Deep learning based motion prediction algorithms for autonomous driving

Version of September 14, 2022

Chenxu Ma

Deep learning based motion prediction algorithms for autonomous driving

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

Computer Engineering

by

Chenxu Ma

Accelerated Big Data Systems Research Group
Department of Computer Engineering
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

AUTOPIA Program
Spain Superior Council of Scientific
Investigations
Polytechnic University of Madrid
Greater Madrid Metropolitan, Spain
https://autopia.car.upm-csic.es/

Deep learning based motion prediction algorithms for autonomous driving

Author: Chenxu Ma
Student id: 5215536
Email: C.Ma-6@student.tudelft.nl

Abstract

In order to ensure that autonomous driving vehicles can make appropriate driving decisions based on the surrounding situation, motion prediction algorithms are used to generate the driving decision output, which will then be used for guiding the trajectory of the vehicle. In general, the output of the motion prediction algorithm is a series that contains the predicted information for the future movement of the vehicle. A traditional approach is using a physics-based model to generate the acceleration prediction series. However, such an approach requires lots of mathematical computation but is only capable to be effective in specific driving scenarios.

To solve that kind of issue, we proposed a data-driven approach by running four different kinds of machine learning models to generate the prediction output series. The results show that the auto-regressive (AR) model has the best prediction performance compared with traditional physics-based models, with a 14.32% improvement on average for the ADE (average displacement error) evaluation metric and 5.93% improvement on average for the FDE (final displacement error) evaluation metric.

Thesis Committee:

Chair:	Dr. Zaid Al-Ars, Faculty EEMCS, TU Delft
University supervisor:	Dr. Zaid Al-Ars, Faculty EEMCS, TU Delft
Committee Member:	Dr. Cynthia Liem, Faculty EEMCS, TU Delft

Preface

With this thesis, my two-year master study comes to an end. These two years have been both extraordinary as well as challenging.

First of all, I want to thank Prof. Zaid Al-Ars and the Prof. Jorge Villagra and PhD candidate Vinicius Trentin who helped me a lot during the thesis both on the development of my engineering as well as academic skills. During the whole thesis period, we discussed and cooperated on a lot of topics and also faced a lot of difficulties in this challenging project. At various points in time, these challenges seemed insurmountable. However, with continued investigation and perseverance, we finally were able to resolved these issues and create solutions that achieved better results than we had.

Secondly, I want to thank my girlfriend Zhuoran Guo. 2020 was an unexpected year. The COVID pandemic influenced the decisions people made for various aspects of their lives. For us, we got to know each other in an unexpected way, and our love journey is also full of unexpected and amazing stories. Although there were also some challenges, my girlfriend was always able to be a source of great happiness for me. That is life, balancing between happiness and sadness. Thank you Zhuoran, you gave me a lot. I also learned a lot.

Thirdly, I want to thank my parents who also cheered me a lot since I was in high school. You cheered me and taught me a lot and made me have a huge, or I would say, earth-shaking changes as a grew from a boy who always played on his cellphone during class to a man who understands his responsibilities.

Last but not least, I'm really thankful for the events that helped shape my journey. These 6 year have really been a small miracle for me. I overcame lots of difficulties. I did not imagine I can be one of the top 10 best students in our school when I first came to high school. I also did not imagine I was the only one student who chose to study abroad in our major when I was in university. I also did not imagine I can find a software engineer job when I started to switch to a software-related major from the microelectronics field two years ago. Thanks to the hard-work I invested during this period. At the end of my journey I learned that there is nobody who can do the work for you to open a new door, but you can use your energy, your passion, and your knowledge to face the difficulties that cross your path.

Chenxu Ma
Delft, the Netherlands
September 14, 2022

Contents

Preface iii

Contents v

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Context and motivation . . . 1
- 1.2 Problem statement . . . 2
- 1.3 Outline . . . 3

2 Preliminaries 5
- 2.1 Time series prediction . . . 5
- 2.2 Autoregressive component . . . 6
- 2.3 Recurrent neural network . . . 6
  - 2.3.1 Traditional RNN . . . 6
  - 2.3.2 Long Short-Term Memory Network . . . 7
  - 2.3.3 Gated Recurrent Unit . . . 8
- 2.4 Convolutional neural network . . . 9
  - 2.4.1 Convolutional layer . . . 10
  - 2.4.2 Pooling layer . . . 11
  - 2.4.3 Fully connected layer . . . 12
- 2.5 Attention scheme . . . 12
  - 2.5.1 Self-attention scheme . . . 12
  - 2.5.2 Multi-head attention scheme . . . 14
- 2.6 Position-wise feed-forward network . . . 16
- 2.7 Dual self-attention network for multivariate time series forecasting . . . 17

3 Interactive awareness framework for roundabout route prediction 19
- 3.1 Interaction awareness . . . 19
  - 3.1.1 Finding corridors . . . 20
  - 3.1.2 Find interactions . . . 21

3.1.3 Compute intentions 22

4 Methodology **25**
4.1 Dataset and data processing 25
4.2 Evaluation matrix 27
4.2.1 Mean square error 27
4.2.2 Root mean square error 27
4.2.3 Mean absolute error 27
4.2.4 Average displacement error 28
4.2.5 Final displacement error 28
4.3 Comparison approach 28
4.4 Implementation details 30
4.5 Experiment setup 31

5 Experimental results **33**
5.1 Project contributions 33
5.2 Impacts of different prediction approaches for timing and prediction performance 33
5.2.1 Timing performance 34
5.2.2 Discussion 35
5.3 Experiment results on the whole test dataset 35
5.3.1 Results for acceleration output 35
5.3.2 Acceleration series prediction effect 36
5.3.3 Vehicle trajectory prediction result in MATLAB for partial simulation 36
5.3.4 Results for velocity output 36
5.3.5 Velocity series prediction effect 40
5.3.6 Vehicle trajectory prediction result in MATLAB for partial simulation 40
5.3.7 Model generalization ability test 40

6 Conclusion and future work **45**
6.1 Conclusion 45
6.2 Future work 46

Bibliography **47**

List of Figures

2.1 Basic structure of traditional RNNs [1] . . . 6
2.2 Detailed structure of RNN [17] . . . 7
2.3 Basic structure of LSTM [23] . . . 8
2.4 Basic structure of GRU [14] . . . 9
2.5 Basic structure of CNN [22] . . . 10
2.6 Max pooling and average pooling . . . 11
2.7 Dot product operation in self-attention scheme [21] . . . 13
2.8 Softmax and context vector computation operation in self-attention scheme . . . 13
2.9 Basic form of scaled dot product [21] . . . 14
2.10 The principle of scaled dot product when computing attention value for a sequence . . . 15
2.11 Basic form of multi-head attention scheme . . . 15
2.12 Basic structure of dual self-attention network (DSANet) [8] . . . 16

3.1 Interaction awareness procedure pipeline . . . 19
3.2 A scenario where the corridor will be removed . . . 20
3.3 A scenario where the corridor has a dependency [19] . . . 21
3.4 Bayesian network [18] . . . 22

4.1 Example of possible corridors for a vehicle in a roundabout . . . 26

5.1 Acceleration prediction visualization for direct generating output approach (in vehicle 33) . . . 37
5.2 Partial simulation result on Matlab simulation framework for the DSANet model in direct 40 acceleration series output approach . . . 38
5.3 Velocity prediction visualization for direct generating output approach (in vehicle 33) . . . 41
5.4 Partial simulation result on MATLAB simulation framework for the AR model in direct 40 velocity series output approach . . . 42
5.5 Entire simulation result on MATLAB simulation framework for all four driving situations with the machine learning models using batch normalization approach and generating acceleration output . . . 44
5.6 Entire simulation result on MATLAB simulation framework for all four driving situations with the machine learning models using global normalization approach and generating acceleration output . . . 44

List of Tables

3.1 Lateral Intention computation [18] 22

4.1 Obtained parameters after modeling 25
4.2 Input Feature 27
4.3 Experiment model and the training configuration 29
4.4 Experiment design for investigativng the timing peformance of different producing output approaches 29
4.5 Hardware Configuration 31
4.6 Software Configuration 31

5.1 Result for directly 40 output approach (output feature: Acceleration) 33
5.2 Result for iterative 40 output approach (output feature: Acceleration) 34
5.3 Result for direct 40 output approach (output feature: Velocity) 34
5.4 Result for iterative 40 output approach (output feature: Velocity) 34
5.5 Evaluation metrics result for 40 acceleration series output 35
5.6 MATLAB partial simulation result based on the acceleration output series . . . 36
5.7 Evaluation metrics result for 40 velocity series output 39
5.8 MATLAB partial simulation result based on the velocity output series 39
5.9 ADE and FDE results of MATLAB entire simulation for all four driving situations for models generating acceleration series output 43

6.1 ADE and FDE results comparison between AR and physics-based model for all four driving situations 46

Chapter 1

Introduction

This chapter introduces some background information and challenges in this graduation research project. In addition, the chapter defines the problem statement of the project, and provides the outline of this thesis.

1.1 Context and motivation

Driving safety is becoming increasingly important as driving-related traffic accidents have been rising gradually in recent years. According to the CDC's data [4], in 2019, more than 36,000 people in the United States died because of vehicle crashes, which means that more than 100 people died due to vehicle crashes every day. Driving distractions such as texting or using a cell phone while driving are the primary reason behind those driving accidents. There are various kinds of solutions to reduce traffic accidents. For example, using more traffic signs or setting more traffic lights could significantly reduce the number of traffic accidents. In addition, having official policies or enforcement to prohibit negative driving habits is also a feasible method. However, those approaches do not manage to ultimately reduce travel accidents caused by driver's slow response or human error. Therefore, it is necessary to figure out an alternative solution to ensure that traffic accidents are reduced further.

Autonomous driving technology has been regarded as a reasonable way to reduce traffic accidents. According to a study by consulting firm McKinsey&Co [13], by interviewing many industry representatives, a self-driving car could cut down 95% of driving accidents in the U.S. and prevent up to $190 billion in damages and health costs annually and save thousands of lives. With the increasing market need for autonomous driving, more and more high-tech companies have started to invest in independent driving research and development. In 2018, SoftBank Vision Fund began to invest $1.35 billion when Cruise vehicles were ready for commercial deployment [6]. On the other hand, exploiting autonomous driving technology will make people more productive because people do not have to pay attention to traffic conditions and pedestrians. Therefore, people could use their commute time to do their work or even have some rest. Based on the discussion above, we can see that developing autonomous driving is necessary.

Autonomous driving is a system that combines several components to make a decision whenever the vehicle is driving, facing obstacles, or involved in various situations. The autonomous driving system generally contains three main steps: perception, decision, and control . Perception means collecting data from vehicle sensors and processing

it so the vehicle can understand its surrounding environment. The primary approaches to collect data from the environment include Radar, LiDAR, and cameras [2]. Among these three approaches, cameras could provide enough information because objects such as roads, pedestrians, and surrounding vehicles captured by cameras would contain different kinds of colors. Therefore, autonomous driving cars could use these different colors as labels. Typically autonomous driving vehicles would combine all these three approaches for perception. The decision is the core component of the autonomous driving system. One approach that could be used in the decision step is to construct a machine learning-based model that would fit the sensor data for position or motion planning. The output of the decision step would be the vehicle position distribution or the acceleration distribution, which represents the future movement of the autonomous driving car. The final step is control. A control system of good quality is also essential to ensure the vehicle can drive smoothly based on the model prediction. For this thesis project, we will concentrate more on developing a machine learning model in the autonomous driving decision step to generate the simulated displacement distribution of the vehicle.

1.2 Problem statement

Currently, autonomous driving systems mainly use physics-based modeling approaches to predict the trajectory of a vehicle based on the surrounding situation of the vehicle, such as the distance to the intersection and the velocity of the leading vehicle. However, it is difficult to create physics-based models due to the high complexity of autonomous driving. As a result, such models are created to represent a rough approximation that abstracts away a lot of the complexity exhibited by car behavior. In this thesis, we aim to use data-driven approaches to create general models that are: 1. easily generated using the driving data collected from cars on the road, and 2. capable to capture the highly complex car behavior exhibited in real-world scenarios.

Therefore, the research questions for this thesis can be defined as follows:

1. Can we use data-driven modeling methods to capture the complexity of autonomous driving tasks?

2. Which data-driven models are most suitable for autonomous driving?

3. What are the limitations of using these models in practice?

To answer these research questions, we need to use multiple machine learning models to predict the future motion path of the vehicle, which is often regarded as a time series prediction problem or, to make it clear, a multi-variate-multi-step time series prediction problem. Given an input series containing multiple features, which represent the surrounding situation of the vehicle, such as the distance to the intersection and the velocity of the leading vehicle, the model output will be the velocity distribution or the acceleration distribution for the future 4 seconds interval. Based on the output from the model, we need to convert the model distribution output into the vehicle coordinate position by feeding that model output to a MATLAB simulation framework developed by the Spanish National Research Council (CSIC) and Polytechnic University of Madrid (UPM) to generate the final result: actual vehicle position prediction. We divide the model output distribution for the future 4 seconds into four parts, with each part representing 1-second distribution.

1.3 Outline

The thesis contains five chapters in total. Chapter 1 briefly introduces autonomous driving technology, related technology trends, and an overview of the thesis topic. Chapter 2 reviews the principle of the technologies and models implemented during the course of this thesis. Chapter 3 presents the details of the modeling framework from our partners, the Spanish National Research Council (CSIC) and Polytechnic University of Madrid (UPM). In Chapter 4, the thesis provides the information related to the experiments performed in this thesis, such as the experiment setting, the feature selection, and the implementation approach for the data processing. Chapter 5 presents the experiment results and discusses the implications of these experiments. Finally, Chapter 6 ends with the conclusions and also describes future work.

Chapter 2

Preliminaries

This chapter gives a description of the ideas used for our machine learning-based model.

2.1 Time series prediction

A time series is a sequence that contains data sorted in time order. Mathematically, a time series $X^{(i)} =< x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)}, ..., x_T^{(i)} >$ is a fully observed sequence of measurements in time where $x_T^{(i)}$ represents the observed measurement at timestamp T and the "(i)" represents a specific feature. From the definition of time series data, we can deduce that the multivariate time series $X =< X^{(1)}, X^{(2)}, X^{(3)}, ..., X^{(i)}, ... >$ means a sequence X whose data contains several features and each feature contains a single variate time series data $X^{(i)} =< x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)}, ..., x_T^{(i)} >$. Time series data always reflects the time-domain relationship among continuous-time data. Therefore, it is important to ensure that the time series data points are sorted in time (i.e., are consecutive in the time horizon).

A time series prediction problem takes the time series data as its input, where the time series data is either multivariate or contains only one variable. The approach of generating prediction output would be different in multi-timestamp sequence prediction.

The first approach at first predicts the output $x_{T+1}^{(i)}$, and then predicts the next output $x_{T+2}^{(i)}$ using the new time series input $< x_2^{(i)}, x_3^{(i)}, x_4^{(i)}, x_5^{(i)}, ..., x_{T+1}^{(i)} >$ until the prediction sequence length is satisfied. One drawback of this approach is that it would generate an accumulated error when predicting the output since every time it will only generate one prediction in the next timestamp. Therefore, as the prediction output length increases, the prediction data error will become larger and larger. Moreover, this approach could not generate a long series within a short time. If, for example, the model needs to use 1 second to generate data prediction at one timestamp, then it would take 40 seconds to generate a prediction distribution that only represents a 4 seconds time interval. These disadvantages give the possibility of trying the second approach.

The second approach directly generates the whole prediction series at one timestamp based on the input series. This approach would save more time in generating the prediction series. Given the time series data $X^{(i)} =< x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)}, ..., x_T^{(i)} >$, the second approach will let the model to directly generate the output $< x_{T+1}^{(i)}, x_{T+2}^{(i)}, x_{T+3}^{(i)}, x_{T+4}^{(i)}, ..., x_{T+t}^{(i)} >$. Using this approach, the model will save more computational time while generating the out-

put. Furthermore, the model could learn all the information in the output series during the training procedure.

2.2 Autoregressive component

The autoregressive component [20] or an autoregressive model is a structure that is mainly used to represent the random process and predict the future state by using the previous state value as the input. A common expression of such a random process is shown in Formula 2.1.

$$y_t = \beta_0 + \beta_1 y_{t-1} + \varepsilon_t \tag{2.1}$$

This formula represents a random progress model whose future state depends on only the previous state at $t-1$ time stamp. β_x means the model's parameters, and ε means the white noise. If we want to use the previous state at both $t-1$ and $t-2$ time stamp, the formula would be changed into Formula 2.2.

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \varepsilon_t \tag{2.2}$$

These formulas show that the autoregressive component is used to compute the linear relation between the previous and current states.

2.3 Recurrent neural network

A recurrent neural network (RNN) [15] is a machine learning network that has the ability to model sequential process data by using its internal memory component and a feedback scheme. Therefore, the recurrent neural network is often used for time series prediction or natural language processing tasks.

2.3.1 Traditional RNN

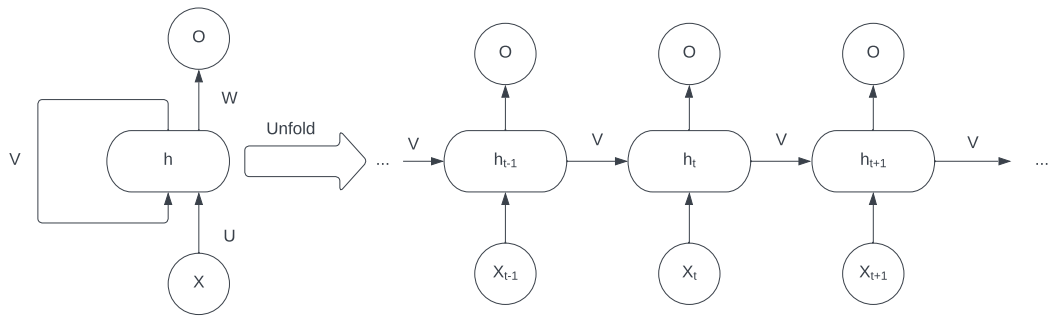


Figure 2.1: Basic structure of traditional RNNs [1]

A simple structure of traditional RNNs is shown in Figure 2.1. This is the most basic type of RNN neural network structure. We can see that, in the RNN structure, the network contains several hidden states. Each hidden state represents the state in the corresponding

time stamp. In every time stamp, the RNN neural network would compute an output at this time stamp and a hidden state vector which would be sent to the next hidden state component.

Figure 2.2 shows a more detailed structure of the RNN hidden state component. The input of that component would be the hidden state vector from the previous hidden state component and the data at the corresponding time stamp. Formula 2.3, 2.4, and 2.5 explain the feed-forward procedure and the mathematical between input and output within the RNN hidden state component. In Formula 2.3, W and U are the parameters of the hidden state component.

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \tag{2.3}$$

$$h^{(t)} = tanh(a^{(t)}) \tag{2.4}$$

$$o^{(t)} = c + Vh^{(t)} \tag{2.5}$$

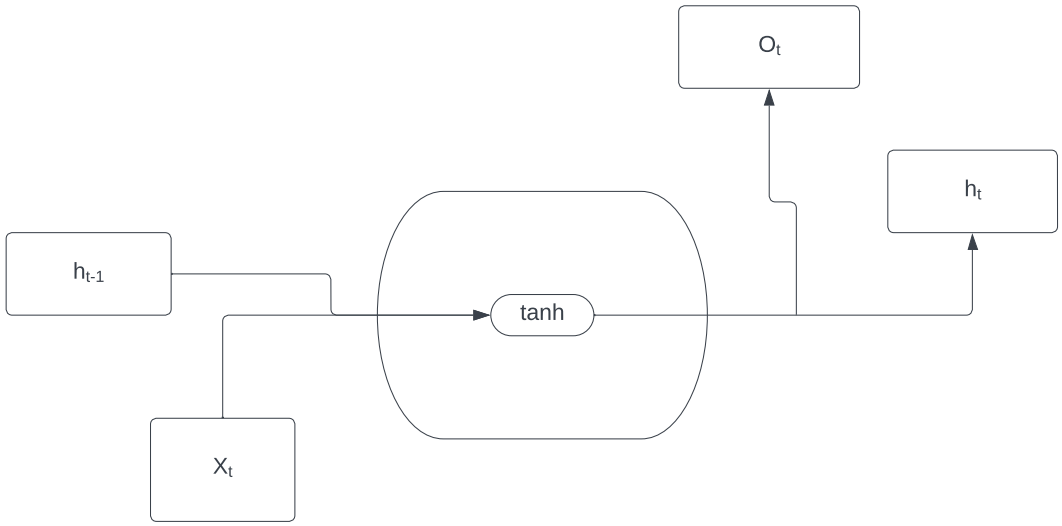


Figure 2.2: Detailed structure of RNN [17]

However, Figure 2.2 also shows the drawback of the traditional RNN. First of all, RNN only has the tanh function, and the structure is too simple. Therefore, traditional RNN could only be able to process short-term dependencies problems. What's more, RNN would also cause gradient vanishing issues. In addition, RNN structure does not have any kinds of cell states to memorize the information, which makes traditional RNN could only depend on the linear computation of the previous hidden state and the data input. These are the reasons to design a better RNN-class model to solve these issues.

2.3.2 Long Short-Term Memory Network

Long Short-Term Memory Network (LSTM) [7] is a variant of the traditional recurrent neural network. Compared with traditional RNN, LSTM has a cell state which can enable LSTM to memorize both long-term and short-term information. That is the reason Figure

2.3 illustrates the structure of the LSTM model. LSTM has both hidden state and cell

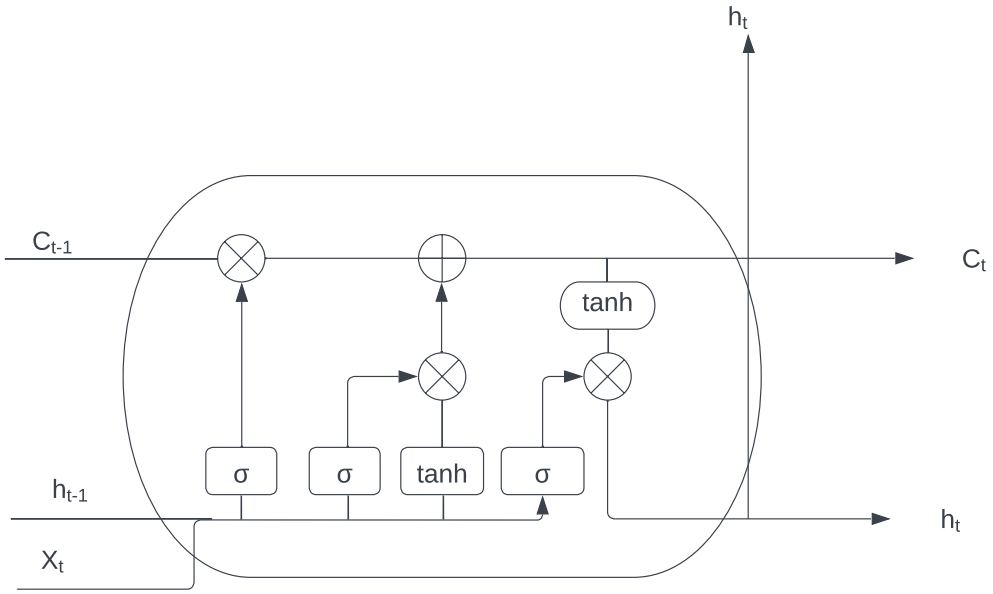


Figure 2.3: Basic structure of LSTM [23]

state, which would be used for computing the output. Besides the tanh function in the traditional RNN, LSTM also uses several gates to store the information and compute the output. These gates are the input gate, forget gate, and output gate. Formula 2.6 to 2.11 express the computation procedure of these three gates. In these formulas, $f^{(t)}$ is the forget gate, $i^{(t)}$ is input gate, $o^{(t)}$ is the output gate, $c^{(t)}$ is the cell state, $h^{(t)}$ is the hidden state, σ_g is the sigmoid function, and σ_c is the tanh function. The weight and bias in these formulas will not change based on time. Therefore we can use the same weight and bias to compute the output at each timestamp.

$$f^{(t)} = \sigma_g(W_f * x^{(t)} + U_f * h^{(t-1)} + b_f) \tag{2.6}$$

$$i^{(t)} = \sigma_g(W_i * x^{(t)} + U_i * h^{(t-1)} + b_i) \tag{2.7}$$

$$o^{(t)} = \sigma_g(W_o * x^{(t)} + U_o * h^{(t-1)} + b_o) \tag{2.8}$$

$$(c^{(t)})' = \sigma_c(W_c * x^{(t)} + U_c * h^{(t-1)} + b_c) \tag{2.9}$$

$$c^{(t)} = f^{(t)} * c^{(t-1)} + i^{(t)} * (c^{(t)})' \tag{2.10}$$

$$h^{(t)} = o^{(t)} * \sigma_c(c^{(t)}) \tag{2.11}$$

2.3.3 Gated Recurrent Unit

Gated Recurrent Unit (GRU) model [5] is another variant of the RNN model. The difference between the LSTM and GRU is that the GRU model does not have the cell state,

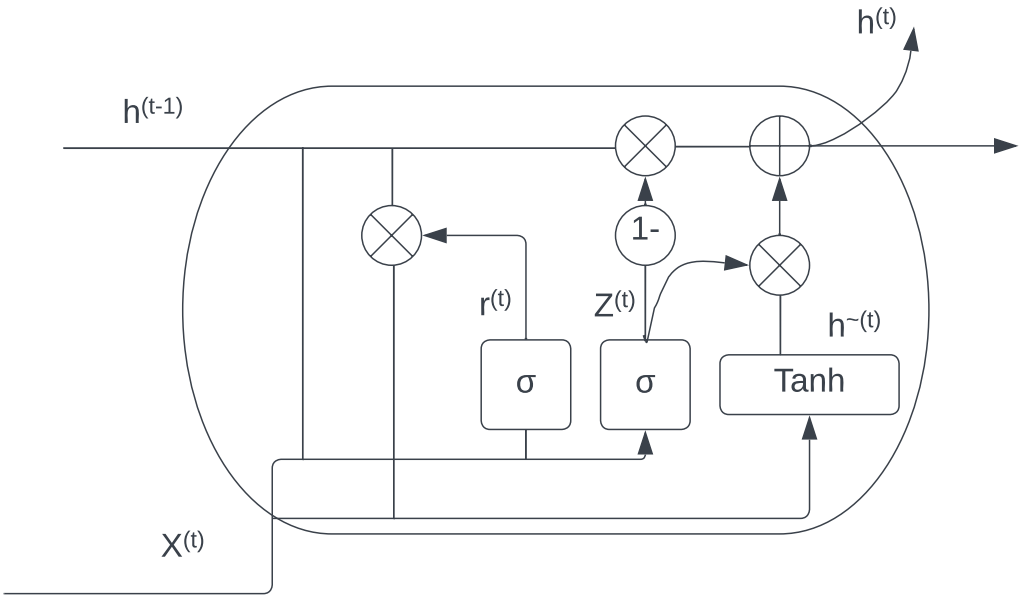


Figure 2.4: Basic structure of GRU [14]

which means the GRU model Figure 2.4 shows the basic structure of the GRU model. Formula 2.12 to Formula 2.15 represents the mathematical principle behind the GRU model. In Formula 2.14, the $\odot$ means the Hadamard product for the matrices.

$$z^{(t)} = \sigma(W_z * x^{(t)} + U_z * h^{(t-1)} + b_z) \tag{2.12}$$

$$r^{(t)} = \sigma(W_r * x^{(t)} + U_r * h^{(t-1)} + b_r) \tag{2.13}$$

$$\tilde{h^{(t)}} = \phi_h(W_h * x^{(t)} + U_h * (r_t \odot h^{(t-1)}) + b_h) \tag{2.14}$$

$$h^{(t)} = z^{(t)} \odot \tilde{h^{(t)}} + (1 - z^{(t)}) \odot h^{(t-1)} \tag{2.15}$$

2.4 Convolutional neural network

The convolutional neural network (CNN) [12] is a machine learning network that is often used for image classification and processing because of its ability for feature extraction. In this thesis work, we mainly used CNN to capture the sequential information in the time series data and obtain a better computational performance. Figure 2.5 shows the most basic structure of the CNN.

From the figure above, we can see that CNN is mainly divided into five parts: input layer, convolution layer, activation function, pooling layer, and fully connected (FC) layer. The convolutional layer is responsible for feature extraction of the input feature vector; the pooling layer realizes the dimensionality reduction processing of the data; finally, the classification is realized through the fully connected layer.

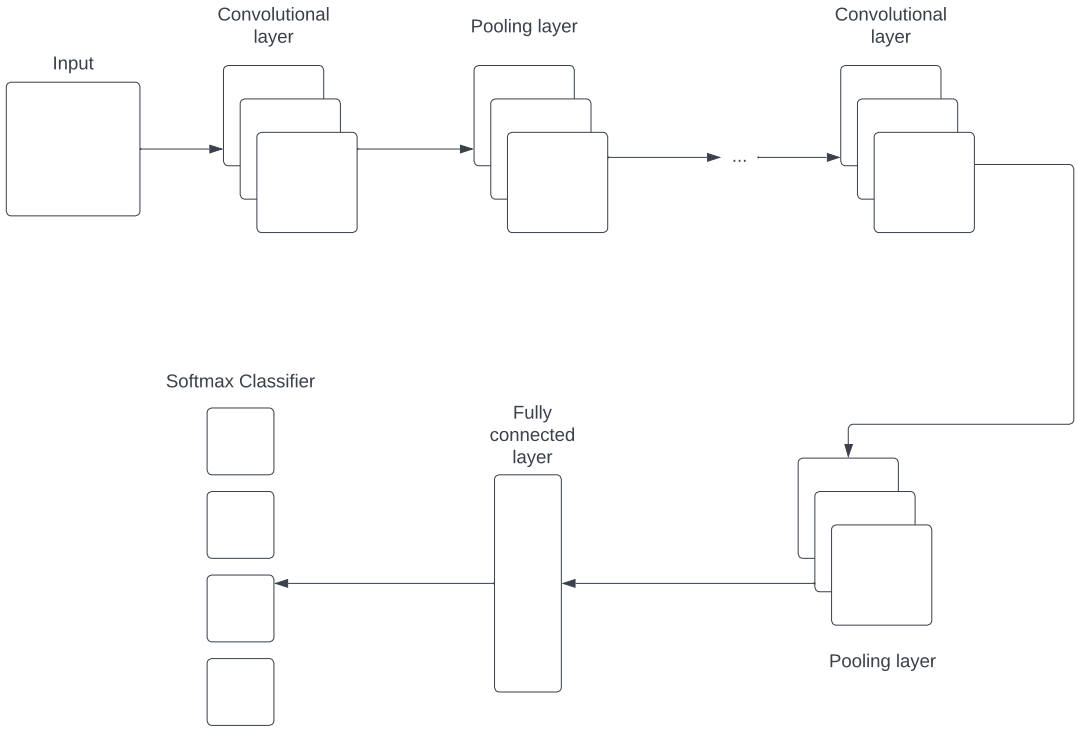


Figure 2.5: Basic structure of CNN [22]

2.4.1 Convolutional layer

The convolutional layer is the most important part of the entire convolutional neural network. Generally speaking, the convolution operation used in image recognition is a two-dimensional convolution operation. That is, a discrete two-dimensional filter (also called a convolution kernel) is convolved with a two-dimensional image. The convolution operation is simply the operation of multiplying and adding the sliding window and the corresponding position on the image. This multiply-add operation is also commonly referred to as an inner product. The convolution operation is mainly used in the field of image processing, and the image features extracted by different convolution kernels in the convolution operation are also different. In general, convolutional layers have the following two characteristics:

Local perception

When performing local perception, the ”convolution kernel” is the most critical element, and it is a tool for local perception. Each neuron is only connected to a region of the input neuron, which is called the receptive field. In image convolution operations, neurons are locally connected in the spatial dimension, but fully connected in depth. For the two-dimensional image itself, the local pixel correlation is also strong. This local connection ensures that the learned filter can have the strongest response to the local input features. The idea of local connections is also inspired by the structure of the visual system in biology, and neurons in the visual cortex receive information locally. For example, a 32×32×3 RGB

image becomes a 28×28×1 feature map after a layer of 5×5×3 convolution, then the input layer has a total of 32×32×3=3072 There are 3072 neurons, and the first hidden layer will have 28×28=784 neurons. These 784 neurons are only partially connected to the neurons of the original input layer.

Shared weights and biases

Suppose there are h hidden layer neurons, each connected to a $w * w$ window, so there will be $h * w * w$ parameters. This brings two problems: (1) each layer will have a large number of parameters; (2) using pixel values as input features is essentially no different from traditional neural networks and do not focus on some input pixels focus on. Therefore, people began to choose to use the weight-sharing operation to avoid this problem. No matter how large the image size is, a fixed-size convolution kernel can be selected. The largest convolution kernel in LeNet is only 5*5*1, while in AlexNet, The largest convolution kernel is just 11*11*3. The convolution operation ensures that each pixel has a weight coefficient, but these coefficients are shared by the entire image, which greatly reduces the number of parameters in the convolution kernel. In addition, the convolution operation takes advantage of the local correlation in the image space, which is one of the biggest differences between CNN and traditional neural networks or machine learning, the automatic extraction of features.

2.4.2 Pooling layer

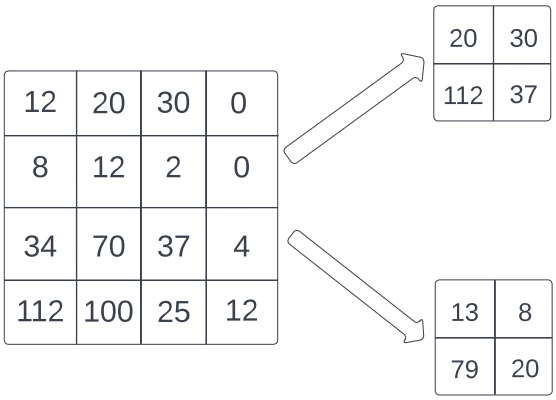


Figure 2.6: Max pooling and average pooling

The main purpose of the pooling layer is to compress the image. The earliest is to model the human visual system for dimensionality reduction (downsampling), and use a higher level of abstraction to represent image features. Figure 2.6 shows the two operations of pooling and the difference in the output produced by each operation. Pooling layers are sandwiched between consecutive convolutional layers and are used to compress the amount of data and parameters and reduce overfitting. Generally speaking, the pooling operation is divided into maximum pooling and average pooling: for maximum pooling, during forwarding propagation, the maximum value of the image area is selected as the pooled value of this area. During backpropagation, the gradient is propagated through the

position of the maximum value, and the gradients at other positions are 0; for average pooling, during forwarding propagation, the average value of the image area is calculated as the pooled value of this area. During backpropagation, the gradients are averaged and distributed to each position.

2.4.3 Fully connected layer

At the end of the convolution, one or more fully connected layers are placed that contain activation functions (e.g., Relu or Sigmoid) that can be used to reshape the size to a vector suitable for the input classifier. For example, if the final convolutional layer outputs a 3x3x128 matrix, but the network only predicts 10 distinct classes, it needs to be reshaped into a 1x1152 vector and gradually reduced in size before feeding it to a classifier (e.g. Softmax classifier) . The fully connected layer will also learn its own function like a typical deep neural network.

2.5 Attention scheme

Attention scheme is an approach that could focus more on some of the input part and ignore other parts. Since the ability to capture the dependencies of the time series, the attention scheme could be able to learn the relation between the data in the time series. To simplify, the attention scheme could help the model to pay more attention to a different piece of the input sequence in order to compute the representation of the sequence. The output of the attention scheme would be a weighted sum for all the input values, based on the different weights assigned to each value [21].

2.5.1 Self-attention scheme

Self-attention scheme is the original attention scheme that is widely used in natural language processing technology. Self-attention scheme contains three main steps. The first step is to compute the dot products between the current input and all the other inputs. Figure 2.7 shows the mathematical pipeline of computing the dot product. Each input sequence will compute the dot product with all the other input sequences and generate a list of results from $X_i^T X_1$ to $X_i^T X_T$ where X_i^T represents the transpose matrix of the current input sequence. After that step, it will use the Softmax function to normalize that dot product value. Formula 2.16 shows the mathematical details of the softmax function.

$$\sigma(Z)_i = \frac{\exp Z_i}{\sum_{j=1}^{K} \exp Z_j} \tag{2.16}$$

The third step is computing the context vector output. In the formula, we can see that every vector normalized by the Softmax function will be multiplied by each input sequence.

$$A_i = \sum_{j=1}^{T} a_{ij} x_j \tag{2.17}$$

However, the self-attention scheme did not have any kinds of parameters, which means it is not quite useful for learning a model. That is a motivation to design the trainable self-attention scheme, which is called a scaled dot product.

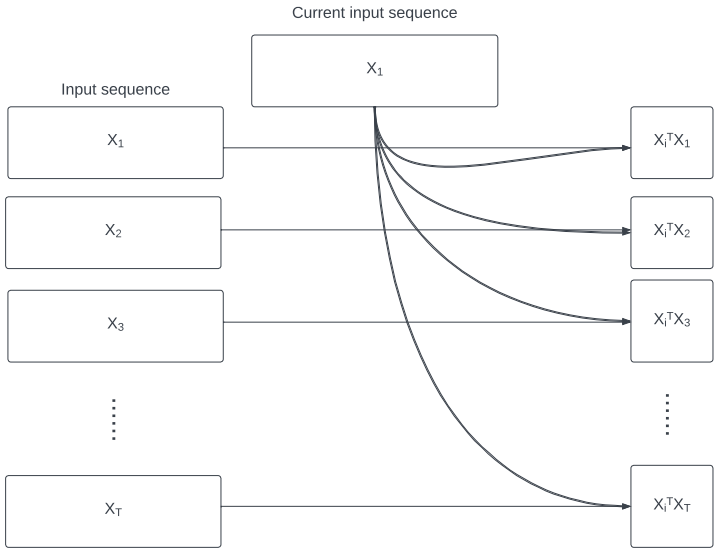


Figure 2.7: Dot product operation in self-attention scheme [21]

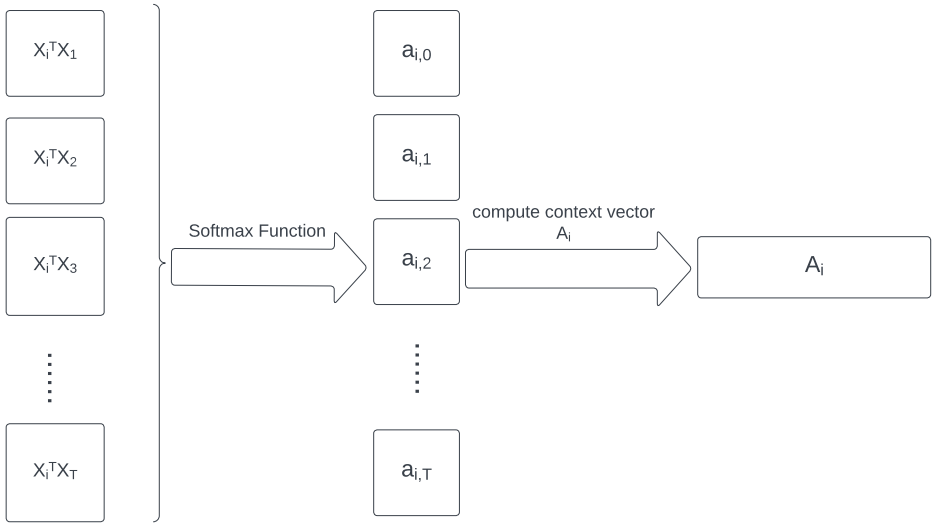


Figure 2.8: Softmax and context vector computation operation in self-attention scheme

We firstly add three weight matrices that can be updated during the training procedure. Those three matrices are W^q, W^k, and W^v. Then we use those three matrices to multiply each input word or the input data. Therefore, each input data will have three multiplication values which are query, key, and value.

Figure 2.9 shows the basic form of scaled dot product. It will take the query, key, and value as the input and compute the matrix multiplication between the value and the Softmax result of the query and key. Formula 2.18 is the mathematical principle behind the scaled dot product.

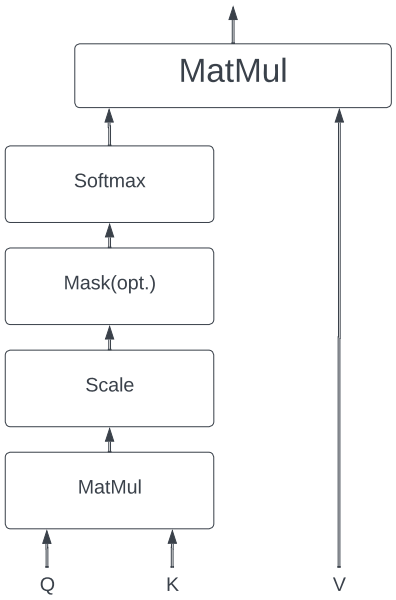


Figure 2.9: Basic form of scaled dot product [21]

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.18}$$

Based on the description above, we can understand how scaled dot product works by using Figure 2.10 as an example. For computing, the attention value for x_1, the scaled dot product uses the query of x_1 as the uniform query value and uses that value to do the multiplication with the key and value of the corresponding input from x_2 to x_T.

2.5.2 Multi-head attention scheme

Multi-head attention scheme uses several self-attention modules in parallel to compute the attention value for the input. In general, multi-head attention uses the scaled dot product as the module. After generating several self-attention values, the multi-head attention scheme will use concatenation and linear transformation to make sure the final output satisfies the expected dimensions. Figure 2.11 shows the basic form of the multi-head attention scheme.

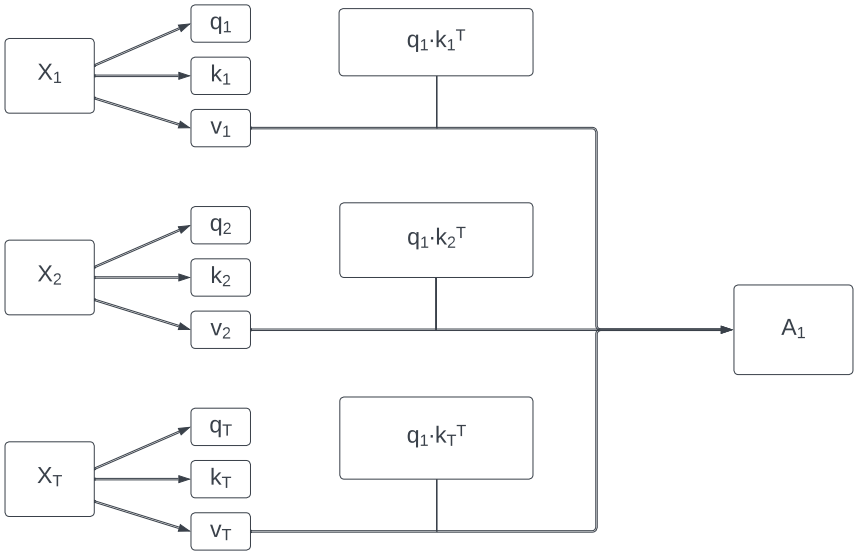


Figure 2.10: The principle of scaled dot product when computing attention value for a sequence

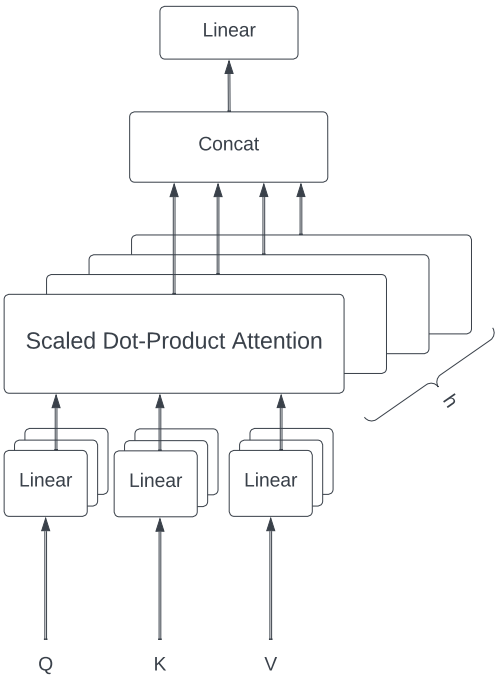


Figure 2.11: Basic form of multi-head attention scheme

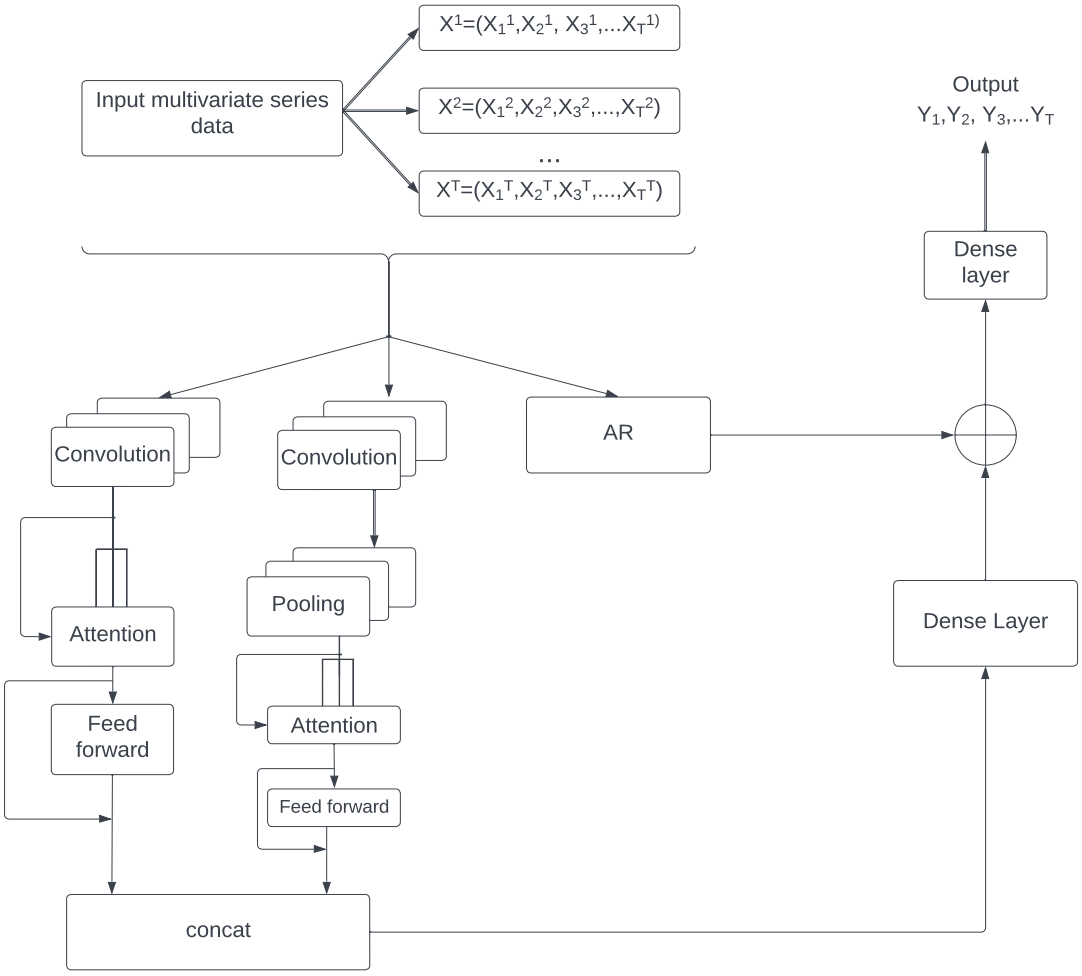


Figure 2.12: Basic structure of dual self-attention network (DSANet) [8]

2.6 Position-wise feed-forward network

Feed-forward network (FFN) is the simplest network in the Artificial intelligence field, which contains two linear computations with a ReLU function. The formula shows the mathematical function of FFN.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.19}$$

The parameters for the FFN function are shared in different input positions but different for each layer.

2.7 Dual self-attention network for multivariate time series forecasting

Dual self-attention network (DSANet) [8] combines several components: convolution layer, self-attention module, and the autoregressive (AR) model. The self-attention module has self-attention layers connected with a position-wise feed-forward layer which has been discussed in Section 2.5 and 2.6. Figure 2.12 shows the basic structure of the DSANet. From that figure, we can see that the idea of the self-attention module originated from the Transformer model, which stacks N times for the same module for learning the dependencies in different time series. From Section 2.2, we can see that the classical Autoregressive model has linear computation, which can be regarded as the linear component of the DSANet model. That is also the reason for adding the AR model to solve the nonlinearity drawback of the convolution layer and self-attention layer. For generating the output, DSANet used the dense layer, which combines the output from two convolutional layers and the self-attention layers and generates the output O_1. Finally, the network added O_1 with the output from the AR network to generate the final output series. We can change the output length by adjusting the output length of the dense layer so that the model can satisfy the output length requirement in this thesis work.

Chapter 3

Interactive awareness framework for roundabout route prediction

This chapter provides a principle description of the route prediction framework. AUTOPIA Group developed the interactive awareness framework, sponsored by the Polytechnic University of Madrid and the Spain National Research Council (CSIC).

3.1 Interaction awareness

When driving into a roundabout, before obtaining the planned route for the ego vehicle, interaction awareness is needed to ensure the decision and planning for the ego vehicle are correct and safe for the surrounding. In real-world driving, autonomous driving should face various kinds of surrounding scenarios. These surrounding scenarios include different kinds of participants, which will be surrounding vehicles, pedestrians, or other things appearing on the road during the driving. A good route planning approach should not be affected by these restrictions and produce a safe route that will also not be conflicted with other surrounding participants. Therefore, it is necessary to estimate the intention of the other drivers or the traffic participants. Based on that estimation, the ego vehicle could make a better decision and plan the route for the next time horizon. That is called interaction awareness. Figure 3.1 shows the primary pipeline for interaction awareness. From that figure, we can see that the whole Interaction awareness procedure contains three main steps: Find/reuse corridors, find interactions and compute intentions. The detailed descriptions for these three phases are given as follows.

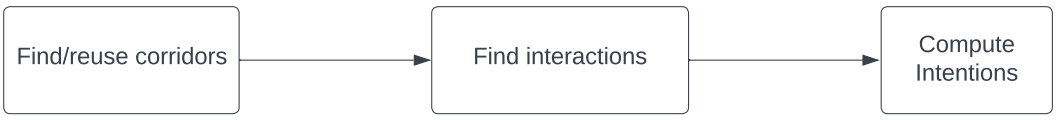


Figure 3.1: Interaction awareness procedure pipeline

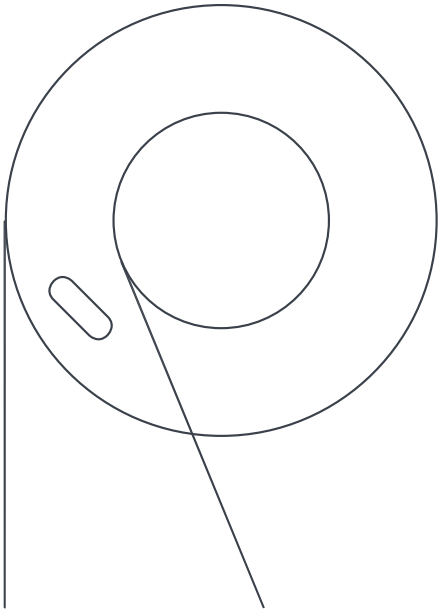


Figure 3.2: A scenario where the corridor will be removed

3.1.1 Finding corridors

The goal of finding/reusing corridors is to obtain all of the navigable corridors for the vehicle in the scene. For obtaining the current position of the ego vehicle, an efficient map for autonomous driving field called Lanelets [10] can be used. After obtaining the vehicle's current Lanelets, a graph search algorithm is implemented to generate the lanelets-sequence for each surrounding corridor. The next thing is to check whether each corridor should be expanded or deleted. When the vehicle is about to reach the end of the corridor for the current time horizon, an expansion execution to the corridor is necessary. To solve this task, a reasonable way is to compute the occupancy probability of each surrounding corridor by using a tool for Set-based Prediction of Traffic Participants (SPOT). Then an expansion execution for each corridor whose occupancy probability is above specific criteria. On the other hand, the corridor can also be deleted or removed if the angle between the centerline of the corridor and the vehicle measured orientation is larger than a criteria. Figure 3.2 illustrates a case when the corridor will be deleted. From that figure, we can see that the measured orientation has a significant difference from the centerline of the corridor. Based on the layout of the road, a grid is developed for each of the surrounding cars' corridors. Finally, the grid is constructed using this knowledge and the assumed route for the ego vehicle.

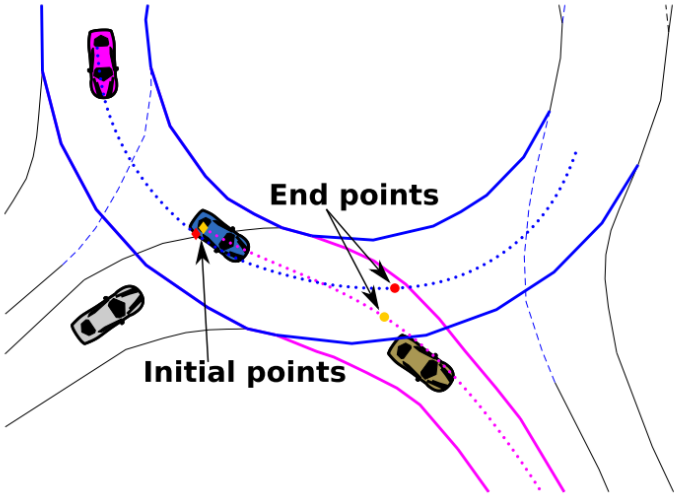


Figure 3.3: A scenario where the corridor has a dependency [19]

3.1.2 Find interactions

The aim of finding interaction is to discover the relationships between each traffic participant and the map elements. Three different kinds of interactions could be found: the corridor dependencies, the distance to the intersections, and the lateral relation.

The aim of finding the corridor dependencies (CD) is to check whether a collision could happen between the different traffic participants. To solve that, we can simply check whether the centerline of the corridor or the road are pairwise intersected, which could result in a collision between traffic participants. Figure 3.3 shows a specific scenario where the two corridors are intersected on the centerline and the corresponding collision point for these two corridors. From that figure, we can see that a pair of corridors is dependent. One is the corridor from the red vehicle, and the other is the corridor from the blue vehicle. The vehicle that firstly arrived at the initial points would lead to a dependency on the vehicle which arrived at the initial point later. Based on the initial points, the possible collision point will be grouped and sorted by considering the distance to the initial points. Then the collision point that is farthest from the initial points will be the dependency of the corridor.

The second kind of interaction would be the distance to the intersections (DI). It is reasonable to use Lanelets [10] identity of the current corridor to check which intersection the current corridor is crossing. To compute the distance to the intersection, first, we need to find the entrance to the center points of another corridor. After finding the entrance, the distance to the intersection is easy to compute by measuring the distance between the entrance and the projected position of the vehicle.

The third kind of interaction is lateral interaction (LI). To compute the lateral interaction, we need to obtain the previous intention (I_{t-1}) and the current expectation (E_t). These two variables will be further discussed in the Section 3.1.3. Then there will be a random value. If that random value is smaller than the probability value in the Table 3.1, then the LI is equal to 1. Otherwise, the LI will be equal to the probability value.

I_{t-1}	E_t	probability
0	0	0.1
0	1	0.5
1	0	0.5
1	1	0.9

Table 3.1: Lateral Intention computation [18]

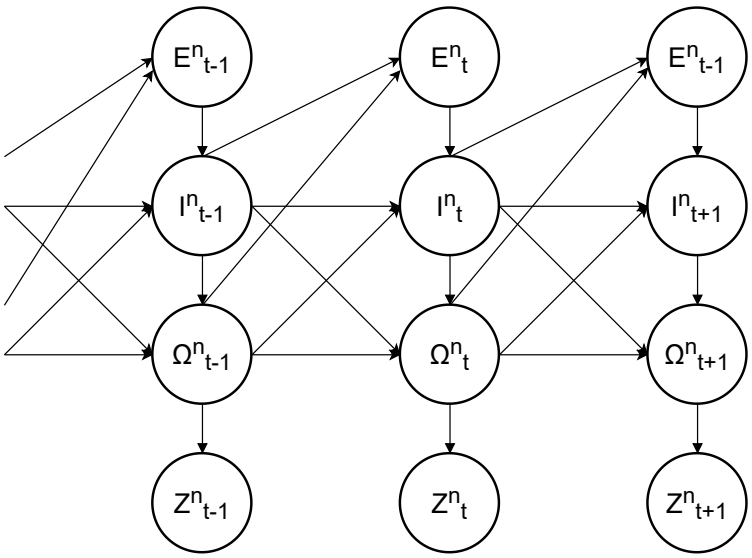


Figure 3.4: Bayesian network [18]

3.1.3 Compute intentions

In order to estimate the intention of the vehicle, the Dynamic Bayesian Network (DBN) [18] is implemented. DBN is a network that combines the original bayesian network with the particle filter algorithm in order to compute the intention estimation, which CSIC and UPM develop. Figure 3.4 shows the original bayesian network that was initiated in each vehicle. In Figure 3.4, several important variables could be further discussed.

E_t^n represents the expected behavior of the vehicle n at the t timestamp. It is divided into two parts. One is the longitudinal part and the other is the lateral part. The longitudinal part is often used to determine whether the vehicle should proceed to the road or stop at the intersection. The lateral part computes the probability of making a lane change. I_t^n represents the intention of the vehicle n at the t timestamp. It also contains two parts. The lateral part of I_t^n represents the corridor that the current vehicle wants to follow. Ω_t^n means the vehicle's position and speed, which is computed in each instant. Z_t^n represents the actual measurement of the vehicle extracted from the autonomous driving vehicle sensor.

Based on the variables in Figure 3.4, Formula 3.1 [19] produces a mathematical princi-

ple for modeling the driving scenario for the autonomous driving vehicle.

$$\begin{aligned} P(E_{0:T}, I_{0:T}, \Omega_{0:T}, Z_{0:T}) = P(E_0, I_0, \Omega_0, Z_0) * \prod_{t=1}^{T} * \\ \prod_{n=1}^{N} [P(E_t^n | I_{t-1} \Omega_{t-1}) * \\ P(I_t^n | \Omega_{t-1}^n I_{t-1}^n E_t^n) * \\ P(\Omega_t^n | \Omega_{t-1}^n I_{t-1}^n I_t^n) * \\ P(Z_t^n | \Omega_t^n)] \end{aligned} \tag{3.1}$$

In Formula 3.1, the $E_{0:T}$ represents the expected behavior from 0 timestamp to t timestamp. Similar with $E_{0:T}$, $I_{0:T}$ means the corridors the current vehicle wants to follow from 0 timestamp to t timestamp, and $\Omega_{0:T}$ means the vehicle position and speed from 0 timestamp to t timestamp, and $Z_{0:T}$ represents the actual measurement of the vehicle from 0 timestamp to t timestamp.

However, it is not possible to compute the vehicle intention by only using Formula 3.1 because only the measurement of the autonomous driving vehicle could be directly extracted. To solve that problem, the particle filter algorithm is necessary to obtain E_t^n, I_t^n, Ω_t^n based on the measurement of the vehicle Z_t^n. Each particle represents the current status of the autonomous driving vehicle, which includes all of the information about the corridors and interactions. Algorithm 1 [19] shows the basic principle of the particle filter pipeline for computing the vehicle intention.

Algorithm 1 Particle filter

Input: Corridor dependency, Distance to Intersection, Lateral Relation
Output: Lateral Intention
initialize particles
while True **do**
compute lateral expectation
compute lateral intention
compute longitudinal expectation
compute longitudinal intention
update pose and velocity
end while

From Algorithm 1, the particle filter contains several computation steps. The main step related to generating the vehicle position will be updating position and velocity. Therefore, in this thesis, we will focus more on the description of that step.

Update the vehicle position

The interactive awareness framework divided the final position P^t into two equal subpart P_1^t and P_2^t. P_1^t represents the displacement over the centerline of the corridor, and P_2^t is the free displacement of the particle. To compute P_1^t, the projection of the current position of the vehicle is required. Formula 3.2 [19] shows the mathematical principle for obtaining that position.

$$P_{proj}^t = P_{proj}^{t-1} + [\frac{v^{t-1} \Delta t}{sl}] \tag{3.2}$$

In Formula 3.2, P^t_{proj} represents the vehicle projected position. v^{t-1} is the vehicle velocity, and the *sl* represents the segment length. Using Formula 3.2, we can obtain P^t_1 by using Formula 3.3. In Formula 3.3 [19], *pv* represents the vector connected between two center points near P^t_1. cp is the matrix variable with $2*n$ dimension that represents the coordinate of the center point.

$$P^t_1 = cp(:, P^t_{proj}) + (v^{t-1}\Delta t \, mod \, sl) * \frac{pv}{||pv||} \tag{3.3}$$

where *mod* represents the modulo computation and the $cp(:, P^t_{proj})$ means a matrix which using the projected position vector to replace one column of the original matrix cp.

We can use Formula 3.4 to compute P^t_2 and the final position will be computing by computing the average of P^t_1 and P^t_2 using Formula 3.5 [19].

$$P^t_2 = P^{t-1} + v^{t-1}\Delta t \begin{bmatrix} cos(\theta^{t-1}) \\ sin(\theta^{t-1}) \end{bmatrix} \tag{3.4}$$

$$P^t = \frac{P^t_1 + P^t_2}{2} + rand \tag{3.5}$$

where *rand* represents the random value.

Update the vehicle velocity

The framework uses the basic physic function in Formula 3.6 to compute the velocity. a_s is the sample value from the initial acceleration distribution generated from the machine learning model implemented in this thesis. Based on the obtained velocity, the vehicle position could be updated and finally visualized in the experiment result.

$$v^t = v^{t-1} + a_s \Delta t \tag{3.6}$$

Chapter 4

Methodology

In this chapter, a description about the implementation approach will be given.

4.1 Dataset and data processing

The dataset that is fit for training the network can be obtained from the publicly available dataset openDD [3]. openDD contains public traffic data from the traffic participants present at the scenes. The frequency of capturing each data point is 100HZ. Each data includes pose, velocity, acceleration, size of each participant vehicle and the frames where each data belongs to. However, only the velocity and acceleration features could be retrieved from that dataset as useful features. Therefore, it is necessary to do a data pre-processing for obtaining curvature of the path and the other features which are also beneficial for model training. To solve that problem, CISC and Polytechnic University of Madrid (UPM) has cooperated to retrieve the curvature of the path and the distance to intersection by implementing a data pre-processing framework which models a roundabout environment for automated driving on Matlab. Figure 4.1 shows an example simulation environment that developed by CSIC and UPM. After binary tagging each frame as containing or not these participants, the intervals are grouped, and the parameters are obtained for each vehicle at each instant by modeling a roundabout environment for automated driving. Table 4.1 shows the features that are obtained after that modeling. In order to use these data for the purpose of thesis work, some of the data have been down-sampled and filtered to remove undesired participants vehicle, such as static vehicles, pedestrians, motorcycles, and large trucks. Therefore, the data between the data files are not consecutive with each other.

Parameters
Velocity time series
Curvature of the path
Distance to next intersection in the path
Distance to the next (leading) vehicle in path
Velocity of the next vehicle in the path

Table 4.1: Obtained parameters after modeling

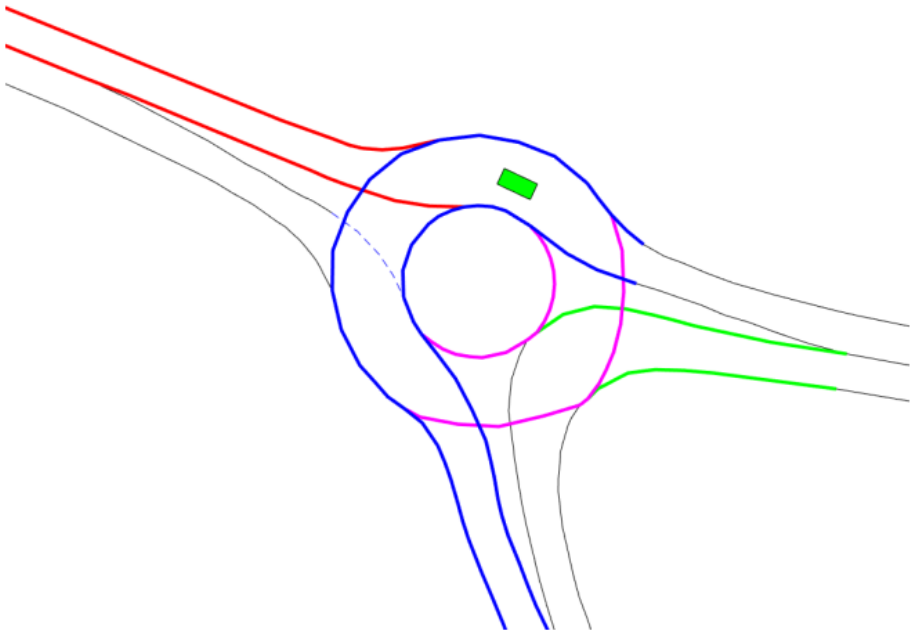


Figure 4.1: Example of possible corridors for a vehicle in a roundabout

The first thing before training the model is doing data preprocessing based on the dataset. For the time series prediction problem, a key point is to make sure the data in each series should be consecutive. What's more, it would be better to ensure the data-label are out of order. However, since the down-sampling process, each file is not contiguous in data. The whole dataset contains multiple participant vehicles, which also leads to the difficulties of creating data-label pair for training. For solving that problem, a reasonable approach will be, first of all, scanning through the whole dataset to search which data-label pair is from which participant vehicle, which means to create a key-value pair in which the key means the ID of the participant vehicle, and the value will be the all the data-label pairs that belongs to that participant vehicle. A HashMap or a dictionary data structure will handle that. Then, for ensuring the out-of-order properties, it will be simpler by only mixing the order of the participant vehicle's ID in the key-value pair. When creating the data-label pair, we used a number of features and each input data sequence has a length of 40, which means every data input contains information about several features in the past 4 seconds. Table 4.2 shows the features that we are currently using in our model. From that table, feature 4 to feature 15 are all the curvature of the path. That is because we divide the roundabout path into six parts, and each part has a pair of path curvature which contains a positive curvature and a negative curvature. The output sequence represents the distribution about the future 4 seconds acceleration. In our thesis work, we divided that distribution into 4 part, each part represents only 1 second acceleration distribution. We regard each distribution as a Gaussian distribution. Therefore, each part of that acceleration distribution will have a corresponding μ and σ, and in total the model would produce four parts of μ and σ to represent the whole 4 seconds acceleration distribution.

Input Feature
Vehicle acceleration or vehicle velocity
Distance to the next intersection in the path
Distance to the next (leading) vehicle in path
Velocity of the next vehicle in the path
Curvature of the path (contains 6 sub-variables)
Distance to the interaction (corridor dependency)
Velocity of the vehicle that leads to a corridor dependency
Priority of the interaction

Table 4.2: Input Feature

4.2 Evaluation matrix

We used four different kinds of evaluation matrix to evaluate and compare our model's performance, which are Mean square error, mean absolute error, average displacement error and final displacement error. The detailed mathematical formula will be explained in the following subsections.

4.2.1 Mean square error

For training the model, mean square error is used to update the network parameter in every training epoch. MSE is an evaluation matrix which measure the difference between the prediction series and the label series by computing the average of the square error of the each data in the predicted series and the corresponding data in the label series. Formula 4.1 shows the basic mathematical computation function of MSE.

$$MSE = \frac{1}{N} * \sum_{i=1}^{N} (\tilde{y}_t^i - y_t^i)^2 \tag{4.1}$$

where $\tilde{y}_t^i$ represents the i-th predicted value at timestamp t, and y_t^i represents the actual label value at timestamp t[1].

4.2.2 Root mean square error

Root mean square error (RMSE) is a reasonable evaluation metric for comparing model's performance. In many case, RMSE would be more useful to capture the large error. The mathematical function of RMSE is quite similar to that of MSE, which means RMSE could be computed by adding root to MSE result. Formula 4.2 shows the function of RMSE.

$$RMSE = \sqrt{\frac{1}{N} * \sum_{i=1}^{N} (\tilde{y}_t^i - y_t^i)^2} \tag{4.2}$$

4.2.3 Mean absolute error

Mean absolute error (MAE) is also an evaluation metric which compute the average error between the predicted series and label series. Although MAE is not able to give

[1] Same as Formula 4.2, 4.3

penalty to the large error, it still have the ability to describe the magnitude of the errors in the prediction series without considering the sign [9]. Formula 4.3 is the mathematical function of MAE.

$$MAE = \frac{1}{N} * \sum_{i=1}^{N} |\tilde{y}_t^i - y_t^i| \tag{4.3}$$

4.2.4 Average displacement error

Average displacement error(ADE) is widely using in self-driving field for measuring the quality of the motion prediction by comparing the ground truth displacement with the predicted displacement. Formula 4.4 shows the formula for ADE.

$$ADE = \frac{1}{T} * \sum_{i=1}^{T} \sqrt{(\tilde{x}^i - x^i)^2 + (\tilde{y}^i - y^i)^2} \tag{4.4}$$

where $\tilde{x}^i$ and $\tilde{y}^i$ represents predicted coordinate of the i-th trajectory point. x^i and y^i represents the ground truth coordinate of the i-th trajectory point.

4.2.5 Final displacement error

Final displacement error(FDE) is another common evaluation metric which measures the difference between the last predicted point and last ground truth trajectory point.

$$FDE = \sqrt{(\tilde{x}^T - x^T)^2 + (\tilde{y}^T - y^T)^2} \tag{4.5}$$

where $\tilde{x}^T$ and $\tilde{y}^T$ represents predicted coordinate of the final trajectory point. x^i and y^i represents the ground truth coordinate of the final trajectory point.

4.3 Comparison approach

We will conduct our experiment in four different kinds of models: Autoregressive, LSTM, GRU, DSANet. For comparing model's performance, several experiments has been established and implemented. First of all, since we need to convert the model output into the actual coordinate position of the vehicle, the model output could be velocity distribution or the acceleration distribution in the future 4 second. Therefore, we conduct an experiment for investigating the prediction performance of different features of model output.

Besides that, as discussed in the Section 2.1, the approach of generating the output will also be different. Therefore, the effect to the model performance using different kinds of output approach is also needed to be considered. Therefore, we compared the effect of performance between two different kinds of out approach: direct generating 40 output series(every input length=40, every output length=40) and iterating generating 40 output series(every input length=40, every output length=1). To test the model, as discussed in the Section 2.1, it will have a timing performance issue when using iterating generating 40 output series approach. Therefore, we first use 5 groups of input and output data of each vehicle to test both the prediction and timing performance of those two different kinds of output approach and then use total test dataset to test the model which using the output approach with better prediction and timing performance.

For comparing the result, we also used the different kinds of approach for processing the data before training the model. The data will be either using the global nomarlization approach or adding the batch nomarlization layer in the model. To compare the model performance, we used five evaluation matrices which has been discussed in the Section 4.2.

For the ADE and FDE, we used the evaluation framework from CSIC and Polytechnic University of Madrid, which will compute the result on the MATLAB platform. For evaluating on the MATLAB, we firstly use 8 groups of data which belongs to one simulation situation as the input to the model and generate corresponding output. We call this step as partial simulation. Each group of input data is from a specific vehicle. That means that in total we have data for 8 vehicles. After generating the outputs, we feed that output into MATLAB simulation environment and obtain the ADE and FDE evaluation result.

For further testing the generalization ability of our machine learning models, CSIC and Polytechnic University of Madrid tested all of our 4 models on MATLAB simulation framework using the whole four driving situation they currently have and make a comparison between our 4 models and their physics-based Dynamic Bayesian Network that mentioned in Section 3.1.3. We call this step as entire simulation. For entire simulation, they use multiple 40 length inputs and generate a long output series. For this testing, they still compute ADE and FDE score to measure the performance.

The other three evaluation metrics: MSE, RMSE, and MAE will be computed in the Pytorch environment. Table 4.3 and 4.4 show the models, the corresponding hyper-parameters of the models and the experiments that we conducted in this thesis.

Models	optimizer	learning rate (lr)	lr scheduler
Autoregressive LSTM GRU DSANet	Adam	0.005	Cosine Annealing

Table 4.3: Experiment model and the training configuration

Experiment	output approach	output feature
sliding window:40 data normalization: batchNorm	directly 40	acceleration velocity
	iteratively 40	accleration velocity
sliding window:40 data normalization: globalNorm	directly 40	acceleration velocity
	iteratively 40	accleration velocity

Table 4.4: Experiment design for investigativng the timing peformance of different producing output approaches

4.4 Implementation details

For down-sampling the scenario where the vehicle is facing the intersections or passengers, the data have been divided into 2000 files. Each file represent a scenario which reflect the surrounding environment of the autonomous driving vehicle. In total, the dataset contains around 800,000 entries which includes the velocity, acceleration and other motion information of the 130 vehicles. To train the model, we use the 80% of the overall dataset as the training set, and then choose 10% of the training set and the validation set. The rest of the dataset are used for testing the model.

We used Python 3.7 embedded in Anaconda library to implement the data processing and the model. For generating the dataset, the pytorch dataloader is used which takes the For training the model, the Pytorch-lightning library has been used. Pytorch-lightning [11] provides an interface for the Pytorch deep learning framework, which enables the development of deep learning experiment to be more efficient and easier. The main api of the Pytorch-lightning that will be often used when developing the model is called LightningModule that define a full deep learning system which could include either an individual model or a sets of models. LightningModule includes all the collections of the methods that will be used for training the models, which is the sub-class of nn.Module. Those collections are training loop, validation loop, test loop, the optimizer and the learning rate schedulers configuration, and then the computational graph of the model definition. The advantage of pytorch-lightning is that user does not need to consider how to feed the input in each epoch or setting the model to evaluate state or enabling the gradients. The pytorch-lightning trainer could handle all of these things automatically. Therefore, user could save more time on the development phase and pay more attention to the machine learning research.

We also test generalization ability of our model and make a comparison with the traditional physics-based model on the MATLAB. For testing the model generalization ability, CSIC and UPM chose four different kinds of driving simulation situations. Since we cannot see the code of the simulation environment of CSIC and UPM, we could only use the ONNX file as the media so that we can use that file format as the input to the MATLAB simulation environment. ONNX is a file format which represents the machine learning models. It helps the developers to easily import models from different training platform and enable the cross-platform development and testing. To do that, first of all, we need to use Pytorch-lightning platform to load the trained model from our local file system. And then export the ONNX file using ONNX package. Finally we can import our machine learning on the MATLAB by calling the command in MATLAB.

From the Section 2.2 we can see that the auto-regressive component is a basic linear computation. Therefore, for implementing the AR model, we can simply using the linear function in Pytorch library to represent the whole computational procedure. By changing the output dimension of the linear function, we can let the model either directly generate 40 output or iteratively generate 40 output.
For the LSTM model, the key is setting the hidden state and cell state. It is necessary to set those two variable to zero tensor for initializing the state of the model. The two different kinds of output approach could be adjusted by setting the different dimension of the linear function after receiving the output from LSTM model.
The implementation for the GRU model is quite similar to the LSTM. In the Section 2.3.3 we have discussed that GRU model does not have the cell state, which means the trainable

parameter will be less than LSTM. For initializing the state of the model, we can simply set the hidden state to zero tensor.
For the DSANet model, the implementation is more complex than those three kinds of model above. In Section 2.7, DSANet has a self-attention module which contains the multi-head attention scheme and the position-wise feed forward. The core principle behind the multi-head attention scheme is the scaled-dot-product function which related to the batch matrix-matrix product. From the Section 2.6 we can see that the position-wise feed forward network contains a ReLu function for choosing the maximal value between the xW_1 and 0, where the xW_1 is a convolution computation. The network uses another convolution computation for generating the output after the receiving the output from ReLu function. After implementing the self-attention module, the DSANet will stack that module for N times.

4.5 Experiment setup

In this thesis, all of the model will be training on the remote GPU in the server in TU Delft QCE department. Table 4.5 shows the main hardware configuration in that server. For training the model, the Anaconda virtual development environment has been install

GPU Name	cores	Memory	CPU speed
RTX 2080Ti	12	128GB	2.4GHz

Table 4.5: Hardware Configuration

in the server which contains the Python 3.9.7 Interpreter with the Pytorch 1.11.0 version. The Pytorch-lightning interface 1.6.5 version is also used on the top of Pytorch framework. After the training procedure, the ADE and FDE evaluation metric will be computed in the R2021b version MATLAB platform. Table 4.6 shows all of the software configuration information.

Software	Version
Pytorch(GPU version)	1.11.0
Pytorch-lightning	1.7.2
MATLAB	R2021a
CUDA	10.2
cuDnn	10.2

Table 4.6: Software Configuration

Chapter 5

Experimental results

This chapter gives an overview of the project's contributions. After this overview, this chapter will present multiple experiments and corresponding results and give some discussions according to the results.

5.1 Project contributions

In this thesis, TU Delft QCE department provides hardware support for all model training. In addition, this thesis project cooperates with the Spain National Research Council and the Polytechnic University of Madrid. The ADE and FDE evaluation matrix will be computed using the MATLAB simulation framework from CSIC and UPM.

5.2 Impacts of different prediction approaches for timing and prediction performance

We used five input groups for each car from the test set to predict the output and measured the computational time of the model used for producing the prediction. Table 5.1 shows the experiment result of all of the models when using two different kinds of prediction approaches[1].

Models	MSE	RMSE	MAE
AR(batchNorm)	0.3987	0.5651	0.4734
AR(globalNorm)	0.5840	0.6860	0.5743
LSTM(batchNorm)	0.4026	0.5668	0.4704
LSTM(globalNorm)	0.3768	0.5474	0.4553
GRU(batchNorm)	0.4122	0.5630	0.4821
GRU(globalNorm)	0.3720	0.5443	0.4531
DSANet(batchNorm)	0.3288	0.5271	0.4333
DSANet(globalNorm)	0.3635	0.5412	0.4470

Table 5.1: Result for directly 40 output approach (output feature: Acceleration)

[1]All the MSE, RMSE, and MAE results represent the average value for all 126 vehicles, same as Table 5.2, 5.3, 5.4, 5.5, 5.7

Models	MSE	RMSE	MAE
AR(batchNorm)	86973.57	282.6038	88.7438
AR(globalNorm)	2.1026	1.4018	0.9571
LSTM(batchNorm)	47119.74	206.1511	68.9448
LSTM(globalNorm)	2.0050	1.3673	0.9369
GRU(batchNorm)	26435.1	158.1275	58.9137
GRU(globalNorm)	0.5646	0.5974	0.5104
DSANet(batchNorm)	53012.36	225.3845	92.9970
DSANet(globalNorm)	0.3635	0.5412	0.4470

Table 5.2: Result for iterative 40 output approach (output feature: Acceleration)

Models	MSE	RMSE	MAE
AR(batchNorm)	1.9718	1.1627	1.0091
AR(globalNorm)	1.1864	0.9045	0.7389
LSTM(batchNorm)	3.6697	1.6926	1.4895
LSTM(globalNorm)	3.1624	1.6199	1.3554
GRU(batchNorm)	2.9144	1.4751	1.3056
GRU(globalNorm)	1.8807	1.1916	1.0216
DSANet(batchNorm)	0.8059	0.8487	0.6944
DSANet(globalNorm)	1.7140	1.1486	0.9550

Table 5.3: Result for direct 40 output approach (output feature: Velocity)

Models	MSE	RMSE	MAE
AR(batchNorm)	89883.9	288.0995	91.7547
AR(globalNorm)	1.6391	1.2511	0.8910
LSTM(batchNorm)	25061.99	150.2193	48.48
LSTM(globalNorm)	1.0988	0.9855	0.6330
GRU(batchNorm)	80268.2400	272.3700	87.4539
GRU(globalNorm)	2.6113	1.5825	0.9949
DSANet(batchNorm)	$4.2*10^8$	7994.053	2752.061
DSANet(globalNorm)	3.8438	1.9171	1.3349

Table 5.4: Result for iterative 40 output approach (output feature: Velocity)

5.2.1 Timing performance

Another important aspect that needs to be considered is the total time used to produce the prediction series. If the model needs to generate the prediction in the future 4 seconds time horizon, the total time used to generate the prediction series should not exceed 4 seconds. The reason is that the autonomous driving vehicle needs to use the prediction series from the output to produce the future vehicle path. In our experiment, the average time

of generating prediction series when using the iterative output approach is 5-10 seconds, which is quite time-consuming compared with the direct output approach.

5.2.2 Discussion

From the experiment results in Section 5.2, we can see that the average loss of the direct output approach is lower than the average loss of the iterative output approach. When using the iterative output approach, whose each input length is 40 and label length is 1, all models have a considerable loss when adding the batch normalization layer in front of the model structure. That is also a reason to avoid using the iterative output approach. Besides that, as discussed in the previous chapter, the iterative output approach could cause an accumulative loss because the model has to do iteration for N times to get the whole N length output series. Table 5.1 and Table 5.2 set a good example to explain that phenomenon .From those two tables, we can find that all models have a relatively lower loss value when using the direct output approach compared with the case when all models use the iterative output approach. Therefore, we can conclude that the direct output approach has a better prediction and timing performance. Considering the timing performance issue, in the next experiment, which uses all of the test datas et, this thesis will only conduct the comparison experiment with all of the models using the direct output approach.

5.3 Experiment results on the whole test dataset

5.3.1 Results for acceleration output

Table 5.5 shows the experiment results of all the models which use direct generating 40 acceleration output approach. Table 5.6 indicates the results for MATLAB simulation results for 8 vehicles (groups) data[2].

Models	MSE	RMSE	MAE
AR(batchNorm)	0.4462	0.5960	0.4997
AR(globalNorm)	0.4668	0.6091	0.6590
LSTM(batchNorm)	0.4696	0.5993	0.5167
LSTM(globalNorm)	0.4140	0.5631	0.4822
GRU(batchNorm)	0.4607	0.5944	0.5107
GRU(globalNorm)	0.4122	0.5630	0.4821
DSANet(batchNorm)	0.3589	0.5258	0.4530
DSANet(globalNorm)	0.3754	0.5316	0.4562

Table 5.5: Evaluation metrics result for 40 acceleration series output

From Table 5.5 we can see that, generally, when using the global normalization data processing method before training the model, the model has a better training effect with lower MSE, RMSE and MAE loss. However, when adding batch normalization into the DSANet network, the performance has a slight improvement compared with the case when DSANet with the global normalization method and all the other models. In Table 5.6, when

[2]In Table 5.6, all of the ADE and FDE results represents the average value for all the 8 vehicles. Same to Table 5.8

Models	ADE	FDE
AR(batchNorm)	1.6476	4.8591
AR(globalNorm)	1.6146	4.7749
LSTM(batchNorm)	1.6478	4.8569
LSTM(globalNorm)	1.5506	4.0614
GRU(batchNorm)	1.6480	4.8581
GRU(globalNorm)	1.5455	4.6452
DSANet(batchNorm)	1.2201	3.5391
DSANet(globalNorm)	1.6176	4.6603

Table 5.6: MATLAB partial simulation result based on the acceleration output series

using the 8 groups of data to test the model, the DSANet has a best performance when adding a batch normalization layer, which means the batch normalization layer helps the DSANet model to improve the generalization ability in some extent. Therefore, we can see that DSANet with batchNorm layer has the best performance for MATLAB to predict the vehicle's future path and relatively positive acceleration prediction accuracy when testing in one driving simulation situation.

5.3.2 Acceleration series prediction effect

For visualizing the prediction effect, we also use #33 vehicle as an example to present the acceleration prediction curve of all these 4 models. Figure 5.1 shows the prediction visualization.

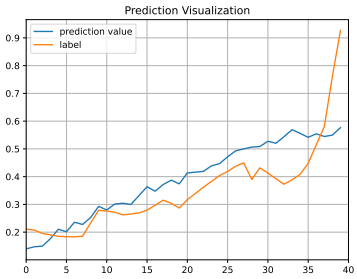
5.3.3 Vehicle trajectory prediction result in MATLAB for partial simulation

We also evaluated our model on the Matlab simulation platform from the Polytechnic University of Madrid to measure the performance of the simulated vehicle on the virtual road. Since the DSANet with batch normalization layer has a best acceleration prediction performance, we use Figure 5.2 to present the loss between the simulated vehicle path based on the model prediction and the actual vehicle position label when using the DSANet model with batch normalization layer and the case when direct generating 40 acceleration series output.

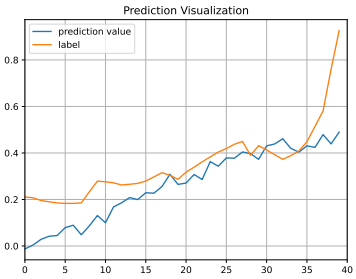
5.3.4 Results for velocity output

Table 5.7 shows the experiment results of all the models which using directly generating 40 velocity series output approach.

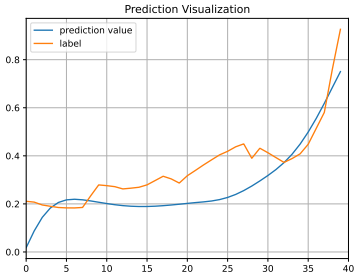
From Table 5.7 we can see that MSE, RMSE, and MAE for all of the models has a significant increase when using velocity serves as the label for model training. What's more, the ADE and FDE values in Table 5.8 also has an increase compared the values in Table 5.6. The possible reason is that we use MSE as the model's loss function to minimize the loss value in each epoch. From Formula 4.1 in Chapter 4, we can see that MSE will focus more on the outliers, which means MSE will give a larger weight to the value with a higher difference to the label. Therefore, the models will update their parameters to minimize the outliers loss. However, that will cause a decrease in the model's performance,



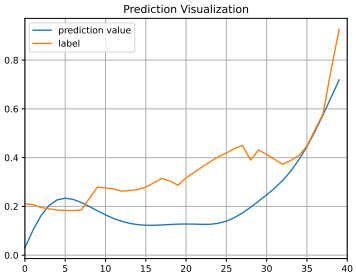
(a) Acceleration prediction visualization of Auto-regressive (batchNorm)



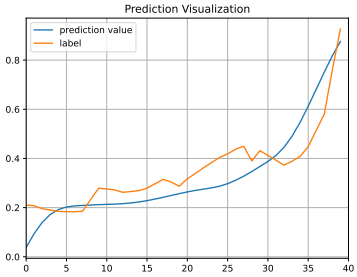
(b) Acceleration prediction visualization of Auto-regressive (globalNorm)



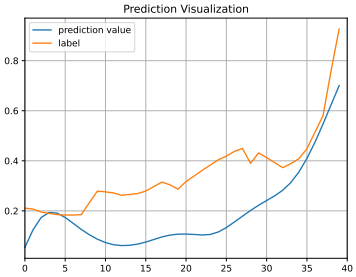
(c) Acceleration prediction visualization of LSTM (batchNorm)



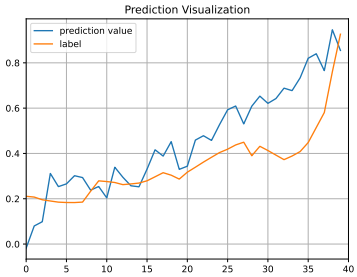
(d) Acceleration prediction visualization of LSTM (globalNorm)



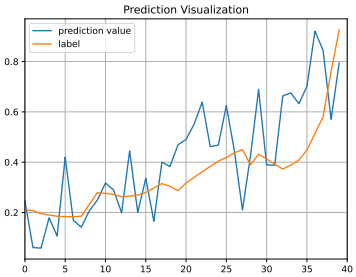
(e) Acceleration prediction visualization of GRU (batchNorm)



(f) Acceleration prediction visualization of GRU (globalNorm)

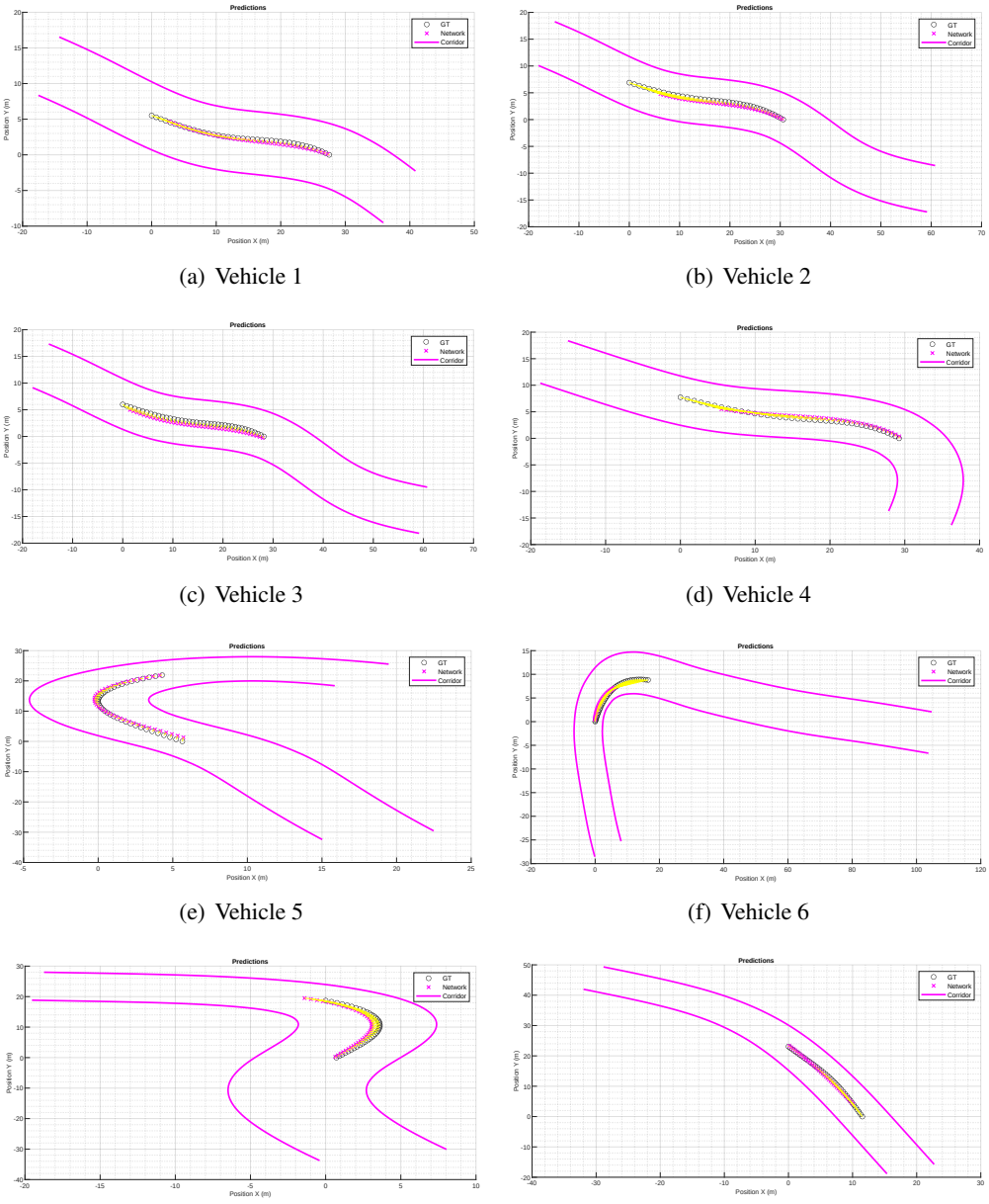


(g) Acceleration prediction visualization of DSANet (batchNorm)



(h) Acceleration prediction visualization of DSANet (globalNorm)

Figure 5.1: Acceleration prediction visualization for direct generating output approach (in vehicle 33)



(a) Vehicle 1

(b) Vehicle 2

(c) Vehicle 3

(d) Vehicle 4

(e) Vehicle 5

(f) Vehicle 6

(g) Vehicle 7

(h) Vehicle 8

Figure 5.2: Partial simulation result on Matlab simulation framework for the DSANet model in direct 40 acceleration series output approach

Models	MSE	RMSE	MAE
AR(batchNorm)	2.4549	1.3159	1.1344
AR(globalNorm)	2.0172	1.1991	1.0063
LSTM(batchNorm)	6.4188	2.2084	1.9736
LSTM(globalNorm)	3.1624	1.620	1.3554
GRU(batchNorm)	3.6345	1.6658	1.4726
GRU(globalNorm)	2.2801	1.3228	1.1393
DSANet(batchNorm)	4.3838	1.8463	1.5652
DSANet(globalNorm)	2.1503	1.2923	1.0763

Table 5.7: Evaluation metrics result for 40 velocity series output

Models	ADE	FDE
AR(batchNorm)	1.5489	4.3884
AR(globalNorm)	1.8467	5.3800
LSTM(batchNorm)	2.3485	6.7520
LSTM(globalNorm)	1.9900	5.7455
GRU(batchNorm)	2.3299	6.4604
GRU(globalNorm)	2.2230	6.4787
DSANet(batchNorm)	2.5564	7.7609
DSANet(globalNorm)	2.4556	7.0782

Table 5.8: MATLAB partial simulation result based on the velocity output series

which means the model could not learn the general knowledge from the data but focus more on the data with higher loss.

When comparing the Table 5.6 and Table 5.8, we can see that all of the models has a better ADE and FDE loss value when using the acceleration feature as the output feature. As discussed in Chapter 3, the vehicle path will be transferred based on the acceleration series. Suppose the model's prediction is velocity series. In that case, MATLAB framework will have to first transfer the velocity prediction series into the acceleration series by computing the formula $acceleration = \frac{velocity}{t}$ and use that computed acceleration series to predict the vehicle path. That will also cause the accumulative error since the error will have during the procedure of transferring the velocity series into the acceleration series.

From the prediction result in Table 5.8, we can see that AR model has the best training simulation evaluation score (ADE and FDE). We can also see that the attention-based model DSANet has a better prediction performance than the RNN-class models, LSTM and GRU. A possible reason is that LSTM and GRU model's generalization ability is worse than DSANet. From the introduction of the DSANet model structure in Chapter 2, we can see that the DSANet has an attention scheme that is stacked for N times. An attention scheme can grasp complex information in a long sentence. Whereas the LSTM and GRU models only have fix number of memory gates, which cannot adapt to the case when the input contains more complex and implicit information such as the motion change in the input sequence of the vehicle in this thesis. That is why RNN-class models do not have a better effect than DSANet.

5.3.5 Velocity series prediction effect

Figure 5.3 shows the velocity prediction curve of all these 4 models. Like acceleration series visualization, we also use #33 as the example to present the velocity prediction effect.

5.3.6 Vehicle trajectory prediction result in MATLAB for partial simulation

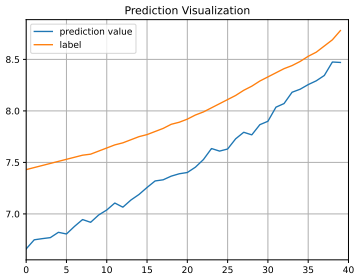
Since the AR with batch normalization data processing method has a best velocity prediction performance in one driving situation, we use Figure 5.4 to present the loss between the simulated vehicle path based on the model prediction and the actual vehicle position label when using the AR model with batch normalization layer and the case when direct generating 40 velocity series output.

5.3.7 Model generalization ability test

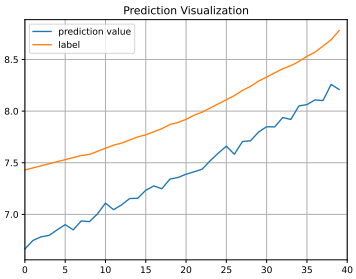
We also use MATLAB platform to test our model in 4 different kinds of simulated driving situation to see generalization ability by taking the ONNX file of all our machine learning models as the input and make a comparison between the machine learning model and the traiditional physics-based model. Table 5.9 shows the experiment results. From the Table 5.9, we can see the AR model with global normalization produces a good performance when predicting acceleration series, which is better than the physics-based model. The possible reason is that the most of the data for the vehicle in our dataset have similar trends. When capturing the data, most of the vehicle are doing similar driving movement: 1. Accelerating speed 2.Uniform speed 3. Decelerating speed. When using global normalization data processing approach, it can contains the global information which can be beneficial for models to learn the global knowledges. In addition, as discussed in the Chapter 2, AR model has a combination of the linear computation, which means the AR model has the ability to use linear regression function for learning similar trends for the data.

Figure 5.5 and 5.6 show the simulation visualization when using MATLAB. Unfortunately, due to the incompatibility between the MATLAB and the Pytoch platform, we cannot successfully import the GRU model to run the simulation test. For each figure, there are three sub-figures. The first sub-figure represents the computational time for each model to generate the output. The second sub-figure visualizes the ADE evaluation result during the whole simulation time interval. The third sub-figure presents the FDE evaluation result during the whole simulation time interval. For all of these three sub-figures, the x-axis represents the iteration of the vehicle. An iteration represents 0.1 second simulation. Therefore, if there are 300 iterations in one simulation situation, the total simulation time for that simulation situation will be 30 seconds. For the second sub-figure and third sub-figure, the y-axis represents the ADE and FDE value. Because the unit of ADE and FDE is distance, we use distance as the unit at y-axis.

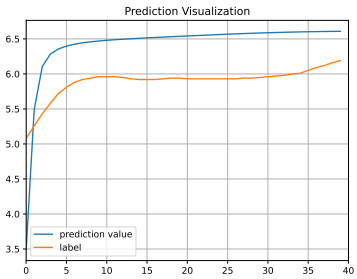
According to the experiment data from Table 5.7 to Table 5.8, we can see that the velocity results, in overall, do not have a better result. Therefore, in this section, we will not run the simulation situation for the model that generates velocity output.



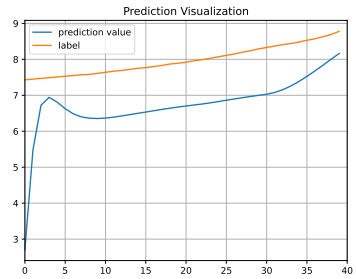
(a) Velocity prediction visualization of Autoregressive (batchNorm)



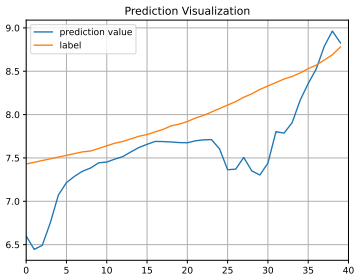
(b) Velocity prediction visualization of LSTM (globalNorm)



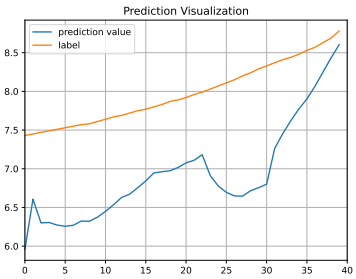
(c) Velocity prediction visualization of LSTM (batchNorm)



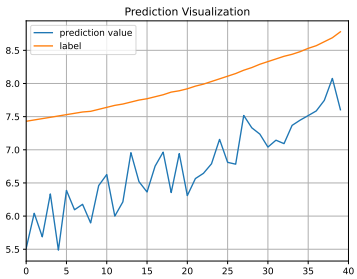
(d) Velocity prediction visualization of LSTM (globalNorm)



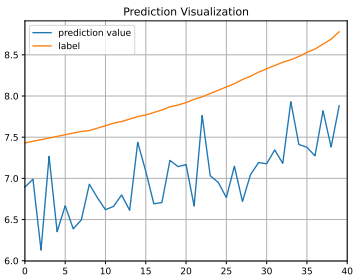
(e) Velocity prediction visualization of GRU (batchNorm)



(f) Velocity prediction visualization of GRU (globalNorm)

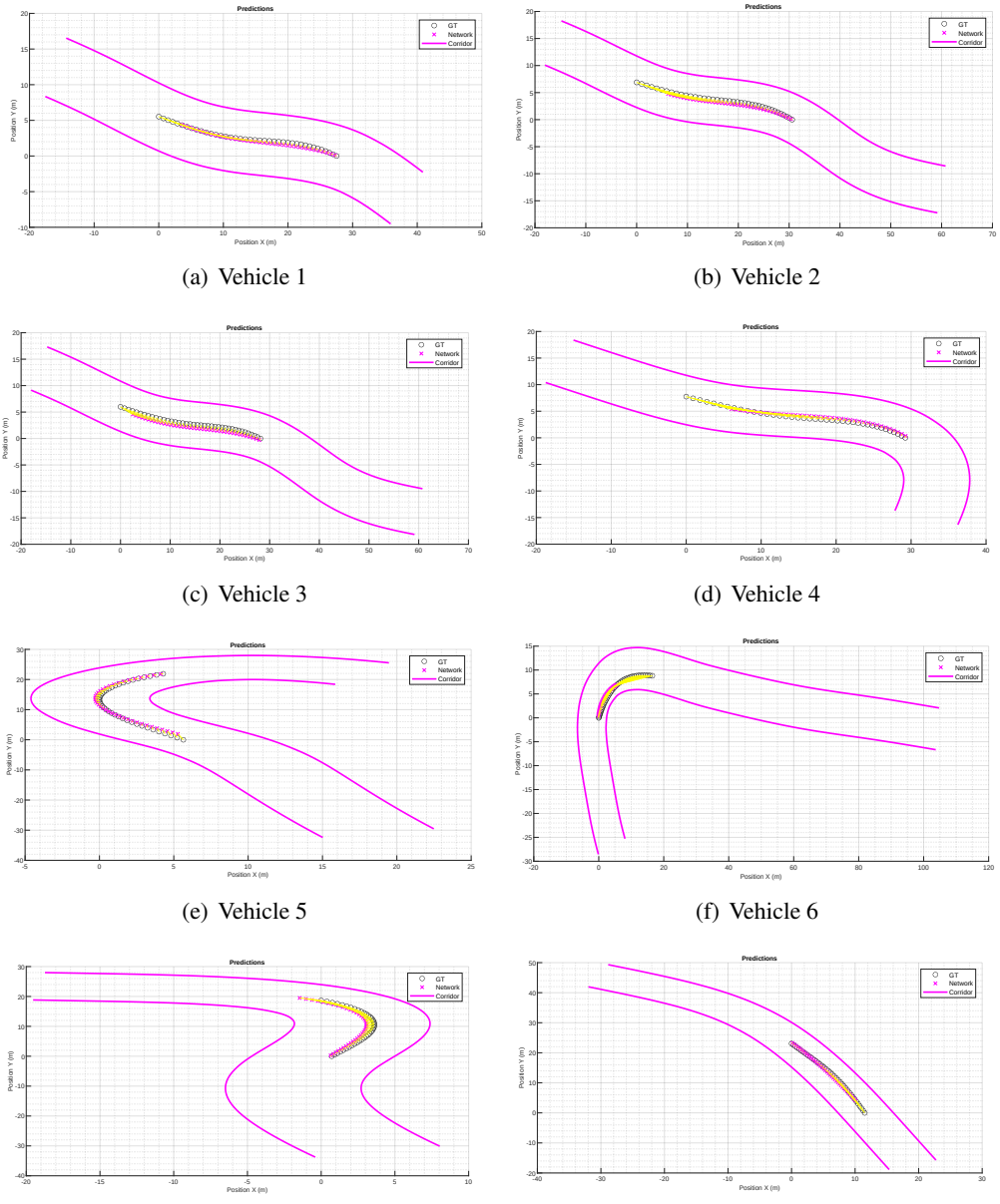


(g) Velocity prediction visualization of DSANet (batchNorm)



(h) Velocity prediction visualization of DSANet (globalNorm)

Figure 5.3: Velocity prediction visualization for direct generating output approach (in vehicle 33)



(a) Vehicle 1

(b) Vehicle 2

(c) Vehicle 3

(d) Vehicle 4

(e) Vehicle 5

(f) Vehicle 6

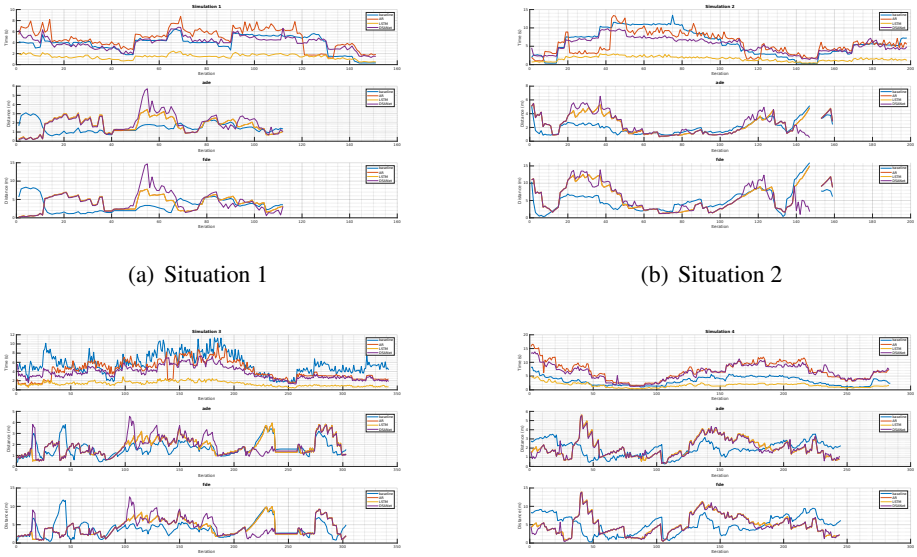
(g) Vehicle 7

(h) Vehicle 8

Figure 5.4: Partial simulation result on MATLAB simulation framework for the AR model in direct 40 velocity series output approach

Simulation situation	Models	ADE	FDE
Situation 1	AR (batchNorm)	1.76	4.01
	AR (globalNorm)	**1.18**	**2.72**
	LSTM (batchNorm)	1.77	4.02
	LSTM (globalNorm)	1.77	4.02
	DSANet (batchNorm)	1.73	3.75
	DSANet (globalNorm)	1.69	3.95
	Physics-based model	1.43	3.22
Situation 2	AR (batchNorm)	2.37	5.81
	AR (globalNorm)	**1.67**	**4.54**
	LSTM (batchNorm)	2.38	5.82
	LSTM (globalNorm)	2.36	5.78
	DSANet (batchNorm)	2.28	5.35
	DSANet (globalNorm)	2.35	5.80
	Physics-based model	1.92	4.83
Situation 3	AR (batchNorm)	1.81	4.24
	AR (globalNorm)	**1.57**	3.88
	LSTM (batchNorm)	1.81	4.25
	LSTM (globalNorm)	1.82	4.27
	DSANet (batchNorm)	1.81	4.09
	DSANet (globalNorm)	1.81	4.24
	Physics-based model	1.58	**3.73**
Situation 4	AR (batchNorm)	2.01	4.96
	AR (globalNorm)	**1.41**	**3.54**
	LSTM (batchNorm)	2.01	4.97
	LSTM (globalNorm)	1.99	4.88
	DSANet (batchNorm)	1.91	4.76
	DSANet (globalNorm)	1.98	4.96
	Physics-based model	1.91	5.05

Table 5.9: ADE and FDE results of MATLAB entire simulation for all four driving situations for models generating acceleration series output



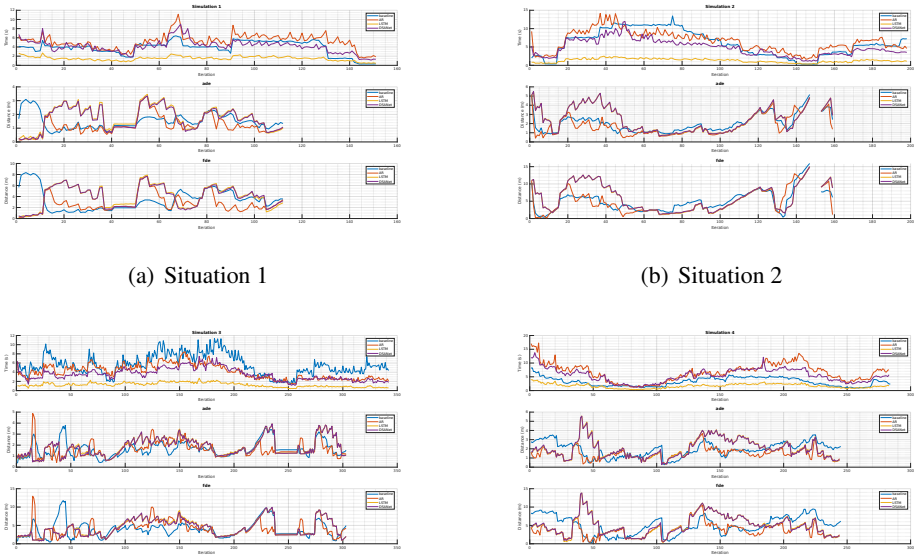
(a) Situation 1

(b) Situation 2

(c) Situation 3

(d) Situation 4

Figure 5.5: Entire simulation result on MATLAB simulation framework for all four driving situations with the machine learning models using batch normalization approach and generating acceleration output



(a) Situation 1

(b) Situation 2

(c) Situation 3

(d) Situation 4

Figure 5.6: Entire simulation result on MATLAB simulation framework for all four driving situations with the machine learning models using global normalization approach and generating acceleration output

Chapter 6

Conclusion and future work

This chapter gives the overall conclusion related to the thesis based on the experimental results in Chapter 5. Besides that, the future improvements for the thesis work will also be discussed the this chapter.

6.1 Conclusion

In this thesis, we used a data-driven modeling approach to capture the complexity of autonomous driving tasks. Our experiments show that machine learning models can indeed learn driving patterns from the dataset and use that knowledge to create an autonomous driving model, which means that machine learning models are capable to capture the complexity of autonomous driving tasks.

From the experiment results in Chapter 5, we can see that, overall, AR models have the best prediction performance when using the global normalization data processing approach. AR has a best generalization ability in all four driving situations and can slightly be improved compared with the Physics-based baseline model. According to Table 6.1, the AR model has the best ADE and FDE result in simulation 1, simulation 2, and simulation 4 where it has the following ADE values in the four simulation situations: 1.18, 1.67, 1.57, 1.41. Although the AR model does not have the best FDE result in the third simulation situation, it does not cause a significant decrease compared with the value in the physics-based model. Although DSANet also has a good ADE and FDE score when testing on one driving situation, from the prediction visualization result in Figure 5.1 (g) and (h), we can see the prediction curve of DSANet contains a lot of noise. Although we have 800,000 data points overall, most of the data has a similar trend. Therefore, a possible reason behind this phenomenon is the over-fitting issue when training the DSANet model.

Although machine learning models have several advantages compared with the traditional physics-based model, neural network based models also have limitations. In many cases, when the models are not performing very well, it is very hard to identify the causes of this limited performance because neural network based models are considered as a black box. In addition, the computational time of the machine learning model still needs to be considered. From Figure 5.5 and Figure 5.6, although the AR model has a good prediction performance, overall, its time latency is the highest compared with all the other models.

Simulation Situation	Models	ADE	FDE
Situation 1	AR (globalNorm)	**1.18**	**2.72**
	Physics-based model	1.43	3.22
Situation 2	AR (globalNorm)	**1.67**	**4.54**
	Physics-based model	1.92	4.83
Situation 3	AR (globalNorm)	**1.57**	3.88
	Physics-based model	1.58	**3.73**
Situation 4	AR (globalNorm)	**1.41**	**3.54**
	Physics-based model	1.91	5.05

Table 6.1: ADE and FDE results comparison between AR and physics-based model for all four driving situations

6.2 Future work

Motion prediction and decision-making control techniques require high accuracy to ensure the safety of the autonomous driving vehicle. However, the results of machine learning models are only accurate in the context of the data used to train the model, which means that the decisions taken by the model may sometimes not be fit to the actual situation or the current road environment. An alternative approach is to use reinforcement learning models to learn the data in an interactive way. The reason is that autonomous driving vehicles require to actively capture surrounding data and interact with the road and other road participants and make decision in real-time [16]. The reinforcement learning model could learn the data from the current driving environment and make predictions according to the current state of the road environment, which makes the prediction output more accurate with respect to the current driving environment and improve the prediction accuracy to some extent. In the future, it is recommended to conduct the experiment for the reinforcement learning model using the same dataset and compare the results with other models used in this thesis.

What is more, the current machine learning models require to be generalized to more driving scenarios, which would result in improving the robustness of the model further. Since we down-sampled the data when the autonomous vehicle is decelerating, the current dataset only contains the data when the vehicle does not have pedestrian or intersection ahead. Therefore, it is recommended to investigate whether the model performance and the generalization ability could be further improved by adding the deceleration data into the current dataset.

Bibliography

[1] Ebru Arisoy et al. “Bidirectional recurrent neural network language models for automatic speech recognition”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 5421–5425. DOI: 10.1109/ICASSP.2015.7179007.

[2] Walter Brenner and Andreas Herrmann. “An Overview of Technology, Benefits and Impact of Automated and Autonomous Driving on the Automotive Industry”. In: *Digital Marketplaces Unleashed*. Ed. by Claudia Linnhoff-Popien, Ralf Schneider, and Michael Zaddach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 427–442. ISBN: 978-3-662-49275-8. DOI: 10.1007/978-3-662-49275-8_39. URL: https://doi.org/10.1007/978-3-662-49275-8_39.

[3] Antonia Breuer et al. *openDD: A Large-Scale Roundabout Drone Dataset*. 2020. DOI: 10.48550/ARXIV.2007.08463. URL: https://arxiv.org/abs/2007.08463.

[4] CDC. *Motor Vehicle Crash Deaths*. https://www.cdc.gov/vitalsigns/motor-vehicle-safety/index.html#:~:text=Major%20risk%20factors%20for%20crash,more%20than%209%2C500%20crash%20deaths.

[5] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. DOI: 10.48550/ARXIV.1409.1259. URL: https://arxiv.org/abs/1409.1259.

[6] Hyunjoo Jin David Shepardson. *GM bets $3.5 billion more on self-driving tech unit as SoftBank exits*. https://www.reuters.com/business/autos-transportation/gm-buys-softbank-vision-funds-stake-cruise-21-bln-2022-03-18/#:~:text=In%202018%2C%20SoftBank%20invested%20%24900,bringing%20its%20stake%20to%2020%25.

[7] Sepp Hochreiter and Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[8] Siteng Huang et al. “DSANet: Dual Self-Attention Network for Multivariate Time Series Forecasting”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 2129–2132. ISBN: 9781450369763. DOI: 10.1145/3357384.3358132. URL: https://doi.org/10.1145/3357384.3358132.

[9] JJ. *MAE and RMSE — Which Metric is Better?* https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d Accessed Mar 3, 2016.

[10] *Lanelets2: A high definition map framework for the future of Automated Driving*. URL: https://ieeexplore.ieee.org/document/8569929.

[11] Himanshu Lawaniya. “Getting Started with PyTorch Lightning”. In: (May 2020).

[12] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.08458. URL: https://arxiv.org/abs/1511.08458.

[13] Mike Ramsey. *Self-Driving Cars Could Cut Down on Accidents, Study Says. The Wall Street Journal*. https://www.wsj.com/articles/self-driving-cars-could-cut-down-on-accidents-study-says-1425567905.

[14] Rajib Rana. “Gated recurrent unit (GRU) for emotion classification from noisy speech”. In: *arXiv preprint arXiv:1612.07778* (2016).

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[16] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving*. 2016. DOI: 10.48550/ARXIV.1610.03295. URL: https://arxiv.org/abs/1610.03295.

[17] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2019.132306. URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

[18] Vinicius Trentin et al. “A Comparison of Lateral Intention Models for Interaction-aware Motion Prediction at Highways.” In: *VEHITS*. 2021, pp. 180–191.

[19] Vinicius Trentin et al. “Interaction-Aware Intention Estimation at Roundabouts”. In: *IEEE Access* 9 (2021), pp. 123088–123102. DOI: 10.1109/ACCESS.2021.3109350.

[20] Oskar Triebe, Nikolay Laptev, and Ram Rajagopal. *AR-Net: A simple Auto-Regressive Neural Network for time-series*. 2019. DOI: 10.48550/ARXIV.1911.12436. URL: https://arxiv.org/abs/1911.12436.

[21] Ashish Vaswani et al. “Attention is All you Need”. In: *ArXiv* (2017).

[22] Jie Wang and Zihao Li. “Research on Face Recognition Based on CNN”. In: *IOP Conference Series: Earth and Environmental Science* 170 (July 2018), p. 032110. DOI: 10.1088/1755-1315/170/3/032110. URL: https://doi.org/10.1088/1755-1315/170/3/032110.

[23] Yong Yu et al. “A review of recurrent neural networks: LSTM cells and network architectures”. In: *Neural computation* 31.7 (2019), pp. 1235–1270.