Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Predictive Techniques for Scene Understanding by using Deep Learning in Autonomous Driving

PhD. Thesis Presented by

Carlos Gómez Huélamo

2023

Universidad de Alcalá

PhD. Program in Electronics: Advanced Electronic Systems. Intelligent Systems

Predictive Techniques for Scene Understanding by using Deep Learning in Autonomous Driving

PhD. Thesis Presented by

Carlos Gómez Huélamo

Advisors

Dr. Luis Miguel Bergasa Pascual

Dr. Rafael Barea Navarro

Alcalá de Henares, TBD

A mi Madre, allá donde esté .....

"En este vasto mundo
navegáis en pos de un sueño,
surcando el ancho mar
que se extiende frente a vosotros.
El puerto de destino es el mañana
cada día más incierto.
Encontrad el camino,
cumplid vuestros sueños,
estáis todos en el mismo barco
y vuestra bandera es la libertad"
Opening 3 de One Piece
Autor: The Babystars

Acknowledgements

Esta Tesis Doctoral supone el culmen a cuatro años (Abril 2019 - Abril 2023) realmente duros, cargado de emociones, triunfos, pandemias, estafas y tropiezos, todo a partes iguales. Este es probablemente (aunque como diría Sean Connery interpretando a James Bond en 1983, *Never Say Never Again*) mi último gran documento individual, académicamente hablando.

Durante mi etapa universitaria (2013 hasta el momento, 2023) he tenido ciertos momentos puntuales en los que he sentido un salto cualitativo como profesional: El primero fue en el segundo cuatrimestre de segundo de carrera, cuando las cosas se pusieron tensas con Control II e Informática Industrial. Vaya sudores. El segundo probablemente fue con el fallecimiento de mi madre durante mi ERASMUS+ en Irlanda. Duros y oscuros momentos, alejado de mis seres queridos. El tercer momento llega en segundo de máster, durante mi querido ERASMUS+ en Finlandia, donde compagino una estancia preciosa en Tampere con el máster y un pre-inicio de doctorado. Me equivoqué al empezar tan pronto con la beca, "queriendo cobrar" cuanto antes, en vez de terminar tranquilamente el TFM y plantear la tesis, pero eso no lo sabría hasta tiempo después. Pero no es hasta la tesis donde empezaron los quebraderos de cabeza reales. Continuamente altibajos, mala planificación por mi parte, momentos puntuales donde me equivoqué rotundamente al empecinarme en soldar una estructura compleja para nuestro vehículo sin ayuda, no estudiar PyTorch tras el congreso WAF 2018 tras la sugerencia de mi tutor, no enfocarme en técnica individual hasta bien entrado el doctorado, no querer hacer nada hasta que no tuviese la teoría perfectamente asimilada, tener demasiado respeto a la Inteligencia Artificial y escurrir el bulto de mi tesis en un compañero mientras yo me dedicaba a integrar y corregir los bugs del grupo que para mí *era lo fácil*. Mal. Todo mal. Pero todo cambió tras mi segunda estancia, en Estados Unidos, cuando tras llorar por no entender el camino a seguir, nadie que me ayudara, decidí crear mi propio camino, con paciencia y fé, práctica y error compaginado con lectura de artículos, para mejorar mi confianza y autoestima, y finalmente logré empezar a entender lo que era el Deep Learning. Gracias a todos mis errores, desventuras y discusiones, a día de hoy, excepto momentos inevitables, me encuentro con muchísima capacidad para atacar y gestionar prácticamente cualquier problema, consultar documentación y organizarme, aunque esta sigue siendo mi tarea

pendiente.

Cada año, desde hace ya varios, mi primera publicación en Instagram viene seguida de la frase "Trabaja duro en silencio y deja que tu éxito haga todo el ruido". Filosofía Kaizen, de mejora y aprendizaje continuo, para así cada día entender el mundo un poquito mejor. Si toda la dedicación y estudio que he depositado en este trabajo sirven para algo en mi futuro, sé que todo el esfuerzo habrá merecido la pena.

Después de este particular monólogo, a lo cual soy muy propenso y de lo cual mis amigos y compañeros no cesan en su empeño de recordádmelo, debo, como no puede ser de otra manera, dar paso a los agradecimientos.

En primer lugar, me gustaría agradecer a mis profesores del grupo RobeSafe, especialmente a mis tutores Luis Miguel Bergasa Pascual y Rafael Barea Navarro, por ofrecerme estar en el grupo (así como aguantarme) durante todos estos años e intentar que tuviésemos el mejor *experimental setup* y *roadmap* en el laboratorio, aunque no fuese siempre sencillo. Sin dudas considero realmente interesante la temática propuesta en esta tesis doctoral, predicción de agentes en el contexto de vehículo inteligente, ya que entra en el plano filosófico sobre cómo razonar el futuro de los objetos y cómo podría afectar a la capacidad ejecutiva del agente que deba tomar una decisión. Mi mente hace tiempo que cambió y me fijo siempre que conduzco de todo lo que intento reproducir con mis estudios. Habrá que seguir esta tendencia muy de cerca en los próximos años, porque personalmente considero que sus aplicaciones son fantásticas.

A mis tutores en las estancias de doctorado, Christoph Stiller y Eduardo Molinos en el Karslruhe Institute of Technology (KIT, Alemania) y Wei Zhan y Masayoshi Tomizuka en la University of California, Berkeley (UCB, Estados Unidos). Se suele decir que unas veces se gana y otras se aprende, y yo en estas estancias quizás aprendí demasiado ... No obstante, me guardo grandísimos momentos (admirar las secuoyas gigantes o hacer mi primera escalada en roca entre ellos) y amigos, como Su Shaoshu o Frank Bieder, con los que aún guardo un cierto contacto.

A mis compañeros, mejor dicho, amigos, de laboratorio: Javier Araluce, Rodrigo, Felipe (chavalín), Santiago, Miguel Antunes, Miguel Eduardo, antiguos compañeros como Javier del Egido, Óscar, Alejandro, Eduardo, Roberto y Pablo Alcantarilla, y a nuevos becarios como Fabio, Navil y Pablo. Gracias de corazón por estar ahí, en nuestras charlas sobre tecnología, empleos, el camino correcto a seguir y la vida en general.

Especial mención vuelvo a hacer a Miguel Eduardo y mi compañero Marcos Conde, cuya apoyo intenso ayudó a centrar mi camino en técnica y escritura. Muchísimas gracias por todo lo que aprendí a vuestro lado. Especial mención a mi querido profesor Ángel Álvarez, por sus valiosos consejos para afrontar mi carrera profesional y mi vida en general de la mejor forma posible. Eres muy grande.

A mis amigos de la universidad, especialmente a Rocío, Juan Carlos, Esther, Sergio, Pablo, Rubén y Adrián Rocandio. Aún me acuerdo de cuando empezamos con la carrera y como el paso inexorable del tiempo nos moldea a conveniencia. Os deseo lo mejor en vuestro futuro.

A mis buenos amigos Samuel y Adrián, con quien gran parte de mi vida he compartido. Con especial cariño guardo las interminables charlas sobre la vida y el futuro después de Karate, de comer, de cenar, en el coche, siempre quejándonos de la hora que marcaba el reloj al final de tan interminables conversaciones.

A mi familia, uno de los pilares de mi vida. A mi padre Juan Antonio y a mi madre Petra, que en paz descanse, les debo todo lo que soy y es por ello por lo que les estaré siempre agradecido. Querido padre, gracias por ser tan Genaro, arisco y pesado. Siempre has sido mi ejemplo a seguir, aunque cuando tenga 42 años me sigas regañando por subir con las zapatillas puestas. Querida madre, no se muere quien se va, sólo se muere el que se olvida, y tú nunca caerás en el olvido. A mi querida hermana-calili-chessmaster Silvia, con quien tantas regañinas he tenido, pero el cariño que nos tenemos las supera a todas. A mi perrita Nuka (a.k.a. Dragón, ChupaChups cuando le cortamos el pelo o Nuki-Nuki), cuyos paseos matutinos son probablemente el ingrediente secreto para la elaboración de esta tesis, dando rienda suelta a mi cabeza para imaginar nuevas propuestas mientras miraba el cielo azul. A mi querido *experimental setup* que me ha acompañado durante toda mi vida académica: silla de madera de la cocina, ratón de Hello Kitty tomado prestado de mi hermana y mi querido portátil táctil. Sois la base de todo mi trabajo.

Al resto de la familia, amigos, compañeros, entrenadores y profesores, gracias por todo.

Y, por último, la persona más importante de mi vida ahora mismo. Mi querida Marta, la persona más maravillosa y buena que conozco. Hemos compartido risas, lloros, besos y abrazos. Nunca me cansaré de repetirte lo suave que tienes la piel tras darte un beso en la mejilla y después hacerte de rabiar. Espero que esta situación esté dentro de un while cuya condición sea *True*.

"Te quiero más que ayer, pero menos que mañana. Hoy, y siempre"

Mi querido lector, disculpa mi monólogo de agradecimientos, es mi forma de ser y la cual tengo por bandera, aunque creo que ha quedado bonito. Podría decir mil anécdotas más de mi doctorado, pero como diría Aragorn, legítimo Rey de Gondor, enfrente de la mismísima Puerta Negra: *Hoy no es ese día.*

Vamos a la lectura importante, que empiece el *Rock and Roll* !!

Resumen

TBD

Palabras clave: Autonomous Driving, Deep Learning, Motion Prediction, Scene Understanding.

Abstract

TBD

Keywords: Autonomous Driving, Deep Learning, Motion Prediction, Scene Understanding.

Contents

Acknowledgements vii

Resumen xi

Abstract xiii

Contents xv

List of Figures xix

List of Tables xxi

List of Source Code xxiii

List of Acronyms xxv

1. Introduction 1

1.1. Motivation 1

1.2. Historical Context 3

1.3. Autonomous Driving architecture 6

1.4. Problem statement 9

1.5. Objectives and Structure of this Thesis 10

2. Related Works 13

2.1. Introduction 13

2.2. Problem Formulation of Motion Prediction 15

2.3. Contextual Factors and Classification of Motion Prediction methods 16

2.3.1. Physics-based Motion Prediction 18

2.3.1.1. Single-Trajectory 19

2.3.1.2. Kalman Filter 20
2.3.1.3. Monte Carlo 21
2.3.2. Classic Machine Learning based Motion Prediction 21
2.3.2.1. Gaussian Process 22
2.3.2.2. Support Vector Machine (SVM) 22
2.3.2.3. Hidden Markov Model (HMM) 23
2.3.2.4. Dynamic Bayesian Network (DBN) 24
2.3.3. Reinforcement Learning based Motion Prediction 25
2.3.3.1. Inverse Reinforcement Learning (IRL) 25
2.3.3.2. Generative Adversarial Imitation Learning (GAIL) 26
2.3.3.3. Deep Inverse Reinforcement Learning (DIRL) 27
2.3.4. Deep Learning based Motion Prediction 27
2.4. Summary 31

3. Theoretical Background 33

3.1. Introduction 33
3.2. Physics-based algorithms 34
3.2.1. Kalman Filter under the hood 34
3.2.2. Hungarian algorithm formulation 36
3.2.3. State Transition Equations of Single-Trajectory Models 36
3.2.3.1. Constant Velocity (CV) 38
3.2.3.2. Constant Acceleration (CA) 39
3.2.3.3. Constant Turn Rate Velocity (CTRV) 40
3.2.3.4. Constant Turn Rate Acceleration (CTRA) 40
3.2.3.5. Constant Speed and Acceleration in the Vertical (CSAV) . 41
3.2.3.6. Constant Speed and Coordinated Acceleration (CSAA) . . 42
3.3. Convolutional Neural Networks 44
3.4. Recurrent Neural Networks 46
3.5. Long Short-Term Memory 46
3.6. Generative Adversarial Networks 48
3.6.1. Summary 49
3.7. Attention Mechanisms 49
3.8. Self-Attention and Positional Encoding 49

3.9. Graphs 50
3.9.1. Graph Neural Networks 50
3.9.2. Graph Neural Networks 51
3.10. Training 51
3.10.1. Regression losses 51
3.10.1.1. Mean Square Error loss 52
3.10.1.2. SmoothL1 loss 52
3.10.2. Softmax loss 53
3.10.3. Negative Log-Likelihood loss 53
3.10.4. Winner-Takes-All loss 54
3.10.5. Hinge loss 54

4. Predictive Techniques for Scene Understanding 57
4.1. Introduction 57
4.2. SmartMOT 58
4.2.1. Monitored Area Attention Module 58
4.3. GAN based Vehicle Motion Prediction 59
4.4. Exploring Map Features 59
4.5. Leveraging traffic context via GNN 59
4.6. Improving efficiency of Vehicle Motion Prediction 60

5. Applications in Autonomous Driving 61
5.1. Motion Prediction Datasets 61
5.2. Multi-Object Tracking 61
5.3. Decision-Making 61
5.4. Holistic Simulation 61

6. Conclusions and Future Works 63
6.1. Conclusions 63
6.2. Future Works 65

Bibliography 67

List of Figures

1.1. Stanley, 2005 DARPA Grand Challenge winner 4

1.2. Number of autonomous test miles and miles per disengagement (Dec 2019 - Nov 2020) 5

1.3. Society of Automotive Engineers (SAE) automation levels 6

1.4. Advanced Driver Assistance systems (ADAS) and Autonomous Driving (AD) revenues in $ billion 7

1.5. Autonomous Driving Stack (ADS) modular vs end-to-end pipeline 8

1.6. Autonomous Driving Stack (ADS) modular pipeline 9

2.1. Example of a Conditional Motion Prediction (CMP) network. 14

2.2. Contextual factors and output types in Vehicle Motion Prediction 16

2.3. Modeling multimodality of future 3-second agent trajectories with one of our algorithms 18

2.4. Contextual factors and output types in Vehicle Motion Prediction 19

2.5. Some typical Classic Machine Learning algorithms in Motion Prediction . . 22

2.6. Reinforcement Learning vs Inverse Reinforcement Learning 25

2.7. Deep Learning methods applied in Motion Prediction 28

3.1. Overview of the Kalman Filter Predict-Update cycle 35

3.2. Overview of Single Trajectory prediction methods 37

3.3. Example of CNN architecture to process 1D-input signals 45

4.1. Path Planning and Map Monitoring Outline 59

List of Tables

2.1. Main state-of-the-art Deep Learning methods for Motion Prediction 29

2.2. Summary of Motion Prediction methods features 31

5.1. Comparison between the Argoverse 2 Motion Forecasting dataset and other recent motion forecasting datasets. Hyphens "-" indicate that attributes are either not applicable, or not available. We define “mined for interestingness” to be true if interesting scenarios/actors are mined *after data collection*, instead of taking all/random samples. † Public leaderboard counts as retrieved on Aug. 27, 2021. 62

List of Source Code

List of Acronyms

AD	Autonomous Driving.
ADS	Autonomous Driving Stack.
AI	Artificial Intelligence.
BEV	Bird's Eye View.
CMP	Conditional Motion Prediction.
CNN	Convolutional Neural Network.
CTRA	Constant Turn Rate and Acceleration.
CTRV	Constant Turn Rate and Velocity.
DL	Deep Learning.
GAN	Generative Adversarial Network.
GNN	Graph Neural Network.
ITS	Intelligent Transportation Systems.
MP	Motion Prediction.
PMP	Passive Motion Prediction.
SOTA	State-of-the-Art.

Chapter 1

Introduction

Aaay, el oro, la fama, el poder.
Todo lo tuvo el hombre que en su día se autoproclamó
el rey de los piratas, ¡GOLD ROGER!
Mas sus últimas palabras no fueron muy afortunadas:
"¿¡MI TESORO!? Lo dejé todo allí, buscadlo si queréis,
ojalá se le atragante al rufián que lo encuentre.

Opening 1 de One Piece: "We are"
Autor original: Hiroshi Kitadani

1.1. Motivation

Autonomous Driving (AD) have held the attention of technology enthusiasts and futurists for some time as evidenced by the continuous research and development in Intelligent Transportation Systems (ITS) over the past decades, being one of the emerging technologies of the *Fourth Industrial Revolution*, and particularly of the Industry 4.0.

The concept *Fourth Industrial Revolution* or Industry 4.0 was first introduced by Klaus Schwab , CEO (Chief Executive Officer) of the World Economic Forum, in a 2015 article in Foreign Affairs (American magazine of international relations and United States foreign policy). A technological revolution can be defined as a period in which one or more technologies are replaced by other kinds of technologies in a short amount of time. Hence, it is an era of accelerated technological progress featured by Researching, Development and Innovation whose rapid application and diffusion cause an abrupt change in society. In particular, the *Fourth Industrial Revolution* conceptualizes rapid change to industries, technology, processes and societal patterns in the 21st century due to increasing inter-connectivity and smart automation. This industrial revolution focuses on operational efficiency, being the following four themes which summarize it:

- Decentralized decisions: Ability of cyber physical systems to make decisions on their own and to perform their tasks as autonomously as possible.
- Information transparency: Provide operators with comprehensive information to make decisions. Inter-connectivity allows operators to gather large amounts of information and data from all points in the manufacturing process in order to identify key areas or aspects that can benefit from improvement to enhance functionality.
- Technical assistance: Ability to assist humans with unsafe or difficult tasks and technological facility of systems to help humans in problem-solving and decision-making.
- Interconnection: Ability of machines, sensors, devices and people to communicate and conect with each other via the Internet of Things (IoT) or the Internet of People (IoP).

Based on the aforementioned principles, this revolution is expected to be marked by breakthroughs in emerging technologies in fields such as nanotechnology, quantum computing, 3D printing, Internet of Things (IoT), fifth-generation wireless technologies (5G), Robotics, Computer Vision (CV), Artificial Intelligence (AI) or the scope of this PhD thesis, Autonomous Driving Stacks (ADSs). The sum of all these advances are resulting in machines that can potentially see, hear and what is more important, think, moving more deftly than humans.

An ADS, also referred in the literature as Intelligent Vehicle (IV), driverless car or autonomous car, is a vehicle tan can sense its surrounding and moving safely with little or even no human input. These ADSs must combine a variety of sensors to understand the traffic scenario, like RADAR (RAdio Detection A Ranging), LiDAR (Light Detection and Ranging), cameras, Inertial Measurement Unit (IMU), wheel odometry, GNSS (Global Navigation Satellite System) or ultrasonic sensors, and detect, track and predict (which is the main purpose of this thesis) the most relevant obstacles around the ego-vehicle. Then, advanced control and planning systems process this sensory information in combination with a predefined global route to calculate the corresponding control commands to drive throughout the environment, ensuring a safe driving.

The dream of seeing fleets of ADSs efficiently delivering goods and people to their destination has fueled billions of dollars and captured consumer's imaginations in investment in recent years. Nevertheless, according to the "Autonomous driving's future: Convenient and connected" report, published by the global management consulting firm McKinsey & Company in January 2023, even after some setbacks have pushed out timelines for AD launches and delayed customer adoption, the transportation community still broadly agrees that AD has the potential to transform consumer behaviour, transportation and

society at large. AD is considered as one of the solutions to the aforementioned problems and one of the greatest challenges of the automotive industry today.

Statistics show that 69 % of the population in the European Union (EU), including associated states, lives in urban areas. According to the World Health Organization, nearly one third of the world population will live in cities by 2030, leading to an overpopulation in most of them. Aware of this problem, the Transport White Paper published by the European Commission in 2011 indicated that new forms of mobility ought to be proposed so as to provide sustainable solutions for people and goods safely. For example, regarding safety, it sets the ambitious goal of halving the overall number of road deaths in the EU between 2010 and 2020. Nevertheless, this goal does not seem to be easy since only in 2014 more than 25,700 people died on the roads in the EU, many of them caused by an improper behaviour of the driver on the road. A similar study made by the National Highway Traffic Safety Administration (NHTSA, tranportation organization of the United States) reported in 2015 that around 94 % of traffic accidents happen because of human error. In that sense, the existence of reliable and economically affordable ADSs are expected to create a huge impact on society affecting social, demographic, environmental and economic aspects. It can produce substantial value for the auto industry, drivers and society, making driving safer, more convenient and more enjoyable. While the human driver or not could select whether to drive, in autonomous mode hours on the road previously spent driving could be used to work, watch a funny movie or even to video call a friend. For employees with long commutes, AD might shorten the workday, increasing worker productivity. Since workers, specially those related to digital jobs or related fields, may perform their jobs from an ADS, they could more easily move further away from the office, which, in turn, could attract more people to suburbs and rural areas. Besides this, it is estimated to cause a reduction in road deaths, reduce fuel consumption and harmful emission associated and improve traffic flow, as well as an improvement in the overall driver comfort and mobility in groups with impaired faculties, such as disable or elderly people, providing them with mobility options that go beyond car-sharing services or public transportation. Other industrial applications of autonomous vehicles are agriculture, retail, manufacturing, commercial and freight transport or mining.

1.2. Historical Context

ADSs have become a challenge for auto competitions and technology companies, which has derived in an intense competition. Though today companies such as Mercedes, Ford or Tesla are racing to build ADSs for a radically changing consumer world, the research

and development of autonomous robots is not new.

In 1500, centuries before the invention of the automobile, Leonardo da Vinci designed a cart that could move without being pulled or pushed. In 1868, Robert Whitehead invented a torpedo that could propel itself underwater in order to be a game-changer for naval fleets all over the world. In terms of robotic solutions for intelligent mobility, the study was started in the 1920s, being the concept of Autonomous Car defined in Futurama, an exhibit at the 1939 New York Wolrd's Fair. General Motors created the exhibit to display its vision of what the world would look like in 20 years, including an automated highway system that would guide ADS. By 1958, General Motors made this concept a reality (at least as a proof of concept) being the car's front end embedded with sensors to detect the current flowing through a wire embedded in the road. The first semi-automated car was developed in 1977 by Japan's Tsukuba Mechanical Engineering Laboratory. The vehicle reached speeds up to 30 km/h with the support of an elevated rail.

Figure 1.1: Stanley, 2005 DARPA Grand Challenge winner
Source: *Stanford university*

Nevertheless, the first truly autonomous cars appeared in the 1980s with Carnegie Mellon University's Navlab and ALV projects funded by the USA company DARPA (Defense Advanced Research Projects Agency) in 1984 and EUREKA Prometheus project (1987) developed by Mercedes-Benz and Bundeswehr University Munich's. By 1985, the ALV project had shown self-driving speeds on two-lane roads of 31 km/h with obstacle avoidance added in 1986 and off-road driving in day and night conditions by 1987. Furthermore, from the 1960s through the second DARPA Grand Challenge in 2005 (212 km off-road course near the California-Nevada state line, surpassed by all but one of the 23 finalists), automated vehicle research in the United States was primarily funded by DARPA, the US Army and US Navy, yielding rapid advances

in terms of speed, car control, sensor systems and driving competence in more complex conditions. This caused a boost in the development of autonomous prototypes by companies and research organizations, most of them from the United States. Figure 1.1 shows Stanley, the 2005 DARPA Gran Challenge winner, from Stanford university.

Even though self-driving cars have not yet displaced conventional cars, there can be found several examples of how it has become a hot topic for powerful companies such as Delphi Automotive Systems, Audi, BMW, Tesla, Mercedes-Benz or Waymo.

In 2005 Delphi broke the Navlab's record achievement (driving 4,584 km while remaining 98 % of the time autonomously) by piloting an Audi, improved with Delphi technology, over 5,472 km through 15 states while remaining in self-driving mode 99 % of the time. Moreover, in 2005 the USA states of Michigan, Virginia, California, Florida, Nevada and the capital, Washington D.C., allowed the testing of automated cars on public roads.

In 2017, Audi stated that its A8 car prototype would be automated at speeds up to 60 km/h by using its perception system named "Audi AI". Also, in 2017 Waymo (self-driving technology development company subsidiary of Alphabet Inc) started a limited trial of a self-driving taxi service in Phoenix, Arizona.

Figure 1.2 shows the total number of autonomous test miles and miles per disengagement in California (Dec 2019 - Nov 2020) by some of the most important AD technology development companies around the world. The concept disengagement is quite useful to assess the quality of an ADS, defined as the deactivation of the autonomous mode when a failure of the autonomous technology is detected or when a safe operation requires that the autonomous vehicle test driver disengages the autonomous mode, resulting in control being seized by the human driver.

Figure 1.2: Number of autonomous test miles and miles per disengagement (Dec 2019 - Nov 2020)
Source: *DMV California, via The Last Driver License Holder*

At the moment of writing this thesis (2023), many vehicles on the road are considered to be semi-autonomous due to safety features like braking systems, assisted parking, lane boundaries detection or predict the long-term behaviour of the users around the vehicle to execute the most optimal action in a safely way. Regarding this, the Society of Automotive Engineers (SAE) published the concept of autonomy levels in 2014, as part of its "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems" report. Figure 1.3 illustrates the six levels of autonomy (the higher the level, the more autonomous the car is), where it can be appreciated that Level Zero means "No Automation", being the acceleration, braking and steering controlled by a human driver at all times, and Level Five represents Full Automation, where there is a full-time automation of all driving tasks on any road, under any conditions, whether there is a human on board or not.

Figure 1.3: Society of Automotive Engineers (SAE) automation levels
Source: *NHTSA (National Highway Traffic Safety Administration)*

In that sense, today most vehicles only included basic Advanced Driver Assistance Systems (ADAS), but major advancements in AD capabilities are on the horizon. According to a 2021 McKinsey consumer survey, growing demand for AD systems could create billions of dollars in revenue. Based on a consumer interest in AD features and commercial solutions available on the market today, ADAS and AD could generate between $300 and $400 billions in the passenger car market by 2035. Figure 1.4 illustrates an interesting study reporting the revenues of ADAS and AD from Level 1 (Driver Assistance) to Level 4 (High Automation). As expected, Level 5 is excluded from this study due to the huge difficulties the automotive companies would have to face to adapt their systems under totally different environmental conditions.

1.3. Autonomous Driving architecture

To sum up what commented above, increasing the level of autonomous navigation in mobile robots (from agriculture to public and private transport) are expected to create

Figure 1.4: Advanced Driver Assistance systems (ADAS) and Autonomous Driving (AD) revenues in $ billion
Source: *McKinsey Center for Future Mobility*

tangible business benefits to those users and companies employing them. However, designing an autonomous navigation system does not seem to be an easy task. In the State-of-the-Art (SOTA) we can distinguish two main kind of software architectures: End-to-End and modular. Figure 1.5 illustrates the entire AD architecture starting from sensing, all the way to longitudinal (throttle/brake) and lateral (steering angle) control of the vehicle, which are the commanded signals that feed the low-level electronic system that moves the vehicle, like a drive-by-wire system [1]. End-to-End are considered black-box models, where a single neural network performs the driving task (throttle/steering/brake) from raw sensor data, in such a way the error be may vanished since intermediate representations are jointly optimized, but these are not very interpretable. On the other hand, modular architectures (considered as glass models as counterpart to End-to-End approaches) separate the driving task into individually programmed or trained modules. This solution is more interpretable, since the know-how of a research group or company is easily transferred, they allow parallel development, being the standard solution in industrial research, but the error is propagated, where intermediate representations can led to suboptimal performance. For example, incorrect object detection can lead to low-quality tracking and motion prediction.

Considering the RobeSafe (Robotics and eSafety) research group and the main projects (Techs4AgeCar, AIVATAR) where this thesis has been developed, we integrate our algorithms in a software modular approach. An example of a software modular approach is shown in Figure 1.6. Despite the fact in literature some authors disagree on the specific software architecture of an ADS, specially the motion prediction module, which is usually classified as a perception algorithm but sometimes is included as part of the planning or decision-making layers, we can hierarchically break down (from raw data to the driving task) a standard AD architecture into the following software layers:

Figure 1.5: Autonomous Driving Stack (ADS) modular vs end-to-end pipeline
Source: *Vrunet: Multi-task learning model for intent prediction of vulnerable road users* [2]

- Localization layer: Positions and locates the vehicle on a map with real-time and centimetric accuracy approach. The main source of information is a robust differential-GNSS, though IMUs, wheel odometry and even cameras are commonly employed.

- Perception layer: Understand the environment around the ego-vehicle thanks to the information collected by the sensors. If defined as multi-stage, the perception layer first detects the most relevant obstacles, then track them over time to finalize long-term predict with plausible predictions. In order to perform object detection, LiDAR, camera and RADAR are the main sensors that provide the corresponding raw data. Additionally, HD map information is frequently used in the motion prediction tasks by most SOTA algorithms.

- Mapping layer: Responsible for creating a topologic, semantic and geographical modeling of the environment through which the vehicle drives, being the HD Map graph the most common source of information.

- Planning layer: This layer is comprised of three components: route, behaviour and trajectory planner. The route planner computes the most optimal (in terms of distance, time and so forth and so on) global route from some predefined start and goal. It uses the localization and mapping output. On the other hand, the behaviour planner, also referred as decision-making layer by some authors, it performs high-level decision-making of driving behaviours such as lane changes or progress through intersections, mostly focused on the previously computed global route and current localization. It can be seen as an atomization of the global route in different behaviors to reach the goal. Finally, the trajectory planner, also known as local planner, generates a time schedule for how to follow a path given constraints such as position, velocity and acceleration in order to meet the previously decided behaviour and taking into account the prediction from the perception layer, avoiding obstacles in optimal direction and speed conditions.

- Control layer: Once the local plan is calculated, the control layer is responsible for generating the commands that are sent to the actuators. It receives as input some waypoints from the calculations made in the trajectory planner. Once these waypoints are received, most authors perform spline interpolations and a velocity profile that ensures a smooth and continous trajectory.

Figure 1.6: Autonomous Driving Stack (ADS) modular pipeline
Source: *Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles* [3]

1.4. Problem statement

As commented in previous sections, in order to operate efficiently and safely in highly dynamic, complex and interactive driving scenarios, ADS need to smartly reason like human beings via predicting future motions of surrounding traffic participants during navigation. Nevertheless, achieving accurate and robust Motion Prediction (MP) in one of the most difficult and interesting challenges to achieve full-autonomy, since it is equivalent to a bridge between the former stages of the perception layer, where the scene is understood detecting and tracking static and dynamic objects of the environment, and the planning and control layer, where the future trajectory of the ego-vehicle is computed

and the driving commands are sent to the physical layer (e.g. Drive-by-Wire [1]). Here are some of the most important challenges:

1. Heterogeneity of traffic participants. Traffic partipants (specially those which are dynamic) can be roughly classified as cyclists, pedestrians or other vehicles. The prediction model should be capable of differentiating the motion patterns of heterogeneous traffic participants, in such a way fine-grained classification (detection module) is quite beneficial to include additional metadata along with the past observations.

2. Complexity of road structure. Road structures are highly diverse and complex, specially in highways and urban areas, which noticeably affect the motion behaviours of traffic participants.

3. Variable number of interactive agents. The prediction model must deal with a number of associated traffic participants within a certain area that can vary from time to time, such as intersections or roundabouts. Then, while driving, a comprehensive representation of the scene must be able to accommodate an arbitrary number of involved traffic participants.

4. Multimodality of driving behaviours. In real-world, despite we know the behaviour our vehicle will carry out, the motion patterns of other traffic participants can be considered inherently multimodal since there is usually more than one reasonable option for a driver to choose, specially in intersections, when the number of lanes increases or even in the same lane with different velocity profiles (constant velocity, sudden break, sudden acceleration). In that sense, a robust and reliable MP model is expected to be human-like and capture different plausible motion modalities where an agent can travel in the prediction horizon.

5. Complex interpendencies among traffic participants and road infrastructure. Agent-Agent, Agent-Road and Road-Road interpendencies are of great importance for MP and interaction modeling, even more taking into account the complexity of road structures and heterogeneity of traffic participants aforementioned. As expected, an agent future trajectory will be affected not only by its own past trajectory and driving objectives (given by the behaviour planner) but also by other surrounding agents past trajectories, traffic rules and physical constraints.

1.5. Objectives and Structure of this Thesis

The main scope of this thesis is to study the SOTA and development of novel and efficient interaction-aware Deep Learning based MP models, focusing on long-term (from 3 to 6 s) prediction horizon and AD, where traffic participants can range from trucks

to pedestrians, instead of models focused on pedestrian trajectory prediction. The main inputs that will be used throughout this work are the physical (map) information and historical states (that may include agent position, velocity, orientation, object type and category) of traffic participants in Bird's Eye View (BEV), assuming these objects have been previously tracked by our ego-vehicle (also referred as the autonomous car). Though the evaluation of these methods will be done using a single target agent, as proposed by some of the most important prediction datasets, like Argoverse 1 [4] and Argoverse 2 [5], some of the proposed methods will be trained considering multi-agent. In this thesis, the solutions to the aforementioned challenges will be discussed and investigated progressively. In order to achieve the main scope, the following objectives will be met:

1. Research of SOTA MP, focused on Deep Learning (DL) and the AD paradigm.
2. Propose of several MP architectures, studying the progressive incorporation of DL mechanisms and different sources of information and metadata, achieving SOTA accuracy while reducing in millions of parameters previous models as well as inference time.
3. Validate the proposed models in downstream applications, such as decision-making or behaviour planning, taking into account former stages of the perception layer (detection and tracking) instead of static files (benchmarks) in hyper-realistic simulation, as a preliminary stage before implementing it in a real-world vehicle.

The organization of this document has been done as follows:

- Chapter 2 reviews the most important features and methods of physics-based and learning-based MP methods. The physics-based methods are reviewed according to a taxonomy similar to existing reviews. The learning-based methods are reviewed based on two classification criteria: scene representation and trajectory decoding.
- Chapter 3 presents a technical background, mostly focused on DL mechanisms to deal with temporal sequences and interactions, to deeply understand the proposed methods.
- Chapter 4 illustrates the different prediction models developed in the thesis using different validation environments, from unimodal physic-based prediction to the final model of the thesis which takes into account agents interactions, map information and past observations using a novel scene representation with heuristic proposals, graph-based encoding, DL-based goal proposals and motion refinement.
- Chapter 5 addresses the integration of the final model of the thesis with upstream and downstream modules to contribute the entire pipeline and closed-loop for AD.
- Chapter 6 summarizes the thesis and provides some promising directions for future work in the areas of MP and validation.

Chapter 2

Related Works

Llegaré a ser el mejor, El mejor que habrá jamás
Mi causa es ser su entrenador, Tras poderlos capturar.
Viajaré a cualquier lugar, Llegaré a cualquier rincón
Y al fin podré desentrañar, El poder de su interior.
¡Pokémon! Hazte con todos (solos tú y yo),
Es mi destino, mi misión
¡Pokémon! Tú eres mi amigo fiel,
Nos debemos defender.

Opening 1 de Pokémon: "Gotta catch 'em all!"
Autor original: Jason Paige

2.1. Introduction

One of the crucial tasks that ADSs must face during navigation, specially in arbitrarily complex urban scenarios, is to predict the behaviour of dynamic obstacles [4], [6]. In a similar way to humans that pay more attention to nearby obstacles and upcoming turns than considering the obstacles far away, the perception layer of an ADS must focus more on the salient regions of the scene, particularly on the more relevant dynamic agents to predict their future behaviour before conducting a maneuver, such as lane changing or accelerating. In that sense, before proceeding with the study of the different methods of the SOTA of MP in the field of AD, one important thing to note is that this thesis is focused on non-conditional motion prediction, also referred as Passive Motion Prediction (PMP), where the prediction of surrounding agents is not influenced by the future decisions of the ego-vehicle or even other agents, referred as Conditional Motion Prediction (CMP) in the literature. Most existing works [7]–[12] focus on a passive prediction scheme, where the future states of a particular agent are predicted given its past information, other surrounding agents information and interactions as well as the physical context. Then, downstream planning modules, specially the behaviour planning module (also referred as decision-making layer, as stated in Section 1.3), the ego-vehicle (our vehicle) future actions are computed according to the predicted trajectories in a

passive manner, that is, without modifying the output of the prediction model, and the global route previously calculated.

Figure 2.1: Example of a Conditional Motion Prediction (CMP) network.
We highlight the influence of the AD in the prediction of surrounding agents.
Source: *Conditional Predictive Behavior Planning with Inverse Reinforcement Learning for Human-like Autonomous Driving* [13]

Nevertheless, to ensure safety under various predicted trajectories of the surrounding agents, our ADS must overly conservative with inefficient maneuvers, specifically in arbitrarily complex traffic scenarios, because PMP models ignore the fact that the future states of an agent can influence the future actions of other agents, what is the most realistic situation. To this end, researchers recently started to explore a more coherent interactive prediction and planning framework which relies on predicting the surrounding agents future trajectories conditioned on the ego-vehicle future actions [14] [15] [16], as a preliminary state to implement a fully-interaction graph where the future states of all agents (either autonomous prototypes or human-driven) influence in the decision of all agents. Under such frameworks, the ADS can reason over potential actions while considering its influence on surrounding agents, as observed in Figure 2.1, inducing less conservative and more efficient maneuvers in highly interactive scenarios. [13] propose a learning-based behaviour planning framework that learns to predict conditional multi-agent future trajectories, evaluating decisions from real-world human data. Moreover, they propose a two-stage learning process where the prediction model is trained first conditioned on the ADS future actions, and then used as an environment model in the learning of the cost function with maximum entropy Inverse Reinforcement Learning (IRL). [17] argue that CMP-based models essentially learns the posterior distribution of future trajectories conditioned on the future states of the ego-vehicle, where this future trajectory is treated as an observation, whilst safe and realistic prediction models should build the MP to approximate the future trajectory distribution

under the intervention of enforcing the ADS future states, referring this new task as Interventional Behaviour Prediction (IBP). As aforementioned, the algorithms studied and developed throughout this thesis do not focus on the joint study of the prediction and behaviour planning modules, but on building efficient and powerful PMP algorithms without considering the future states of the autonomous agents as an additional condition.

Once the differences between CMP and PMP have been illustrated, we proceed with the problem formulation, main contextual factors and classification of prediction methods.

2.2. Problem Formulation of Motion Prediction

Given a sequence of past trajectories $a_P = [a_{-obs'_{len}+1}, a_{-obs'_{len}+2}, ..., a_0]$ for an agent, we aim to predict its future steps $a_F = [a_1, a_2, ..., a_{pred_{len}}]$ up to a fixed time step $pred_{len}$. Running in a specific traffic scenario, each agent will interact with static HD maps m and the other dynamic actors, meeting the corresponding traffic and social rules. Therefore, the probabilistic distribution that we want to capture is $p(a_F|m, a_P, a_P^O)$, where a_P^O denotes the other agents observed states. The output of most existing methods is a set of trajectories $A_F = \{a_F^k\}_{k\in[0,K-1]} = \{(a_1^k, a_2^k, ..., a_{pred_{len}}^k)\}_{k\in[0,K-1]}$ for each agent, where K represents the number of modes or plausible future directions, due to the inherent uncertainty associated to the prediction problem. This set of trajectories for each agent will be used by downstream decision modules. On top of that, TNT (Target-driven trajectory prediction) [18] is one of the first methods that introduces specific preliminary future positions in the problem formulation, also referred as goals, being TNT [18]-like methods distribution approximated as:

$$\sum_{\tau\in T(m,a_P,a_P^O)} p(\tau|m, a_P, a_P^O)p(a_F|\tau, m, a_P, a_P^O) \tag{2.1}$$

where $T(m, a_P, a_P^O)$ is the space of candidate goals depending on the driving context.

However, the map space m is large, and the goal space $T(m, a_P, a_P^O)$ requires careful design. In that sense, some methods expect to accurately predict the actor motion by extracting good features. For example, LaneGCN [12] tries to approximate $p(a_F|m, a_P, a_P^O)$ by modeling $p(a_F|M_{a_0}, a_P, a_P^O)$, where M_{a_0} is a "local" map features that is related to the actor's state a_0 at final observed step $t = 0$. To extract M_{a_0}, they use a_0 as an anchor to retrieve its surrounding map elements and aggregate their features. We found that not only the "local" map information is important, but also the goal area maps information is of great importance for accurate trajectory prediction. So, we reconstructed the probability as:

$$\sum_{\tau} p(\tau|M_{a_0}, a_P, a_P^O)p(M_\tau|m, \tau)p(a_F|M_\tau, M_{a_0}, a_P, a_P^O) \tag{2.2}$$

We directly predict possible goals τ based on actors' motion histories and driving context. Therefore, GANet is genuinely end-to-end, adaptive, and efficient. Then, we apply the predicted goals as anchors to retrieve the map elements in goal areas explicitly and aggregate their map features as M_τ.

2.3. Contextual Factors and Classification of Motion Prediction methods

This section studies the contextual factors (inputs) and classification of MP in the field of AD according to its encoding method of the different inputs and output types. Figure 2.2 [19] summarizes the main inputs and outputs of these methods.

Figure 2.2: Contextual factors and output types in Vehicle Motion Prediction
Source: *A survey on trajectory-prediction methods for autonomous driving* [19]

In order to model the future states of surrounding agents, MP models must pay attention to the current environment, where some contextual factors can be clearly identified:

- Physics-related factors refer to the kinematic and dynamic variables of the agents, specifically to their spatio-temporal variables (such as position, velocity, acceleration, object type or mass) as well as past behaviours.
- Interaction-related factors include the inter-dependencies and social regulations between agents maneuvers. It is important to consider that traffic agents can be either non-relevant pedestrians on the sidewalk as well as an extremely relevant truck in front of the ego-vehicle.

- Road-related factors include the corresponding traffic rules (lane type, traffic signals, stops, etc.) as well as the modeling of the map (usually HD-map), including its topological, semantic and geometrical information.

On the other hand, regarding the output types, MP methods need to provide the future trajectories of traffic participants. Nevertheless, these methods can provide these future trajectories in different ways, even though these outputs can be unified as a single output, depending on the application:

- Unimodal prediction. In the unimodal case, the prediction method only returns a single future trajectory, without taking into account other possible behaviours.
- Multimodal prediction. Models that generate a multimodal prediction compute multiple K future trajectories (also referred as modes in the literature) with the probability of each future trajectory. The higher the mode probability (also referred as score), the more probable this particular behaviour (turn left with a certain velocity and acceleration) should be. This output type is specially useful for fast-changing and highly interactive situations where multiple options are available. One important thing to note is that multimodal methods must be designed in order to not only reason in terms of different maneuvers (keep straight, turn right, lane change, etc.) but also different velocity profiles (constant velocity, acceleration, sudden break, etc.) regarding the same maneuver.
- Maneuver, also referred as intention, can be part of the final output or just be an intermediate step in the method. MP methods usually produce maneuver intention to assist in the subsequent prediction.

As we will study throughout this section, most prediction methods focus on the multimodal output with an associated probability for each mode, since this is the most realistic way to imitate the human brain during navigation. First of all, there are plenty of problems during navigation, since there is a high uncertainty of traffic behavior and a large number of different situations. That means that one cannot use a discrete number of situations and a discrete number of car movements. Second, the main tasks of ADS, like ensuring safe and efficient operations and anticipating a multitude of possible behaviors of traffic actors in its surroundings, provide a large need in knowing the position of all vehicles beforehand. Multimodal means that we have multiple predictions for each timeframe. Figure 2.3 illustrates an interesting traffic scenario in the Argoverse 1 [4] Motion Forecasting dataset, processed by one our algorithms. The **target agent** is getting close to an intersection, where several future maneuvers are plausible. In both cases (unimodal and multimodal, which are the most common ones), the model must reason the future trajectory of the target agent based on its past observations, interactions with other agents and physical context. On the left (2.3a) illustrates the unimodal case, where a single trajectory is predicted. On the right (2.3b), multiple trajectories (or modes) with associated

(a) Unimodal prediction (b) Multimodal prediction

Figure 2.3: Modeling multimodality of future 3-second agent trajectories with one of our algorithms; We represent: our vehicle (**ego**), the **target agent**, and **other agents**. We can also see the **ground-truth** trajectory of the target agent, our **multimodal predictions** (with the corresponding confidences) and **plausible centerlines**. Circles represent last observations and diamonds last future positions.

probabilities are predicted, with most modes turning left and one mode keeping straight since in similar situations the agent could also perform this behaviour with a similar context. The main point of having this multimodality is to remove too generalized solutions. A lot of different algorithms study this problem, where quite similar inputs would produce similar predictions, while in reality this does not happens.

After defining the contextual factors and output types, a classification of the prediction methods according to different modeling approaches is illustrates. Over the last two decades, MP can be divided into four parts in chronological order: Physics-based, classic Machine Learning-based, Reinforcement Learning-based and Deep Learning-based, as shown in Figure 2.4. The remaining content of this section, based on the study performed by [19], illustrates the main algorithms used in each part, especially focusing on DL since in this work we focus on predictive techniques for scene understanding based on deep models.

2.3.1. Physics-based Motion Prediction

Physics-based methods are the first and simplest methods used by researchers. Although the accuracy of these methods is relatively low, more and more models use the idea of physics-based models to improve the accuracy. Physics-based methods have more accurate results when the movement of vehicles can be accurately described by kinematics or dynamics models, but the physical model of the traffic participants is constantly changing, such that most of these methods are only suitable for short-term prediction (no

Figure 2.4: Contextual factors and output types in Vehicle Motion Prediction

more than 1-s). Dynamics models can be quite complex, including many inherent parameters and introducing an extra computation burden, in such a way that researchers prefer simple dynamic methods for motion prediction. In terms of vehicle MP, the bicycle model is usually employed to model the vehicle physics, driven by the front wheels [20], [21]. In the literature three main types of physics-based models are distinguished, where the main different is the way in which the uncertainty is handled: Single Trajectory, Kalman Filter-based and Monte Carlo-based.

2.3.1.1. Single-Trajectory

One of the most straightforward methods to predict an agent trajectory is to directly apply the agent current state to the physic model. In order to increase the accuracy and stability of the estimation, the vehicles are mostly assumed to comply with motion models that describe their dynamic behavior. In the past, numerous motion models (with different degrees of complexity) have been proposed for this task [21]–[23]. A first systematization can be achieved by defining different levels of complexity. At the lower end of such a scale, linear motion models are situated. These models assume a Constant Velocity (CV) or a Constant Acceleration (CA). Their major advantage is the linearity of the state transition equation which allows an optimal propagation of the state probability distribution. On the other hand, these models assume straight motions and are thus not able to take rotations (especially the yaw rate) into account.

A second level of complexity can be defined by taking rotations around the z-axis into account. The resulting models are sometimes referred to as curvilinear models. They can be further divided by the state variables which are assumed to be constant. The most

simple model of this level is the Constant Turn Rate and Velocity (CTRV) model. By defining the derivative of the velocity as the constant variable, the Constant Turn Rate and Acceleration (CTRA) model can be derived. Both CTRV and CTRA assume that there is no correlation between the velocity v and the yaw rate ω. As a consequence, disturbed yaw rate measurements can change the yaw angle of the vehicle even if it is not moving.

In order to avoid this problem, the correlation between v and ω can be modeled by using the steering angle Φ (angle between the axis of motion and the direction of the front wheels) as constant variable and derive the yaw rate from v and Φ. The resulting model is called Constant Steering Angle and Velocity (CSAV). Again, the velocity can be assumed to change linearly, which leads to the Constant Curvature and Acceleration (CCA) model.

From a geometrical point of view, nearly all curvilinear models are assuming that the vehicle is moving on a circular trajectory (either with a constant velocity or acceleration). The only exception is the CTRA model which models a linear variation of the curvature and thus assumes that the vehicle is following a clothoid.

While in theory curvilinear models describe the motion of road vehicles very accurately, errors may result from highly dynamic effects such as drifting or skidding. While models which are able to cope with such effects do exist, they will not be considered here for two reasons: Firstly, most ITS applications are designed for scenarios with non-critical dynamics. Secondly, the required information for estimating the additional parameters (e.g. slip from every tire, lateral acceleration) are not observable by exteroceptive sensors. Furthermore, these methods are not able to consider the road-related factors and the uncertainty of the current state is unreliable for long-term prediction in such a way these single trajectory models should only be used for estimating unimodal trajectories of the surrounding agents in the short-term. We further study the state transition equations of physics-based models in Chapter 3 since they will be used in the algorithms proposed in this thesis as preliminary proposals for the DL models.

2.3.1.2. Kalman Filter

Kalman Filter (KF)-based methods aim to solve one of drawbacks of physics-based models: In real-world, the states of agents are not perfectly known since they present an associated noise. KF-based methods model the uncertainty oir noise of the current agent state and its physic model by means of a Normal (Gaussian) distribution. Compared to the single trajectory methods, the main advantage is that KF methods consider the uncertainty of the predicted trajectory, specially when using its Extended (EKF) or Unscented (UKF) where non-linearities can be modeled. As proposed by the original

algorithm [24], the prediction and update steps are combined into a loop where the mean value and covariance matrix of the agent state is computed for each future step, calculated as an average trajectory with related uncertainty.

Nevertheless, these KF-based methods are unimodal Gaussian distributions which are not enough to represent agents interactions. In that sense, [25] propose an Interactive Multiple Model (IMM) to compute a multimodal prediction. Moreover, [26] model a set of Kalman Filters used to describe physical models of the vehicles and switch between them, defined as Switched Kalman Filter (SKF). [27] propose IMM-KF, a novel Interacting Multiple Model Kalman Filter which takes interaction-related factors (social regulation, inter-dependencies) into consideration, as shown in Figure 2.2.

2.3.1.3. Monte Carlo

In the same way KF methods aimed to solve the associated noise to the physics state of the agent, Monte Carlo method aims to simulate the state distribution approximately since an analytical expression for the predicted state distribution is usually unknown without any assumptions of the linearity or the model's Gaussian nature. This method randomly samples the input variables and applies the physics model to compute potential future trajectories. In order to ensure the plausibility of the future behaviour in the context of AD, the generated future states are usually filtered with a lateral acceleration lower than the actual allowable lateral acceleration [28], though other vehicle physical limitation can also be used such that the input of the model will be more realistic. [29] present a model that identifies a preliminary maneuver and then applies the Monte Carlo method to compute future trajectories by the identified maneuver. Furthermore, [30] first use the Monte Carlo algorithm to predict future trajectories and then utilize MPC (Model Predictive Control) algorithm to refine these preliminary future trajectories.

2.3.2. Classic Machine Learning based Motion Prediction

Classic ML MP methods make use of data-driven models to predict trajectories instead of only considering the physical model. These methods determine the probability distribution by mining data features. These methods provide new ideas for MP, which promote the development of learning-based approaches. With more factors to be considered, the accuracy of these methods keeps increasing, being most of them maneuver-based (which is provided or identified in advance) in order to subsequently estimate the future states of the agents by first judging the corresponding maneuver. Regarding the field of vehicle MP, we can find in the literature four main types of classic ML methods: Gaussian Process, Support Vector Machine (SVM), Hidden Markov Model (HMM) and Dynamic Bayesian (Network). Figure 2.5b illustrates some these methods.

(a) Hidden Markov Model

(b) Dynamic Bayesian Network

Figure 2.5: Some typical Classic Machine Learning algorithms in Motion Prediction, like Hidden Markov Model and Dynamic Bayesian Network.
Source (a): *Improved driving behaviors prediction based on fuzzy logic-hidden markov model (fl-hmm)* [31]
Source (b): *Probabilistic intention prediction and trajectory generation based on dynamic bayesian networks* [32]

2.3.2.1. Gaussian Process

Gaussian Process (GP) applied in ML [33] is identified as an stochastic process (collection of random variables indexed time or space) such that every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is normally distributed. This stochastic process may be used to solve the prototype trajectory method [34], [35], which a maneuver-based method that divides agents trajectories into a collection of several types of prototype trajectories. First, past trajectories are regarded as the samples of the GP, sampled along the time axis, being these samples represented by N discrete points to map the N-dimensional space, which is equivalent to satisfy the N-dimensional Gaussian distribution. Therefore, the main task of the GP model regarding the prototype trajectory method is to determine the parameters of GP through the samples. Once these parameters have been obtained by means of the stochastic, the prototype trajectory method measures the similarity between the input historical trajectory and the computed prototype set to predict the most plausible future motion. GP can also be used to model interaction-related factors, as shown in Figure 2.2. [36] solve the frozen robot problem, where the environment surpasses a certain level of complexity and the robot planner decides to freeze in place to avoid collisions, by means of GP for joint collision avoidance. Furthermore, [37] apply GP and Dirichlet Process (DP) to define motion processes and apply a non-parametric Bayesian network to extract potential motion patterns.

2.3.2.2. Support Vector Machine (SVM)

Support Vector Machine (SVM) increases the level of complexity over previous methods, being able to learn and recognize an agent maneuver in a complex environment. The

main idea of SVM is to find the support vector that meets the classification requirements and determine the optimal hyperplane which can maximize the interval of the classified data. In particular, when applied to the MP problem in AD, driving maneuvers are usually classified into several categories: keep straight, turn left, turn right, break, etc. In that sense, it uses the kernel function to convert the input data to high-dimensional and perform linear classification in the space to identify the current driving maneuvers so as to predict the future steps. [38] apply SVM to identify a lane changing maneuver, using the position, velocity, acceleration and steering wheel angle of the corresponding vehicle as input features for identification. [39] propose a layered architecture method combining SVM and Bayesian filtering to identify lane-changing maneuvers so as to obtain more accurate identification results. Furthermore, [40] make use of SVM to identify the maneuvers of traffic participants. Nevertheless, according to the SVM definition which output the characteristics of classification probability, the user must present the categories or possible maneuvers in advance which will will also impact the final prediction results.

2.3.2.3. Hidden Markov Model (HMM)

As aforementioned, SVM can be effective in classification problems, but in the field of MP in AD not as effective as a Hidden Markov Model (HMM). This is one of the most popular maneuver-based classic ML MP methods, which uses Markov Chain. The Markov Chain refers to a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event, i.e. , the state at time $t+1$ of the system is only related to the previous time t, and the state transition probability is not related to time. Figure 2.5a illustrates this model. The mathematical expression is:

$$\begin{aligned} &\mathrm{P}\left(S_{n+1}=s \mid S_1=s_1, S_2=s_2 \cdots, S_n=s_n\right) \\ &\quad=\mathrm{P}\left(S_{n+1}=s \mid S_n=s_n\right). \end{aligned} \tag{2.3}$$

In real life, we can only observe the distinct state that is exposed on the surface, but no intuitive representation of its hidden states exists. Therefore, it is necessary to establish a Markov process with hidden states and get the essential states of events through the observable states set related to the hidden states probability, which is the so-called Hidden Markov Model. HMM is represented by (S, O, A, B, π)[50], as shown in Figure 2.5a:

- $S=\{S_1, S_2, \cdots S_N\}$ represents the hidden states sequence. - $O=\{O_1, O_2, \cdots O_M\}$ represents the observation sequence. - A represents the transition probability matrix between hidden states. - B is the output matrix, representing the transition probability of hidden states to output states. - π is the initial probability matrix, representing the initial probability distribution in hidden states.

When HMM is used in the trajectory prediction, the historical states of traffic participants are represented by observation sequence O, and HMM solves the most likely

future observation sequence. [31] combine HMM with Fuzzy Logic for driver maneuver prediction. In [41], HMM is used for trajectory prediction and risk assessment, and the results of are fed into the decision-making and planning system. Although traditional HMM methods have achieved a great success in predicting driver's maneuvers, they do not consider the impact of interaction-related factors in the prediction process, such that its prediction results are not accurate enough in actual traffic scenes. To solve this issue, [42] propose a vehicle trajectory prediction model based on HMM and Variational Gaussian Mixture Models (GMM) considering interaction-related factors. The vehicle interaction information is obtained by finding the optimal solution of the energy function.

2.3.2.4. Dynamic Bayesian Network (DBN)

Dynamic Bayesian Networks (DBNs) [43] solve the aforementioned issue of HMM when modeling the inter-dependencies among traffic participants, since in order to improve the accuracy of MP, the prediction model should consider at least both vehicle states and the interaction effect between traffic participants. While Bayesian Networks describe static systems, DBNs introduce the concept of time segments to solve timing issues in probabilistic models. Time segment refers to a time template materialized according to DBN, which discretizes continuous time into countable points with preset time granularity.

Generally, the preset time granularity should be consistent with the actual state acquisition frequency, and DBN is trained according to the sensor sampling frequency as the time segment. Besides, the inference and learning methods of DBN need to be converted into Bayesian Networks before they can be directly applied. The architecture of DBN includes a behavior layer, a hidden layer, and an observation layer, as shown in Figure 2.5b. The behavior layer represents the network input information, and the observation layer represents the driver's maneuver. DBN models the effect of interaction between traffic participants when applied to trajectory prediction and perform well in classic machine learning-based methods. Using this architecture, Gindele et al. [59] model the driving maneuvers of multiple vehicles. The input information includes all vehicle states, vehicle interaction relationships, road structures, observation states, etc. [44] apply DBN to judge driving maneuvers and utilize the kinematics model corresponding to each driving maneuver to predict the trajectory. In [45], the vehicle maneuver is predicted by game theory, and then the vehicle motion is judged by DBN which considers the interaction-related factors. In [46], DBN is designed to consider physics-related factors, road-related factors, and interaction-related factors. As maneuver-based methods, DBN models obtain high recognition performance and have been used in several real-world tests [47]. However, DBN still faces the error problem from recognizing maneuvers to generating trajectories. Many methods can only judge two or three maneuvers, such as lane-keeping and lane-changing, and the model's generalization ability is not strong.

(a) (a) (b) (b)

Figure 2.6: Description of (a) Reinforcement Learning and (b) Inverse Reinforcement Learning
Source: *A survey on trajectory-prediction methods for autonomous driving* [19]

2.3.3. Reinforcement Learning based Motion Prediction

Reinforcement learning (RL) is one of three basic ML paradigms, alongside supervised learning and unsupervised learning. RL is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. When RL is used in the field of MP for AD, most methods use the Markov decision process (MDP) to maximize the expected cumulative reward and generate optimal driving policies by learning expert demonstrations, most of which are planning-based methods. A MDP is a tuple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \gamma)$, where $\boldsymbol{S}$ is a finite set of states, $\boldsymbol{A}$ is a finite set of actions, $\boldsymbol{P}$ is a state transition probability matrix, $P^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \;\; S_t = s, A_t = a\right]$, $\boldsymbol{R}$ is a reward function, $R^{\mathrm{a}}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$, and γ is a discount factor. To find the best decision process over all policies, the optimal state-value function $v_\star(s)$ and the optimal action-value function $q_\star(s, a)$ can be calculated as:

$$
\begin{aligned}
v_*(s) &= \max_a \left[R^a_s + \gamma \sum_{s' \in A} P^a_{ss'} v_*\left(s'\right) \right], \\
q_*(s, a) &= R^a_s + \gamma \sum_{s' \in A} P^a_{ss'} \max_{a'} q_*\left(s', a'\right).
\end{aligned}
\tag{2.4}
$$

Using MDP, the RL-based methods can be classified as Inverse Reinforcement Learning (IRL) methods, Generative Adversarial Imitation Learning (GAIL) methods, and Deep IRL (DIRL) methods, which will be discussed below.

2.3.3.1. Inverse Reinforcement Learning (IRL)

Usually, MDP assumes that the reward function is already provided. However, the driver's behavior is always complicated such that manually specifying the weight of the

reward function is inappropriate [48]. IRL learns the reward function according to the expert demonstration (trajectory) to generate the corresponding optimal driving policy as shown in Figure 2.6b, in contrast to the RL approach where we must know the weight of a reward associated to a specific action. These IRL methods may be divided into groups: maximum margin-based and maximum entropy-based methods, where the main difference is the way of learning the weights of the reward function.

Maximum margin-based methods optimize the reward function weights by minimizing the feature expectations between the expert demonstration and the predicted trajectory. For example, [49] use maximum margin planning framework to learn reward functions and learn driving maneuvers for ADSs. However, most margin-based methods are ambiguous in the matching of feature expectations, because some degeneracies can also satisfy the optimal policy of expert demonstration.

On the other hand, Maximum entropy-based methods are more popular because they can use multiple reward functions to explain the ambiguity of experts's behavior [50], most of which are based on linear mapping and can be formulated as:

$$r(\Phi(s)) = \theta^\top \Phi(s) \tag{2.5}$$

where r is the approximation of reward function; Φ is a function to output the features of the state s, and the weight θ will be acquired by training. Several works apply maximum entropy-based IRL (MaxEnt-IRL) to behavior prediction for AVs. In [51], using MaxEnt-IRL acceptability-dependent behavior models are learned from expert's trajectories to generate the stochastic behavior, then the optimum behavior model is chosen by maximizing the social acceptability. [52] leverage IRL with Deep Q-Networks (DQN) to extract the rewards with large state spaces. In [53], interaction-related factors are considered to accomplish probabilistic prediction for ADSs. The distribution for future trajectories is formulated by driving maneuvers. Based on the decision-making mechanism, reward functions are learned using a polynomial trajectory sampler with discrete latent driving intentions in [54].

2.3.3.2. Generative Adversarial Imitation Learning (GAIL)

Instead of learning the reward function from experts' demonstration with IRL, Generative Adversarial Imitation Learning (GAIL) [55] uses the method of GAN to do imitation learning in RL. GAIL directly extracts policies from data where, as proposed by a Generative Adversarial Network (GAN) [56], the core idea of GAIL is that the generator generates a trajectory similar to the expert trajectory as much as possible, and the discriminator tries to judge whether it is an expert trajectory as much as possible. Many articles use

GAIL to complete trajectory prediction for AD. [57] extend GAIL to the optimization of RNN to demonstrate human driver behaviors, and policies and actions are evaluated by the discriminator. In [58], a parameter-sharing extension of GAIL is proposed to model the interaction between multi-agent and can provide agents with domain-specific knowledge. To overcome the shortcomings of GAIL, which only models the next state using the current state, [59] propose a method combining a partially observable Markov decision process (POMDP) within the GAIL framework, and the model is trained using the reward function from the discriminator.

2.3.3.3. Deep Inverse Reinforcement Learning (DIRL)

Since the prediction problem in AD is usually non-linear, i.e. the agent does not traverse in a straight path, it is necessary to use non-linear mapping for generalizable function approximations. In that sense, Deep Inverse Reinforcement Learning (DIRL) is proposed [60] to approximate complex and nonlinear reward functions, which can be expressed as:

$$r(\Phi(s)) = f(\theta, \Phi(s)) \tag{2.6}$$

where f is a nonlinear function. Some DIRL methods take historical trajectories as input. [61] consider the driving style and the road geometry, where the authors first use RL to design MDP, then learn the optimal driving policy from IRL, and use the deep neural network (DNN) to approximate the reward function. In [62], trajectories of traffic participants are encoded by LSTM and the reward network is learned by FCN. Currently, more DIRL-based methods directly use raw perception data. [63] apply FCN for mapping the lidar data to traversability maps. The network is pre-trained to regress to a manual prior cost map and the initialize weights will be fine-tuned by the maximum entropy DIRL network. [64] use RL ConvNet and state visiting frequency (SVF) ConvNet to encode the vehicle's kinematics and obtain the weight of the reward function by back-propagating the loss gradient [65] between expert SVF from expert demonstration and policy SVF from lidar data.

2.3.4. Deep Learning based Motion Prediction

The Deep Learning (DL) methods are the last type of MP methods covered in Chapter since its different approaches are, by far, the most used at this moment in the field of MP in AD to predict the future trajectory of traffic participants., being this thesis focused in these particular methods. Most traditional predictions methods [19], which usually only consider physics-related factors (like the velocity and acceleration of the target vehicle that is going to be predicted) and road-related factors (prediction as close as possible to the road centerline), are only suitable for short-time prediction

tasks [19] and simple traffic scenarios, such as constant velocity (CV) in a highway or a curve (Constant Turn Rate Velocity, CTRV) where a single path is allowed, i.e. multiple choices computation are not required. Recently, MP methods based on DL have become increasingly popular since they are able not only to take into account these above-mentioned factors but also consider interaction-related factors (like agent-agent [66], agent-map [67] and map-map [12]) in such a way the algorithm can adapt to more complex traffic scenarios (intersections, sudden breaks and accelerations, etc.). It must be consider that multimodal, specially in the field of vehicle motion prediction, does not refer necessarily to different directions (e.g. turn to the left, turn to the right, continue forward in an intersection), but it may refer to different predictions in the same direction that model a sudden positive or negative acceleration, so as to imitate a realistic human behaviour in complex situations. As expected, neither classical nor machine learning (ML) methods can model these situations [19].

Figure 2.7: Deep Learning methods applied in Motion Prediction
Source: *A survey on trajectory-prediction methods for autonomous driving* [19]

The main DL-based approaches are: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), GANs, Attention mechanisms and Graph Neural Networks (GNNs). Figure 2.7 illustrates the main idea of using DL to predict the future trajectories of the agents. Since this thesis is focused on developing efficient and accurate DL-based MP models in the field of AD, the theoretical explanation of these different neural networks used in our pipelines will be explained in the Chapter 3 3 along with some physics-based models theory to fully-understand the proposed models. The input is represented by, at least, the physics-factors of the corresponding, though road-factors and interaction-factors are present in most DL algorithms. Then, a neural network extracts the most important features and outputs a future trajectory according to the training process in a supervised way.

In order to classify DL based MP methods, we can identify the different factors (physics, interaction and road) and determine which type of neural network is employed to extract features of the corresponding input. In the literature we mainly distinguish the following inputs and outputs: Motion history (physics-based factors), Social information (interaction-based factors), Map information (road-relared factors), how the model re-

turns the output trajectory and its corresponding distribution. Table 2.1 summarizes several SOTA methods.

Table 2.1: Main state-of-the-art Deep Learning methods for Motion Prediction. Main categories are Encoder (splitted into motion history, social info (agent interactions) and map info (physical information)), Decoder, Output representation and Distribution over future trajectories

Method	**Encoder**			**Decoder**	**Output**	**Trajectory Distribution**
	Motion history	Social info	Map info			
SocialLSTM [68]	LSTM	spatial pooling	–	LSTM	states	samples
SocialGan [66]	LSTM	maxpool	–	LSTM	states	samples
Jean [69]	LSTM	attention	–	LSTM	states	GMM
TNT [18]	polyline	maxpool, attention	polyline	MLP	states	weighted set
LaneGCN [12]	1D-conv	GNN	GNN	MLP	states	weighted set
WIMP [16]	LSTM	GNN+attention	polyline	LSTM	states	GMM
VectorNet [70]	polyline	maxpool, attention	polyline	MLP	states	unimodal
SceneTransformer [71]	attention	attention	polyline	attention	states	weighted set
HOME [7]	raster	attention	raster	conv	states	heatmap
GOHOME [8]	1D-conv+GRU	GNN	GNN	MLP	states	heatmap
MP3 [72]	raster	conv	raster	conv	cost function	weighted samples
CoverNet [73]	raster	conv	raster	lookup	states	GMM w/ dyn. anch.
DESIRE [74]	GRU	spatial pooling	raster	GRU	states	samples
MFP [14]	GRU	RNNs+attention	raster	GRU	states	samples
MANTRA [75]	GRU	–	raster	GRU	states	samples
PRANK [76]	raster	conv	raster	lookup	states	weighted set
IntentNet [67]	raster	conv	raster	conv	states	unimodal
Multimodal [77]	raster	conv	raster	conv	states	weighted set
MultiPath [78]	raster	conv	raster	MLP	states	GMM w/ static anchors
MultiPath++ [9]	LSTM	RNNs+maxpool	polyline	MLP	control poly	GMM
PLOP [79]	LSTM	conv	raster	MLP	state poly	GMM
Trajectron++[6]	LSTM	RNNs+attention	raster	GRU	controls	GMM
CRAT-PRED[11]	LSTM	GNN+attention	–	MLP	states	weighted set
R2P2 [80]	GRU	–	polyline	GRU	motion	samples
DKM [81]	raster	conv	raster	conv	controls	weighted set

- **Motion history**: Most methods encode the sequence of past observed states using 1D-convolution [12] [69], able to model spatial information, or via a recurrent net [68] (LSTM, GRU), which are more useful to handle temporal information. Other methods that use a raster version of the whole scenario represent the agent states rendered as a stack of binary mask images depicting agent oriented bounding boxes [7]. On the other hand, other approaches encode the past history of the agents in a similar way to the road components of the scene given a set of vectors or polylines [18], [70] that can model the high-order interactions among all components, or even employing attention to combine features across road elements and agent interactions [71].

- **Social information**: In complex scenarios, motion history encoding of a particular target agent is not sufficient to represent the latent space of the traffic situation, but the algorithm must deal with a dynamic set of neighbouring agents around the target agent. Common techniques are aggregating neighbour motion history with a permutation-invariant set operator: soft attention [71], a combination of soft attention and RNN [9] / GNN [11] or social pooling [66], [68]. Raster based approaches rely on 2D convolutions [78] [72] over the spatial grid to implicitly capture agent interactions in such a way long-term interactions are dependent on the neural network receptive fields.

- **Map information**: High-fidelity maps [82] have been widely adopted to provide offline information (also known as physical context) to complement the online information provided by the sensor suite of the vehicle and its corresponding algorithms. Recent learning-based approaches [67], [83], [84], which present the benefit of having probabilistic interpretations of different behaviour hypotheses, require to build a representation to encode the trajectory and map information. Map information is probably the feature with the clearest dichotomy: raster vs vector treatment. The raster approach encodes the world around the particular target agent as a stack of images (generally from a top-down orthographic view, also known as Bird's Eye View). This world encoding may include from agent state history, agent interactions and usually the road configuration, integrated all this different-sources information as a multi-channel image [7], in such a way the user can use an off-the-shelf Convolutional Neural Network (CNN) based pipeline in order to leverage this powerful information. Nevertheless, this representation has several downsides: constrained field of view, difficulty in modeling long-range interactions and even difficulty in representing continuous physical states due to the inherent world to image (pixel) discretization. On the other hand, the polyline approach may describe curves, such as lanes, boundaries, intersections and crosswalks, as piecewise linear segments, which usually represents a more compact and efficient representation than using CNNs due to the sparse nature of road networks. Some state-of-the-art algorithms not only describe the world around a particular agent as a set-of-polylines [16] [18] in an agent-centric coordinate system, but they also leverage the road network connectivity structure [12] [85] treating road lanes as a set of nodes (waypoints) and edges (connections between waypoints) in a graph neural network so as to include the topological and semantic information of the map.

- **Decoder**: Pioneering works of DL based MP usually adopt the autoencoder architecture, where the decoder is often represented by a recurrent network (GRU, LSTM, etc., specially designed to handle temporal information) to generate future trajectories in an autoregressive way, or by CNNs [7] [8] / MLP [12] [11] using the non-autoregressive strategy. The method may use an autoregressive strategy where the pipeline generates tokens (in this case, positions or relative displacements) in a sequential manner, in such a way the new output is dependent on the previously generated output, whilst MLP [11], CNN [7] or transformer [71] based strategies usually follow a non-autoregressive strategy, where from a latent space the whole future trajectory is predicted.

- **Output**: The most popular model output representation is a sequence of states (absolute positions) or state differences (relative displacements for any dimension considered). The spacetime trajectory may be intrinsically represented as a continous polynomial representation or a sequence of sample points. Other works [7] [8]

first predict a heatmap and then decode the corresponding output trajectories after sampling points from the heatmap, whilst [72] [86] learn a cost function evaluator of trajectories that are enumerated heuristically instead of being generated by a learned model.

- **Trajectory Distribution**: The choice of output trajectory distributions has several approaches on downstream applications. Regardless the agent to be predicted is described as a (non-)holonomic [87] platform, an intrinsic property of the motion prediction problem is that the agent must follow one of a diverse set of possible future trajectories. A popular choice to represent a multimodal prediction are Gaussian Mixture Models (GMMs) due to their compact parameterized form, where mode collapse (associated frequently to GMMs) is addressed through the use of trajectory anchors [78] or training tricks [77]. Other approaches model a discrete distribution via a collection of trajectory samples extracted from a latent space and decoded by the model [80] or over a set of trajectories (fixed or a priori learned) [12].

2.4. Summary

In order to finish this Chapter, we perform a brief comparison between the different methods (Physics-based, Classic ML, Reinforcement Learning and DL) in terms of accuracy, prediction horizon, computation cost and applications in the AD field.

Table 2.2: Summary of Motion Prediction methods features. Short-term and long-term characterize prediction horizons of no more than 1-s and no less than 3-s, respectively.

Methods	Accuracy	Prediction Horizon	Computation Cost	Applications
Physics-based	High in short-term prediction, low in other prediction horizon	Short	Small	Colision risk analysis
Classic Machine Learning-based	Good at recognizing maneuvers but generalization ability is poor	Medium	Medium	Maneuver recognition
Deep Learning-based	High in considering some factors	Long	Relatively high	More and more applied in real-world
Reinforcement Learning-based	Relatively high, prediction methods are relatively few	Long	High	More applied in planning

- Physics-Based Methods are suitable for the movement of vehicles, which can be accurately described by kinematics or dynamics models. Given a suitable physics model, these methods can be applied to a variety of scenarios at small computational cost and in a short time but without training. However, the prediction results based on such models heavily depends on the inputs and the model selection. The inputs are closely related to human or machine drivers, influenced by the driving environment or the interactions with other participants. Therefore, without the capability to describe such factors, physics-based models are limited to short-term prediction and in static scenes. Because of its simplicity and fast response, these methods can be easily used in real applications for ADSs, such as collision risk analysis.

- Classic Machine Learning-Based Methods, compared with physics-based methods, are able to consider more factors and its accuracy is relatively high with a longer prediction length at a higher computing cost. Most of these methods are maneuver-based methods, which predicts the trajectory with the maneuver known as a prior. However, vehicle maneuvers of human drivers are usually diverse and vary greatly in different scenarios such that the generalization ability of is poor. In real applications for AVs, such methods are used in scenarios such as lane change studies, leveraging their advantages in maneuver recognition.

- Reinforcement Learning-Based Methods imitate the human decision-making process and obtain the reward function through learning the expert demonstration to generate the corresponding optimal driving policy. They can continuously evolve through learning and adapt to complex environments and long prediction horizons. Such methods probably generate higher accuracy trajectories than deep learningmethods in a longer time domain. However, most of these methods are typically computationally expensive in their recovery of an expert cost function and require long training times. In real applications for AVs, reinforcement learning-based trajectory prediction methods are more applied to trajectory planning, taking its advantages in the decision-making process.

- Deep Learning-Based Methods can perform accurate predictions in a longer time horizon with respect to traditional methods that are only suitable for simple scenes and short-term prediction. By means of powerful neural networks, such as RNNs, CNNs, GANs, Attention mechanisms or GNNs for feature extraction, physics-related, interaction-related and road-related factors are processed as inputs to the model. Furthermore, they can adapt to more complex environments and a longer prediction horizon. DL-based methods require to use a large amount of data for training. Besides, with the increase of consideration factors and the increase of the number of network layers, the computing costs and time increases sharply. Such methods can naturally generate multi-modal trajectories, which is consistent with the diversity of vehicles' maneuvers. In real applications for AVs, it is necessary to reach a balance between calculation time and model complexity to ensure the real-time performance and safety of AVs. At present, more and more real-world trials use thesemethods to predict thefuture trajectory of traffic participants.

As observed in Table 2.2 and discussed in this section summarizing the different MP algorithms, we focus this thesis on DL methods since they are the most suitable methods for long-term prediction with a relatively high computation cost, specially focusing on the ability of extracting and combining the latent spaces of the different inputs by means of SOTA algorithms.

Chapter 3

Theoretical Background

Desde que el mundo cambió, estamos mucho más unidos
con los Digimon, luchamos juntos contra el mal.
Algo extraño pasaba, Digievolucionaban,
en tamaño y color, ellos son los Digimon.

Opening 1 de Digimon: "Butterfly"
Autor original: Kōji Wada

3.1. Introduction

As commented in previous sections, the MP algorithms covered by this thesis range from tracking multiple objects and subsequent prediction with physics-based methods to the most recent SOTA techniques to compute the deep traffic context and then decode multimodal predictions with associated confidences, assuming the physical information is given and surrounding participants have been multi-tracked beforehand. Throughout this Chapter, an in-depth theoretical study will be made of those algorithms, neural networks or heuristics that form a direct part of the development of this work in order to address the proposed methods in Chapter 4.

First of all, we will start with the mathematical formulation of the methods to perform physics-based Multi-Object Tracking, from the well-known techniques Kalman Filter [24] for the agents states estimation to the Hungarian algorithm [88] for the association of detections and trackers, which represents the preliminary stage before carrying out the subsequent prediction. On top of that, since several single-trajectory models (Constant Turn Rate and Velocity (CTRV), Constant Turn Rate and Acceleration (CTRA)) are used to compute the most plausible centerlines in the Argoverse 1 [4] and Argoverse 2 [5], we will review the state transition equations to properly understand the constraints for each model. Furthermore, the principal DL techniques (e.g. 1D-CNN, LSTM, GCN, Attention mechanisms) and training losses used in this work to encode and decode the aimed

multimodal will be stated, first a general mathematical formulation and applications and then how the corresponding technique is used in the MP field.

3.2. Physics-based algorithms

As stated in Chapter 2 Section 2.1, initially researchers rely on physics-based methods which are basic and straightforward. These methods may not offer high accuracy, but many models use the underlying idea of physics-based models to improve their accuracy. Physics-based approaches yield better results when the movement of vehicles is described by kinematics or dynamics models accurately. Nevertheless, the physical model of traffic participants is constantly evolving, so most physics-based models are only applicable for short-term predictions of no more than one second. In this Section we will study the mathematical formulation of the physics-based prediction algorithms, as well as the data association problem regarding the Multi-Object Tracking stage, that are directly related to our algorithms proposed in Chapter 4.

3.2.1. Kalman Filter under the hood

The Kalman Filter [24] is a recursive algorithm used for estimating the state of a dynamic system in the presence of noise. It is widely used in various fields such as engineering, control systems, and robotics. The algorithm works by combining a prediction of the system state based on a mathematical model with measurements (updates) from sensors to improve the accuracy of the state estimation, as shown in Figure 3.1. The Kalman Filter is a powerful algorithm for state estimation, and it has many variations and extensions that can handle various types of systems and measurements. The filter can be applied to non-linear systems using the Extended Kalman Filter or the Unscented Kalman Filter, and it can handle multiple models using the multiple model Kalman Filter. The filter can also be used for smoothing, which involves estimating the state of the system based on past and future measurements.

In its most simple version, the Kalman filter assumes that the system can be described using a set of linear equations, where the state of the system can be represented by a vector $\mathbf{x}_k$, and the measurements can be represented by a vector $\mathbf{z}_k$. The Kalman filter also assumes that the noise in the system follows a Gaussian distribution.

The state estimation process is performed in two steps: the prediction step and the update step. In the prediction step, the current state of the system is predicted based on the previous state and a mathematical model of the system. In the update step, the predicted state is corrected based on a measurement from a sensor.

The prediction step can be represented using the following equations:

Figure 3.1: Overview of the Kalman Filter Predict-Update cycle. The Kalman filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate. The estimate is updated using a state transition model and measurements
Source: *An introduction to the kalman filter* [89]

$$\begin{aligned} \hat{\mathbf{x}}_k^- &= \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_k \\ \mathbf{P}_k^- &= \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \end{aligned} \tag{3.1}$$

where $\hat{\mathbf{x}}_k^-$ is the predicted state of the system at time k, $\hat{\mathbf{x}}_{k-1}$ is the estimated state of the system at time $k-1$, $\mathbf{A}$ is the state transition matrix, $\mathbf{B}$ is the control-input matrix, $\mathbf{u}_k$ is the control input at time k, $\mathbf{P}_k^-$ is the predicted error covariance matrix, and $\mathbf{Q}$ is the process noise covariance matrix. Note that the state transition, control-input and process noise covariance matrix values do not depend on the timestep time k.

On the other hand, the update step can be represented using the following equations:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R})^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-) \\ \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^- \end{aligned} \tag{3.2}$$

where $\mathbf{K}_k$ is the Kalman gain, $\mathbf{H}$ is the measurement matrix, $\mathbf{R}_k$ is the measurement noise covariance matrix and $\mathbf{I}$ the identity matrix of the corresponding dimension. The Kalman gain determines the relative weight given to the predicted state and the measurement, and is adjusted based on the measurement noise covariance matrix. The measurement matrix relates the measurements to the state variables, and is used to convert the measurements into the same units as the state variables.

3.2.2. Hungarian algorithm formulation

The Hungarian algorithm [88], also known as the Munkres algorithm or the Kuhn-Munkres algorithm, is an efficient algorithm for solving the assignment problem in combinatorial optimization. The main concept behind this algorithm is an assignment problem that involves finding the optimal assignment of n workers to n jobs, given the cost of assigning each worker to each job.

The Hungarian algorithm works by iteratively finding a set of independent zero-cost assignments, which correspond to a perfect matching in a bipartite graph. These assignments are then used to reduce the problem size and find a new set of independent zero-cost assignments, until all workers are assigned to jobs.

The Hungarian algorithm has a time complexity of $\mathrm{O}(n^3)$, which makes it one of the most efficient algorithms for solving the assignment problem. In this algorithm, the cost matrix C is assumed to be an $n \times n$ matrix, where each element $C_{i,j}$ represents the cost of assigning worker i to job j. The matrix M represents the matching, where each element $M_{i,j}$ is 1 if worker i is assigned to job j, and 0 otherwise.

As observed in Algorithm 1, the Hungarian algorithm iteratively finds an uncovered zero in the cost matrix C and adds it to the matching M, while also adjusting the cost matrix to ensure that all rows and columns are covered by the matching. This is done by finding the minimum uncovered cost in each row and subtracting it from all uncovered costs in the row, and finding the minimum cost in each uncovered column and adding it to all uncovered costs in the column. Once all rows and columns are covered, the algorithm returns the final matching M.

In conclusion, the Hungarian algorithm is a powerful optimization algorithm that solves the assignment problem in polynomial time. The algorithm has a wide range of applications and can be easily adapted to handle various constraints and objectives. The algorithm is also guaranteed to find the optimal solution to the assignment problem, which makes it a valuable tool in many practical settings.

3.2.3. State Transition Equations of Single-Trajectory Models

As stated in Chapter 2 Section 2.3.1.1, Single-trajectory prediction methods are used in the field of motion estimation and control, to predict the future state of an object

Algorithm 1: The Hungarian algorithm for solving the minimum cost perfect matching problem.

```
Input  : A cost matrix C of size n × n
Output: A minimum cost perfect matching
Initialize the label vectors u_i = min_{j∈1,...,n} C_{i,j} and v_j = 0 for all i, j; Initialize the empty
 matching M; while |M| < n do
    Choose an unmatched row i; Initialize the set T = i and the predecessor vector P = ∅; while
     true do
        Let S be the set of columns j such that i ∈ T and C_{i,j} = u_i + v_j; If |S| > 0, choose any
         column j in S; else
            Choose a column j such that v_j = min_{k∈1,...,n} v_k and let S be the set of rows i such
             that j ∈ M or C_{i,j} = u_i + v_j; Increment each u_i for i ∈ T by
             δ = min_{i∈T,j∈S}(u_i + v_j − C_{i,j}); Decrement each v_j for j ∈ S by δ; for each row i ∈ T
             and each column j ∈ S do
                if C_{i,j} = u_i + v_j then
                    Add the edge (i, j) to the alternating tree represented by M and P; if j is
                     unmatched then
                        Augment M along the alternating tree to create a larger matching; return
                         the updated matching M;
                    end
                    else
                        Add j to T and continue the while loop;
                    end
                end
            end
        end
    end
end
```

based on its current state and motion. In these methods, the agents are mostly assumed to comply with motion models that describe their dynamic behavior in such a way these are not able to consider the road-related factors and the uncertainty of the current state is unreliable for long-term prediction. Then, these models should only be used for estimating unimodal trajectories of the surrounding agents in the short-term (no more than 1-s).

Figure 3.2: Overview of Single Trajectory prediction methods. Every sophisticated model can be transformed into a simpler one by setting one state veriable to zero
Source: *Comparison and evaluation of advanced motion models for vehicle tracking* [90]

As illustrated in Figure 3.2, several single-trajectory prediction models have been proposed in the literature, including Constant Velocity (CV), Constant Acceleration (CA), Constant Turn Rate and Velocity (CTRV), Constant Turn Rate and Accelera-

tion (CTRA), Constant Speed and Angular Velocity (CSAV), and Constant Curvature Acceleration (CCA)/Constant Speed and Angular Acceleration (CSAA). In this Section we focus on the state transition equations of the different methods and how a sophisticated model (e.g. CTRA) can be transformed into a simpler one by setting one state veriable to zero. For example, CTRA is transformed to CTRV if the agent acceleration a is set to zero. Similarly, CTRA is transformed to CA if the angular speed ω is set to 0.

3.2.3.1. Constant Velocity (CV)

The Constant Velocity (CV) model is a popular single-trajectory prediction model used in the field of motion planning and control. It assumes that the object moves at a constant velocity in a straight line. The state vector of this model includes position and velocity. This means that the position of the object at time t can be calculated by adding the product of the velocity and time to the position at time $t-1$. The velocity at time t is assumed to be the same as the velocity at time $t-1$.

The state vector for the CV model includes position and velocity:

$$x = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{3.3}$$

where (x, y) is the position of the object and (v_x, v_y) is the velocity of the object in the x and y directions, respectively.

The state transition model for the CV model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

where Δt is the time step.

The measurement model for the CV model is:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.5}$$

which means that the measurements are the position of the object in the x and y directions.

3.2.3.2. Constant Acceleration (CA)

The Constant Acceleration (CA) assumes that the object moves at a constant acceleration in a straight line. The state vector of this model includes position, velocity, and acceleration. This means that the position of the object at time t can be calculated by adding the product of the velocity and time to the position at time $t-1$, and adding half the product of the acceleration and the square of the time step to the position at time $t-1$. The velocity at time t is assumed to be the sum of the velocity at time $t-1$ and the product of the acceleration and the time step.

The state vector for the CA model includes position, velocity, and acceleration:

$$x = \begin{bmatrix} x \\ y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \tag{3.6}$$

where (x, y) is the position of the object, (v_x, v_y) is the velocity of the object in the x and y directions, respectively, and (a_x, a_y) is the acceleration of the object in the x and y directions, respectively.

The state transition model for the CA model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

where Δt is the time step.

The measurement model for the CA model is the same as the CV model:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

which means that the measurements are the position of the object in the x and y directions.

3.2.3.3. Constant Turn Rate Velocity (CTRV)

The Constant Turn Rate and Velocity (CTRV) assumes that the object moves at a constant velocity with a constant turn rate. The state vector of this model includes position, velocity, turn rate, and yaw angle. This means that the position of the object at time t can be calculated by integrating the velocity over time, and the heading angle can be calculated by adding the product of the turn rate and time to the heading angle at time $t-1$. The velocity and turn rate are assumed to be constant over time.

The state vector for the CTRV model includes position, velocity, turn rate, and yaw angle:

$$x = \begin{bmatrix} x \\ y \\ v \\ \dot{\psi} \\ \psi \end{bmatrix} \tag{3.9}$$

where (x, y) is the position of the object, v is the velocity of the object, $\dot{\psi}$ is the turn rate of the object, and ψ is the yaw angle of the object.

The state transition model for the CTRV model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t \cos(\psi) & -v \sin(\psi)\Delta t & 0 \\ 0 & 1 & \Delta t \sin(\psi) & v \cos(\psi)\Delta t & 0 \\ 0 & 0 & 1 & \dot{\psi}\Delta t & 0 \\ 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

where Δt is the time step.

The measurement model for the CTRV model is the same as the CV and CA models:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

which means that the measurements are the position of the object in the x and y directions.

3.2.3.4. Constant Turn Rate Acceleration (CTRA)

The Constant Turn Rate and Acceleration (CTRA) assumes that the object moves at a constant velocity with a constant turn rate and a constant acceleration. The state vector of this model includes position, velocity, turn rate, acceleration, and yaw angle.

This means that the position of the object at time t can be calculated by integrating the velocity over time, and the heading angle can be calculated by adding the product of the turn rate and time to the heading angle at time $t-1$. The velocity, turn rate, and acceleration are assumed to be constant over time.

The state vector for the CTRA model includes position, velocity, turn rate, acceleration, and yaw angle:

$$x = \begin{bmatrix} x \\ y \\ v \\ \dot{\psi} \\ a \\ \psi \end{bmatrix} \tag{3.12}$$

where (x, y) is the position of the object, v is the velocity of the object, $\dot{\psi}$ is the turn rate of the object, a is the acceleration of the object, and ψ is the yaw angle of the object.

The state transition model for the CTRA model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t \cos(\psi) & -v \sin(\psi)\Delta t & 0.5a \cos(\psi)\Delta t^2 & 0 & & & & & \\ 0 & 1 & \Delta t \sin(\psi) & v \cos(\psi)\Delta t & 0.5a \sin(\psi)\Delta t^2 & 0 & & & & & \\ 0 & 0 & 1 & \dot{\psi}\Delta t & a\Delta t & 0 & & & & & \\ 0 & 0 & 0 & 1 & \Delta t & 0 & & & & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & & & & & & & & & & \end{bmatrix} \tag{3.13}$$

where Δt is the time step.

The measurement model for the CTRA model is the same as the CV, CA, and CTRV models:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.14}$$

which means that the measurements are the position of the object in the x and y directions.

3.2.3.5. Constant Speed and Acceleration in the Vertical (CSAV)

The Constant Speed and Acceleration in the Vertical (CSAV) model is a single-trajectory prediction model used in the field of motion planning and control. It is similar to the CV and CA models, but accounts for changes in altitude or elevation by assuming

a constant speed and acceleration in the vertical direction. This means that the position of the object at time t can be calculated by integrating the velocity over time, and the altitude or elevation can be calculated by adding the product of the acceleration and time squared to the altitude or elevation at time $t-1$. The horizontal speed is assumed to be constant over time.

The state vector for the CSAV model includes position, horizontal speed, vertical speed, and altitude or elevation:

$$x = \begin{bmatrix} x \\ y \\ v_x \\ v_z \\ z \end{bmatrix} \tag{3.15}$$

where (x, y) is the position of the object, v_x is the horizontal speed of the object, v_z is the vertical speed of the object, and z is the altitude or elevation of the object.

The state transition model for the CSAV model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & & & & & \end{bmatrix} \tag{3.16}$$

where Δt is the time step.

The measurement model for the CSAV model is the same as the CV and CA models:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

which means that the measurements are the position of the object in the x and y directions.

3.2.3.6. Constant Speed and Coordinated Acceleration (CSAA)

The Constant Speed and Coordinated Acceleration (CSAA) (also referred as Constant Coordinated Acceleration (CCA) in the literature) assumes coordinated acceleration or deceleration between two vehicles or objects, and can be used for collision avoidance or cooperative maneuvering. The CSAA model assumes a constant speed in the horizontal direction and a coordinated acceleration or deceleration in the vertical direction.

The state vector for the CSAA model includes position, horizontal speed, vertical speed, and altitude or elevation for both vehicles or objects:

$$x = \begin{bmatrix} x_1 \\ y_1 \\ v_{x1} \\ v_{z1} \\ z_1 \\ x_2 \\ y_2 \\ v_{x2} \\ v_{z2} \\ z_2 \end{bmatrix} \tag{3.18}$$

where (x_1, y_1) and (x_2, y_2) are the positions of the two vehicles or objects, v_{x1} and v_{x2} are the horizontal speeds of the two vehicles or objects, v_{z1} and v_{z2} are the vertical speeds of the two vehicles or objects, and z_1 and z_2 are the altitudes or elevations of the two vehicles or objects.

The state transition model for the CSAA model can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

where Δt is the time step.

The measurement model for the CSAA models is the same as the CTRV and CTRA models:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.20}$$

which means that the measurements are the position of the object in the x and y directions.

3.3. Convolutional Neural Networks

Convolutional neural networks (CNNs) are commonly used for image and signal processing tasks. The type of CNN architecture used depends on the nature of the input data. Here are the differences between the three types of CNNs:

1D Convolutional Neural Networks (1D CNNs): 1D CNNs are used for processing sequential data such as time series data, speech signals, and text data. The input to a 1D CNN is a one-dimensional sequence of data, such as a time series of sensor readings. 1D CNNs typically have fewer parameters and are computationally efficient compared to 2D and 3D CNNs.

2D Convolutional Neural Networks (2D CNNs): 2D CNNs are used for processing 2D images. The input to a 2D CNN is a two-dimensional image represented as a matrix of pixels. The convolutional layers in a 2D CNN apply filters that slide over the image to extract local features. The output of each convolutional layer is a set of 2D feature maps. 2D CNNs are widely used for image classification, object detection, and image segmentation.

3D Convolutional Neural Networks (3D CNNs): 3D CNNs are used for processing 3D volumetric data such as CT scans, MRI images, and video data. The input to a 3D CNN is a three-dimensional volume represented as a sequence of 2D images. The convolutional layers in a 3D CNN apply filters that slide over the 3D volume to extract local features. The output of each convolutional layer is a set of 3D feature maps. 3D CNNs are used for tasks such as medical image analysis, video classification, and action recognition.

In summary, 1D CNNs are used for processing sequential data (such as time series data, audio signals, and text), 2D CNNs are used for image processing, and 3D CNNs are used for volumetric data. As we will see in Chapter 4, in this work we focus on 1D CNNs. The basic building blocks of a 1D CNN are convolutional layers, which learn local patterns in the input data by applying a set of filters to it, as illustrated in Figure 3.3. Each filter slides over the input sequence and performs a dot product operation at each position to generate a feature map. These feature maps capture the presence of certain patterns at different positions in the input sequence.

The architecture of a 1D CNN typically consists of several convolutional layers, followed by one or more fully connected layers for classification or regression. The output of each convolutional layer is fed into the next layer, with optional pooling layers in between to reduce the spatial dimension of the feature maps. The final output of the network is obtained by passing the output of the last fully connected layer through a suitable activation function, such as softmax for classification or linear for regression.

Let x be a 1-dimensional input sequence of length n, represented as a vector $x = [x_1, x_2, ..., x_n]$. Let k be a filter of length m, represented as a vector $k = [k_1, k_2, ..., k_m]$. We can apply the filter k to the input sequence x by computing a convolution operation:

Figure 3.3: Example of CNN architecture to process 1D-input signals
Source: *Multiclass damage identification in a full-scale bridge using optimally tuned one-dimensional convolutional neural network* [91]

$$(k * x)_i = \sum_{j=1}^{m} k_j x_{i+j-m}$$

where $*$ denotes the convolution operation, and i ranges from m to n. This operation produces a new sequence of length $n - m + 1$, representing the local features extracted from the input sequence x by the filter k.

To learn multiple filters and extract different types of features, we use multiple convolutional layers in the network. Each layer applies a set of filters, and produces a set of feature maps. The feature maps can be computed as:

$$h_{i,j} = f(\sum_{k=1}^{m} W_{j,k} x_{i+k-m} + b_j)$$

where h is the feature map at position i and filter j, W is the weight matrix of the j-th filter, b is the bias term for the j-th filter, and f is an activation function (such as ReLU or sigmoid).

After each convolutional layer, we typically apply a pooling layer to reduce the dimensionality of the features and increase the translation invariance of the network. The most common pooling operation is max pooling, which selects the maximum value within a window of size p:

$$y_{i,j} = \max_{k=0}^{p-1} h_{i+k,j}$$

where y is the output of the pooling layer at position i and filter j.

Finally, we can use one or more fully connected layers to perform the final classification or regression task. The output of the pooling layer is flattened into a vector, and fed into a set of fully connected layers, with optional dropout regularization and batch normalization:

$$z_j = f(\sum_{i=1}^{n} W_{i,j} y_i + b_j)$$

where z is the output of the j-th fully connected layer, W is the weight matrix, y is the flattened output of the pooling layer, b is the bias term, and f is an activation function.

3.4. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network that are designed to process sequential data, where the order of the input matters. They are widely used in a variety of applications such as natural language processing, speech recognition, image captioning, and time series prediction.

The general theory of RNNs is that they have a feedback loop in their architecture that allows information to be passed from one time step to the next, thereby enabling the network to capture dependencies between the inputs at different time steps. The input at each time step is fed into the network along with the output from the previous time step, and the network uses this information to generate a new output.

There are several types of RNNs, including the basic RNN, the Gated Recurrent Unit (GRU), and the Long Short-Term Memory (LSTM). The basic RNN is the simplest form of RNN, where the output at each time step is a function of the input at that time step and the output from the previous time step. However, it suffers from the vanishing gradient problem, where the gradients become exponentially small as they propagate through time, making it difficult to learn long-term dependencies.

The GRU is a variation of the basic RNN that uses gating mechanisms to control the flow of information through the network. It has fewer parameters than the LSTM and can be faster to train, but it may not perform as well as the LSTM on tasks that require more complex temporal dependencies.

3.5. Long Short-Term Memory

LSTM networks were introduced to address the vanishing gradient problem in standard RNNs. They achieve this by using a memory cell to store information over long periods of time, and three gating mechanisms to control the flow of information through the network.

The three gates are called the input gate, forget gate, and output gate, and they are responsible for deciding which information to store in the memory cell, which information to forget, and which information to output, respectively.

The equations for the input, forget, and output gates are as follows:

- Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- Forget gate: $f_t = sigmoid(W_f * [h_{t-1}, x_t] + b_f)$
- Input gate: $o_t = sigmoid(W_o * [h_{t-1}, x_t] + b_o)$

Moreover,

- Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- Forget gate: $f_t = sigmoid(W_f * [h_{t-1}, x_t] + b_f)$
- Input gate: $o_t = sigmoid(W_o * [h_{t-1}, x_t] + b_o)$

$$\text{t} = \tanh(W_c \cdot [ht-1, x_t] + b_c) \; C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \; h_t = o_t \cdot \tanh(C_t)$$

Here, x_t is the input at time t, h_{t-1} is the hidden state from the previous time step, i_t is the input gate activation, f_t is the forget gate activation, and o_t is the output gate activation. W_i, W_f, and W_o are weight matrices, and b_i, b_f, and b_o are bias vectors. $\tilde{C}t$ is the candidate value, C_t is the new memory cell value, and $Ct-1$ is the previous memory cell value. W_c is a weight matrix and b_c is a bias vector.

The memory cell is updated based on the input, forget, and output gates, as well as a new candidate value that is computed based on the current input and hidden state. The equations for the memory cell and candidate value are as follows:

Candidate value: $\text{t} = \tanh(\text{W}_c * [ht-1, x_t] + b_c) Memorycell : C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Here, t is the candidate value, $\text{C}_t isthenewmemorycellvalue, andCt-1istheprevious memorycellvalu$

Finally, the hidden state at time t is computed based on the output gate and the new memory cell value, using the following equation:

Hidden state: $\text{h}_t = o_t * tanh(C_t)$

This equation scales the memory cell value by the output gate activation, then applies a hyperbolic tangent function to obtain the new hidden state.

In summary, LSTMs use a memory cell and three gating mechanisms to learn long-term dependencies in sequential data. The input, forget, and output gates control the flow of information through the network, while the memory cell stores information over time. The equations for LSTMs involve computing the input, forget, and output gate activations, the candidate value, the new memory cell value, and the new hidden state.

3.6. Generative Adversarial Networks

Throughout most of this book, we have talked about how to make predictions. In some form or another, we used deep neural networks to learn mappings from data examples to labels. This kind of learning is called discriminative learning, as in, we'd like to be able to discriminate between photos of cats and photos of dogs. Classifiers and regressors are both examples of discriminative learning. And neural networks trained by backpropagation have upended everything we thought we knew about discriminative learning on large complicated datasets. Classification accuracies on high-res images have gone from useless to human-level (with some caveats) in just 5-6 years. We will spare you another spiel about all the other discriminative tasks where deep neural networks do astoundingly well.

But there is more to machine learning than just solving discriminative tasks. For example, given a large dataset, without any labels, we might want to learn a model that concisely captures the characteristics of this data. Given such a model, we could sample synthetic data examples that resemble the distribution of the training data. For example, given a large corpus of photographs of faces, we might want to be able to generate a new photorealistic image that looks like it might plausibly have come from the same dataset. This kind of learning is called generative modeling.

Until recently, we had no method that could synthesize novel photorealistic images. But the success of deep neural networks for discriminative learning opened up new possibilities. One big trend over the last three years has been the application of discriminative deep nets to overcome challenges in problems that we do not generally think of as supervised learning problems. The recurrent neural network language models are one example of using a discriminative network (trained to predict the next character) that once trained can act as a generative model.

In 2014, a breakthrough paper introduced Generative adversarial networks (GANs) :cite:'Goodfellow.Pouget-Abadie.Mirza.ea.2014', a clever new way to leverage the power of discriminative models to get good generative models. At their heart, GANs rely on the idea that a data generator is good if we cannot tell fake data apart from real data. In statistics, this is called a two-sample test - a test to answer the question whether datasets $X = \{x_1, \ldots, x_n\}$ and $X' = \{x'_1, \ldots, x'_n\}$ were drawn from the same distribution. The main difference between most statistics papers and GANs is that the latter use this idea in a constructive way. In other words, rather than

just training a model to say "hey, these two datasets do not look like they came from the same distribution", they use the [two-sample test](https://en.wikipedia.org/wiki/Two-sample$_hypothesis_testing)toprovidetrainingsignalstoagenerativemodel.Thisallowsustoimprovethedat$

It then applies a function to generate $\mathbf{x}' = G(\mathbf{z})$. The goal of the generator is to fool the discriminator to classify $\mathbf{x}' = G(\mathbf{z})$ as true data, *i.e.*, we want $D(G(\mathbf{z})) \approx 1$. In other words, for a given discriminator D, we update the parameters of the generator G to maximize the cross-entropy loss when $y = 0$, *i.e.*,

$$\max_G \{-(1-y)\log(1-D(G(\mathbf{z})))\} = \max_G \{-\log(1-D(G(\mathbf{z})))\}.$$

If the generator does a perfect job, then $D(\mathbf{x}') \approx 1$, so the above loss is near 0, which results in the gradients that are too small to make good progress for the discriminator. So commonly, we minimize the following loss:

$$\min_G \{-y\log(D(G(\mathbf{z})))\} = \min_G \{-\log(D(G(\mathbf{z})))\},$$

which is just feeding $\mathbf{x}' = G(\mathbf{z})$ into the discriminator but giving label $y = 1$.

To sum up, D and G are playing a "minimax" game with the comprehensive objective function:

$$\min_D \max_G \{-E_{x\sim\text{Data}} \log D(\mathbf{x}) - E_{z\sim\text{Noise}} \log(1-D(G(\mathbf{z})))\}.$$

3.6.1. Summary

3.7. Attention Mechanisms

3.8. Self-Attention and Positional Encoding

In deep learning, we often use CNNs or RNNs to encode sequences. Now with attention mechanisms in mind, imagine feeding a sequence of tokens into an attention mechanism such that at each step, each token has its own query, keys, and values. Here, when computing the value of a token's representation at the next layer, the token can attend (via its query vector) to each other token (matching based on their key vectors). Using the full set of query-key compatibility scores, we can compute, for each token, a representation by building the appropriate weighted sum over the other tokens. Because each token is attending to each other token (unlike the case where decoder steps attend to encoder steps), such architectures are typically described as *self-attention* models :cite:'Lin.Feng.Santos.ea.2017,Vaswani.Shazeer.Parmar.ea.2017', and elsewhere described as *intra-attention* model :cite:'Cheng.Dong.Lapata.2016,Parikh.Tackstrom.Das.ea.2016,Paulus.Xiong.

In this section, we will discuss sequence encoding using self-attention, including using additional information for the sequence order.

Given a sequence of input tokens $\mathbf{x}_1, \ldots, \mathbf{x}_n$ where any $\mathbf{x}_i \in \mathbb{R}^d$ $(1 \leq i \leq n)$, its self-attention outputs a sequence of the same length $\mathbf{y}_1, \ldots, \mathbf{y}_n$, where

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d$$

according to the definition of attention pooling in :eqref:‘eq$_a ttention_p ooling‘.Usingmulti-headattention, thefollowingcodesnippetcomputestheself-attentionofatensorwithshape(batchsize, numberoftimestepsorsequencelengthintokens$,d).$Theoutputt$

3.9. Graphs

A Graph is the type of data structure that contains nodes and edges. A node can be a person, place, or thing, and the edges define the relationship between nodes. The edges can be directed and undirected based on directional dependencies.

Graphs are excellent in dealing with complex problems with relationships and interactions. They are used in pattern recognition, social networks analysis, recommendation systems, and semantic analysis. Creating graph-based solutions is a whole new field that offers rich insights into complex and interlinked datasets.

Nevertheless, Graph-based data structures have drawbacks, and researchers must understand them before developing graph-based solutions.

- A graph exists in non-euclidean space. It does not exist in 2D or 3D space, which makes it harder to interpret the data. To visualize the structure in 2D space, you must use various dimensionality reduction tools.
- Graphs are dynamic; they do not have a fixed form. There can be two visually different graphs, but they might have similar adjacency matrix representations. It makes it difficult for us to analyze data using traditional statistical tools.
- Large size and dimensionality will increase the graph's complexity for human interpretations. The dense structure with multiple nodes and thousands of edges is harder to understand and extract insights.

3.9.1. Graph Neural Networks

Graph Neural Networks are special types of neural networks capable of working with a graph data structure. They are highly influenced by Convolutional Neural Networks (CNNs) and graph embedding. GNNs are used in predicting nodes, edges, and graph-based tasks.

- A graph exists in non-euclidean space. It does not exist in 2D or 3D space, which makes it harder to interpret the data. To visualize the structure in 2D space, you must use various dimensionality reduction tools.

- Graphs are dynamic; they do not have a fixed form. There can be two visually different graphs, but they might have similar adjacency matrix representations. It makes it difficult for us to analyze data using traditional statistical tools.

- Large size and dimensionality will increase the graph's complexity for human interpretations. The dense structure with multiple nodes and thousands of edges is harder to understand and extract insights.

3.9.2. Graph Neural Networks

3.10. Training

Training is the process of iterating through a dataset to make the network learn the optimal mapping (combination of weights) from input to desired output in the data samples. In each iteration, a forward pass through the network is performed, computing the output of each layer until the end. This produces the output response of the network, which is then compared to a desired output with a defined Loss function (L). This function estimates the amount of error, which is back-propagated through the network to update its weights with the aim of minimizing the error.

Regarding the supervised training paradigm, Backpropagation is an algorithm used for training artificial neural networks (ANNs) used to update the weights and biases of the neurons in the network based on the error between the predicted output and the actual output.

The backpropagation algorithm consists of two phases: the forward pass and the backward pass. During the forward pass, the input is fed into the network and propagated through the layers to obtain the predicted output. During the backward pass, the error between the predicted output and the actual output is computed and used to update the weights and biases of the neurons in the network.

The backpropagation algorithm is based on the chain rule of calculus, which allows the gradient of the loss function with respect to each weight and bias to be computed recursively from the output layer to the input layer of the network. The gradient descent algorithm is then used to update the weights and biases in the direction of the negative gradient of the loss function. In that sense, the main losses used in this work are described throughout the remaining content of this Chapter.

3.10.1. Optimizer and learning rate

The ADAM optimizer is a popular optimization algorithm used in deep learning, particularly for training neural networks. It is an adaptive learning rate optimization algorithm that is well suited for large datasets and high-dimensional parameter spaces.

The basic idea behind ADAM is to compute adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. The algorithm computes a moving average of the gradients and their squared values, and uses these estimates to update the parameters with a learning rate that adapts to the local curvature of the loss function. The algorithm also includes bias-correction terms to ensure that the estimates are unbiased, especially in the early stages of training when the estimates are highly uncertain.

The update rule for ADAM can be expressed mathematically as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \; v_t \quad = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \; \hat{m}_t = \frac{m_t}{1-\beta_1^t} \; \hat{v}t \quad = \frac{v_t}{1-\beta_2^t} \; \theta t + 1 = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}$$

Note that the ampersand symbol ’’ is used to align the equations at the equal sign, and the double backslash ’ís used to separate the lines of the equation.

where θ_t is the parameter vector at time step t, g_t is the gradient vector, m_t and v_t are the first and second moment estimates at time step t, $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected moment estimates, α is the learning rate, β_1 and β_2 are the decay rates for the moment estimates, and ϵ is a small constant to prevent division by zero.

The algorithm starts with initializing m_0 and v_0 as zero vectors, and θ_0 as the initial parameter vector. At each iteration, the gradient vector g_t is computed using a batch of training data, and the moment estimates m_t and v_t are updated according to the first two lines of the update rule. The bias-corrected moment estimates $\hat{m}_t$ and $\hat{v}_t$ are then computed using the next two lines of the update rule. Finally, the parameter vector θ_t is updated using the last line of the update rule.

The hyperparameters α, β_1, β_2, and ϵ can be tuned to optimize performance on a particular dataset and neural network architecture.

On the other hand, The learning rate is a key hyperparameter in the optimization process of machine learning algorithms, including optimizer updates like the ADAM optimizer. The learning rate controls the step size taken in the direction of the negative gradient during each iteration of the optimization process.

If the learning rate is too small, the optimization process will be slow and may get stuck in local minima. On the other hand, if the learning rate is too large, the optimization process may overshoot the minimum and oscillate back and forth, or even diverge.

In the context of the ADAM optimizer, the learning rate is used to adjust the size of the update step taken in the direction of the estimated gradient. The update step is multiplied by the learning rate, which determines the size of the step. A larger learning rate will result in larger update steps, and a smaller learning rate will result in smaller update steps.

In practice, the learning rate is usually set through a process called hyperparameter tuning, where different values of the learning rate are tried on a validation set to find the optimal value that results in the best performance of the model on the test set.

One common technique to adjust the learning rate during training is called learning rate scheduling. This involves decreasing the learning rate over time, often according to a predetermined schedule or based on the performance of the model on a validation set. This technique can help improve the convergence and stability of the optimization process, particularly in the later stages of training when the model is close to the optimal solution.

Overall, the learning rate plays a crucial role in the optimization process of machine learning algorithms, including optimizer updates like the ADAM optimizer. Selecting an appropriate learning rate is important for achieving fast and stable convergence to the optimal solution.

3.10.2. Regression losses

A regression loss is a type of loss function used in regression problems to measure the difference between the predicted and true values of a continuous variable. In other words, it is a way to quantify how well a machine learning model is able to predict numerical values based on input data.

In regression problems, the goal is to learn a function that maps input features to output values. A regression loss is used to train the model by penalizing the difference between the predicted and true output values. The loss function is typically minimized during training, so that the model learns to make more accurate predictions. There are several types of regression loss functions, such as the Quantile loss, Log-Cosh or Mean Absolute Error (MAE). In this work we mainly focus on the Mean Square Error (MSE) and SmoothL1, also known as Huber loss.

3.10.2.1. Mean Square Error loss

The Mean Squared Error (MSE) loss is a commonly used loss function in regression problems. It measures the average squared difference between the predicted and true values of a continuous variable. The MSE loss is given by:

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.21}$$

where y_i is the true value of the i-th data point, $\hat{y}_i$ is the predicted value, and n is the number of data points.

The MSE loss penalizes large errors more strongly than small errors, since it uses the square of the difference between the predicted and true values. This makes it sensitive to outliers and can lead to overfitting if the data contains extreme values.

The MSE loss is used in many regression problems, such as linear regression, polynomial regression, and neural networks. It is often used as a performance metric to evaluate the quality of the model's predictions.

3.10.2.2. SmoothL1 loss

SmoothL1 loss, also known as Huber loss, is a loss function used in regression problems to measure the difference between the predicted and true values of a continuous variable. It is a variant of the L1 loss that is less sensitive to outliers and has a smooth gradient near zero.

The SmoothL1 loss function can be defined as follows:

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{3.22}$$

where y is the true value, $\hat{y}$ is the predicted value, and $|\cdot|$ represents the absolute value function.

The SmoothL1 loss function behaves like the L1 loss for small errors and like the L2 loss for large errors. Specifically, for errors smaller than 1, it uses the squared difference between the predicted and true values, which has a smooth gradient. For larger errors, it uses the absolute difference, which is less sensitive to outliers than the squared difference.

The SmoothL1 loss is used in regression problems when the data contains outliers or when the model needs to be less sensitive to large errors. It is commonly used in object detection and localization tasks, where the predicted bounding boxes can be highly sensitive to small changes in the input data.

3.10.3. Softmax loss

The softmax loss, also known as the cross-entropy loss, is a commonly used loss function in classification problems. It measures the difference between the predicted probability distribution and the true probability distribution of a categorical variable. The softmax loss is given by:

$$CE(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{i,j}\log(\hat{y_{i,j}}) \tag{3.23}$$

where $y_{i,j}$ is the true probability of the i-th data point belonging to class j, $\hat{y_{i,j}}$ is the predicted probability, n is the number of data points, and k is the number of classes.

The softmax function is applied to the output of the model to obtain a probability distribution over the classes. The predicted probability of class j is given by:

$$\hat{y_{i,j}} = \frac{e^{z_{i,j}}}{\sum_{l=1}^{k} e^{z_{i,l}}} \tag{3.24}$$

where $z_{i,j}$ is the unnormalized score or logit for class j of the i-th data point.

The softmax loss penalizes the model more heavily for predictions that are far from the true probabilities. It encourages the model to assign higher probabilities to the correct classes and lower probabilities to the incorrect classes.

The softmax loss is used in many classification problems, such as image classification, natural language processing, and speech recognition. It is often used as a performance metric to evaluate the quality of the model's predictions.

3.10.4. Negative Log-Likelihood loss

Negative Log Likelihood (NLL) loss is a loss function commonly used in classification problems, particularly in deep learning models that output probabilities for each class. It is a type of maximum likelihood estimation (MLE) loss, which means it attempts to maximize the likelihood of the predicted probabilities given the true labels.

The NLL loss function can be defined as follows:

$$L(y_i, \hat{y}_i) = -\log(\hat{y_{i,y_i}}) \tag{3.25}$$

where y_i is the true label of the i-th data point, $\hat{y}_i$ is the predicted probability distribution for that point, and $\hat{y_{i,y_i}}$ is the predicted probability for the true label.

The NLL loss penalizes the model for assigning low probabilities to the true label, while rewarding it for assigning high probabilities. It is a logarithmic loss function, which means that the penalty for low probabilities increases exponentially as the predicted probability approaches zero.

The NLL loss is used in classification problems because it provides a gradient that can be used to update the model parameters during training, in order to improve the accuracy of the predicted probabilities. It is commonly used in conjunction with softmax activation function, which ensures that the predicted probabilities sum to one across all classes.

3.10.5. Winner-Takes-All loss

Winner-Takes-All (WTA) loss is a loss function used in clustering problems, particularly in competitive learning models. It is a type of unsupervised learning, which means that it does not require labeled data for training.

The WTA loss function can be defined as follows:

$$L(y_i, f(x_i)) = \begin{cases} 0 & \text{if } y_i = \arg\max_j f_j(x_i)\ 1 \\ \text{otherwise} \end{cases} \tag{3.26}$$

where y_i is the true cluster of the i-th data point, $f_j(x_i)$ is the activation of the j-th neuron in the output layer for that point, and $\arg\max_j f_j(x_i)$ is the index of the neuron with the highest activation.

The WTA loss function penalizes the model for assigning a data point to the wrong cluster. It works by forcing each neuron in the output layer to specialize in a particular cluster, such that the neuron with the highest activation for a given point corresponds to the true cluster of that point.

The WTA loss is used in clustering problems because it encourages the model to learn a set of representative clusters that capture the structure of the data, without requiring any prior knowledge of the true labels. It is commonly used in conjunction with competitive learning algorithms, such as self-organizing maps (SOMs), that use local competition between neurons to learn a topology-preserving mapping from the input space to the output space.

3.10.6. Hinge loss

Hinge loss is a loss function used in classification problems, particularly in support vector machines (SVMs). It is a type of max-margin loss, which means it attempts to maximize the margin between the decision boundary and the data points.

The hinge loss function can be defined as follows:

$$L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i)) \tag{3.27}$$

where y_i is the true label of the i-th data point, $f(x_i)$ is the predicted score for that point, and 1 is a margin hyperparameter that determines the width of the margin.

If $y_i f(x_i) \geq 1$, then the point is correctly classified and the loss is zero. If $y_i f(x_i) < 1$, then the point is misclassified and the loss is proportional to the distance between the predicted score and the correct score. The loss function penalizes misclassifications linearly, with a slope of -1 for negative misclassifications and 0 for positive misclassifications.

The hinge loss is used in SVMs because it encourages the model to find a decision boundary that maximizes the margin between the classes, while still correctly classifying the data points. This results in a more robust and generalizable model.

3.10.7. Regularization techniques

Regularization techniques in deep learning are used to prevent overfitting and improve the generalization performance of a model. Overfitting occurs when a model fits the training data too closely and captures noise or irrelevant patterns, resulting in poor performance on new, unseen data. Here are some commonly used regularization techniques in deep learning:

L1 and L2 regularization: L1 and L2 regularization are two popular regularization techniques that add a penalty term to the loss function during training. L1 regularization adds the sum of the absolute values of the weights to the loss function, while L2 regularization adds the sum of the squares of the weights. This encourages the model to learn simpler and more generalizable representations by shrinking the weights towards zero.

Dropout: Dropout is a regularization technique that randomly drops out some units (neurons) in a layer during training. This forces the remaining units to learn more robust and diverse representations that generalize better to new data. Dropout has been shown to be effective in reducing overfitting, particularly in deep neural networks with many layers.

Data augmentation: Data augmentation is a technique that artificially increases the size of the training set by generating new examples from existing ones. This can be done by applying transformations such as rotations, translations, flips, or adding noise to the input data. Data augmentation can help reduce overfitting by increasing the diversity of the training data and improving the generalization performance of the model.

Early stopping: Early stopping is a technique that monitors the performance of the model on a validation set during training and stops the training process when the performance starts to degrade. This prevents the model from overfitting to the training data and allows it to generalize better to new data.

Batch normalization: Batch normalization is a technique that normalizes the activations of a layer by subtracting the batch mean and dividing by the batch standard deviation. This helps stabilize the distribution of the activations and reduces the internal covariate shift, which can improve the training process and reduce overfitting.

Weight decay: Weight decay is a technique that adds a penalty term to the loss function during training, similar to L2 regularization. The penalty term is proportional to the square of the weights, and it encourages the model to learn smaller and simpler weights, which can reduce overfitting.

These are some of the commonly used regularization techniques in deep learning. They can be combined and tuned to improve the performance of a model on a specific task and dataset.

Moreover, another interesting technique to help the model improve during training is hard-mining. Hard-mining is a technique in deep learning used to improve the training of a model by focusing on the samples that are most difficult to classify correctly.

During training, the model makes predictions on a batch of training data, and the loss function is computed based on the difference between the predicted values and the true values. In hard-mining, the training samples that contribute the most to the loss function are identified, and these samples are given more importance during training.

The idea behind hard-mining is that by focusing on the difficult samples, the model is forced to learn more discriminative features that can better distinguish between the different classes. This can lead to better generalization performance and improved accuracy on the test data.

There are several ways to implement hard-mining in deep learning, including:

Hard negative mining: This involves selecting the training samples that are misclassified with the highest confidence by the model and including them in the next batch of training data. By focusing on the difficult negative samples, the model can learn to better distinguish between the different classes and reduce the number of false negatives.

Hard positive mining: This involves selecting the training samples that are misclassified with the lowest confidence by the model and including them in the next batch of training data. By focusing on the difficult positive samples, the model can learn to better distinguish between the different classes and reduce the number of false positives.

Curriculum learning: This involves gradually increasing the difficulty of the training data over time. The model is first trained on easy samples and then gradually exposed to more difficult samples. This can help the model learn more robust and generalizable features by gradually increasing the complexity of the training data.

Overall, hard-mining is a useful technique in deep learning for improving the training of a model by focusing on the difficult samples. By incorporating hard-mining into the training process, the model can learn more discriminative features and improve its generalization performance on new, unseen data.

Chapter 4

Predictive Techniques for Scene Understanding

Avanzad, sin temor a la oscuridad.
Luchad jinetes de Theoden.
Caerán las lanzas, se quebrarán los escudos.
Aún restará la espada.
Rojo será el día, hasta el nacer del sol.
Cabalgad, cabalgad, cabalgad hacia la desolación
y el fin del mundo. Muerte, muerte, muerte.

Discurso de Theoden, Rey de Rohan
El Señor de los Anillos: El Retorno del Rey

4.1. Introduction

Autonomous driving has gained significant attention in recent years, with the development of intelligent vehicles and the increasing need for safe and efficient transportation systems. However, one of the main challenges in the field of autonomous driving is scene understanding, which involves the ability of a vehicle to recognize and interpret the objects and events in its environment. Accurate scene understanding is critical for safe and efficient driving, as it enables the vehicle to make informed decisions and take appropriate actions.

Deep learning has emerged as a powerful tool for scene understanding in autonomous driving, as it can automatically learn representations of complex and high-dimensional data. In particular, deep neural networks have shown remarkable performance in various tasks related to scene understanding, such as object detection, segmentation, and classification. However, the design and implementation of effective deep learning models for scene understanding in autonomous driving is still an active research area, and there is a need for more advanced predictive techniques to improve the performance of autonomous vehicles.

The main focus of this PhD thesis is to explore predictive techniques for scene understanding in autonomous driving using deep learning. The goal is to develop advanced models that can accurately recognize and interpret the objects and events in a vehicle's environment, and provide reliable predictions for safe and efficient driving. The thesis will investigate different deep learning architectures and techniques, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention mechanisms, and evaluate their performance on various datasets and benchmarks. In addition, the thesis will investigate the use of transfer learning and domain adaptation to improve the generalization and robustness of deep learning models for scene understanding in diverse driving conditions and environments. The ultimate goal of this research is to contribute to the development of intelligent and safe autonomous driving systems that can operate effectively in real-world scenarios.

4.2. SmartMOT

4.2.1. Monitored Area Attention Module

AV need to locate itself in the environment to know what is happening close to it, in order to make decisions and execute a correct navigation like a human driver would. When we talk about localization, the first thing we need is a map where to be located and, particularly for vehicles, this is road map. Road maps used by current navigators, e.g. Google Maps, are not valid for AV because of data type and the fact that they are not accurate enough to be used for autonomous driving. Here is where HD maps appear to solve this problem.

The concept of HD map is a text file describing the real-world features related to the road map and its location within a 2D/3D space, and can do things that other sensors cannot [92]: First, they have an "infinite range" and, therefore, can "see" even into occluded areas. Second, maps will never fail due to environmental conditions. Lastly, maps contain highly refined data. This information can be used by different modules of an AV, (including self-localization, vehicle control, path planning, perception and system management) drastically reducing the computational load and complexity in comparison to other more complex methods, providing robustness and reliability to the system.

There is not an standard definition about which data an HD map may include, but in the case of road maps, we could be talking about: roads, lanes, regulatory elements, topological information or any other information that can be considered relevant to drive in a safety way. HD map can be used to alleviate the computational load of other modules and add confidence to the real-world model. In our case, it is used for two different purposes, as illustrated in Figure 4.1:

Figure 4.1: Path Planning and Map Monitoring Outline

- Global Path Planning, which uses a specific path planner which inputs are the HD map information and the ego-vehicle current location to retrieve an optimal (usually optimized based on the travelled distance) global route towards an specific goal.
- Map monitoring, responsible for monitoring the most relevant static and dynamic map elements around the ego-vehicle at each timestep, such as standard lanes (current, back), intersection lanes (merge, split and cross) and regulatory elements (e.g. give way, stop, pedestrian crossing, traffic light).

4.3. GAN based Vehicle Motion Prediction

4.4. Exploring Map Features

We use the standard Negative Log-Likelihood (**NLL**) loss to train our social and map baselines in order to compare the ground-truth points $Y \in \mathbb{R}^{pred_{len} \times data_{dim}} = \{(x_0, y_0)...(x_{pred_{len}}, y_{pred_{len}})\}$ with our multimodal predictions ($\hat{Y} \in \mathbb{R}^{k \times pred_{len} \times data_{dim}}$), given k modalities (hypotheses) $\mathbf{p} = \{(\hat{x}_0^1, \hat{y}_0^1)...(\hat{x}_{pred_{len}}^k, \hat{y}_{pred_{len}}^k)\}$, with their corresponding confidences $\mathbf{c} = \{c_1...c_k\}$ using the following equation:

$$\text{NLL} = -\log \sum_k e^{\log c^k - \frac{1}{2}\sum_{t=0}^{pred_{len}} (\hat{x}_t^k - x_t)^2 + (\hat{y}_t^k - y_t)^2} \tag{4.1}$$

4.5. Leveraging traffic context via GNN

$$\mathrm{L}_{class,Hinge} = \frac{1}{(K-1)} \sum_{m=1\backslash m \neq m^*}^{K} \max(0, c_k + \epsilon - c_{m^*}) \quad (4.2)$$

4.6. Improving efficiency of Vehicle Motion Prediction

Chapter 5

Applications in Autonomous Driving

La fuerza de tus convicciones determina tu éxito,
no el número de tus seguidores.

Reamus Lupin
Harry Potter y Las Reliquias de la Muerte, Parte 2

5.1. Motion Prediction Datasets

5.2. Multi-Object Tracking

5.3. Decision-Making

5.4. Holistic Simulation

Table 5.1: Comparison between the Argoverse 2 Motion Forecasting dataset and other recent motion forecasting datasets. Hyphens "-" indicate that attributes are either not applicable, or not available. We define "mined for interestingness" to be true if interesting scenarios/actors are mined *after data collection*, instead of taking all/random samples. † Public leaderboard counts as retrieved on Aug. 27, 2021.

	Argoverse 1 [4]	Interaction [93]	Lyft [94]	Waymo [95]	NuScenes [96]	Yandex [97]	Argoverse 2 [97]
# Scenarios	324k	-	170k	104k	41k	600k	250k
# Unique Tracks	11.7M	40k	53.4M	7.6M	-	17.4M	13.9M
Average Track Length	2.48 s	19.8 s	1.8 s	7.04 s	-	-	5.16 s
Total Time	320 h	16.5 h	1118 h	574 h	5.5 h	1667 h	763 h
Scenario Duration	5 s	-	25 s	9.1 s	8 s	10 s	11 s
Test Forecast Horizon	3 s	3 s	5 s	8 s	6 s	5 s	6 s
Sampling Rate	10 Hz	10 Hz	10 Hz	10 Hz	2 Hz	5 Hz	10 Hz
# Cities	2	6	1	6	2	6	6
Unique Roadways	290 km	2 km	10 km	1750 km	-	-	2220 km
Avg. # tracks per scenario	50	-	79	-	75	29	73
# Evaluated object categories	1	1	3	3	1	2	5
Multi-agent evaluation	×	✓	✓	✓	×	✓	✓
Mined for Interestingness	✓	×	-	✓	×	×	✓
Vector Map	✓	×	×	✓	✓	×	✓
Download Size	4.8 GB	-	22 GB	1.4 TB	48 GB	120 GB	58 GB
# Public Leaderboard Entries†	194	-	935	23	18	3	-

Chapter 6

Conclusions and Future Works

El mundo no es todo alegría y color, es un lugar terrible y por muy duro que seas es capaz de arrodillarte a golpes y tenerte sometido a golpes permanente si no se lo impides; Ni tú ni yo ni nadie golpea mas fuerte que la vida. Pero no importa lo fuerte que golpeas, sino lo fuerte que pueden golpearte y los aguantas mientras avanzas, hay que soportar sin dejar de avanzar.
¡Así es como se gana!
Si tú sabes lo que vales, vé y consigue lo que mereces pero tendrás que soportar los golpes y no puedes estar diciendo que no estás donde querías llegar por culpa de él o de ella, eso lo hacen los cobardes y tú no lo eres.
TÚ ERES CAPAZ DE TODO.

Discurso de Rocky a su hijo
Rocky Balboa

6.1. Conclusions

In this thesis, a series of interaction-aware trajectory prediction methods, including single-agent trajectory prediction, multi-agent trajectory prediction, and multimodal trajectory prediction, are developed for autonomous driving. Besides, the impacts of trajectory prediction on trajectory planning are also investigated. In Chapter 3, a novel framework with consideration of vehicle-infrastructure heterogeneous interactions is proposed for trajectory prediction of a single target vehicle. In the proposed scheme, a heterogeneous graph is developed to represent the interactions, where the nodes contain features extracted from corresponding encoders. Besides, a novel heterogeneous graph social pooling (HGS) module is designed to extract high-level interaction features. The framework can be easily expanded for highway driving scenarios. Experimental results obtained using real-world driving datasets show that the proposed HGS method outperforms existing interaction-aware methods in terms of prediction accuracy. Besides, ablative studies demonstrate that the consideration of vehicle-infrastructure heterogeneous interactions

effectively improves the prediction accuracy compared to those methods only considering inter-vehicle interactions. Then, the above prediction method for single-agent is generalized and expanded for heterogeneous multi-agent trajectory prediction in Chapter 4. To do this, a novel three-channel framework is designed to jointly consider traffc participants' dynamics, interaction, and map features. The driving scene is represented in a hybrid way, where the inter-agent interaction in the traffc system is represented with an edge-featured heterogeneous graph, and the shared local map is represented with a Bird's Eye View (BEV) image. Two shared Recurrent Neural Networks (RNNs) are adopted to capture vehicles' and pedestrians' dynamics features from their historical states, respectively. A novel heterogeneous edge-enhanced graph attention network (HEAT) is proposed to model the inter-agent interactions, and a map-sharing technique based on the gate mechanism is also leveraged to share the local map across all target agents. Experimental validations on real-world driving datasets of both urban and highway scenarios show that the proposed method not only achieves state-of-the-art performance but also can provide simultaneous predictions of multi-agent trajectories for a variable number of heterogeneous agents. Besides the unimodal predictions for single and multiple agents, this thesis also tackles the inherent multimodality problem of driving behaviors for prediction in Chapter 5. A novel map-adaptive multimodal trajectory prediction framework is proposed. Within this framework, through a single graph operation, a variable number of map-compliant trajectories and a non-map-compliant trajectory can be generated. Map-compliant predictions are conditioned on either a single candidate centerline (CCL) or a bunch of all CCLs, making the predictor adaptive to different road structures. The non-map-compliant prediction captures the irrational driving behavior for safety concerns. The driving scene is represented with a heterogeneous hierarchical graph containing both agents and their CCLs. A hierarchical graph operator (HGO) with an edge-masking technology is proposed to encode the driving scene. Validation on the Argoverse motion forecasting benchmark shows that the proposed method achieves state-of-the-art performance with the advantage of mapadaptive capacity. Beyond pure prediction, in Chapter 6, predictive planning and the impacts of prediction on downstream trajectory planning are also investigated. An interactionaware predictive planner, which is trained to imitate human driving behaviors, is designed to investigate the problem of how prediction would affect the performance of motion planning. The predictive planner is obtained by training an oracle planner, which is aware of target agents' ground truth future trajectories, and replacing the ground truth with the predicted trajectories for inference during implementation. Experimental results on a real-world dataset show that the proposed predictive planner achieves better performance over other baselines in terms of displacement error, miss rate, and collision rate. The gap between the predictive planner and the oracle planner shows that it is promising to further enhance the planning performance by improving the prediction accuracy. To be implemented in real-world self-driving systems, the proposed methods require upstream localization, perception, and tracking results, since the historical states of other

traffc participants are needed as the input of the proposed methods. Perception can be realized using either or both of camera and LiDAR. Other sensors, such as radar, can also be used for better perception via sensor fusion. For the single-agent and multi-agent prediction methods in Chapter 3 and Chapter 4, we are using BEV maps, where only the map is needed. We do not need a BEV image to show the real-time tra!c. The map can be obtained from main-stream map providers and converted into images for the usage of our method. The mapadaptive multimodal method in Chapter 5, however, requires a high-definition map (HD map) of the local area since we need the candidate centerlines of vehicles of interest. The methods can run on both CPUs or GPUs, and using GPUs is suggested for faster inference.

6.2. Future Works

Although many studies have been done in trajectory prediction and path planning in the past years, there are still many aspects that need to be further investigated in the future. Scene representation and encoding. Researchers have proposed many methods to encode driving scenes with different representations. However, there is no unified representation of various driving scenes so far. The lack of a universal representation limits the generalizability of prediction methods with large-scale deployment in autonomous vehicles in the real world because a method can hardly be applied to a situation that cannot be described. Among many representation approaches proposed so far, graph-based representations are promising because a graph can accommodate an arbitrary number of heterogeneous objects and represent their interdependencies via directed edges. For example, when modeling a driving scene in the context of traffc systems, a node can represent a vehicle, a pedestrian, a lanelet, a junction, a traffc signal, etc. A new object can always be added to the existing graph. There are three important steps that need to be done to further improve the graph-based scene representation and encoding in the future. The first aspect is to construct the graph with proper connections, that is, to determine the edge set of the graph. This step needs to identify interdependencies between pairs of nodes and connect nodes with directed edges for information flow in the graph. Once the graph structure is settled, the second step is to assign the node, and edge features properly. This requires researchers to select or design proper encoders for different kinds of nodes and edges. Then the third step is required to design graph operators to handle the heterogeneity in the scene graphs. In this step, the advances in heterogeneous graph neural networks can be leveraged. Trajectory decoding. For future work, an immediate step is to generalize the map-adaptive multimodal prediction method proposed in Chapter 5 with uncertainty estimations. The uncertainty includes both motion and mode uncertainties. The motion uncertainty captures the distribution of agents' position over a planar map at each time step, and the mode uncertainty captures the possibility of driving modalities. The former can be modeled via bivariate Gaussian distributions, and the latter can

be treated as a multi-class classification problem over a variable number of modalities. Then the next step can focus on generalizing uncertainty-aware multimodal predictions to multi-agent settings by modeling the joint distribution of multiple agents' behaviors. Social consistency should be considered in this step such that there is no conflict between any pair of trajectories in a joint modality in normal cases. Besides, trajectory predictors should be designed from the ego vehicle's point of view. One possible way is to design a decoder that can output trajectories upon the ego's request. For example, the predictor can focus on a small set of target agents requested by the ego vehicle rather than all the agents in sight. For a specific target agent, the predictor can focus on predicting its driving options that may affect the ego's planned trajectories. This attentive approach can reduce computation efforts for real-time implementations. Predictive planning. The ultimate goal of trajectory prediction is to further improve the performance of decision-making and motion control of autonomous vehicles with respect to safety, smartness, and effciency. So prediction must be integrated into the planning module, and therefore predictive planning is worthwhile exploring. There are many problems that should be addressed for the development of predictive planners. First, predictive planners should be able to address prediction uncertainty since prediction can never be exactly the same as the ground truth. Second, the relationship between prediction and planning needs to be further studied in order to answer the following questions: 1) How would the improvement in prediction affect the downstream planning performance? 2) Is there a floor of prediction error below which improving prediction accuracy leads to no improvement or even a negative effect on planning? 3) Can we design a predictive planner that is scalable to predictors of different uncertainties as long as these uncertainties are known? Third, learning-based motion planners should be further investigated since they have great potential to be incorporated with data-driven predictions. However, the current limitations in explainability and reliability need to be addressed. In general, plenty of effort is needed in these research areas.

Bibliography

[1] J. F. Arango, L. M. Bergasa, P. A. Revenga, *et al.*, "Drive-by-wire development process based on ros for an autonomous electric vehicle", *Sensors*, vol. 20, no. 21, p. 6121, 2020.

[2] A. Ranga, F. Giruzzi, J. Bhanushali, *et al.*, "Vrunet: Multi-task learning model for intent prediction of vulnerable road users", *arXiv preprint arXiv:2007.05397*, 2020.

[3] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles", in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8806–8813.

[4] M.-F. Chang, J. Lambert, P. Sangkloy, *et al.*, "Argoverse: 3d tracking and forecasting with rich maps", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8748–8757.

[5] B. Wilson, W. Qi, T. Agarwal, *et al.*, "Argoverse 2: Next generation datasets for self-driving perception and forecasting", *arXiv preprint arXiv:2301.00493*, 2023.

[6] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data", in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, Springer, 2020, pp. 683–700.

[7] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Home: Heatmap output for future motion estimation", in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 500–507.

[8] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Gohome: Graph-oriented heatmap output for future motion estimation", in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 9107–9114.

[9] B. Varadarajan, A. Hefny, A. Srivastava, *et al.*, "Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction", in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 7814–7821.

[10] M. Wang, X. Zhu, C. Yu, *et al.*, "Ganet: Goal area network for motion forecasting", *arXiv preprint arXiv:2209.09723*, 2022.

[11] J. Schmidt, J. Jordan, F. Gritschneder, and K. Dietmayer, "Crat-pred: Vehicle trajectory prediction with crystal graph convolutional neural networks and multi-head self-attention", in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 7799–7805.

[12] M. Liang, B. Yang, R. Hu, *et al.*, "Learning lane graph representations for motion forecasting", in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, Springer, 2020, pp. 541–556.

[13] Z. Huang, H. Liu, J. Wu, and C. Lv, "Conditional predictive behavior planning with inverse reinforcement learning for human-like autonomous driving", *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[14] C. Tang and R. R. Salakhutdinov, "Multiple futures prediction", *Advances in neural information processing systems*, vol. 32, 2019.

[15] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "Precog: Prediction conditioned on goals in visual multi-agent settings", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2821–2830.

[16] S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan, "What-if motion prediction for autonomous driving", *arXiv preprint arXiv:2008.10587*, 2020.

[17] C. Tang, W. Zhan, and M. Tomizuka, "Interventional behavior prediction: Avoiding overly confident anticipation in interactive prediction", in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 11 409–11 415.

[18] H. Zhao, J. Gao, T. Lan, *et al.*, "Tnt: Target-driven trajectory prediction", in *Conference on Robot Learning*, PMLR, 2021, pp. 895–904.

[19] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen, "A survey on trajectory-prediction methods for autonomous driving", *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 652–674, 2022.

[20] N. Kaempchen, B. Schiele, and K. Dietmayer, "Situation assessment of an autonomous emergency brake for arbitrary vehicle-to-vehicle collision scenarios", *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 4, pp. 678–687, 2009.

[21] R. Pepy, A. Lambert, and H. Mounier, "Reducing navigation errors by planning with realistic vehicle model", in *2006 IEEE Intelligent Vehicles Symposium*, IEEE, 2006, pp. 300–307.

[22] R. Miller and Q. Huang, "An adaptive peer-to-peer collision warning system", in *Vehicular Technology Conference. IEEE 55th Vehicular Technology Conference. VTC Spring 2002 (Cat. No. 02CH37367)*, IEEE, vol. 1, 2002, pp. 317–321.

[23] J. Hillenbrand, A. M. Spieker, and K. Kroschel, "A multilevel collision mitigation approach—its situation assessment, decision making, and performance tradeoffs", *IEEE Transactions on intelligent transportation systems*, vol. 7, no. 4, pp. 528–540, 2006.

[24] R. E. Kalman, "A new approach to linear filtering and prediction problems", 1960.

[25] N. Kaempchen, K. Weiss, M. Schaefer, and K. C. Dietmayer, "Imm object tracking for high dynamic driving maneuvers", in *IEEE Intelligent Vehicles Symposium, 2004*, IEEE, 2004, pp. 825–830.

[26] B. Jin, B. Jiu, T. Su, H. Liu, and G. Liu, "Switched kalman filter-interacting multiple model algorithm based on optimal autoregressive model for manoeuvring target tracking", *IET Radar, Sonar & Navigation*, vol. 9, no. 2, pp. 199–209, 2015.

[27] V. Lefkopoulos, M. Menner, A. Domahidi, and M. N. Zeilinger, "Interaction-aware motion prediction for autonomous driving: A multiple model kalman filtering scheme", *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 80–87, 2020.

[28] A. Broadhurst, S. Baker, and T. Kanade, "Monte carlo road safety reasoning", in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, IEEE, 2005, pp. 319–324.

[29] K. Okamoto, K. Berntorp, and S. Di Cairano, "Driver intention-based vehicle threat assessment using random forests and particle filtering", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 860–13 865, 2017.

[30] Y. Wang, Z. Liu, Z. Zuo, Z. Li, L. Wang, and X. Luo, "Trajectory planning and safety assessment of autonomous vehicles based on motion prediction and model predictive control", *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8546–8556, 2019.

[31] Q. Deng and D. Söffker, "Improved driving behaviors prediction based on fuzzy logic-hidden markov model (fl-hmm)", in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 2003–2008.

[32] G. He, X. Li, Y. Lv, B. Gao, and H. Chen, "Probabilistic intention prediction and trajectory generation based on dynamic bayesian networks", in *2019 Chinese Automation Congress (CAC)*, IEEE, 2019, pp. 2646–2651.

[33] C. E. Rasmussen, "Gaussian processes in machine learning", in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, Springer, 2004, pp. 63–71.

[34] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A bayesian nonparametric approach to modeling motion patterns", *Autonomous Robots*, vol. 31, pp. 383–400, 2011.

[35] Q. Tran and J. Firl, "Online maneuver recognition and multimodal trajectory prediction for intersection assistance using non-parametric regression", in *2014 ieee intelligent vehicles symposium proceedings*, IEEE, 2014, pp. 918–923.

[36] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds", in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 797–803.

[37] Y. Guo, V. V. Kalidindi, M. Arief, *et al.*, "Modeling multi-vehicle interaction scenarios using gaussian random field", in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 3974–3980.

[38] H. M. Mandalia and M. D. D. Salvucci, "Using support vector machines for lane-change detection", in *Proceedings of the human factors and ergonomics society annual meeting*, SAGE Publications Sage CA: Los Angeles, CA, vol. 49, 2005, pp. 1965–1969.

[39] P. Kumar, M. Perrollaz, S. Lefevre, and C. Laugier, "Learning-based approach for online lane change intention prediction", in *2013 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2013, pp. 797–802.

[40] G. S. Aoude and J. P. How, "Using support vector machines and bayesian filtering for classifying agent intentions at road intersections", Tech. Rep., 2009.

[41] Y. Wang, C. Wang, W. Zhao, and C. Xu, "Decision-making and planning method for autonomous vehicles based on motivation and risk assessment", *IEEE Transactions on Vehicular Technology*, vol. 70, no. 1, pp. 107–120, 2021.

[42] N. Deo, A. Rangesh, and M. M. Trivedi, "How would surround vehicles move? a unified framework for maneuver classification and motion prediction", *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, pp. 129–140, 2018.

[43] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[44] M. Schreier, V. Willert, and J. Adamy, "An integrated approach to maneuver-based trajectory prediction and criticality assessment in arbitrary road environments", *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 10, pp. 2751–2766, 2016.

[45] M. Bahram, A. Lawitzky, J. Friedrichs, M. Aeberhard, and D. Wollherr, "A game-theoretic approach to replanning-aware interactive scene prediction and planning", *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3981–3992, 2015.

[46] J. Li, B. Dai, X. Li, X. Xu, and D. Liu, "A dynamic bayesian network for vehicle maneuver prediction in highway driving scenarios: Framework and verification", *Electronics*, vol. 8, no. 1, p. 40, 2019.

[47] G. Weidl, A. L. Madsen, D. Kasper, and G. Breuel, "Optimizing bayesian networks for recognition of driving maneuvers to meet the automotive requirements", in *2014 IEEE International Symposium on Intelligent Control (ISIC)*, IEEE, 2014, pp. 1626–1631.

[48] Y. Guan, S. E. Li, J. Duan, W. Wang, and B. Cheng, "Markov probabilistic decision making of self-driving cars in highway with random traffic flow: A simulation study", *Journal of Intelligent and Connected Vehicles*, vol. 1, no. 2, pp. 77–84, 2018.

[49] D. Silver, J. A. Bagnell, and A. Stentz, "Learning autonomous driving styles and maneuvers from expert demonstration", in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, Springer, 2013, pp. 371–386.

[50] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals", in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2011, pp. 1561–1566.

[51] M. Herman, V. Fischer, T. Gindele, and W. Burgard, "Inverse reinforcement learning of behavioral models for online-adapting navigation strategies", in *2015 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2015, pp. 3215–3222.

[52] S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers, "Learning to drive using inverse reinforcement learning and deep q-networks", *arXiv preprint arXiv:1612.03653*, 2016.

[53] L. Sun, W. Zhan, and M. Tomizuka, "Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning", in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2111–2117.

[54] Z. Huang, J. Wu, and C. Lv, "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning", *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 10 239–10 251, 2021.

[55] J. Ho and S. Ermon, "Generative adversarial imitation learning", *Advances in neural information processing systems*, vol. 29, 2016.

[56] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks", *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[57] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks", in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 204–211.

[58] R. Bhattacharyya, B. Wulfe, D. J. Phillips, *et al.*, "Modeling human driving behavior through generative adversarial imitation learning", *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[59] S. Choi, J. Kim, and H. Yeo, "Trajgail: Generating urban vehicle trajectories using generative adversarial imitation learning", *Transportation Research Part C: Emerging Technologies*, vol. 128, p. 103 091, 2021.

[60] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning", *arXiv preprint arXiv:1507.04888*, 2015.

[61] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning", *Robotics and Autonomous Systems*, vol. 114, pp. 1–18, 2019.

[62] T. Fernando, S. Denman, S. Sridharan, and C. Fookes, "Deep inverse reinforcement learning for behavior prediction in autonomous driving: Accurate forecasts of vehicle motion", *IEEE Signal Processing Magazine*, vol. 38, no. 1, pp. 87–96, 2020.

[63] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning", *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[64] Z. Zhu, N. Li, R. Sun, D. Xu, and H. Zhao, "Off-road autonomous vehicles traversability analysis and trajectory planning based on deep inverse reinforcement learning", in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 971–977.

[65] M. Wulfmeier, D. Z. Wang, and I. Posner, "Watch this: Scalable cost-function learning for path planning in urban environments", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 2089–2095.

[66] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2255–2264.

[67] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data", in *Conference on Robot Learning*, PMLR, 2018, pp. 947–956.

[68] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[69] J. Mercat, T. Gilles, N. El Zoghby, G. Sandou, D. Beauvois, and G. P. Gil, "Multi-head attention for multi-modal joint vehicle motion forecasting", in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 9638–9644.

[70] J. Gao, C. Sun, H. Zhao, *et al.*, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 525–11 533.

[71] J. Ngiam, V. Vasudevan, B. Caine, *et al.*, "Scene transformer: A unified architecture for predicting future trajectories of multiple agents", in *International Conference on Learning Representations*, 2022.

[72] S. Casas, A. Sadat, and R. Urtasun, "Mp3: A unified model to map, perceive, predict and plan", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 403–14 412.

[73] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom, and E. M. Wolff, "Covernet: Multimodal behavior prediction using trajectory sets", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 074–14 083.

[74] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 336–345.

[75] F. Marchetti, F. Becattini, L. Seidenari, and A. D. Bimbo, "Mantra: Memory augmented networks for multiple trajectory prediction", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7143–7152.

[76] Y. Biktairov, M. Stebelev, I. Rudenko, O. Shliazhko, and B. Yangel, "Prank: Motion prediction based on ranking", *Advances in neural information processing systems*, vol. 33, pp. 2553–2563, 2020.

[77] H. Cui, V. Radosavljevic, F.-C. Chou, *et al.*, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks", in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 2090–2096.

[78] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction", *arXiv preprint arXiv:1910.05449*, 2019.

[79] T. Buhet, E. Wirbel, A. Bursuc, and X. Perrotton, "Plop: Probabilistic polynomial objects trajectory prediction for autonomous driving", in *Conference on Robot Learning*, PMLR, 2021, pp. 329–338.

[80] N. Rhinehart, K. M. Kitani, and P. Vernaza, "R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting", in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 772–788.

[81] H. Cui, T. Nguyen, F.-C. Chou, *et al.*, "Deep kinematic models for kinematically feasible vehicle trajectory predictions", in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 10 563–10 569.

[82] Y. B. Can, A. Liniger, O. Unal, D. Paudel, and L. Van Gool, "Understanding bird's-eye view of road semantics using an onboard camera", *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3302–3309, 2022. DOI: 10.1109/LRA.2022.3146898.

[83] R. Mahjourian, J. Kim, Y. Chai, M. Tan, B. Sapp, and D. Anguelov, "Occupancy flow fields for motion forecasting in autonomous driving", *IEEE Robotics and Automation Letters*, 2022.

[84] B. Ivanovic, K.-H. Lee, P. Tokmakov, *et al.*, "Heterogeneous-agent trajectory forecasting incorporating class uncertainty", *arXiv preprint arXiv:2104.12446*, 2021.

[85] W. Zeng, M. Liang, R. Liao, and R. Urtasun, "Lanercnn: Distributed representations for graph-centric motion forecasting", in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 532–539.

[86] W. Zeng, W. Luo, S. Suo, *et al.*, "End-to-end interpretable neural motion planner", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8660–8669.

[87] B. Triggs, "Motion planning for nonholonomic vehicles: An introduction", 1993.

[88] H. W. Kuhn, "The hungarian method for the assignment problem", *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[89] G. Bishop, G. Welch, *et al.*, "An introduction to the kalman filter", *Proc of SIGGRAPH, Course*, vol. 8, no. 27599-23175, p. 41, 2001.

[90] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking", in *2008 11th international conference on information fusion*, IEEE, 2008, pp. 1–6.

[91] S. Sony, S. Gamage, A. Sadhu, and J. Samarabandu, "Multiclass damage identification in a full-scale bridge using optimally tuned one-dimensional convolutional neural network", *Journal of Computing in Civil Engineering*, vol. 36, no. 2, p. 04 021 035, 2022.

[92] K. Wong, Y. Gu, and S. Kamijo, "Mapping for autonomous driving: Opportunities and challenges", *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 1, pp. 91–106, 2020.

[93] W. Zhan, L. Sun, D. Wang, *et al.*, "Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps", *arXiv preprint arXiv:1910.03088*, 2019.

[94] H. John, Z. Guido, B. Luca, *et al.*, "One thousand and one hours: Self-driving motion prediction dataset", *arXiv preprint arXiv: 2006.14480*, 2020.

[95] S. Ettinger, S. Cheng, B. Caine, *et al.*, "Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9710–9719.

[96] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, "Nuscenes: A multimodal dataset for autonomous driving", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 621–11 631.

[97] A. Malinin, N. Band, G. Chesnokov, *et al.*, "Shifts: A dataset of real distributional shift across multiple large-scale tasks", *arXiv preprint arXiv:2107.07455*, 2021.