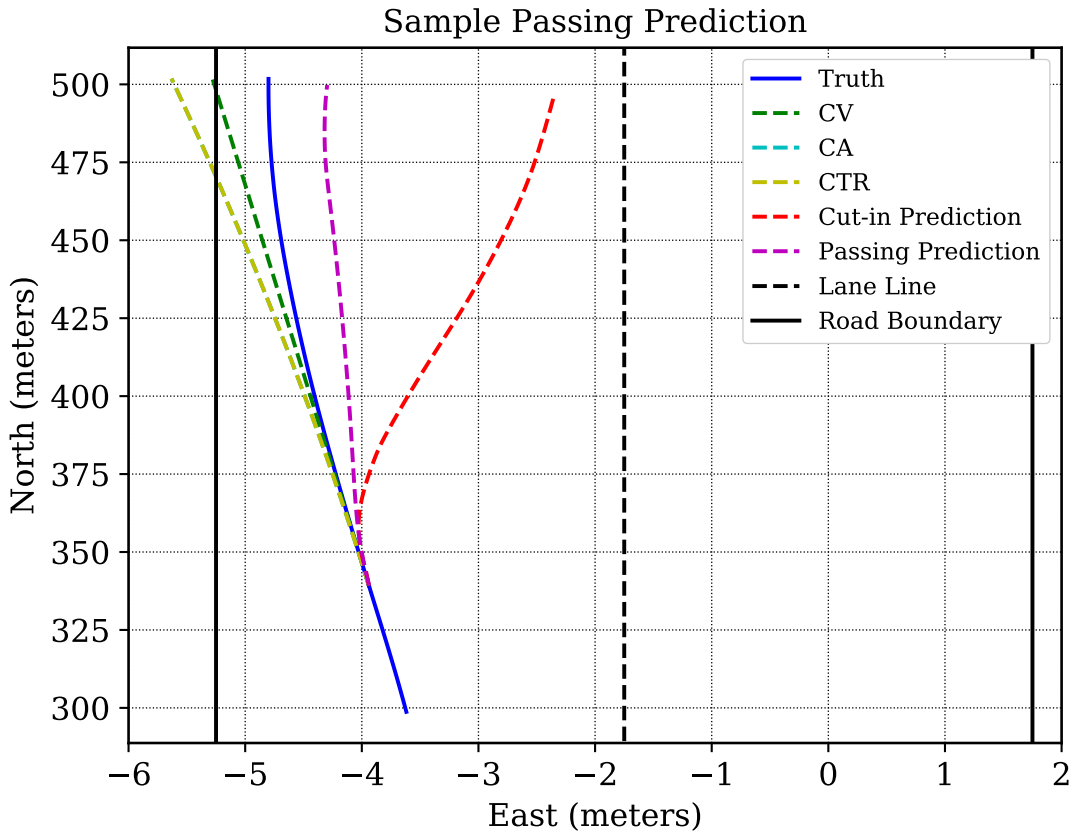


Figure 4.20: Sample Passing Mode Trajectory Prediction with 5 Second Horizon

Table 4.8: Sample Passing Mode Prediction Performance

Predictor Performance			
Predictor	***Probability***	***RMSE (m)***	***Error at t = 5s (m)***
Cut-in Network	0.105	25.93	5.89
Passing Network	0.894	9.54	2.024
Constant Velocity	–	2.153	0.517
Constant Accel.	–	3.488	0.837
Constant Turn	–	3.487	0.837

4.5 Conclusion and Discussion

This work has demonstrated the viability of using deep learning methods to predict behavior of vehicles around truck platoons in a simplified approach. The LSTM method successfully predicts the behavior of cut-in vehicles and is able to accurately assign probabilities between modes. The resulting Ensemble Model predicts future vehicle behavior with a mean RMSE of 1.461 meters with a standard deviation of 1.439 meters. A cut-in detector was designed and tested over several thousand test trajectories using each predictor. The state-based models overall outperformed the network, although the network has room to improve performance in subsequent design passes.

The Ensemble network was particularly good at predicting the behavior of cut-in trajectories, outperforming the state-based predictions as shown in 4.12. It struggled to beat the state-based predictors at predicting passing trajectories, leading to mixed RMSE results. This is certainly an area of needed improvement for the network, as it will lead to better results in the cut-in detection algorithm. The Ensemble was successfully trained to predict which predicted

trajectory was more likely to occur in a well calibrated fashion, however it remained very cautious with most of its probability predictions ranging between 50 and 60 percent. Therefore another area of significant improvement for the network going forward would be to better train the network to maintain good calibration while making higher confident predictions. Improving both of these weak areas in the network may lead to out performance in cut-in detection in future iterations.

Additional improvements to the network in future passes can come in the design of the experience distribution. It may be desirable to adapt the simulation to a high fidelity environment like CARLA in which network performance can be trained and tested in a more complex environment. Collecting a real data set of trajectories of vehicles next to the platoon in highway settings such as is done in Chapter 5 would also be useful to provide training and test sets in future iterations.

Beyond the LSTM Ensemble network, the state-based models have been shown to provide good predictions, especially in the short term. These models are also significantly easier to implement than the LSTM network, making them the recommended first prediction models to try for cut-in detection on the real truck platoon. If better performance is desired, a sophisticated network such as the one herein can then be designed.

5

Experimental Validation of Simulation Trained Neural Network

5.1 Introduction

This chapter aims to build off of the results given in Chapter 4 by adapting the LSTM and state based predictors to an empirical data set collected on the Auburn truck platoon. First, a description of the data collection hardware setup will be provided. Then, the processes of Detection and Tracking of neighboring vehicles will be discussed. Next, the filtered data set will be shown and described. Lastly the network implementation will be detailed and results will be given.

5.2 Data Collection

The data collected for this study came from runs of the Auburn Truck Platoon along interstate 85 near Auburn, Alabama. During this span of highway, the road has two lanes on each side, separated by a wide median, with a speed limit of 70 miles per hour and a nominal lane width of 3.6 meters.

5.2.1 Truck Platoon Setup

The platoon, comprised of two Peterbilt 579 and two military Freighliner M915 trucks, can be seen in Figure 5.1. These trucks are equipped with software developed by Auburn University to allow for autonomous longitudinal control capabilities on each truck. The software was written in Python and C++ and implemented on the trucks with the Robotic Operating System (ROS). The trucks employ a sensor suite including Novatel GPS, Dedicated Short Range

Communication (DSRC) Radios, Delphi ESR Radar, and a Nuvo-509GC computer. The DSRC Radios provide communication between trucks, relaying GPS pseudorange and carrier phase measurements, and the truck's states including velocity, acceleration, and brake status. The Delphi radars on the follower trucks have a wide span of 45 degrees with a range of 60 meters, a narrow span of 15 degrees with a range of 120 meters, and an update frequency of 20 Hz [77] [23].

Figure 5.1: Auburn University Truck Platoon

Together, the range measurements from the radar on the follower trucks and the GPS positioning gathered on the lead trucks are used to estimate a relative position vector between each follower and lead in the platoon. This relative position vector is updated with measurements from Dynamic Base Real Time Kinematic positioning (DRTK), which uses differential GPS positioning between truck receivers to provide cm level relative positioning accuracy between trucks without a base station [74]. DRTK operates at a frequency of 2 Hz. In between DRTK measurements, the relative position vector estimate is updated with the radar range measurements taken at 20 Hz.

5.3 Detection and Tracking of Neighboring Vehicles

The detection and tracking of neighboring vehicles was developed and written by Dan Pierce but has not yet been documented, thus this section will take the liberty of describing

his work. The neighboring vehicle detection and tracking solution uses radar returns as well as DTRK provided solutions of relative positioning to the lead truck. Given knowledge of the relative position of the lead truck, lanes can be drawn using Pure Pursuit between the follower and lead truck. These lanes serve to filter radar points of interest, which are then clustered before being tracked via a NCV Kalman Filter. These methods will be further described below.

5.3.1 Pure Pursuit Lane Drawing

The existing lane estimation solution employs a Pure Pursuit model to estimate the road curvature between the lead and follower truck. This solution assumes the trucks are within the same lane, that the follower and lead trucks are at or near the center of the lane, and that the trucks are close enough to one another such that the road space between them only curves in one direction. Violating these assumptions may causes the Pure Pursuit lane drawing solution to inaccurately estimate lane boundaries [77]. In future work, finding more robust lane estimation solutions is certainly a must; however, this solution works well for the highway runs used in this study.

For a given range, R, and azimuth, θ, estimate of the lead truck from the follower truck, the relative position vector is calculated to be

$$\begin{bmatrix} y \\ x \end{bmatrix} = R \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix}. \tag{5.1}$$

Where y and x are the cartesian components of the relative position vector. The path curvature C between the trucks is then calculated as

$$C = \frac{2y}{x^2 + y^2} \tag{5.2}$$

This curvature forms the estimate of the center of the lane between the follower and lead truck. This estimate is used to constrain radar returns to the road as is shown in Section 5.3.2.

5.3.2 Road Constraining

To constrain radar returns to the road, the relative position of the tracked points are projected to a circle, using the estimated road curvature calculated above. This is given such that

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{C^{-1}}{\sqrt{y^2 + (x - C^{-1})^2}} \begin{bmatrix} y \\ x - C^{-1} \end{bmatrix} + \begin{bmatrix} 0 \\ C^{-1} \end{bmatrix}, \tag{5.3}$$

and

$$\begin{bmatrix} x_{path} \\ y_{path} \end{bmatrix} = \begin{bmatrix} y' \\ x - x' \end{bmatrix}. \tag{5.4}$$

The resulting x_{path} and y_{path} are the coordinates of the tracked radar returns with respect to the circular lane path drawn by Pure Pursuit. x_{path} is the longitudinal distance the tracked return along the lane path, and y_{path} is the lateral distance of the tracked return from the lane path. These tracks are constrained such that x_{path} is less than the distance of the nose of the leader truck and y_{path} is within $\frac{3}{2}$ lane widths from the path. Tracks that do not satisfy these constraints are thrown out, while the surviving tracks are then clustered into a single point per vehicle.

5.3.3 Kalman Filtering

Filtered track measurements from the radar are fed into a Nearly Constant Velocity Kalman Filter with continuous Wiener process noise. The Kalman Filter and Covariance Matrix are given as

$$\underbrace{\begin{bmatrix} r_{k+1} \\ \dot{r}_{k+1} \\ \theta_{k+1} \end{bmatrix}}_{x_{k+1}} = \underbrace{\begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{A_k} \underbrace{\begin{bmatrix} r_k \\ \dot{r}_k \\ \theta_k \end{bmatrix}}_{x_k} + \underbrace{\begin{bmatrix} \frac{1}{2\sqrt{3}} dt^{3/2} & \frac{1}{2} dt^{3/2} \\ 0 & \sqrt{dt} \end{bmatrix}}_{B_k} v_k \tag{5.5}$$

Where r and $\dot{r}$ are estimates of range and range rate, and θ is the estimated azimuth angle between the follower truck and the tracked vehicle.

$$Q_k = \begin{bmatrix} \tilde{q}_1 & 0 & 0 \\ 0 & \tilde{q}_1 & 0 \\ 0 & 0 & \tilde{q}_2 \end{bmatrix} \tag{5.6}$$

where $\tilde{q}_1$ is the variance of the range acceleration given by the radar, and $\tilde{q}_2$ is the square of the variance of the bearing rate. These are set such that

$$\begin{aligned} \tilde{q}_1 &= \sigma^2_{RangeAcceleration} = 0.25 \tfrac{m}{s^2} \\ \tilde{q}_2 &= \sigma^2_{BearingRate} = 0.1 \tfrac{rad}{s} \end{aligned} \tag{5.7}$$

The process noise covariance matrix for the system can be given as

$$B_k Q_k B_k^T = \begin{bmatrix} \frac{1}{3}dt_k^3 & \frac{1}{2}dt_k^2 & 0 \\ \frac{1}{2}dt_k^2 & dt & 0 \\ 0 & 0 & dt \end{bmatrix} \begin{bmatrix} \tilde{q}_1 & 0 & 0 \\ 0 & \tilde{q}_1 & 0 \\ 0 & 0 & \tilde{q}_2 \end{bmatrix} \tag{5.8}$$

This model is run through the Kalman Filter given by Equation 3.28 at a frequency of 20 Hz to provide NCV Filtered range, range rate, and bearing for each neighboring vehicle. Once measurements are no longer received for a tracked vehicle, it is removed from the stack of tracked vehicles.

5.4 Data set

Usually the truck platoon takes place on the NCAT test track in a closed environment, but sometimes runs are recorded on public highways. As stated earlier, the data collected for this study was taken from platoon runs on interstate 85 near Auburn, Alabama. A map of a data collection run is shown in Figure 5.2.

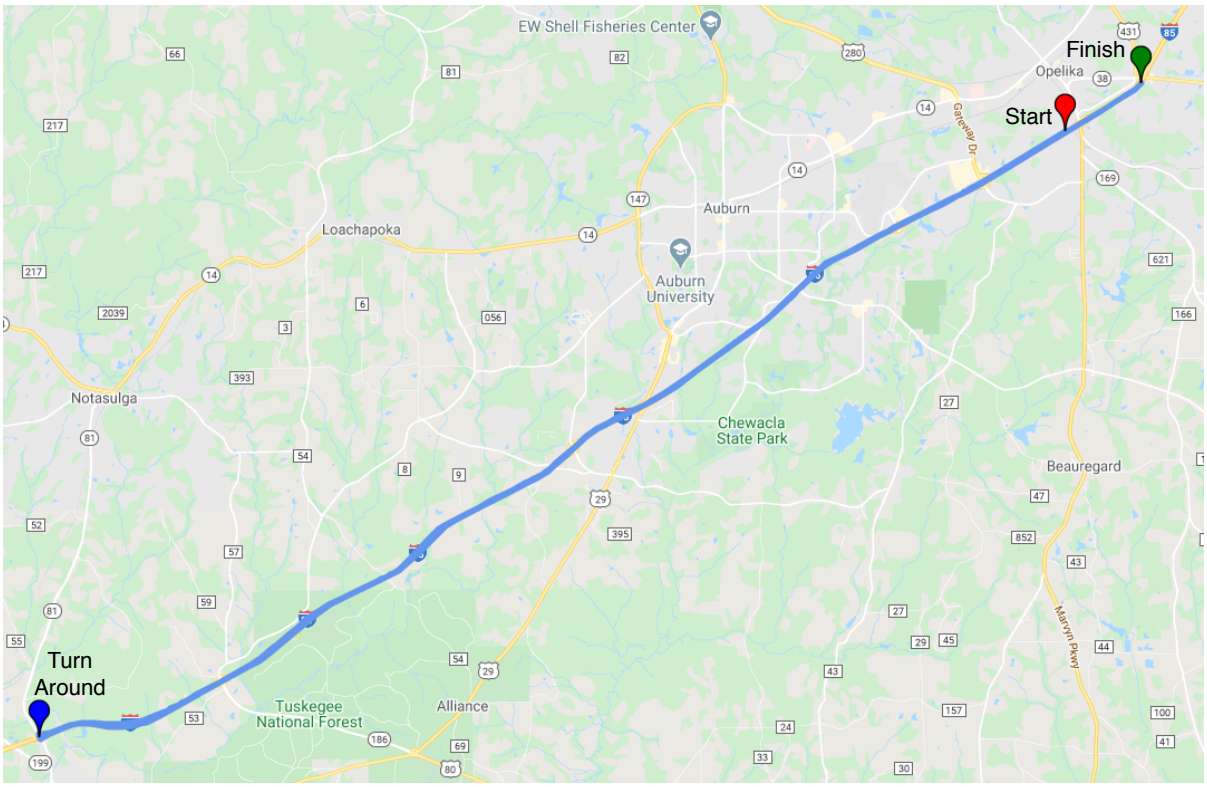


Figure 5.2: GPS Track of Data Collection Run along Interstate 85

5.4.1 Trajectory Visualization

As described above, radar returns on the follower truck were filtered for points that are dynamic and constrained within the "road" drawn by Pure Pursuit. Points satisfying these constraints were filtered through a Kalman Filter using a Nearly Constant Velocity and continuous Weiner process noise model. After removing tracks that are shorter than 2.5 seconds, the resulting positions of neighboring vehicles are shown in Figure 5.3 in the road frame. The trajectories are divided into cut-in and passing trajectories where cut-in trajectories are determined as tracks that crossed the Pure Pursuit lane boundary.

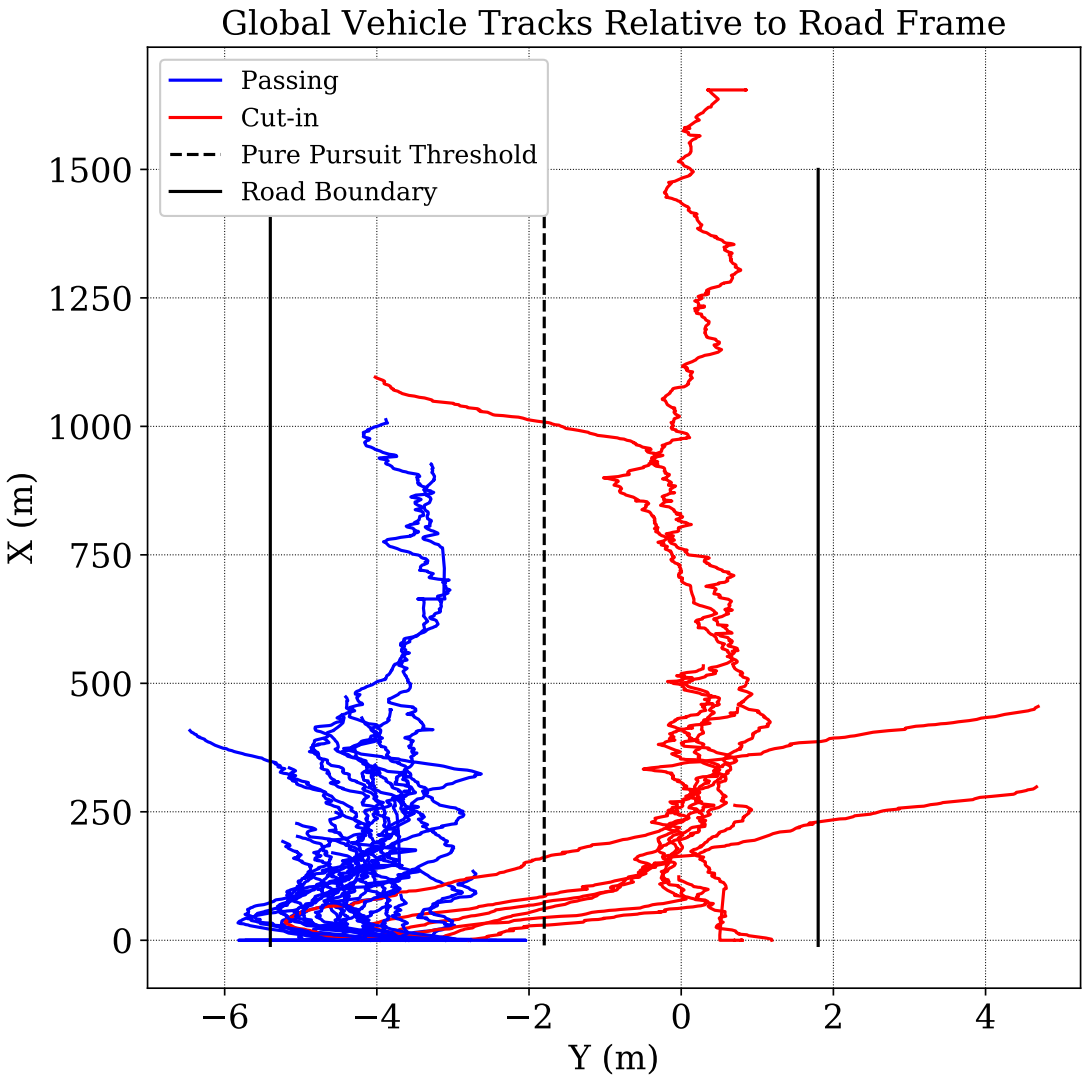


Figure 5.3: Tracked vehicle trajectories from the 10/01/2019 interstate 85 run using the NCV Filter

From Figure 5.3 it can be seen that passing trajectories tend to be around 4 meters to the left of the follower truck. Two vehicles merging off the highway is also observed, as there are two trajectories of which drivers merged into the right lane in between the truck platoon before merging into a right hand exit lane. The trajectories resemble those plotted in 4.10 but with noise added.

5.4.2 Cut-in Trajectories

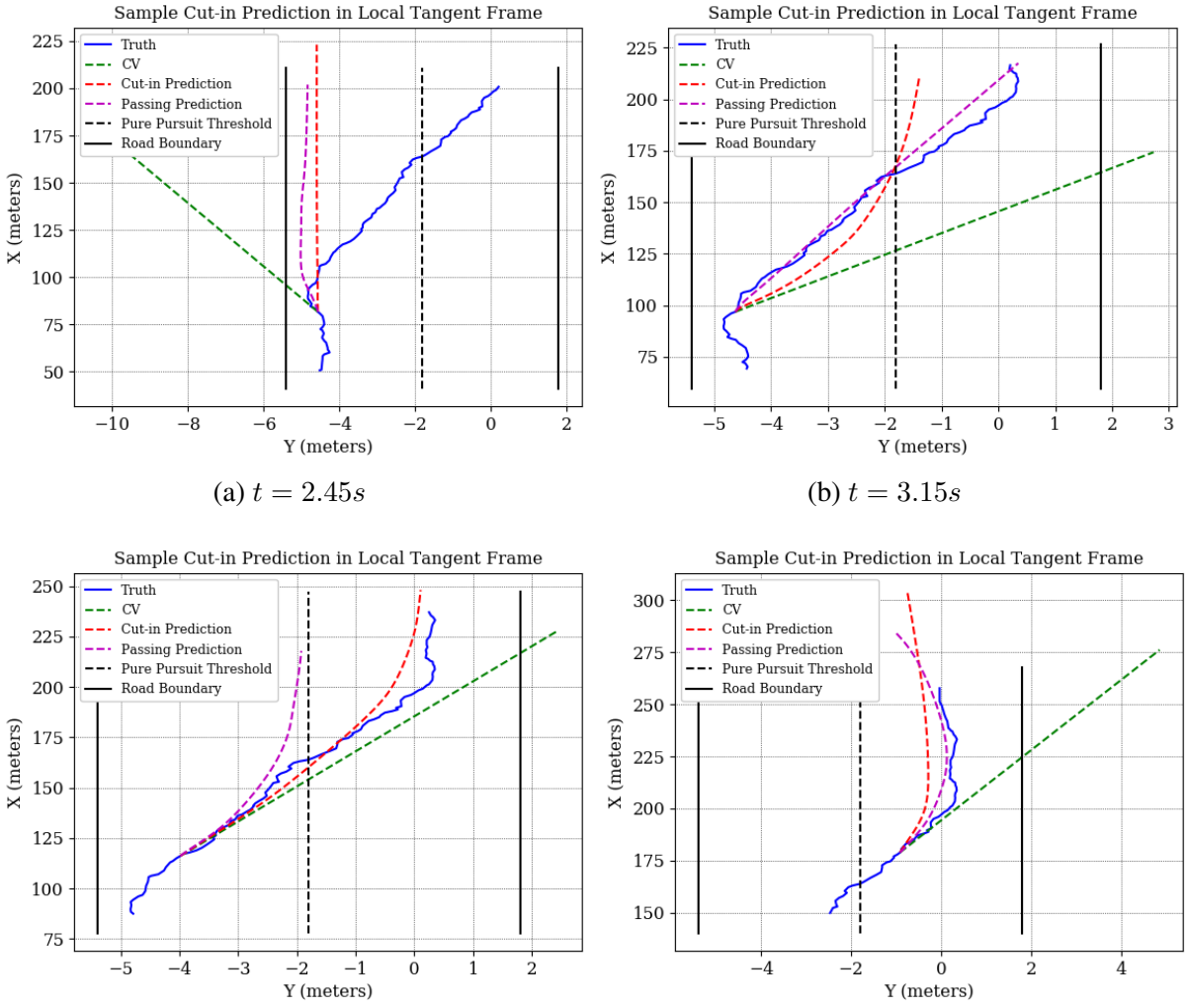
There were six observed cut-ins, with a minimum time until cut-in of 1.800 seconds and a maximum time until cut-in of 5.5 seconds. The mean and standard deviation were 3.753 seconds and 1.950 seconds respectively. This suggests the network designed in Chapter 4 with an input of 1.25 seconds and output of 5 seconds may be suitable to model cut-ins, as the minimum cut-in time observed is greater than the network input time by 0.55 seconds.

Table 5.1: Estimated Times until Cut-in

Time Until Cut-in			
Cut-in 1	2.1328 (s)	Cut-in 4	7.204 (s)
Cut-in 2	2.7365 (s)	Cut-in 5	1.800 (s)
Cut-in 3	3.1425 (s)	Cut-in 6	5.504 (s)
Mean	3.753 (s)	Standard Deviation	1.950 (s)

5.5 Network Results

The ensemble network performance was tested on the empirical data obtained from the truck platoon by comparing predicted trajectories from the LSTM and CV predictors to actual behavior. Running the network trained and tested in Chapter 4 on a cut-in trajectory shown in Figure 5.3 gives promising results. A sequence of predictions across a single cut-in are given in Figure 5.4.



(a) $t = 2.45s$ (b) $t = 3.15s$

(c) $t = 3.95s$ (d) $t = 6.65s$

Figure 5.4: Network Predictions on a Cut-in Trajectory at Different Time Epochs

Figure 5.4 shows the network predictions on an observed cut-in trajectory. The blue line represents the Kalman Filtered estimates of the vehicle's positions in the road frame. The network begins predicting after observing 1.25 seconds of position estimates, and predictions are shown at various time stamps since the vehicle was tracked.

Figure 5.4a shows the network predictions after the vehicle has been tracked for 2.45 seconds. Despite the short term trend guiding the CV predictor off the road, the network is able to predict from the previous time steps that the vehicle is going to continue along the road. In Figure 5.4b, the network sees an additional 0.6 seconds of data and sees the shift in velocity trend of the neighboring vehicle. The Cut-in predictor responds by beginning to predict cut-in behavior, and, oddly, the passing predictor predicts the vehicle to continue on a straight path. This could potentially be due to the network being confused by observing noise it hasn't seen before.

Figure 5.4c shows predictions closer to what is expected from the predictors. The cut-in network predicts the cut-in behavior rather accurately, while the passing predictor attempts to turn the vehicle back into its lane. Finally, Figure 5.4d shows the predictor performance as the vehicle completes its merging maneuver. It is shown that the Cut-in predictor recognizes the cut-in maneuver is almost complete and predicts that the vehicle will head north along the road again. In this plot it appears the cut-in and passing predictions both predict the vehicle to travel further than it actually does, but in this case the vehicle track ended sooner than the 5 second prediction horizon, probably due to the vehicle leaving radar range.

5.6 Conclusion and Discussion

This chapter has demonstrated that the principles discussed and applied in Chapters 2 through 4 can be applied to real vehicle trajectories surrounding the truck platoon. Rotating each trajectory into the Normal-Tangent frame about the Pure Pursuit lane line yields trajectories that are generally confined to two straight lanes, as seen in Figure 5.3. This rotation allows the network and state-based predictors to observe two degrees of freedom for the vehicle, distance along the Pure Pursuit track and distance radially from it, making predictions easier. The

network was run on the data set of real trajectories and was able to recognize and predict cut-in behavior, suggesting that the network is able to learn from ideal experiences designed in Chapter 4 to potentially predict real traffic behavior.

There are many improvements to be made to the network predictions. First, the robustness of predictions can be improved. This can come from two avenues: first the simulation environment can be improved to better approximate the distribution of vehicle trajectories observed on the truck platoon, or better, a cohesive data base of neighboring vehicle radar returns can be gathered from current and future runs of the truck platoon to form a training and test set that can be used to retrain the network. Additionally, the track filters used to estimate the vehicle positions can be extended to predictors, and thus the state-based predictors can be tested on the empirical set in the same manner as was done in Chapter 4.

6

Conclusions

6.1 Summary

Chapter 2 introduced the overarching concepts in Machine Learning and honed in on principles in Deep Learning, a subset of Machine Learning in which neural networks are stacked with many layers. The Universal Approximation Theorem and the idea of "No Free Lunch" were introduced in section 2.2.4. In this theorem lies the key to Machine Learning success. First is to know what task is needed to be performed. Many potential tasks were outlined in the chapter, with the tasks herein to 1) predict future positions using regression, and 2) classify which prediction was most likely to be correct using classification. Second is to choose a machine learning algorithm well suited for the problem at hand. This thesis chose to use an LSTM to perform these tasks due to its success in prediction sequences such as handwriting, basketball shot trajectories, and image captioning. Third is to design relevant experiences for the network to observe, which is further described in Chapter 4. These steps, coupled with testing and validation of the network to identify areas needing improvement formulates one pass of the Active Learning Pipeline, as described in Section 2.2.5. As a whole, this thesis represents one pass of Active Learning Pipeline, and therefore aims to lend plenty of insight into future passes.

Chapter 3 introduces the field of Time Series Forecasting, particularly how it pertains to the prediction of neighboring vehicles around a truck platoon. Several different methods of

predicting vehicle positions are presented including state-based and stochastic methods, goal-based methods, and machine learning methods. Within the state-based methods are the Constant Velocity, Constant Acceleration, and Constant Turn Radius predictors. These predictors served as the base-line for results in Chapter 3, and also formulate dynamics models for common stochastic track models as is described in section 3.3. Several controllers for goal-based predictors were presented, and then related work of other machine learning algorithms used in Time Series Forecasting applications was presented. Practical considerations for LSTM implementation were then discussed, with careful attention to the process of choosing features and preprocessing procedures for training data.

Chapter 4 details the modeling and simulation of neighboring vehicles around a moving truck platoon, and provides network prediction results in the simulated environment. A neighboring vehicle was simulated using a bicycle model for lateral dynamics and a second order transfer function was used to approximated longitudinal dynamics. A pure pursuit controller was used to steer the vehicle in between the truck platoon for cut-in vehicles. A Monte Carlo trajectory generation process was used to generate training, validation, and test trajectories to the network to process. Among the varied parameters were longitudinal velocity, reference distance behind the lead truck for the merging vehicle, lateral biases inside the lane, noisiness of the reference waypoints and reference longitudinal velocities, look ahead distance of the Pure Pursuit Controller, and starting position of the neighboring vehicle. The results of the trained network show that the network is particularly good at predicting cut-in behavior over the state-based benchmark predictors. The passing predictor performance on passing trajectories appeared to lag that of the state-based methods, although its average distance to the final end point of test trajectories after the 5 second prediction horizon was less than 3 meters. Incorporating the performance of both predictors by using the trajectory of highest confidence as rated by the ensemble network, the ensemble network tended to outperform the state-based predictors, especially after horizons of 3.5 seconds. The results of using the ensemble network to predict time until cut-in showed that the state-based predictors did really well at predicting cut-in times for time until cut-ins of up to 1 second, however, after that horizon the network outperformed, predicting times till cut-in with an average of 2 seconds error at 5 seconds until

cut-in. A cut-in detection algorithm was designed which used the prediction models to predict if a cut-in would occur and when. The performance of the detector was measured for each predictor by determining the balanced accuracy, rate of false alarms, and rate of missed detections. The state based predictors performed the best, although the LSTM Ensemble Predictor wasn't far behind.

Chapter 5 details the collection and processing of data from the Auburn truck platoon. The truck platoon was driven for a couple hours on a round trip trajectory on i85 near Auburn, Alabama. Within the radar data collected was roughly 50 passing trajectories and 6 cut-in trajectories. The radar points were static-dynamic filtered, rotated into the road frame as estimated by the curvature between the two trucks, and discriminated based off whether they were on the estimated road or not. Points that survived were clustered and Kalman Filtered using a Nearly Constant Velocity model. These filtered tracks were used as test trajectories to demonstrate the viability of the network designed in Chapter 4. Results show that the network is able to identify and predict cut-in behavior, and that although improved models can likely improve network behavior, collecting roughly 1000 cut-in trajectories in total should allow for good training generalization.

6.2 Conclusion

A framework for the use of Long-Short Term Memory Networks in the prediction of cut-ins for the Auburn Truck Platoon was presented. The results suggest that the network is able to predict trajectories of cut-in vehicles much more accurately than current state-based predictors can. Results also show that the network is successful in predicting the trajectory output with the lowest root mean squared error with predicted probabilities corresponding to the correct probability of occurrence. A cut-in detection algorithm was designed using the various predictors and found that the state-based predictors perform well. With a truth horizon of 5 seconds, the LSTM predictor had a balanced accuracy of 87.61 percent. This was an improvement over the constant velocity predictor which had a balanced accuracy of 83.39 percent. However, the constant acceleration and constant turn radius predictors performed the best with balanced accuracies of 90.34 percent and 90.33 percent respectively.

Given the comparable performance of the LSTM to the state-based predictors and considering the large difference in ease of implementation between the state-based and LSTM predictors, it is therefore recommended that the state-based predictors be used first for real implementation, then a network can be design and implemented if better performance is needed. It is recommended that the constant velocity predictor be implemented first, for two main reasons. First, it is currently the model used in the track filter working on the truck platoon for cut-in detection, so it will be the easiest to implement. Second, the CV predictor should perform better than the CA and CTR predictors with noisy radar returns, since the CA and CTR predictors require noisy range solutions be derived twice, they may suffer in accuracy due to noisy range acceleration estimates. Nonetheless, the CA and CTR predictors worked better than the CV predictor over longer prediction horizons, so their implementation is worth exploring. Additionally, the CV predictor performed the best overall within the 2 second truth horizon with a balanced accuracy of 96.2 percent. Therefore it may be useful to design a detection system that implements several different models, weighting them based on their accuracy at different prediction horizons.

Ultimately, this thesis recommends exploring the state-based methods for real implementation on the truck platoon first and foremost. During this implementation, a data set of vehicle trajectories may continually be gathered that eventually may serve as a fertile ground for testing machine learning implementations. Once this point is reached, this thesis recommends continuing the multi-modal trajectory prediction method, as it has been shown that an LSTM is able to accurately predict trajectories in a multi-modal fashion, leading to better overall predictions.

6.3 Future Work

The work performed in this thesis provides fertile ground for future work. Some areas for future work are presented below.

- Improve upon the simulation designed in Chapter 4. This could mean moving to a higher fidelity simulation such as Anvel or CarSim, or improving the Monte Carlo Simulations to incorporate varying vehicle parameters or to simulate scenarios with more than one

neighboring vehicle. Additionally, the Monte Carlo distributions in this thesis used uniform distributions for all varying parameters. It may be more accurate to change some of these distributions to normal distributions. For example, in the simulation in this thesis the lateral bias from the center of the lane was uniform. This means that simulated vehicles were as likely to drive within the center of the lane as on the edges of the lane. Switching this to a normal distribution would mean the majority of vehicles drive near the center of the lane.

- Incorporate the abilities of CNNs like they are used in [21] to model interactions between traffic agents. This could improve predictions if the networks are able to understand interactions between vehicles neighboring the truck platoon. For example, the network could learn not to predict a neighboring vehicle to merge into a vehicle already in between the platoon.

- Expand the network experiences beyond that of a two lane highway. Trucks travel on many different types of highways. This thesis focused on predictions on traffic behavior on a two lane interstate with a speed limit of 70 miles per hour. Future work can expand the experience distributions to include highways with more than two lanes, include vehicles merging into the platoon and then off an off ramp, and roads with different speed limits.

- Explore the use of other models for the tracking of neighboring vehicles. Currently used is the Nearly Constant Velocity Model but other Models such as the Nearly Constant Acceleration Model could be used as well. One issue with the NCA model, however, is the need for an accurate estimate of range acceleration. Given the noisy nature of radar measurements, deriving noise twice can lead to erroneous estimates.

- Use other sensors in the position solution. This could be lidar or camera. Sensor fusion of lidar and/or camera with radar could lead to much more robust tracking and prediction solutions of neighboring vehicles. This could manifest in better positioning as well as gathering signals such as blinkers and brake lights from vehicles.

- Add additional radars to the sides of the follower trucks. Additional radar facing forward on the sides of the follower trucks can provide earlier vehicle detection, allowing networks to begin observing data sooner. This would allow them to make predictions more robustly, as one of the failure modes of the current design is that the network doesn't have enough time to observe a full input trajectory if the vehicle cuts in too quickly.

- Gather a large training set of cut-in vehicles. This could come gradually over time from various highway tests the truck team does. Given diligent recording of bag files through ROS, someone could go back through each test and segment the individual trajectories into passing and cut-in trajectories. Code from this work can easily be recycled to do this and can provide a starting point for more studies involving all methods under the sun that could potentially be used to solve this problem.

- Implement the CV and CA predictors on data collected from the truck platoon and perform a cut-in detection analysis similar to that performed in Section 4.4.3. Performing this first will avoid much of the development time involved with getting prediction networks to work, and will give a solid baseline of which future designed networks can compete against.

- Design a more robust lane line estimate. Ward et. al in [77] show the feeble nature of the current solution. While it works well on the data collected in Chapter 5, it breaks down when the trucks aren't in the center of the lane, are in different lanes, or if the road curves in more than one direction in between the trucks. One potential aid to this could be to place cameras on the back of the lead truck and the front of the follower to estimate where the trucks are in their respective lane, as well as the width of the lane.

- Explore goal based prediction methods. A nominal lane change model can be designed for each modal behavior. This could include a model to keep the vehicle straight on the lane, a model to merge right, merge left, or to exit an off ramp. A network can be reading the past positional behavior as well as additional information such as neighboring vehicle positions and off ramp locations and output a probability of occurrence attached to each

model similar to the functionality of the Ensemble Network designed in Section 4.4.1.3. A recent work that explored this method similar to this approach can be found in [65].

Bibliography

[1] International Trade Administration. *Logistics & Transportation Spotlight*. en. Nov. 2019. URL: https://www.selectusa.gov/logistics-and-transportation-industry-united-states (visited on 01/13/2020).

[2] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple Object Recognition with Visual Attention". In: *arXiv:1412.7755 [cs]* (Apr. 2015). arXiv: 1412.7755. URL: http://arxiv.org/abs/1412.7755 (visited on 02/19/2020).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". en. In: *arXiv:1409.0473 [cs, stat]* (Sept. 2014). arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473 (visited on 08/26/2019).

[4] Yaakov Bar-Shalom, Xiao-Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation*. New York: Wiley, 2001. ISBN: 978-0-471-41655-5.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[6] Richard Bishop et al. "Evaluation and Testing of Driver-Assistive Truck Platooning: Phase 2 Final Results". en. In: *Transportation Research Record: Journal of the Transportation Research Board* 2615.1 (Jan. 2017), pp. 11–18. ISSN: 0361-1981, 2169-4052. DOI: 10.3141/2615-02. URL: http://journals.sagepub.com/doi/10.3141/2615-02 (visited on 10/28/2019).

[7] George E. P. Box et al. *Time series analysis: forecasting and control*. Fifth edition. Wiley series in probability and statistics. Hoboken, New Jersey: John Wiley & Sons, Inc, 2016. ISBN: 978-1-118-67502-1.

[8] R Brothers and D Bevly. "A Comparison of Vehicle Handling Fidelity Between the Gazebo and ANVEL Simulators". In: *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium*. Aug. 2019, pp. 13–15.

[9] Jason Brownlee. *How to Normalize and Standardize Your Machine Learning Data in Weka*. en-US. July 2016. URL: https://machinelearningmastery.com/normalize-standardize-machine-learning-data-weka/ (visited on 10/07/2019).

[10] Jason Brownlee. *How to Remove Trends and Seasonality with a Difference Transform in Python*. en-US. July 2017. URL: https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/ (visited on 11/14/2019).

[11] Jason Brownlee. *How to Use Power Transforms for Time Series Forecast Data with Python*. en-US. Jan. 2017. URL: https://machinelearningmastery.com/power-transform-time-series-forecast-data-python/ (visited on 11/14/2019).

[12] Jason Brownlee. *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*. Machine Learning Mastery, 2017.

[13] Jason Brownlee. *Machine Learning Mastery with Python: Understand Your Data, Create Accurate Models and Work Projects End-to-end*. Machine Learning Mastery, 2016.

[14] Jason Brownlee. *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. en-US. July 2016. URL: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ (visited on 02/19/2020).

[15] Cang Ye, N. H. C. Yung, and Danwei Wang. “A fuzzy controller with supervised learning assisted reinforcement learning algorithm for obstacle avoidance”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 33.1 (2003), pp. 17–27.

[16] Irene Cara, Jan-Pieter Paardekooper, and TNO Helmond. “The potential of applying machine learning for predicting cut-in behaviour of surrounding traffic for truck-platooning safety”. In: *25th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration*. 2017.

[17] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[18] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *arXiv:1409.1259 [cs, stat]* (Oct. 2014). arXiv: 1409.1259. URL: http://arxiv.org/abs/1409.1259 (visited on 04/03/2020).

[19] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[20] Akansel Cosgun et al. “Towards full automated drive in urban environments: A demonstration in GoMentum Station, California”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, June 2017, pp. 1811–1818. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995969. URL: http://ieeexplore.ieee.org/document/7995969/ (visited on 02/04/2020).

[21] Henggang Cui et al. “Multimodal trajectory predictions for autonomous driving using deep convolutional networks”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2090–2096.

[22] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. en. In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194, 1435-568X. DOI: 10.1007/BF02551274. URL: http://link.springer.com/10.1007/BF02551274 (visited on 08/27/2019).

[23] *Delphi Electronically Scanning Radar — RADAR*. en-US. Library Catalog: autonomoustuff.com. URL: https://autonomoustuff.com/product/delphi-esr-2-5-24v/ (visited on 03/02/2020).

[24] US Department of Energy. *Average Annual Fuel Use by Vehicle Type*. Nov. 2018. URL: https://afdc.energy.gov/data/10308 (visited on 01/13/2020).

[25] US Department of Energy. *Average Annual Vehicle Miles Traveled by Major Vehicle Categories*. Apr. 2019. URL: https://afdc.energy.gov/data/10309 (visited on 01/13/2020).

[26] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback control of dynamic systems*. 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002. ISBN: 978-0-13-032393-4.

[27] Lex Fridman. *Deep Learning State of the Art (2020)*. Jan. 2020. URL: https://www.youtube.com/watch?v=0VH1Lim8gL8 (visited on 04/06/2020).

[28] Ian J. Goodfellow et al. "Generative Adversarial Networks". en. In: *arXiv:1406.2661 [cs, stat]* (June 2014). arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661 (visited on 08/26/2019).

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0-262-03561-3.

[30] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *arXiv:1308.0850 [cs]* (Aug. 2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850 (visited on 10/09/2019).

[31] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017). arXiv: 1503.04069, pp. 2222–2232. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2016.2582924. URL: http://arxiv.org/abs/1503.04069 (visited on 04/24/2020).

[32] GW Groves, WD Blair, and JE Gray. *Some concepts for target trajectory predictions*. Tech. rep. NAVAL SURFACE WARFARE CENTER DAHLGREN DIV VA, 1994.

[33] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)". In: *arXiv:1606.08415 [cs]* (June 2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415 (visited on 09/25/2019).

[34] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91.1 (1991).

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[36] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Netw.* 2.5 (July 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. URL: http://dx.doi.org/10.1016/0893-6080(89)90020-8 (visited on 08/27/2019).

[37] Luke Humphreys et al. *Lateral Offset and Its Potential Effects on Platooning*. Apr. 2016. URL: http://eng.auburn.edu/~dmbevly/.

[38] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. English. OCLC: 1057236722. Heathmont, Vic.: OTexts, 2018. ISBN: 978-0-9875071-1-2.

[39] Nal Kalchbrenner and Phil Blunsom. "Recurrent continuous translation models". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 1700–1709.

[40] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". en. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. URL: https://asmedigitalcollection.asme.org/fluidsengineering/article/82/1/35/397706/A-New-Approach-to-Linear-Filtering-and-Prediction (visited on 02/04/2020).

[41] Andrej Karpathy. *Building the Software 2.0 Stack*. San Francisco, CA, June 2018. URL: https://databricks.com/session/keynote-from-tesla (visited on 04/07/2020).

[42] Andrej Karpathy. *Convolutional Neural Networks: Architectures, Convolution / Pooling Layers*. URL: http://cs231n.github.io/convolutional-networks/#conv (visited on 08/27/2019).

[43] Andrej Karpathy. *Neural Networks Part 1: Setting up the Architecture*. URL: http://cs231n.github.io/neural-networks-1/#actfun (visited on 11/14/2019).

[44] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. May 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (visited on 08/27/2019).

[45] Andrej Karpathy and Li Fei-Fei. "Deep Visual-Semantic Alignments for Generating Image Descriptions". In: *arXiv:1412.2306 [cs]* (Apr. 2015). arXiv: 1412.2306. URL: http://arxiv.org/abs/1412.2306 (visited on 02/23/2020).

[46] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 02/19/2020).

[47] Kris M Kitani et al. "Activity forecasting". In: *European Conference on Computer Vision*. Springer, 2012, pp. 201–214.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[49] Keita Kurita. *An Overview of Normalization Methods in Deep Learning*. Nov. 2018. URL: https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/ (visited on 02/25/2020).

[50] Yann Lecun. “Generalization and network design strategies”. English (US). In: *Connectionism in perspective* (1989). URL: https://nyuscholars.nyu.edu/en/publications/generalization-and-network-design-strategies (visited on 08/27/2019).

[51] Fei-Fei Li, Justin Johnson, and Serena Yeung. “Lecture 3: Loss Functions and Optimization”. en. In: (2018), p. 97.

[52] Heng Luo et al. “Texture Modeling with Convolutional Spike-and-Slab RBMs and Deep Extensions”. en. In: *arXiv:1211.5687 [cs, stat]* (Nov. 2012). arXiv: 1211.5687. URL: http://arxiv.org/abs/1211.5687 (visited on 08/26/2019).

[53] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: http://tensorflow.org/.

[54] Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[55] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602 (visited on 08/26/2019).

[56] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, pp. 663–670.

[57] OpenAI. *OpenAI Five Defeats Dota 2 World Champions*. en. Library Catalog: openai.com. Apr. 2019. URL: https://openai.com/blog/openai-five-defeats-dota-2-world-champions/ (visited on 04/07/2020).

[58] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. 2013, pp. 1310–1318.

[59] Mohammad Pezeshki. *What are some general tips on feature selection and engineering that every data scientist should know?* Mar. 2014. URL: https://www.quora.com/What-are-some-general-tips-on-feature-selection-and-

engineering-that-every-data-scientist-should-know (visited on 03/08/2020).

[60] Tanner Ray. "Pedestrian navigation using particle filtering and a priori building maps". In: (2019).

[61] Roberto Reif. *Importance of Feature Scaling in Data Modeling (Part 1)*. en-US. Library Catalog: www.robertoreif.com. Dec. 2017. URL: https://www.robertoreif.com/blog/2017/12/16/importance-of-feature-scaling-in-data-modeling-part-1-h8nla (visited on 03/08/2020).

[62] Roberto Reif. *Importance of Feature Scaling in Data Modeling (Part 2)*. en-US. Library Catalog: www.robertoreif.com. Jan. 2018. URL: https://www.robertoreif.com/blog/2017/12/21/importance-of-feature-scaling-in-data-modeling-part-2 (visited on 03/08/2020).

[63] M. A. Richards et al., eds. *Principles of modern radar*. Raleigh, NC: SciTech Pub, 2010. ISBN: 978-1-891121-52-4 978-1-891121-53-1 978-1-891121-54-8 978-1-61353-201-0.

[64] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0 (visited on 08/27/2019).

[65] Matthias Schreier, Volker Willert, and Jurgen Adamy. "An Integrated Approach to Maneuver-Based Trajectory Prediction and Criticality Assessment in Arbitrary Road Environments". In: *IEEE Transactions on Intelligent Transportation Systems* 17.10 (Oct. 2016), pp. 2751–2766. ISSN: 1524-9050, 1558-0016. DOI: 10.1109/TITS.2016.2522507. URL: http://ieeexplore.ieee.org/document/7412746/ (visited on 02/05/2020).

[66] Rajiv Shah and Rob Romijnders. "Applying Deep Learning to Basketball Trajectories". In: *arXiv:1608.03793 [cs]* (Aug. 2016). arXiv: 1608.03793. URL: http://arxiv.org/abs/1608.03793 (visited on 08/26/2019).

[67] Ryan Shaw. “Obstacle Avoidance of an Unmanned Ground Vehicle using a Combined Approach of Model Predictive Control and Proportional Navigation”. In: (2018).

[68] Patrick Smith et al. “Experimental Results and Analysis of a Longitudinal Controlled Cooperative Adaptive Cruise Control (CACC) Truck Platoon”. In: vol. Volume 1: Advanced Driver Assistance and Autonomous Technologies; Advances in Control Design Methods; Advances in Robotics; Automotive Systems; Design, Modeling, Analysis, and Control of Assistive and Rehabilitation Devices; Diagnostics and Detection; Dynamics and Control of Human-Robot Systems; Energy Optimization for Intelligent Vehicle Systems; Estimation and Identification; Manufacturing. Dynamic Systems and Control Conference. 2019. URL: https://doi.org/10.1115/DSCC2019-9135.

[69] Jarrod M Snider. “Automatic steering methods for autonomous automobile path tracking”. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).

[70] Robert F. Stengel. *Optimal control and estimation*. Dover books on advanced mathematics. New York: Dover Publications, 1994. ISBN: 978-0-486-68200-6.

[71] Thomas Streubel and Karl Heinz Hoffmann. “Prediction of driver intended path at intersections”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. MI, USA: IEEE, June 2014, pp. 134–139. ISBN: 978-1-4799-3638-0. DOI: 10.1109/IVS.2014.6856508. URL: http://ieeexplore.ieee.org/document/6856508/ (visited on 02/04/2020).

[72] Chen Sun et al. “Revisiting unreasonable effectiveness of data in deep learning era”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 843–852.

[73] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[74] Thomas Troupe Tabb. “Multi-Antenna GPS for Improved Carrier Phase Positioning in Autonomous Convoys”. en. In: (July 2019). URL: https://etd.auburn.edu/handle/10415/6900 (visited on 02/27/2020).

[75] J.M. Wang, D.J. Fleet, and A. Hertzmann. “Gaussian Process Dynamical Models for Human Motion”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2 (Feb. 2008), pp. 283–298. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1167. URL: http://ieeexplore.ieee.org/document/4359316/ (visited on 02/05/2020).

[76] Sen Wang, Daoyuan Jia, and Xinshuo Weng. “Deep Reinforcement Learning for Autonomous Driving”. In: *arXiv:1811.11329 [cs]* (Nov. 2018). arXiv: 1811.11329. URL: http://arxiv.org/abs/1811.11329 (visited on 08/26/2019).

[77] Jacob Ward et al. “Cooperative Adaptive Cruise Control (CACC) in Controlled and Real-World Environments: Testing and Results”. In: 2019.

[78] David H Wolpert, William G Macready, et al. *No free lunch theorems for search*. Tech. rep. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[79] S. H. I. Xingjian et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[80] Keisuke Yoneda, Akisue Kuramoto, and Naoki Suganuma. “Convolutional neural network based vehicle turn signal recognition”. In: *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. Okinawa: IEEE, Nov. 2017, pp. 204–205. ISBN: 978-1-5090-6664-3. DOI: 10.1109/ICIIBMS.2017.8279693. URL: http://ieeexplore.ieee.org/document/8279693/ (visited on 04/14/2020).

[81] Julius Ziegler et al. “Making Bertha Drive—An Autonomous Journey on a Historic Route”. In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20. ISSN: 1939-1390. DOI: 10.1109/MITS.2014.2306552. URL: http://ieeexplore.ieee.org/document/6803933/ (visited on 10/28/2019).

[82] Alex Zyner et al. “Long short term memory for driver intent prediction”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, June 2017, pp. 1484–

1489. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995919. URL: http://ieeexplore.ieee.org/document/7995919/ (visited on 04/03/2020).

Appendix A

Vanilla RNN Backward Propagation

The derived equations for back propagation through an RNN are written out below.

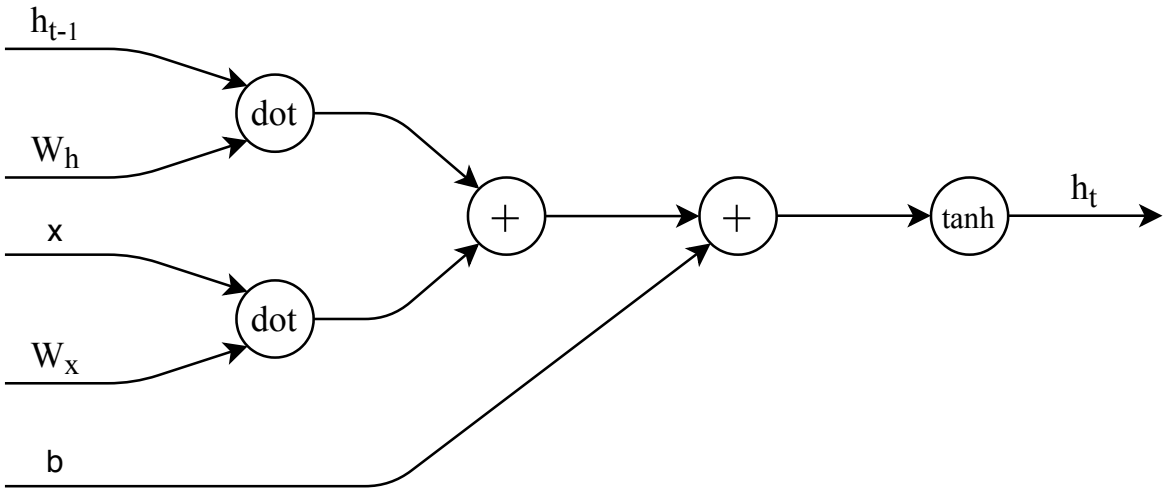


Figure A.1: Diagram of RNN Forward Propagation

$$a = W^l \begin{bmatrix} h_t^{l-1} \\ h_{t-1}^l \end{bmatrix} \tag{A.1}$$

$$\frac{dh}{dh_{t-1}} = W_h \frac{dh}{da} \tag{A.2}$$

$$\frac{dh}{dW_h} = h_{t-1} \frac{dh}{da} \tag{A.3}$$

$$\frac{dh}{dx} = W_x \frac{dh}{da} \tag{A.4}$$

$$\frac{dh}{dW_x} = x \frac{dh}{da} \tag{A.5}$$

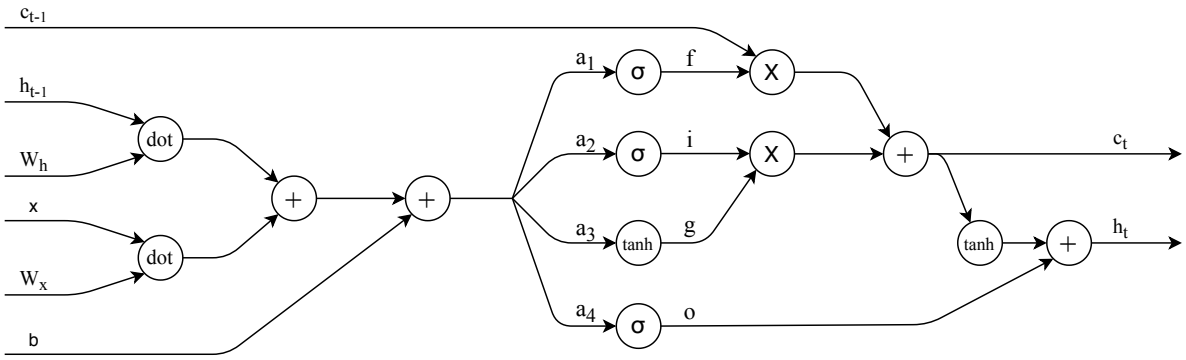
$$\frac{dh}{db} = \frac{da}{db}\frac{dh}{da} \tag{A.6}$$

$$\frac{dh}{da} = (1 - tanh^2(a))dh \tag{A.7}$$

Appendix B

LSTM Backward Propagation

The derivation for LSTM Back-Propagation is given below.



B.1 Diagram of LSTM Forward Propagation

$$a = h_{t-1}W_n + xW_x + b = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \tag{B.1}$$

$$\frac{dh}{do} = tanh(c_t)dh_t \tag{B.2}$$

$$\frac{dh}{dc} = o(1 - tanh^2(dc))dh \tag{B.3}$$

$$\frac{dh}{df} = c_{t-1}\frac{dh}{dc} \tag{B.4}$$

$$\frac{dh}{di} = g\frac{dh}{dc} \tag{B.5}$$

$$\frac{dh}{dc}_{t-1} = f\frac{dh}{dc} \tag{B.6}$$

$$\frac{dh}{dg} = i\frac{dh}{dc} \tag{B.7}$$

$$\frac{dh}{da} = \begin{bmatrix} \sigma(a_1) \\ (1-\sigma(a_1))\frac{dh}{df} \\ \sigma(a_2)(1-\sigma(a_2))\frac{dh}{di} \\ \sigma(a_4)(1-\sigma(a_4))\frac{dh}{do} \\ (1-tanh^2(a_3))\frac{dh}{dg}] \end{bmatrix} \tag{B.8}$$

$$\frac{dh_t}{dh_{t-1}} = W_n\frac{dh}{da} \tag{B.9}$$

$$\frac{dh}{dW_n} = h_{t-1}\frac{dh}{da} \tag{B.10}$$

$$\frac{dh}{dx} = W_x\frac{dh}{da} \tag{B.11}$$

$$\frac{dh}{W_x} = x\frac{dh}{da} \tag{B.12}$$