université
PARIS-SACLAY

Motion forecasting of the objects in road scenes

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Automatique
Unité de recherche: Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France
Référent: : Centrale Supélec

Thèse présentée et soutenue à Gif-sur-Yvette, le 19 Mars 2021, par

Jean Mercat

Composition du jury:

Mathilde Mougeot Professeur, ENSIIE, ENS Paris-Saclay	Présidente
David Filliat Professeur, ENSTA Paris, U2IS	Rapporteur et Examinateur
Fabien Moutarde Professeur, Mines ParisTech, Centre de Robotique	Rapporteur et Examinateur
Yann LeCun Professeur, New York University, Facebook	Examinateur
Guillaume Sandou Professeur, Centrale Supélec, L2S	Codirecteur et Examinateur
Nicole El Zoghby Docteur, Renault, Guyancourt	Coencadrante et Examinatrice
Dominique Beauvois Professeur, Centrale Supélec, L2S	Codirecteur et invité
Guillermo Pita Gil Docteur, Renault, Guyancourt	Coencadrant et invité

Thèse de doctorat

NNT: 2020IUPASG008

ABSTRACT

The automotive industry is moving toward intelligent systems. The aim for more safety, comfort or even full self-driving systems requires the vehicles to take actions instead of the driver. It may use light signals, brake, accelerate, turn, and shift gears. Performing safely such actions requires a decision making process that anticipate the resulting situation. This is broken down in three "p" steps: perception, prediction, and planning. The perception system recognizes the current surrounding road-scene situation. The prediction system forecasts the future of the road-scene including the other road users. The planning system produces a driving intention.

If the future was perfectly known, planning would be straight forward. However, the perception system makes partial and noisy observations. The future cannot be perfectly predicted because of the uncertainties in the observation and because the human driver's decisions are modeled imperfectly. Therefore, improving the motion forecasting model that can work with a noisy observation system in complex situations is a critical point for the safety, comfort and future applications toward autonomous vehicles.

Motion forecasting can be modeled from various perspectives. The three main ones are kinematics and heuristics, statistics, and decision process. The kinematics and heuristics are the historical approaches that offer efficiency and robustness. Most vehicles using short-term forecasting systems on the road nowadays rely on such forecasting models. However, it leads to an overly conservative behavior when used in complex situations. The approach using decision processes shows good performances in complex scenarios. It is able to perform well in simulations and controlled environments. It combines forecasting and planning into a single task and may account for the reactions of the other drivers in the decisions it plans to take. However, in a noisy and wider, uncontrolled environment, statistical approaches still produce the best performances.

In this work, we use neural networks to learn the statistics of the vehicle trajectories. We progressively build a motion forecasting model from a constant velocity baseline to a complex interaction-aware neural network model. The evaluation criteria that we establish are used to judge the quality of the forecast and to compare the models. The likelihood of the future observation for the forecasted distribution is the main criterion that is commonly used in the applications. However, there is no universal forecasting quality criterion and it remains an open problem.

We begin our tests with vanilla neural networks for maximum likelihood motion forecasting. These models are then modified to also learn to fit the expected forecasting error. They optimize the NLL. It is a likelihood criterion for a given dataset of trajectories. This does not lead to much improvement over the constant velocity model. We go on to add interaction awareness to the model using multi-head attention. This brings much improvement to the forecasts but it is still insufficient. We improve the model further by considering different future possibilities. Finally, the multi-head attention model is extended to also attend to the surrounding lane shapes. The resulting neural network counts several blocks that account for different aspects of the task. Experiments show that each part of the model is useful. We show that the trained model has learned specialized attention patterns and is able to make multi-modal forecasts. Our results were the winning entry at the two first Argoverse forecasting competition.

Keywords: *Road scene*, *Vehicles*, *Neural networks*, *Forecasting*, *Prediction*.

RÉSUMÉ

L'industrie automobile se pare de systèmes intelligents. Les objectifs de sécurité, de confort, et même de systèmes autonomes demandent que le véhicule s'actionne de lui-même à la place du conducteur. Prendre le contrôle du véhicule de manière sécurisée demande une prise de décision qui anticipe les situations futures. Cette anticipation se fait en trois étapes : perception, prédiction, et planification. La perception reconstitue la scène routière environnante. La prédiction estime les états futurs de la scène. La planification produit une intention de trajectoire.

Si les états futurs de la scène routière étaient parfaitement connus, la planification serait simple. Cependant, le futur ne peut pas être prédit parfaitement à cause des incertitudes liées au bruit d'observation mais surtout parce que l'on ne sait pas modéliser parfaitement le comportement des conducteurs humains. Ainsi, pour garantir plus de sécurité et de confort, il est important d'améliorer les modèles de prédiction de scènes routières dans les situations complexes avec des observations bruitées.

Plusieurs approches peuvent être adoptées pour prédire le mouvement des objets des scènes routières. Les trois principales sont l'approche heuristique et cinématique, l'approche statistique et l'approche par processus de décision. L'approche cinématique et heuristique est celle adoptée historiquement et offre efficacité et robustesse. La plupart des véhicules actuels qui emploient une prédiction de court terme utilisent cette approche. Cependant, dans des situations complexes, ces méthodes sont trop conservatives. L'approche par prise de décision prédit les actions de chaque objet pour définir sa trajectoire future. Elle a de bonnes performances dans les scénarios complexes mais se limite à un environnement contrôlé ou aux simulations. Dans un environnement réel bruité, l'approche statistique offre actuellement le meilleur compromis.

Cette thèse emploie les réseaux neuronaux pour apprendre les statistiques de trajectoires de véhicules. Nous établissons des critères d'évaluation permettant de juger et de comparer les différents modèles. Le critère principal est la vraisemblance des données au vu des distributions prédites. Cependant, il n'y a pas de critère universel jugeant de la pertinence des prédictions et cela reste une importante question ouverte.

Nous débutons nos expériences avec des réseaux neuronaux classiques prédisant le maximum de vraisemblance des positions futures. Ces modèles sont ensuite modifiés pour prédire l'erreur de prédiction attendue en minimisant le critère NLL. C'est un critère évaluant la vraisemblance des données observées pour les distributions de trajectoires futures prédites. Ces premières approches ne surpassent pas la prédiction à vitesse constante. Nous ajoutons alors au modèle la capacité de représenter les interactions entre les véhicules grâce à l'architecture d'attention à plusieurs têtes. Les résultats obtenus sont alors bien meilleurs mais toujours insatisfaisants. Nous ajoutons donc au modèle la capacité de considérer plusieurs hypothèses futures et une capacité d'interaction avec le tracé des voies navigables. Nos expériences montrent l'utilité des blocs composant le réseau neuronal et représentant différents aspects du problème. Nous constatons une spécialisation du modèle pour certaines interactions types et que des hypothèses de trajectoires futures très différentes sont prédites dans certaines situations. Les résultats obtenus ont remporté les deux premières compétitions Argoverse.

Mots clés: *Scène routière*, *Vehicules*, *Réseaux neuronaux*, *Prédiction*.

ACKNOWLEDGEMENT

I wish to express my thanks to my supervisors at Renault **Guillermo Pita Gil, and Nicole El Zoghby** and at Paris Saclay L2S, **Guillaume Sandou,** and **Dominique Beauvois**. Nicole, thank you for the continuous implication you showed, and for the time you have given me even when your work and family did not leave you enough for yourself. Dominique, you helped me make this work more rigorous and precise, thank you for your long hours and your patience. Guillaume, your positive appreciations, questions, and optimistic guidance have given me confidence and motivation. Guillermo, you are a true mentor. You pushed me to trust in myself. Your help me inside and outside of Renault and you have given me the best opportunities. Thank you so much for your trust.

I am grateful to the Paris Saclay University, and to Renault for providing me with infrastructural facilities and other resources needed for my research.

I wish to extend my profound sense of gratitude to my colleagues, friends, and family for providing me with moral support and encouragement whenever required. Thank you to my colleagues from Renault: Boubker, Iris, Federico, Irene, Maxime, Bruno, Omer, Hana, Katherine, Simon, Chrysanthi, Salim, Bastien, Jorge, Franck, Romain. Thank you to my colleagues from L2S: Vincent, Jérémy, Thomas, Joffrey, Martin, Maxime, Joy, Matthieu, Baptiste, Dario. Thank you to Edouard Leurent, Zongwei Wu, Louis Guerlin, and Thomas Gilles for our work discussions and coffee breaks (that were often the same thing).

Merci Maman pour ton soutien particulièrement pendant le confinement. Merci Albane et Ilia ainsi que Félix et Claire pour votre accueil quand j'étais sans logis. Merci à mes sœurs, beaux-frères et à mon frère : Albane, Ilia, Tiphaine, Clément, Armelle, Alexis, Sibylle, Yoann, Paul.

Un merci particulier à Matthieu Patrizio, Marlène de Bank, Félix Gomez, Samuel Girardin, Gentien Marois, Lucas Lestandi, Tristan Rey, et Éric Martin. Nos discussions m'ont construit et m'ont guidé dans les choix de vie les plus importants. Je ne peux rien vous demander de plus et pourtant j'ai encore besoin de vous. Sans vous je serais inadapté à ce monde. Avec vous je suis toujours inadapté mais ça n'a plus d'importance.

Jean Mercat
Clermont-Ferrand
November 18, 2020

Foreword

This work is the result of the collaboration between the Laboratoire des Signaux et Systèmes at Université Paris-Saclay and the car manufacturer Renault. It is written in English instead of French in the hope of being read internationally by parts of the large and growing community of academics and engineers in the fields of autonomous driving and applied neural networks. We structure our writing as an iterative process for solving the problems that arise when the previous one is solved. This leaves out some of the broad explorations to focus our efforts on the practical solutions.

In our writing structure, we try our best to follow the recommendations from [Min09]: from the top down in a pyramid. This means that at some points, the conclusions might be given first, and the elements leading to that conclusion follow. This is almost reversed from the commonly used scientific structure: definitions, properties, applications. It seemed more appropriate to use the pyramid structure because this work is not about theoretical demonstrations but applied research and experiments.

Introduction

It is a hazardous thought to imagine our future. This might be why it is a common question at job interviews: "Where do you see yourself in ten years?" How demanding is that question! Are we even sure where we are now? So much could happen. What the world could become in ten years... How different could we become? The interviewer will make a hiring decision based on the information we give. So we make up a strategy. We consider what the interviewer wants to hear. We reason in abstractions. We make suppositions about the world and add a layer of our desires in these fantasies... and we get it wrong. Why do we get it wrong? Because we do not think rationally. Our fast thinking process activated by the urgency of the interviewer's question makes short cuts instead of slow rational reasoning. The fast and slow reasoning processes are described in [Kah11]. This way of thinking is fundamentally biased, as shown in [AJ08]. The proposed solution to avoid getting it wrong is to compare our situation with the one of others: see where they ended up ten years later and imagine the same thing for yourself.

In this work, we follow the same path as the interviewee in our story. We are a car, not the driver, the car itself. We sense the road and the surroundings in our own way. The interviewer is a part of the system that takes the decisions. And we must answer its question: "where do you see yourself relatively to the others in the next five seconds?" The same interrogations arise: Are we even sure where we are and where the others are now? So much could happen. How different could the road users behave? The interviewer must make a driving decision with the information we give. So we make up a strategy. We consider what the interviewer wants to hear. We reason in abstraction. We make suppositions about the world. But we must avoid getting it wrong. We think rationally. We only use the practical solution: what do the road users usually do? This question is answered by statistical learning. In this work, we build deep neural networks for statistical learning. We use it to make statistical forecasts of the movements of road scene agents.

Framing

Prediction and Forecast Prediction is a broad subject; it is with the Occam's razor, the criterion for a good scientific model: How predictive is it? We restrict our study to forecasts. A forecast predicts the outcome of a situation that was already observed. It is restricted to the future development of past events. By contrast, a prediction may predict the existence of an event that was never observed such as the presence of a planet at a certain point in time or the existence of a particle. Predicting is a battle against the unknown while forecasting is a battle against time.

Statistical Forecast Since our knowledge is limited, and because the events cannot be known before their unfolding, we must cheat. We cheat by forecasting only locally and partially. We cheat by forecasting only approximately. And, we cheat by giving multiple answers when only one is correct. However, cheating must be done with care and methodology. The partiality must be sufficient; the approximations must be controlled; each forecast must be likely.

Application in Autonomous Driving On a down to earth consideration, forecasting is a tool. It is a powerful tool in any decision process. One may use it to beat an opponent in a game, make money in a trade, decide whether to take an umbrella or make the best decision to avoid a car accident. One million, three hundred and fifty thousand lives were lost in traffic accidents in the year before the beginning of this thesis (2016). Most of these fatalities are due to human factors. The Advanced Driving Assistance Systems (ADAS) can help to prevent some traffic collisions. Forecasting the motion of the people and objects in road scenes is a step in that direction. The present work tries to contribute to the research for the improvement of such forecasting tools.

Road scene forecasting

Forecasting road scene movements is useful to make a decision and take action. This action may require some time to be executed safely and therefore requires a forecast time horizon between 3 to 5 seconds. In traffic scenes, this is considered a long term forecast. There may be many agents in the scenes, and any of them could behave in various ways. This is a very unstable setting in the sense that similar road scenes could lead to very different outcomes. For this reason, the forecast cannot be extended too far in the future and a range of likely outcomes must be considered.

Range of likely outcomes Technically, the range of outcomes is limited by the dynamics of the road scene agents. Each agent has a maximum acceleration, it cannot go faster than a maximum velocity, and it occupies an amount of space that it cannot share. However, this range of outcomes contains the possibility that every road user would try to destroy your vehicle and jump under your wheels. In these conditions, it would be impossible to do anything. To narrow the range of outcomes that should be considered, the notion of responsibility has been mathematicized in [SSS17]. The authors' recommendation is to only explore the range of outcomes that would not hold you responsible in the case of an accident. Then any outcome that would not be forecasted in this range would be blamed on the others. If everyone followed these rules, and if nothing was left out from the rules, no accident should be possible. Of course, some situations could have been left out, and it is a fact that the rules are not respected perfectly. Not following the rules should not be punishable by death; otherwise, nobody would survive adolescence. To avoid this cruel reality, the range of outcomes should be limited by another consideration: statistics. We want to limit the exploration of the outcomes using the likelihood that these events might happen. This means that we want to trade being sure of not being responsible for any accidents with being confident of not having accidents.

Problems with the statistics The statistics that must be considered to produce the forecasts should of course relate to a present situation. However, there is an infinite number of situations, and all of them are unique. Therefore, the statistics of the future development of a current situation is intractable. If we consider each agent separately, some statistics can be given: pedestrians usually walk at 5km/h, cars follow their lanes etc... However, this does not consider the overall situation. A maneuver to avoid an obstacle would be judged very unlikely because usually there is no obstacle to avoid. But, if we know that there is an obstacle, then it is very likely that it will be avoided.

To summarize, the statistics relating to the whole scene are intractable, and the statistics relating to each agent are insufficient. In this situation, how can we be confident that the accidents will be avoided? For now, the commonly accepted answer is to test the system with vast amounts of data and hope for the best.

Learning the statistics For the same reason that the statistics are difficult to verify, they are difficult to estimate. To avoid errors, we begin with the most straightforward approaches and complexify them step by step. In our first chapter, we build the simplest motion forecasting model to make a first statistical estimation of the road scene outcomes. Then, since we want to validate our system with data, we briefly present how they are acquired in chapter 2 before presenting how to use them for validation in chapter 3. Many different criteria are defined because none of them is sufficient to judge all aspects of the forecasts. We continue with the exploration of the solutions proposed by others in chapter 4. They differ in many aspects, and the various applications of ADAS systems may require specific types of forecasts. Overall, it seems that one approach is producing the best results for the evaluation that we defined: statistical learning with neural networks.

Neural networks

Before beginning this work, the author and his supervisors were not knowledgeable in machine learning and neural networks. Therefore, we introduce neural networks progressively and apply them in trivial examples in chapter 6. From chapter 6 to the end, more and more aspects of the problem are added and solved with different types of neural networks. In chapter 7, we describe and apply neural network architectures that model the interactions between road users. However, these forecasts do not describe all the possible futures. In chapter 8, we study how to fit a forecast distribution with many possibilities using generative models and mixture of Gaussians. Finally, in chapter 9, we apply all of the above in a single neural network architecture that forecasts the trajectories of the agents in a road scene with many possibilities, interacting agents, and interactions with the road network.

Throughout our progress, related work from the literature is presented and analyzed. Each incremental improvement is tested on real data (NGSIM dataset), and the final model is used with another dataset of real urban road scenes (Argoverse dataset). We used it to win twice the Argoverse motion forecasting competition, which allows us to compare our results with many other solutions from research and industry around the world.

Contents

Chapter 1

A first forecasting model

1.1 Introduction

Kinematic models are widely used in the automotive industry for motion forecasting. They rely on estimated kinematic states from noisy observations. The most used vehicle state estimator is the Kalman filter or its extensions. Some simple hypotheses about vehicle behaviors are made, such as constant velocity, constant acceleration, or constant turn rate. Forecasts can be made by merely integrating the kinematic equation. This first chapter produces such a kinematic forecasting model with the constant velocity hypothesis. It gives an easy to interpret baseline that serves as a reference for the improvements proposed in the following chapters. These forecasts are computed with real trajectories and compared to the actual measured outcomes. To this end, we use the NGSIM dataset.

1.2 Constant Velocity as a Forecasting Baseline

The simplest kinematic forecasts use the hypothesis that a quasi-constant action is applied during the whole forecast sequence. This assumption creates a variety of linear or linearized models described by Yaakov Bar-Shalom *et al.* in their book [YT13].

Since it is easy to implement and interpret, the constant velocity model is often chosen as a baseline in the motion forecasting literature. In [Sch+20], the authors show that the simplest constant velocity model (CVM) can outperform results from highly complex models, namely SR-LSTM [Zha+19] and RED [Bec+18], that were presented as State-Of-The-Art (SOTA) for pedestrian trajectory forecasting. This is true when multiple forecast possibilities are emitted. The approaches considering multiple possibilities are discussed in chapter 8. In the case of the NGSIM highway dataset, the constant velocity is also a good approximation because the roads are straight and, apart from the insertion zones and as long as the traffic is not too dense, the velocity on highways is near-constant. The CVM is simple and easy to implement, but different results may be found in the published articles with no explanation of the specific implementation and parameter choice. For instance, [DT18] and [Xu+20] implemented two different models that are both called constant velocity and produce forecasts with associated error covariance matrices, but no details are given to reproduce them. Consequently, the different results that they obtain cannot be fully interpreted.

The first step to making a kinematic forecast is to estimate the kinematic state from the (x, y) observations. Moreover, in addition to the forecast, we want to estimate the uncertainty of that forecast. The estimation should account for both uncertainty sources: the forecasting model and the state estimation. The errors of the constant velocity assumption and the perception errors are modeled with a random distribution over the state. This error distribution is modeled as a discrete-time centered Gaussian noise acceleration. It means that we consider a discrete timeline sampled at times t_k for $k \in \mathbb{N}$. Between t_{k-1} and t_k a random acceleration $\tilde{a}_k = (\tilde{a}_x, \tilde{a}_y)_k$ occurs. The accelerations $\tilde{a}$ are uncorrelated in time and follow a centered Gaussian distribution $\tilde{a} \sim \mathcal{N}(0, \Sigma_a)$. The acceleration sample at each time step k is noted $\tilde{a}_k$. The parameter Σ_a can be estimated empirically with a dataset of measures. The perception error is modeled as a centered Gaussian noise that is added to the real positions.

To produce a constant velocity forecast from noisy positional observation, we proceed in two steps: First, estimating the current kinematic state, then forecasting based on the state estimation.

1.2.1 State Estimation

As stated earlier, the perception is noisy. Moreover, the kinematic state is computed indirectly. The velocity is indeed retrieved from the position observations. To smooth out the noise and to estimate the kinematic state, we use the Kalman filter.

1.2.1.1 Need for Kalman

To estimate a vehicle position and velocity using noisy position observations, the noise must be filtered. Smoother position estimations could be produced with the average of the last two observations. However, it creates a time latency between the smoothed signal and the current state. The time window on which the average is computed is de-centered. For the average of the last two positions, the time window is centered half a time step in the past. A solution to avoid this latency is to re-center the averaging window on the present time. To achieve this, the future positions are needed, but they are not observed yet. To obtain the future positions at present time, a short-term forecast must be made. Thankfully, this can be produced using a model for the short-term kinematics applied to the current vehicle state estimation. Only the inertia is considered for this very short-term forecast. Thus, it tends to move in straight lines at near-constant velocity. Instead of computing the average of the current and previous observations, the average of the current observation and a *forecast* of the previous state estimation is computed. This can be improved again in two ways: Firstly, the uncertainty can be estimated. Secondly, it can be used to compute a weighted average between the predicted state and the observation. This improved filter is called the Kalman filter and was developed in [Kal60]. The uncertainty is estimated using a Bayesian update of a prior distribution over the state uncertainty. Two kinds of noises are considered: the observation noise, and the unknown acceleration. We model both as centered discrete-time Gaussian noises. This implies a zero mean acceleration and considers the variation of velocity in particular samples to be additional noise.

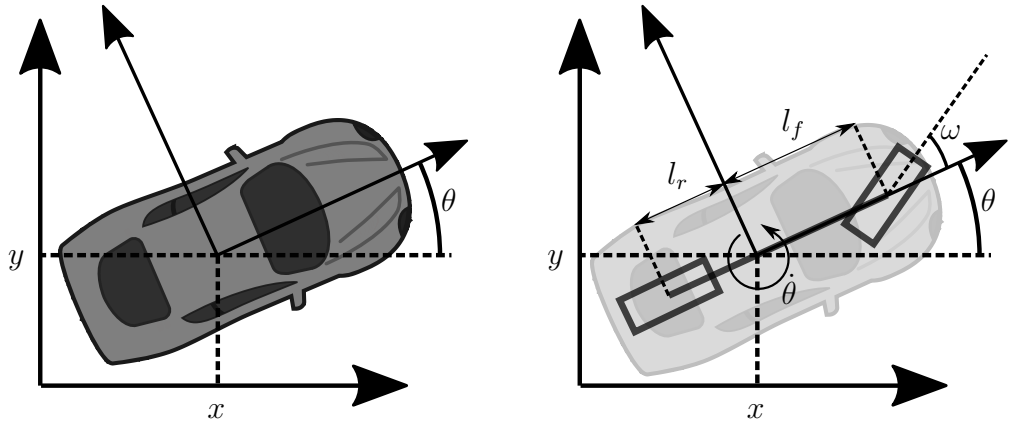


Figure 1.1: The coordinate systems used for the state representation of a vehicle.

To summarize, the filtering process filters out the noise but also a part of the information. This impacts the estimated amplitude and the phase that is delayed. The Kalman filter mitigates the information loss using a short-term forecast and a Bayesian update of the state estimation. We give details about this in appendix A.

Several expressions of constant velocity models for Kalman filtering with positions as observations can be made. We use the state vector $X = (x, v_x, y, v_y)^T$, with position (x, y) and velocity (v_x, v_y) because it produces the simplest linear Kalman filter for a constant velocity forecast. However, other work might choose $X = (x, y, \theta, v)^T$, with the non-linear equality : $(v_x, v_y) = (\cos(\theta)v, \sin(\theta)v)$. The bicycle model, with a state vector $(x, y, \theta, v, \omega)$, respectively position, heading angle, velocity, and wheel angle, describes the actual vehicle kinematics with a no-slip approximation. Both these states are represented in figure 1.1. The constant velocity model might have a different meaning with this last state vector because the wheel angle could be considered constant or, as done in[Xu+20], it can be set to 0 along with the acceleration. Many variations of these models are used. Consequently, even if it is a simplistic model, the name constant velocity and a brief description are not sufficient to reproduce the results.

1.2.1.2 Constant Velocity Model

The observations are noisy sequences of vehicle positions. The vehicle positions are modeled as samples from a kinematic process driven by a discrete-time centered noise acceleration. At each time t_0, an observation history sampled at a fixed frequency over a few seconds is used. It forms $N_H + 1$ coordinates called the past trajectory and noted $\{\tilde{Z}_k\}_{k=-N_H,0}$ with $\tilde{Z}_k = (\tilde{x}, \tilde{y})_k$. The position observation can be written as a true position plus the noise:

$$(\tilde{x}, \tilde{y})_k = (x, y)_k + (\tilde{m}_x, \tilde{m}_y)_k$$

For each sequence, the coordinate system is centered on the vehicle position at t_0 thus $(\tilde{x}, \tilde{y})_0 = (0, 0)$.

The observation is used sequentially to update a Kalman filter from its initial point; a satisfactory state estimation should be reached at t_0. The initial state uses the two first position observations to estimate the position and velocity. With our linear model, the evolution of the state $X = (x, v_x, y, v_y)^T$ from step k to step $k+1$ is written as follow:

$$X_{k+1} = AX_k + E\tilde{a}_k \tag{1.1}$$

A is the transition matrix; it represents the evolution model. In our case, $A = \begin{pmatrix} A_x & 0 \\ 0 & A_y \end{pmatrix}$ and $A_x = A_y$. With a time step dt, $A_x = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$.
E is the noise weighting matrix and $\tilde{a}_k = (\tilde{a}_{xk}, \tilde{a}_{yk})^T$ is the noise. We chose to represent the noise as an acceleration thus $E = \begin{pmatrix} E_x & 0 \\ 0 & E_y \end{pmatrix}$ and $E_x = E_y$.
With the time step dt, $E_x = \left(\frac{dt^2}{2}, dt\right)^T$.

We divide the Kalman filter process into three steps: forecast, innovation, and update. The forecast uses the model to produce the future state approximation $\hat{X}_{k+1|k}$ from the current state estimation $\hat{X}_{k|k}$. The state error covariance matrix P is updated with the transition matrix A and with a process noise covariance Q. It is independent of the observation. The observation at time t_k is $\tilde{Z}_k = (\tilde{x}_k, \tilde{y}_k)^T$. It is matched with the estimated positions of the state vector X using the matrix $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ in the innovation step. The innovation is an intermediary result that computes the difference between the forecast of the observation from the previous state estimation and the actual observation. The innovation covariance S is computed as the sum of the part of the state noise covariance that is observed HPH^T, and the observation noise covariance R. The update step uses the innovation to perform a Bayesian update of the forecasted state.

Forecasting:

$$\begin{aligned} \hat{X}_{k+1|k} &= A\hat{X}_{k|k} \\ P_{k+1|k} &= AP_{k|k}A^T + Q \end{aligned} \tag{1.2}$$

Innovation:

$$\begin{aligned} e_{k+1} &= \tilde{Z}_{k+1} - H\hat{X}_{k+1|k} \\ S_{k+1} &= HP_{k+1|k}H^T + R \end{aligned} \tag{1.3}$$

Update:

$$\begin{aligned} K_{k+1} &= P_{k+1|k}H^T S_{k+1}^{-1} \\ \hat{X}_{k+1|k+1} &= \hat{X}_{k+1|k} + K_{k+1}e_{k+1} \\ P_{k+1|k+1} &= P_{k+1|k} - K_{k+1}H_{k+1}P_{k+1|k} \end{aligned} \tag{1.4}$$

Using equations (1.2), (1.3), and (1.4) at every time step of the context sequence, we reach a good state estimation (position and velocity) at time t_0. Then, the kinematic forecast can be produced. It merely uses the short-term forecasting step of the filter. Thus, the next section only describes how the filter is initialized and used with multiple steps.

Algorithm 1: Kalman forecast

dataset : $[\tilde{Z}_{k\in[\![-N_H,N_F]\!]}]^{(i)} \quad i \in [\![1, N]\!]$
input : Q, R
for $\tilde{Z}^{(i)}$ *in dataset* **do**
  set: $\hat{X}^{(i)}_{-N_H}, P_{-N_H}$
  for k ***from*** $-N_H + 1$ ***to*** 0 **do**
    $\hat{X}^{(i)}_k, P_k \leftarrow \text{Kalman}_{\text{filter}}(\tilde{Z}^{(i)}_k, \hat{X}^{(i)}_{k-1}, P_{k-1}, Q, R)$
  for k ***from 1 to*** N_F **do**
    $\hat{X}^{(i)}_k, P_k \leftarrow \text{Forecast}_{\text{cv}}(\hat{X}^{(i)}_{k-1}, P_{k-1}, Q)$
    $\hat{Z}^{(i)}_k \leftarrow H\hat{X}^{(i)}_k$
    $\Sigma^{(i)}_k \leftarrow HP_kH^T$
return $(\hat{Z}^{(i)}_k, \Sigma^{(i)}_k)^{i=1,N}_{k=1,N_F}$

1.2.2 Constant Velocity Forecasting

The algorithm 1 describes the state estimation using the Kalman filter, followed by the constant velocity forecast. The forecast function in the second loop is the Kalman filter forecasting step using equation (1.2). For each sample from the dataset, a state $\hat{X}_{-N_H}$ and the covariance matrix P_{-N_H} are initialized using the approximations (1.5) below:

$$\begin{aligned} \hat{X}_{-N_H} &\leftarrow (\tilde{x}_{-N_H}, \frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt}, \tilde{y}_{-N_H}, \frac{\tilde{y}_{-N_H+1} - \tilde{y}_{-N_H}}{dt})^T \\ P_{-N_H} &\leftarrow \text{diag}(\sigma_{xx}, \sigma_{v_xv_x}, \sigma_{yy}, \sigma_{v_yv_y}) \end{aligned} \tag{1.5}$$

Then, all three steps of the Kalman filter are computed on the history sequence using the parameters: Q, R, the initial covariance P_{-N_H} and state $\hat{X}_{-N_H}$, and the observation sequence: $\tilde{Z}_{k=-N_H+1,0}$. The history allows the filter to make a filtered state estimation $\hat{X}_0$ at time t_0. From this time on, only the forecasting step is used without any observation to update it. The forecasted track for the next time steps, $k > 0$, is $\hat{Z}_k = H\hat{X}_{k|0}$. The associated error covariance matrix estimation is $\Sigma_k = HP_{k|0}H^T$. Since there is no new observation after this step, the notation showing the conditioning on the last observation can be omitted because it is always 0. As many steps as necessary to fill the desired forecasting horizon are made. This produces N_F forecast steps at the same sampling rate.

In the last two sections, we have defined a state estimator and a kinematic forecast using this state and its estimated covariance to produce a motion forecasting model. They rely on a few parameters that we define in the next section.

1.2.3 Parameter Estimation

The parameters to be defined are the process noise covariance Q, the observation noise covariance R, and the initial covariance P_{-N_H}. The process noise represents the position and velocity changes due to the acceleration that is unaccounted for in the forecasting model. It is modeled as discrete-time centered Gaussian acceleration thus we estimate its variance as the mean quadratic acceleration variation over one time step $q \approx \overline{\tilde{a}^2}$ in $m^2.s^{-4}$. Then the process covariance is the block-diagonal matrix:

$$Q = \begin{pmatrix} dt^4/4 & dt^3/2 & 0 & 0 \\ dt^3/2 & dt^2 & 0 & 0 \\ 0 & 0 & dt^4/4 & dt^3/2 \\ 0 & 0 & dt^3/2 & dt^2 \end{pmatrix} \mathrm{diag}(q_x, q_x, q_y, q_y)$$

The matrix is block-diagonal because the model considers that x and y are decoupled.

We have now fully defined our forecasting model. Its parameters depend on the observation noise and vehicle behaviors. Therefore, they are set using a dataset of real measurements. The covariance $P_{-N_H} = \mathrm{diag}(\sigma_{xx}, \sigma_{v_x v_x}, \sigma_{yy}, \sigma_{v_y v_y})$ is set using the data. However, the perception noise is unknown. We use the equality $\sigma_{v_x v_x} = \frac{2\sigma_{xx}}{dt^2}$ to approximate σ_{xx} and we estimate $\sigma_{v_x v_x}$ with the following estimation:

$$\sigma_{v_x v_x} = \overline{(\tilde{v}_x - \overline{\tilde{v}_x})^2} = \overline{\left(\frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt} - \overline{\frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt}} \right)^2}$$

This estimation would be exact if all the vehicles were going at the same velocity in all the data sequences and if we neglect the discretization error. The discretization error is expected to be small but we know that the velocities vary. Therefore, the covariance approximation is an overestimation of the actual perception noise.

$\overline{\tilde{a}^2}$ is computed as the mean quadratic acceleration in the dataset. The acceleration is estimated as follows:

$$\tilde{a}^{(i)}_{k+1} = \frac{x^{(i)}_{k+2} - 2x^{(i)}_{k+1} + x^{(i)}_k}{dt^2}$$

$\overline{\tilde{a}^2}$ approximates the variance because the mean acceleration is almost null. The observation noise R is set arbitrarily with a low value.

Improving the parameter estimations - The perception noise could be evaluated with much better accuracy from the dataset. In the application of chapter 4, we propose a solution to iteratively improve the estimation of the Kalman filter parameters. However, the resulting parameters cannot be interpreted as perception noise covariance. It would be possible to estimate the perception noise covariance with an approximation of the ground truth. A filter using both the past and future perceptions could produce a precise state estimation. This ground truth approximation could be compared with the perceptions to estimate the perception error.

The observation noise could be set using the innovation covariance estimation. R is the difference between this covariance and HPH^T.

We do not use these ideas to improve the parameter estimation because in this first chapter, we only aim to produce a simple baseline.

In the next section, we introduce the NGSIM dataset used to apply our model, set its parameters, and test its performance.

Figure 1.2: Recording system, sub-sections of the raw and processed samples for the NGSIM I-80 dataset. (source: [CH07])

1.3 The NGSIM Dataset

The Next Generation SIMulation (NGSIM) dataset [CH07] is a collection of vehicle trajectory data. They were observed from above with cameras in four locations of the US: southbound 101 and Lankershim Boulevard in Los Angeles, eastbound I-80 in Emeryville, and Peachtree Street in Atlanta. Figure 1.2 shows the setting, a row acquisition and a processed acquisition for the I-80 dataset. The observed zone is about 500m long. In that area, the vehicle positions are recorded at 10Hz and tracked. We use the preprocessing function of [DT18] that downsample the observations to 5Hz.

This dataset was built more than ten years ago, and since 2007, it has been widely studied for traffic flow and motion forecasting applications. This is an excellent opportunity to compare our results with other works from the literature. Of course, the data is not perfect. A consistency analysis has been made by [PB09] that showed a poor estimation of the acceleration and inconsistencies with the velocities, especially when studying relative velocities. The authors also pointed out the Lankershim dataset as less reliable.

For our forecasting applications, we use the I-80 and 101 datasets that are simple highway segments. We only use the (x, y) positions tracked in time. Our first model considers each track separately (without interaction). However, in chapter 7, we also consider the interactions between the vehicles in the local scene.

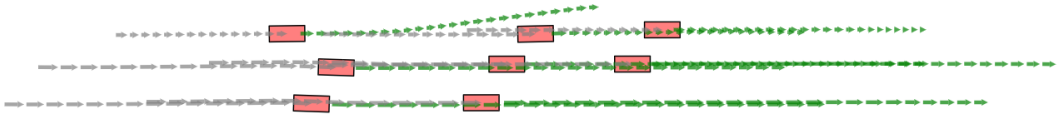


Figure 1.3: A sample from the NGSIM dataset with past observations over 3s in gray and the future observations over 5s in green.

The preprocessing from [DT18] breaks down the dataset into 8 second sequences of sub-parts of the observation centered on selected vehicles. It considers the agents within a neighborhood of 60 meters in length and 12 meters in width centered on the selected vehicle at a given time. All the vehicles positions within that frame are recorded for 3 seconds before the selected time and 5 seconds after. This constitute a *road scene*. Three partitions are formed: the training set containing about 6 million sequences, the validation set 0.8 million sequences, and a test set 1.5 million sequences.

All the vehicles are alternatively selected as the central vehicle. The sequences centered around the same vehicle are taken 4 seconds apart. This means that the last 4 seconds from one sequence are the first 4 seconds in another. Moreover, unless the sequence only contains one vehicle, each vehicle is tracked as the center vehicle in one sequence and also as a neighboring vehicle in other sequences. Therefore, for each scene, there exist a subset of other scenes that are correlated. However, we selected the training set, the validation set, and the test set so that they are disjoint. The correlated subsets are small compared to the whole dataset. Therefore, we consider that the road scene sequences are independent of each other.

Figure 1.3, shows a trajectory sample with the 3 first seconds used for context and the 5 following seconds used as ground truth to be compared with the forecast. It would be possible to use consecutive 8 seconds sequences to take advantage of the temporal continuity and avoid redundant computations. However, using independent sequences also has advantages. It ensures that every forecast is made using a similar state estimation with the same number of steps separating them from the initialization. The filter should reach a steady-state before the end of the context sequence. Then, because the initial Q and R are independent of the current observation, the history length should not make much difference, but using a fixed sequence size guarantees a similar state estimation for all samples with any parameter choice.

A set of 8 seconds road scenes forms a standard data usage in machine learning and will help with their applications later in this work. Other work and other datasets also use independent sequences for similar applications. Making this choice from the start unifies the experiments at the cost of some redundant computations and losing the possibility to study long sequences.

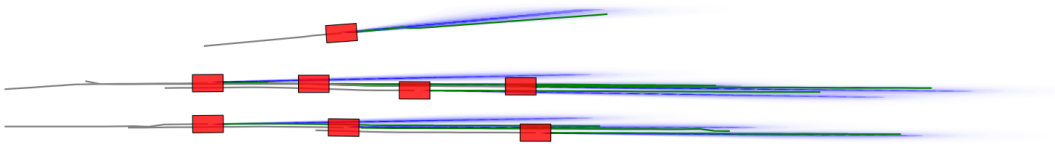


Figure 1.4: A sample from the NGSIM dataset with past observations over 3s in gray and the future observations over 5s in green. The blue shades represent the probability density described with the constant velocity Gaussian forecasts.

1.4 Evaluation

In the previous sections, we defined a first motion forecasting model. It uses the hypothesis that the acceleration is quasi-constant and can be modeled as a centered discrete-time Gaussian noise. It also considers that the observation is affected by a centered discrete-time Gaussian noise. From these approximations, a state estimation is made with the Kalman filter. Then, a constant velocity forecast is made using the estimate. The parameters of the model are computed using the measurements collected in the training dataset and are given in table 1.1.

Table 1.1: Constant velocity model parameters.

Parameter	σ_x	σ_y	σ_{v_x}	σ_{v_y}	q_x	q_y
Computed values	0.04	0.57	0.20	4.01	0.93	2.82

Parameter	r_{11}	r_{22}	r_{12}
Set values	0.01	0.01	0

The resulting model produces constant velocity forecasts as sequences of Gaussian distributions, as represented in figure 1.4.

In this section, we evaluate the forecasts on a separate part of the dataset called the test set. If the hypotheses are correct and if the parameters are well estimated, the forecast should be unbiased, and the empirical standard deviation of the forecast errors should match the standard deviations from the covariance prediction.
We consider that the observation noise is unbiased. Then, the empirical forecasting bias is computed at each forecasting step k on all the test data ($N = 1.5\ 10^6$) using the following equations:

$$\begin{aligned} \overline{\text{bias}_x}_N(k) &\overset{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \tilde{x}_k^{(i)} - \hat{x}_k^{(i)} \\ \overline{\text{bias}_y}_N(k) &\overset{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \tilde{y}_k^{(i)} - \hat{y}_k^{(i)} \end{aligned} \tag{1.6}$$

If the bias is small compared to the standard deviation, the Root Mean Squared Error (RMSE) is almost equal to the empirical standard deviation.

The RMSE is computed with the following equation:

$$\begin{aligned} \text{RMSE}_x(k) &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\tilde{x}_k^{(i)} - \hat{x}_k^{(i)})^2} \\ \text{RMSE}_y(k) &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_k^{(i)} - \hat{y}_k^{(i)})^2} \end{aligned} \tag{1.7}$$

The position covariance $\Sigma_k = HP_kH^T$ is diagonal because of the (x, y) independence caused by the model choices. It is independent of the specific forecast, but it depends on the parameters computed using the data. The square-roots of its diagonal terms $\sigma_x^{(\text{pred})}, \sigma_y^{(\text{pred})}$ are the predictor standard deviation estimations. The RMSE accounts for the error standard deviation but is also affected by the bias and the observation noise. Both the bias and the observation noise are small compared to the forecast error. Therefore, the standard deviation estimations and the RMSE should be close.

The first results from table 1.2 show the bias. There is a higher bias in the x direction than the y direction. In the dataset, the roads are almost aligned with the y direction. This means that the bias on the x is caused by lateral movements. The lateral motion is somewhat biased, whereas the longitudinal motion is not. In other words, the dataset is centered on a road portion where one direction is predominant. It is not negligible, but it remains small compared to the RMSE.

Table 1.2: Global indicators assessing the constant velocity model over the test dataset.

Time horizon	1s	2s	3s	4s	5s
$\overline{\text{bias}_{x\,N}}$ (m)	0.01	0.05	0.12	0.20	0.31
$\overline{\text{bias}_{y\,N}}$ (m)	0.00	0.00	0.00	0.01	0.01
RMSE_x (m)	0.34	0.69	1.05	1.39	1.74
RMSE_y (m)	1.33	2.85	4.56	6.45	8.57
$\sigma_x^{(\text{pred})}$ (m)	0.18	0.52	0.94	1.45	2.03
$\sigma_y^{(\text{pred})}$ (m)	0.85	2.42	4.44	6.83	9.54

The RMSE and the predictor standard deviation are close. The forecasting model underestimates the variance at short forecasting time under 2 seconds, and it overestimates the variance at forecasting time after 4s. This could be improved by tuning the filter parameters instead of using the computations from the data statistics explained above. The results are given in table 1.2 and represented on the graph 1.5.

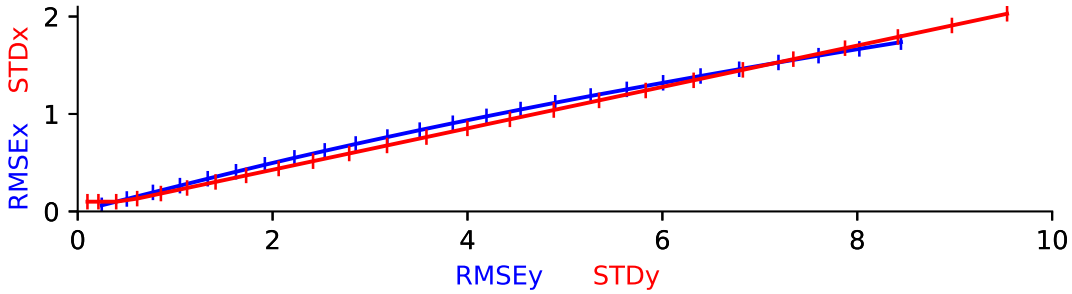


Figure 1.5: Parametric curves of RMSE in blue and standard deviation, $\sigma^{(\text{pred})}$ in red as functions in meters of the forecasting time.

In this chapter, we built a first forecasting model. Even if it is elementary in principle, producing only straight-line trajectory forecasts is a good baseline that is still widely used in the literature. We also introduced the need for uncertainty estimation that accounts for the noise and the unpredicted actions. We produced and applied this model on the NGSIM dataset. This is a first solution to the motion forecasting task that we study in this work. However, we have not clearly stated what our goal is. The two next chapters formulate our objective as a functional characterization: what the outputs should be and from which input. Chapter 3 gives ways to evaluate a forecasting model. This evaluation defines the motion forecasting objectives and helps to judge and compare the models produced in this work and the existing literature.

In this first chapter, the vehicle trajectories were observed from a camera above. However, most applications rely on a perception system embedded in a car observing the road scene from within. In the next chapter, the embedded perception system is presented to define how it forms the tracks in the road scene that the forecasting model uses to make the forecasts.

Chapter 2

Perception of a Complex Road Scene

The perception of the road scene is needed to feed many ADAS modules, including forecasting. Perception technologies figure among the prime efforts made in the development of self-driving vehicles. The present work does not build a perception and tracking system but relies heavily on the existing ones. This implies a modular system as represented in diagram 2.1.

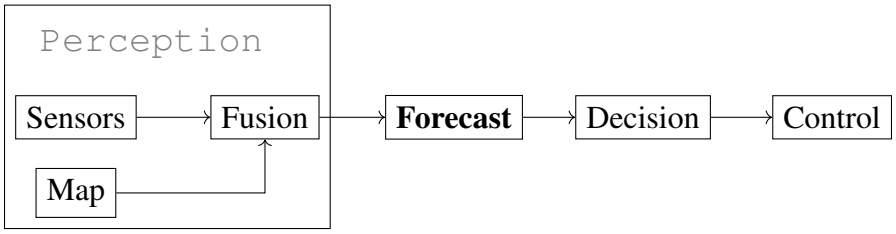


Figure 2.1: High level software architecture of the driver assistance system.

The perception system produces the road scene data that we use in the forecasting module and, as we will see, it affects its expected outputs. A trade-off between two extremes must be chosen: low-level fusion that might perform better but that is not modular, and high-level fusion that is modular. The low-level fusion, or early fusion, takes in raw perceptions from all the sensors of the vehicle and produces a road scene tracking. It is often a black-box system that is specific to the whole set of sensors. In this work, a modular perception system is used instead. It relies on smart sensors that individually interpret their perceptions before the fusion is performed. This is called high-level fusion or late fusion. It is modular and more easily adapted to different sensor sets than the low-level fusion. In the present chapter, we give an overview of the high-level fusion perception module.

2.1 Sensor Perceptions

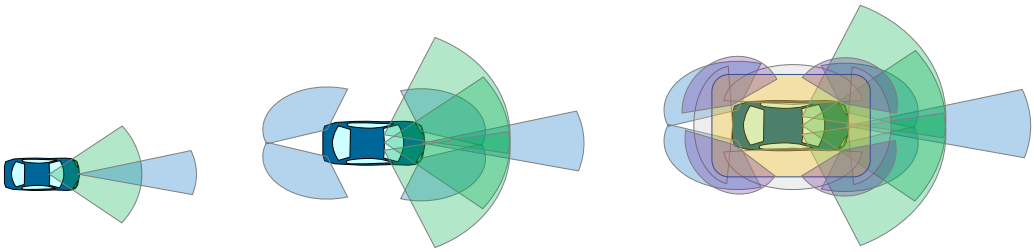


Figure 2.2: Representation of the perception field of the vehicle with different sensor sets. (Source: *Renault*)

Field Of View Perceptions are made with different numbers and types of sensors: ultra-sonic sensors, cameras, radars, and lidars. figure 2.2 shows different sensor sets with the corresponding Field Of Views (FOV). The FOV is complex, redundant in some areas and may present blind spots. Within the observed zones, depending on the sensor technology, different nature of information is acquired: the lidar measures distance accurately. The radars measure distances and velocity with more noise. The camera identifies objects but is easily occluded and is not very reliable to estimate depth. The ultrasonic sensors measure short-range distances at a low frequency and are only useful for parking maneuvers.

Object Tracking Within the field of view, the smart sensors measure raw data, make the first interpretation, and produce object-level detections that we call raw objects. These detections are made periodically by the sensors but are not tracked in time. This means that the fusion algorithm must recognize when the same object is seen several times and associate its detection in time to form a track. This gives a sense of motion to the road scene reconstitution.

Measurement Noise Each sensor has its own characteristics and is subject to different kinds of noise and errors. These errors are anisotropic and depend on the relative position of the observed object. Sensor manufacturers cannot give the error specifications because they depend on too many parameters to be guaranteed: the cleanness of the sensor, external disturbances, meteorological conditions, etc... Thus, the noise level is estimated by the fusion algorithm.

The raw object data are specific to the measurement technology of the sensor. In the next paragraphs we describe and illustrate the different perceptions of each sensor technology. This should give a sense of what the vehicle "sees."

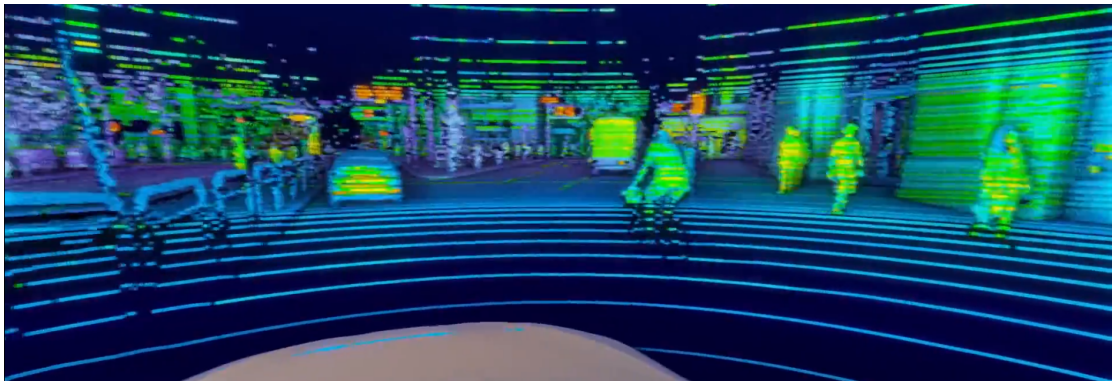


Figure 2.3: Visualization of the point cloud produced by the lidar velodyne V-128. (source: *Velodyne Lidar*)

The lidar (light detection and ranging) sends rotating laser beams and measures the intensity and time of flight of their reflections. A typical Field Of View (FOV) for a corner lidar extends to 80 meters over a $145°$ angle. A Velodyne VLS-128 has a $360°$ horizontal FOV and a $40°$ vertical FOV over a 300-meter distance. It produces a point-cloud with the reflective intensities that depend on the surface characteristics. Figure 2.3 represents the point-cloud generated by a velodyne V-128 lidar. The lidar perception uses active illumination. Thus, it does not depend on external lighting and has the same performances in the dark. However, it is affected by fog and rain. It is effective for *obstacles and road borders detection* and thus to define the drivable area. However, lidar sensors are costly and use fragile moving parts. This leads most manufacturers to avoid their use in large scale industrial applications. They are still abundantly employed to form research and development datasets such as the Argoverse dataset [Cha+19b] that we use in the final application of this work.

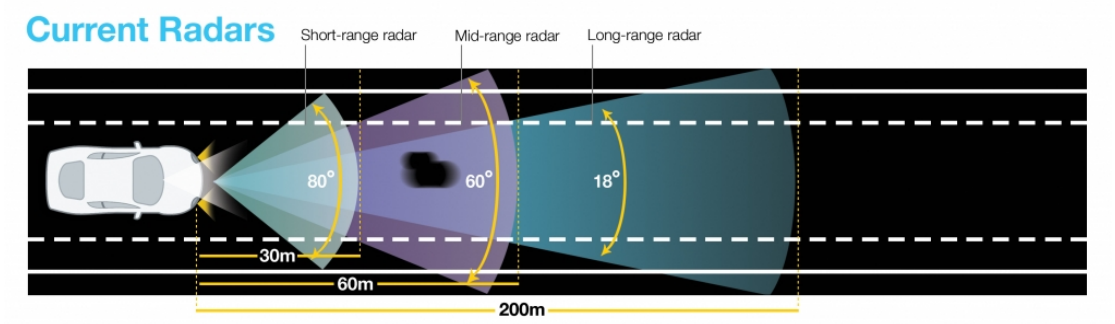


Figure 2.4: Visualization of a radar field of view. (source: *Arbe robotics*)

The radar (radio detection and ranging) emits radio waves and measures the reflections. It can measure the distance and velocity of objects and detects well the heavy metal objects. Being an active sensor, it sees in the dark, and its radio frequency is not affected by rain and fog. Its range and field of view are coupled and the same sensor switches periodically from near to far ranges, as illustrated in figure 2.4. It suffers from a lot of noise, leading to a compromise between undetected objects and false alarms. The radar is most useful for the detection of *moving surrounding vehicles*.

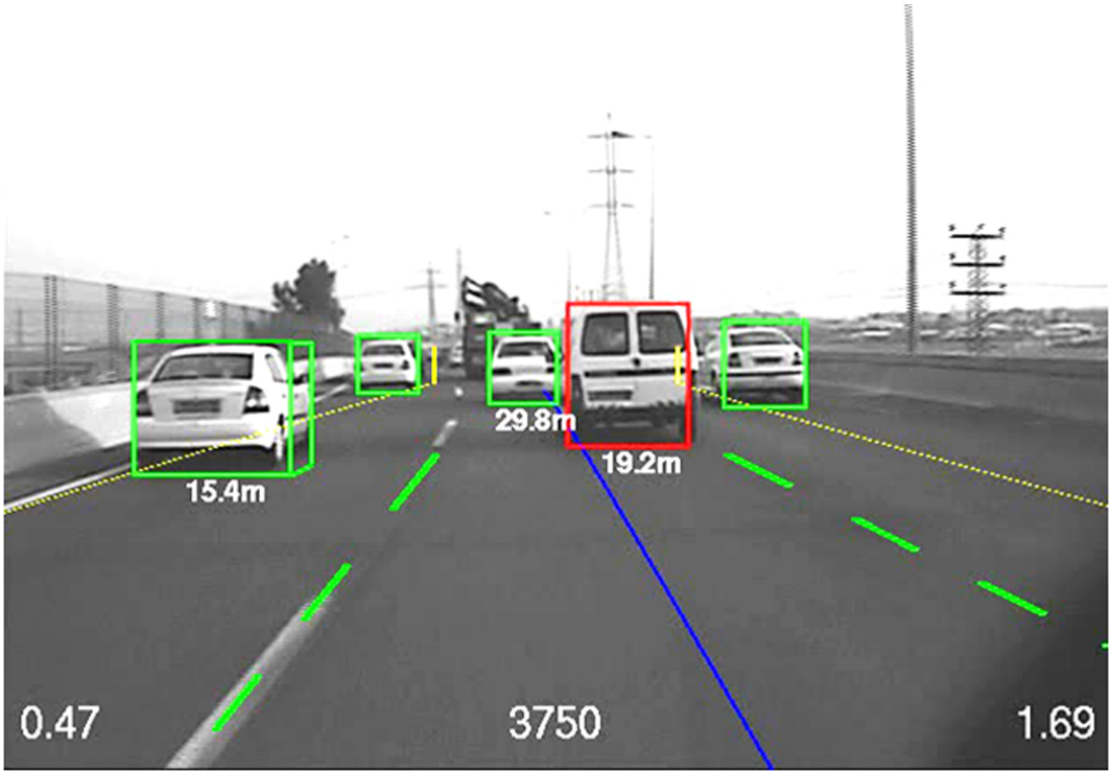


Figure 2.5: Representation of the object identifications from the smart camera. (Source: *Renault*)

The camera is used as a smart sensor. Its outputs are not raw images but, as represented in figure 2.5, it identifies objects such as lanes, free-space, and obstacles. The smart camera returns the identities and positions of the detected objects. This is the perception system that is closest to the human vision. Thus, it is well adapted to *identify signs, lines and road participants*. However, it is less reliable with night scenes, tunnel, and any situation needing a high dynamic range. It is also affected by cleanness, fog, and rain. Its object detection capabilities are evolving quickly but are still inferior to the human vision, mainly because it does not focus on important information. Many systems still do not use the temporality of observations to interpret the images with the movement. With one camera, the depth perception is still weak, but it can be improved with stereo cameras and with the camera movements.

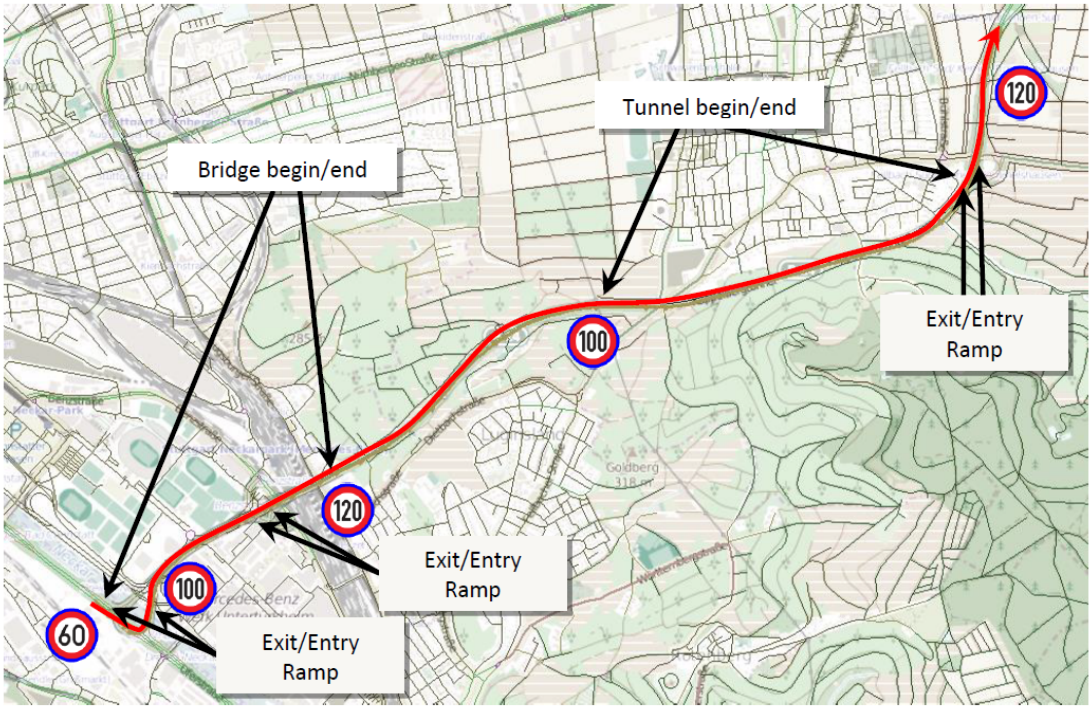


Figure 2.6: Representation of the information retrieved from an HD-map along a path. (Source: *Renault*)

The HD-map is not a sensor, but similarly to sensors, it is used as an information input. An example of information it contains is represented in figure 2.6. Key points from the surroundings, GPS, and lane assignments allow the system to self-localize in the map absolute coordinates. The knowledge from the map at the vehicle position is used to complete the representation of the road scene.

There is no standard perception - These descriptions of the sensors match the ones used by current driving assistance system and self-driving vehicles. However, the field is rapidly evolving. New sensors, such as imaging radars or movement based cameras, are very likely to change the perception capabilities and their characteristics. All manufacturers use different combinations of sensors placed at different positions on the vehicle. Most of the time, there is at least one or two front cameras. This can be completed with corner radars or lidars or a roof lidar. The different sensors each identify a specific set of objects in their field of view. Overall, the information observed in the aggregated field of view is unevenly distributed.

Raw perceptions are not a road scene - The perception hardware, even with smart-sensors, only produces raw objects that describe the road scene as an unordered list of features at different frequencies. Then, the fusion algorithm must form a global scene representation for all detected objects. It merges several perceptions of the same object from various sensors to gain accuracy and minimize uncertainty. It also identifies and tracks the objects in time. We call "road scene" the collection of information about the ego vehicle, the surrounding vehicles, other road users, and road equipments all identified in an ordered time sequence. In our final model, the collection of information is composed of the road agent position tracks and the lane centerlines.

2.2 Data Fusion

Figure 2.7: Fusionned road scene representation with the corresponding camera images.(Source: *Renault*)

The inputs of the data fusion module are the raw object perceptions from the smart sensors and map information. It also uses a fine knowledge of the ego vehicle kinematic state. It produces a coherent, unified representation and tracking of the road scene, as represented in figure 2.7. This representation is refreshed at a regular frequency chosen between 5 to 30Hz. On the left part of the figure, the FOVs of the sensors are represented with colored cones; the pink rectangle is the ego vehicle, other rectangles are fusionned targets representing vehicles, the orange ellipses represent radar detections. The lines are represented in gray, and the ego-motion forecasting is represented with the blue path. To produce this result, the fusion module must solve two main challenges: synchronicity and coherence. This is represented in the diagram 2.8.

Synchronicity - The measurements are asynchronous and the sensors measurement frequencies may vary. This can go as far as data being received in the wrong order. Each sensor uses its own clock and works at its own frequency. Thus, the different sensors time stamps might not be directly comparable and must be matched to a central clock. The input data might be received in a mixed order. Thus, the fusion module must either store a history of its input and re-order it or it should account for asynchronous data.

Coherence - The observed objects are not identified. There is no coherence between the consecutive raw objects measurements. The unordered list of objects includes duplicates and false detections produced by the sensors. The objects must be identified in a coherent representation in space and time. The spatial coherence is obtained by merging duplicates, filtering noise, and ignoring false detection. The timely coherence is made by the identification and tracking of the objects. The identification system either matches the detection with targets or considers them to be a false positive and ignores them. For pre-existing targets, a Bayesian filter such as the Kalman filter is already instantiated. It produces the expected observation that can be compared with the actual ones. Then, the perceived objects that are close to the expected ones are used to update

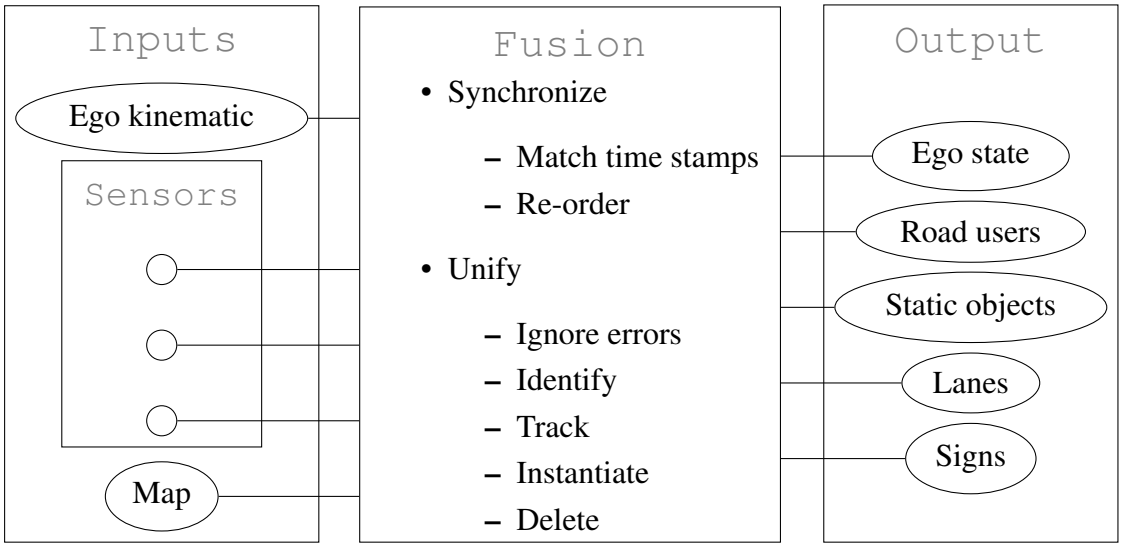


Figure 2.8: Diagram of the fusion module

the corresponding targets. Other detections are either errors or new targets. A history of the recent detections is kept and used to separate the errors from the new targets. If the detected objects are coherent with previous detections, a new target is instantiated. When for a long enough time, no observation matches an existing target, it is terminated. The target timelines reconstitutions are called tracks. This process is represented in figure 2.9.

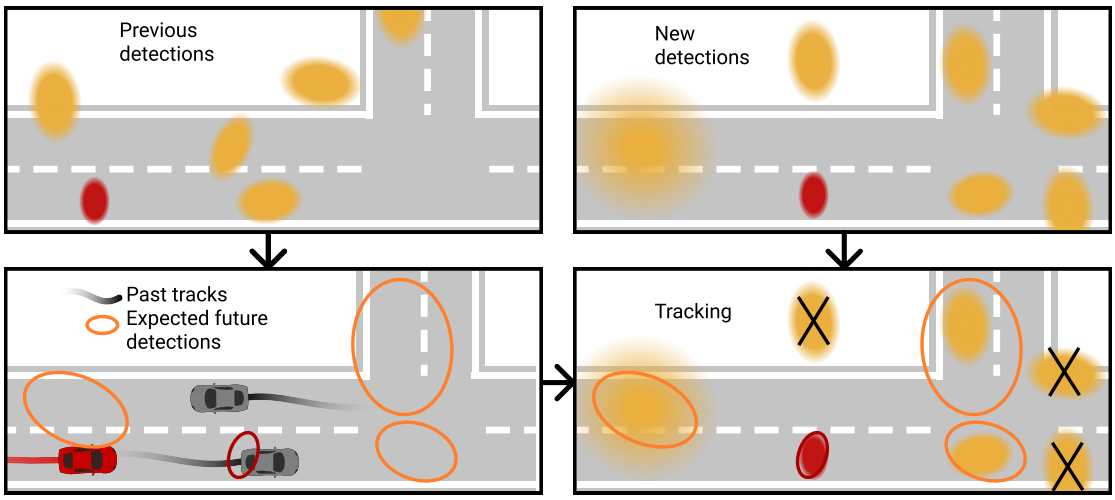


Figure 2.9: Illustration of the tracking process. The ego vehicle is colored in red. The top left figure represent the sensory input. The object detections are represented as orange and red ellipses with the size correlated to the uncertainty zone. The bottom left figure shows the updated road scene and the expected future positions of the tracked objects. The top right figure represents the next detections. In the bottom right figure, the new detections are compared with the expected road scene from the previous observation. It is used to ignore the false detections and to associate the detections to objects. These object states are updated with the new observations.

Object and model associations - Each track uses a specific model depending on the object type. For instance, the track of a pedestrian relies on a simple Kalman filter as presented in chapter 1. Vehicles also use Kalman filters based on a bicycle model that estimates their orientation, velocity, acceleration, and wheel angle. The estimated ego position, represented in red in figure 2.9 uses a different process than the position estimation of the other objects. The ego-vehicle localizes itself using an Inertial Measurement Unit (IMU), a GPS, and an HD-map using key-point features. Its kinematic state is estimated from the wheel rotation velocities and front-wheels angle. This process produces a reliable ego kinematic estimation. Static objects and signs are tracked using very different models updating their existence and properties without a forecast. Lines also have a specific tracking model that updates their parameters. They are either represented as polynomials or Euler spirals.

Moving observer - Once a model is associated to each object, it can be used to make short-term forecasts. The short-term forecasts are used to define the expected future observations. The association between the forecasts and the new detections must consider the ego-motion because the new detections are made from the viewpoint of a new ego position.

Fusioned road scene is all you need - The fusion model produces the road scene representation. The sensor acquisitions are unified; noise and errors are filtered and corrected. However, it cannot be perfect. The models relying on the fusioned road scene should account for errors and noise. In our case, the long term forecasting uses the road scene tracks as an input in its past observations and as a ground-truth to validate the forecasts. Thus, the road scene representation is both the forecasting model input and expected output. The absence of a real ground truth forces to be especially careful about the uncertainty estimation when measuring model performances. However, the minimization of the noise and error is already part of the fusion module and should not be repeated. To respect the modular architecture that is used in the vehicle software, the forecasting module should not try to improve what is already done in other modules. Therefore, it should rely on the scene representation that is produced.

A first forecasting model is defined in chapter 1. It is reliable for very short-term forecasts but insufficient for a longer forecast horizon. The complex nature of the perception system is simplified by a modular approach that abstracts the forecasting model from the raw sensor perception. It produces a road scene representation that can be used directly even if it still contains some noise and errors. The uncertainty of the perception system is too difficult to characterize because there are too many sources of error. Thus, only a statistical validation of this module and the subsequent systems is possible.

In this chapter, we presented the perception and tracking system that produces the road scene representation. This defines the input of our long-term forecasting model. Moreover, the same road scene representation defines the outcome to be forecasted. In the next chapter, we define how to evaluate a long-term forecast using these road scenes as inputs and as reference outcomes.

Chapter 3

Evaluating Gaussian Mixture Forecasts

The most challenging part of this work, and probably of most applied work, is to settle on a properly defined goal to pursue. This is equivalent to defining how to evaluate the solutions. In our case, it seems relatively easy to think of what a forecast should be: a sequence of positions matching the future observations. It also seems natural to evaluate: wait and see. However, it is difficult to formalize.

Achievable objective - The difficulty to evaluate the forecasts arises because what is achievable is not defined. Of course, producing forecasts that would match exactly the future observations would be a perfect solution and would be easy to validate, but it is not achievable. When formalizing what is achievable, the notions of uncertainties, multi-modality, and multi-objective arise. They blur the definition of our objective.

In this chapter, we settle on the expression of the forecast as a Gaussian mixture sequence. It forms a probabilistic and multi-modal forecast. This formulation forces the choice of a trade-off between different objectives. We are not able to define the best trade-off nor to properly formalize and define one unique objective. Thus, several evaluation criteria are defined and allow the judgment of different aspects of the results. Most of these criteria are used in other works from the literature. This will allow us to compare our solutions.

3.1 The Uncertainty Model

Gaussian assumption - As seen in the previous chapter, a moving agent in the road scene, (vehicle, pedestrian, bike) is observed with some noise. Each agent makes unpredictable small adjustments to its kinematic state (trajectory, velocity…), and makes decisions (brake, switch lane, give way…). We consider that the observation noise and small adjustments are both unpredictable. Thus, we model the (x, y) position uncertainty with a centered bivariate Gaussian. Its variance depends on both uncertainty sources.

Multi-modal assumption - We also consider that different decisions from the agents cause a discrete number of distinct possible futures called modes. The multi-modality means that the probability density of possible future positions contains local maxima. Therefore, the natural way to represent the forecast is with a bivariate Gaussian mixture. A Gaussian mixture is a weighted sum of several Gaussian distributions. Each one is called a component. To simplify, we fix the number of component per mixture to n_{mix}

independently of the scene to forecast. If the mixture components centroids are far apart compared to their variance, they represent probability density maxima called modes. On the contrary, they might be almost superposed. Then, several components may contribute to the same mode. The case where there are more modes than components is also possible but unlikely (see [CW03]).

Gaussian mixture definition - The Gaussian components are expressed with a mean value, a covariance matrix and a probability coefficient $((\hat{x}, \hat{y}), \Sigma, p)_m$. It defines a Gaussian distribution $\mathcal{N}((\hat{x}, \hat{y})_m, \Sigma_m)$ and its contribution to the whole mix p_m. One component represents an expected vehicle position $(\hat{x}, \hat{y})_m$ at one time in the future and its covariance Σ_m, that represents uncertainties and adjustments. Several components may describe different possible decisions that are more or less likely. We define the estimated probability of the decision with p_m. p_m is the mixture weight such that for n_{mix} components, $\sum_{m=1}^{n_{mix}} p_m = 1$. The whole Gaussian mixture probability density function is the sum of the components weighted with p. With $\mu = (\hat{x}, \hat{y})$, the Gaussian PDF is expressed for all $z \in \mathbb{R}^2$ as:

$$G_{\text{PDF}}(\mu, \Sigma)(z) = \frac{1}{2\pi \left|\Sigma\right|^{1/2}} \; e^{-\frac{1}{2}(z-\mu)^\top \Sigma^{-1}(z-\mu)} \tag{3.1}$$

The Gaussian mixture PDF is written:

$$GM_{\text{PDF}}(\{\mu_m, \Sigma_m, p_m\}_{m \in [\![1, n_{\text{mix}}]\!]}) = \sum_{m=1}^{n_{\text{mix}}} p_m G_{\text{PDF}}(\mu_m, \Sigma_m) \tag{3.2}$$

Independent forecasts - The complete trajectory forecast is a list of Gaussian mixtures for each forecast time step. If a model produces a joint forecast for several vehicles, the vehicle trajectories are not independent. However, we neglect the interdependence between the forecast distributions. This means that the forecasted probability density functions conditioned on the whole observed road scene that contains other agents are considered to be independent. For example, in the situation shown in figure 3.1, if one vehicle gives way the other will pass through. There might be a situation where it is unclear which vehicle is going to give way and which one will pass. Thus, the forecast for each vehicle should contain those two modes. Yet, only two combinations of those modes are likely. This can be phrased as: *making interdependent forecasts of vehicles is different than making forecasts of interdependent vehicles*. These interdependent forecast modes are challenging to produce but also to interpret for the subsequent modules. This is why we do not address this issue. Our models produce mode probabilities conditioned on the road scene (containing the other vehicle tracks, map information, and object perception), not on the occurrence of different modes from other vehicle forecasts nor any source of hypothesis.

To summarize, the forecasts are expressed independently for each vehicle as Gaussian Mixture sequences. They describe the possible future positions of the agents in the scene as bivariate probability density functions.

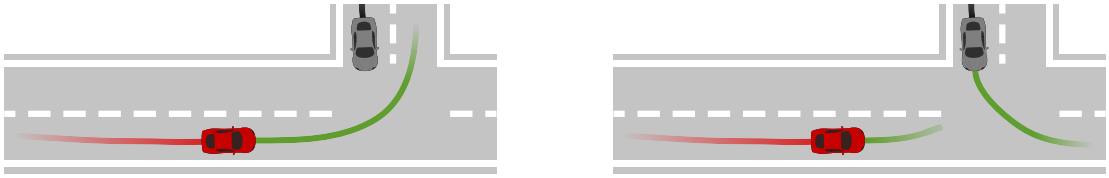


Figure 3.1: Illustration of a situation with two modes for each vehicle and an incompatible mode combination.

3.2 The Model Interface

The inputs are sequences of all vehicle (x, y) positions in a road scene. At each time t_0, we consider an observation history with a fixed observation frequency and a fixed number of observations n_{hist}. The past trajectory is written $\{(x, y)_k\}_{k=-n_{\text{hist}}+1,0}$. The coordinate system is centered on the observed ego vehicle position at t_0.

The outputs are sequences of Gaussians mixtures for each vehicle. They are expressed with $(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho, p)_{v,k,m}$ for each vehicle v, at each forecast step k and for each mixture component m. It defines a Gaussian component $(\mathcal{N}((\hat{x}, \hat{y}), \Sigma), p)_{v,k,m}$ with

$$\Sigma_{v,k,m} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}_{v,k,m}$$

$\Sigma_{v,k,m}$ is the covariance matrix, and $p_{v,k,m}$ is the mixture weight such that for n_{mix} components, $\sum_{m=1}^{n_{mix}} p_{v,k,m} = 1$.

The forecasting model is a set of functions pred_θ : inputs $\rightarrow$ outputs. The inputs and outputs sets are defined with the cartesian products:

$$\text{inputs} \in \left(\mathbb{R}^2\right)^{n_{\text{hist}} \times n_{\text{veh}}}$$

$$\text{outputs} \in \left(\left(\underbrace{\mathbb{R}^2}_{\hat{x},\hat{y}} \times \underbrace{\mathbb{R}_+^2}_{\sigma_x,\sigma_y} \times \underbrace{[-1,1]}_{\rho} \right)^{n_{\text{mix}}} \times \underbrace{\Delta^{n_{\text{mix}}}}_{p} \right)^{n_{\text{pred}} \times n_{\text{veh}}}$$

$\Delta^{n_{\text{mix}}}$ is the n_{mix} element probability simplex:

$$\Delta^{n_{\text{mix}}} = \left\{ (p_1, \ldots, p_{n_{\text{mix}}}) \in \mathbb{R}^{n_{\text{mix}}} \;\middle|\; \sum_{i=1}^{n_{\text{mix}}} p_i = 1,\ p_i \geq 0 \ \forall i \right\}$$

The pred_θ function set is defined with the parameter set θ. It may be defined for a given number of vehicles n_{veh} and a given number of forecast steps n_{pred}. However, it may also be a unique function defined for any number of vehicles and time steps.

We could give a straight forward extension of the forecasting model definition to any function that produce a probability density of the vehicle future positions. The modes that we identify with each mixture component would be extended to the local probability density maxima of this model. However, in this work we do not need this extension because all the models that we produce in our applications produce Gaussian forecasts and Gaussian mixture forecasts.

3.3 Performance Indicators

This section defines and interprets the performance indicators that we use to judge the quality of the Gaussian mixture forecasts. The validation dataset contains N road scene sequences. We compare the forecasts and future observations in each sequence using these indicators. Most of the time, the error is estimated only for the ego vehicle. Averaging the indicators for all vehicles in the scene is possible but is difficult to interpret because it contains incomplete observations with an unevenly distributed noise. The average indicator values over the N sequences give an overall performance evaluation. The indicators are functions of the forecasting time step k. They evaluate the performance of a forecast:

$$\left((\hat{x}_k^{(i)}, \hat{y}_k^{(i)})_m, \Sigma_m, p_m\right)_{k\in[\![1,n_{\text{pred}}]\!],m\in[\![1,n_{\text{mix}}]\!]}^{i\in[\![1,N]\!]}$$

The forecasts are compared with the observed future positions:

$$\left(\tilde{x}_k^{(i)}, \tilde{y}_k^{(i)}\right)_{k\in[\![1,n_{\text{pred}}]\!]}^{i\in[\![1,N]\!]}$$

This neglects the observation noise covariance that is expected to be small compared to the long-term forecast uncertainty. Biased observations would not change our results because the forecasts are relative to the past observations that would contain the same bias.

3.3.1 Distance Indicators

In this first paragraph, we describe the Root Mean Squared Error and Final Displacement Error for a uni-modal forecast ($n_{\text{mix}} = 1$) with a unique bivariate Gaussian forecast centered on $(\hat{x}_k^{(i)}, \hat{y}_k^{(i)})_{k\in[\![1,n_{\text{pred}}]\!]}^{i\in[\![1,N]\!]}$.

The Root Mean Squared Error (RMSE) computation is made with equation (3.3)

$$\text{RMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\tilde{x}_k^{(i)} - \hat{x}_k^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_k^{(i)})^2} \tag{3.3}$$

The Final Displacement Error (FDE) is the average distance between the forecast and the observation at time t_k. It is computed with equations (3.4).

$$\text{FDE}(k) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(\tilde{x}_k^{(i)} - \hat{x}_k^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_k^{(i)})^2} \tag{3.4}$$

RMSE and FDE comparison - In the particular case where $\tilde{x}_k - \hat{x}_k$ and $\tilde{y}_k - \hat{y}_k$ are uncorrelated centered Gaussians with the same standard deviation ($\sigma = \sigma_x = \sigma_y$), the ratio of the FDE and the RMSE is fixed. Under this hypothesis, the FDE is the mean of a Rayleigh distribution of parameter σ: $\sigma\sqrt{\frac{\pi}{2}}$. The RMSE is the square root of the sum of the variances, thus its value is $\sqrt{2}\sigma$. This means that $\frac{\text{FDE}}{\text{RMSE}} = \frac{\sqrt{\pi}}{2} \approx 0.89$. This shows that in this particular case, the FDE and RMSE give exactly the same information.

The RMSE and FDE characterize the expected forecast error but not the error for the whole distribution. Moreover, we need to extend these definitions for a mixture distribution.

Multi-modal FDE and RMSE are defined for the general model that we chose. When several trajectories with the associated probabilities are produced, three estimations of the RMSE and FDE are given for a set of n_{mix} proposed trajectories:

• The forecast with the maximum estimated probability:

$$\text{RMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} \mathbb{1}_{p_m^{(i)}=p_{\text{max}}^{(i)}} \left((\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2\right)}$$

$$\text{FDE}(k) = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} \mathbb{1}_{p_m^{(i)}=p_{\text{max}}^{(i)}} \sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2}$$

• A weighted average of the error for each proposition:

$$p\text{RMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} p_m^{(i)} \left((\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2\right)}$$

$$p\text{FDE}(k) = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} p_m^{(i)} \sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2}$$

• The error for the trajectories that produces the minimum final displacement error:

$$\text{minRMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\tilde{x}_k^{(i)} - \hat{x}_{k,\text{min}}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,\text{min}}^{(i)})^2}$$

$$\text{minFDE}(k) = \frac{1}{N}\sum_{i=1}^{N} \sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,\text{min}}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,\text{min}}^{(i)})^2}$$

Pessimistic and optimistic evaluations - This last evaluation requires the knowledge of the future trajectory to select the mode whose last position is the closest to the last future position. This leads to an overly optimistic error estimation but it is informative about the capacity of the model to fit the different modes. The weighted average and the maximum probability are overly pessimistic because they may include the error of far-off forecasts that might have been. The observed future is not always the most probable outcome that could have happened. Moreover, it mixes the capacity to estimate the probabilities of the different modes and the capacity to fit them. Thus, these indicators bound the error but are not sufficient to characterize it. We use three other indicators to characterize the performance.

The Miss Rate (MR) is the rate with which the forecasts miss the final position by more than 2m. This threshold is chosen to be smaller than the lane width while allowing an acceptable error margin. This ensures that an unpredicted lane change is counted as missed.

$$\text{MR}(k) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\sqrt{(\tilde{x}_k^{(i)}-\hat{x}_k^{(i)})^2+(\tilde{y}_k^{(i)}-\hat{y}_k^{(i)})^2}>2} \tag{3.5}$$

The miss rate is an important indicator; it gives a sense of "usefulness" of the forecasting model. A model unable to forecast a critical event such as an acceleration or deceleration or a lane change is indeed not very useful. However, a model that misses such events could still produce sound results with distance indicators such as RMSE. For example, in a case where a lane change might occur, forecasting a position between the lanes for the whole future sequence would produce a small error but is not a plausible forecast. The position between the lane is a transitional state that is not maintained over long sequences. A model producing such averaged forecasts is likely to have a low error on distance indicators and a high miss rate.

3.3.2 Probabilistic Indicators

Several Gaussian mixture components can be quasi-superposed. If each component describes a different mode, the output represents a broader exploration of the less likely modes, and it becomes easier to interpret. Thus, we propose a similarity indicator for the forecast propositions. A low similarity indicates a proper differentiation of the forecast propositions. Thus, for similar values of the performance indicators, a lower similarity is preferable.

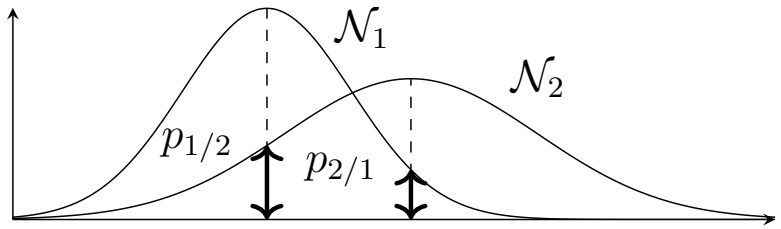


Figure 3.2: Graph of two Gaussian probability density functions (PDF). $p_{1/2}$ and $p_{2/1}$ are the values of one PDF computed at the mean value of the other.

The similarity indicator represents the average likelihood of the pairs of mixture components for one another. Figure 3.2 is a representation of the intermediate computations used to compute the similarity indicator. For each pair of Gaussian mixture components i, and j, with $i \neq j$, the probability density function of one component is computed at the centroid (the position of the mean value) of the other. This produces the two values $p_{i/j}$ and $p_{j/i}$. The similarity indicator involving these values is defined in equation (3.6).

$$\text{SIM}(k) = \frac{1}{n_{\text{mix}}(n_{\text{mix}}-1)}\sum_{i}^{n_{\text{mix}}}\sum_{j\neq i}^{n_{\text{mix}}} p_{i/j}(k)p_{j/i}(k) \tag{3.6}$$

The Negative Log-Likelihood (NLL) is a probabilistic indicator that relies on the forecast error and an estimation of the error covariance for each forecast. We use the NLL as described in [Bis94].

Let $\tilde{\mathbf{Z}}_k$ be the random variable of the observed position at time t_k, and $\hat{\mathbf{Z}}_k|_{\tilde{\mathbf{Z}}^h}$ the random variable of the forecast at time t_k knowing the history. The forecast is modeled as a bivariate Gaussian probability, $\mathcal{N}(\hat{Z}_k, \Sigma_k)$. We note $f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}$ the estimated probability density function of the position at t_k. Then the NLL is written:

$$\text{NLL}(k) = -\ln\left(f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_\mathbf{h}}(\tilde{\mathbf{Z}}_k)\right) \tag{3.7}$$

The covariance at time t_k is defined with the covariance matrix Σ_k. It can be defined with $(\sigma_x, \sigma_y)_k \in \mathbb{R}_+^2$ and the correlation coefficient $\rho_k \in]-1, 1[$ as follow:

$$\Sigma_k = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}_k$$

For a bivariate Gaussian distribution, the probability density function $f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}$ is written:

$$f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}(\tilde{z}_k) = \frac{1}{2\pi\sqrt{|\Sigma_k|}} \exp\left(-\frac{1}{2}(\tilde{z}_k - \hat{z}_k)^T\Sigma_k^{-1}(\tilde{z}_k - \hat{z}_k)\right) \tag{3.8}$$

The errors along x and y axis at time t_k are written d_x and d_y. The coefficients σ_x, σ_y, and ρ are identified with the coefficients of Σ at time t_k. Then the NLL_k of the forecast at a fixed time step k is given by the equation (3.9).

$$\begin{aligned} \text{NLL}_k^{(i)}(dx, dy, \Sigma) = & \underbrace{\frac{1}{2}\frac{1}{(1-\rho^2)}\left(\frac{d_x^2}{\sigma_x^2} + \frac{d_y^2}{\sigma_y^2} - 2\rho\frac{d_x d_y}{\sigma_x\sigma_y}\right)}_{(\tilde{z}_k - \hat{z}_k)^T\Sigma_k^{-1}(\tilde{z}_k - \hat{z}_k)} \\ & + \underbrace{\ln\left(\sigma_x\sigma_y\sqrt{1-\rho^2}\right)}_{\ln(\sqrt{|\Sigma_k|})} \\ & + \ln(2\pi) \end{aligned} \tag{3.9}$$

For a mixture of bivariate Gaussian distributions, the NLL is computed as follows (note the overloading of the function $\text{NLL}_k^{(i)}$):

$$\text{NLL}_k^{(i)}\left(\{dx, dy, \Sigma, p\}_{m\in[\![1,n_{\text{mix}}]\!]}\right) = -\ln\left(\sum_{m=1}^{n_{\text{mix}}} p_m e^{-\text{NLL}_k^{(i)}(dx_m, dy_m, \Sigma_m)}\right) \tag{3.10}$$

Over the dataset, the mean NLL value at time t_k is given by equation (3.11).

$$\text{NLL}_k = \frac{1}{N}\sum_{i=1}^{N} \text{NLL}_k^{(i)} \tag{3.11}$$

3.3.3 Interpretation of the NLL

The value of the bivariate NLL with a non-zero correlation ρ is difficult to visualize. However, the univariate expression gives a good idea of the NLL behavior in x and y directions for $\rho = 0$. If $\rho \neq 0$, the covariance can be diagonalized. Then, the interpretation is similar in a rotated frame aligned with the directions of its eigenvectors. The univariate NLL of the centered error z of covariance σ is expressed by equation 3.12 and represented as a contour plot in figure 3.3.

$$\text{NLL}_{\text{univariate}} = -\ln(f(z,\sigma)) = \frac{z^2}{2\sigma^2} + \ln(\sigma) + \frac{1}{2}\ln(2\pi) \tag{3.12}$$

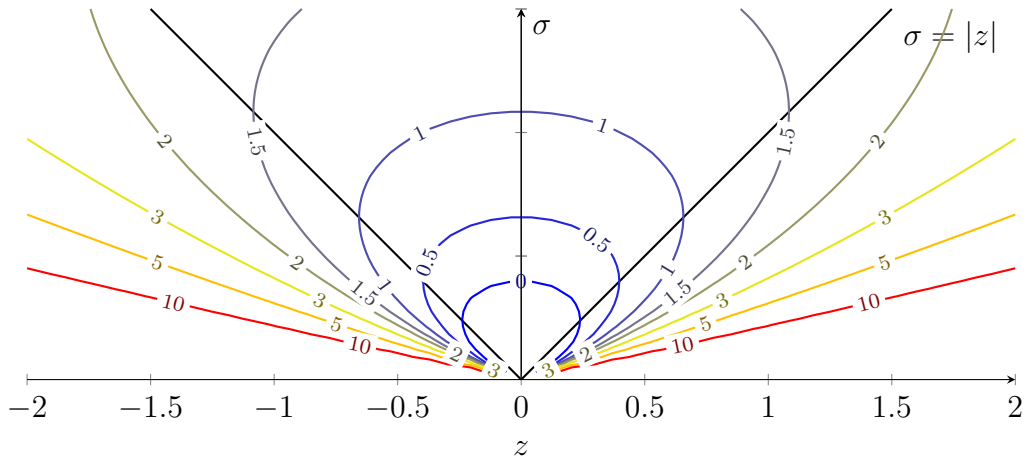


Figure 3.3: Contour plot of the univariate Gaussian NLL

Minimizing the NLL - The NLL derivatives are given by equations 3.13. It shows that the minimum in the z direction is obtained for $z = 0$. A growing z value causes a quadratic growth of the NLL with the slope $\frac{z}{\sigma^2}$. In the σ direction, the minimum is obtained for $\sigma = |z|$ and, if σ is large compared to z, the growth is logarithmic as σ is getting larger than $|z|$. This is shown for $z = 1$ in figure 3.4.

$$\begin{aligned} \frac{d\text{NLL}_{\text{univariate}}}{dz} &= \frac{z}{\sigma^2} \\ \frac{d\text{NLL}_{\text{univariate}}}{d\sigma} &= \frac{\sigma^2 - z^2}{\sigma^3} \end{aligned} \tag{3.13}$$

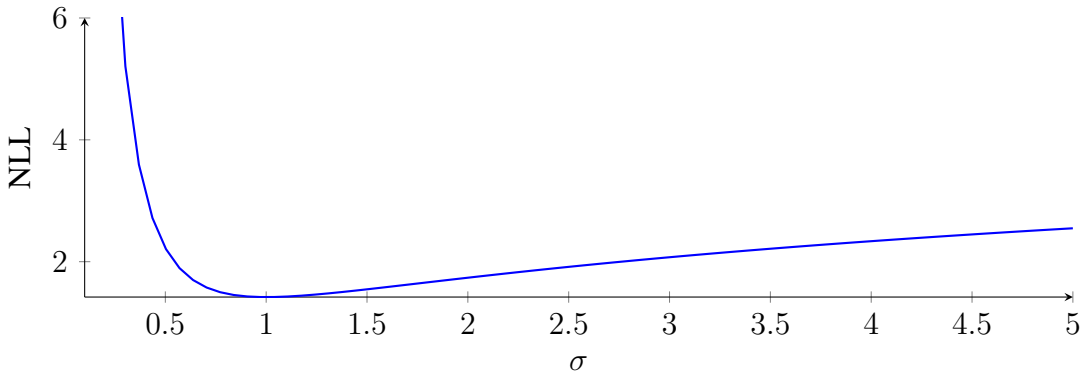


Figure 3.4: NLL as a function of σ for a fixed error $z = 1$.

Only one future - The true future distribution and even the covariance around the true future mode are unknown. Only the true future observation can be used to assess the estimated covariance σ. We consider that it is correctly estimated if it is close to the error $|z|$ because it minimizes the NLL. An underestimated covariance produces a much larger NLL value than an overestimated one.

Balancing $\frac{z^2}{2\sigma^2}$ and $\ln(\sigma)$ - For a given RMSE value, a low NLL value means that the estimated standard deviation is high when the forecast error is high. Increasing the forecast error increases the first term of equation (3.9) or (3.12) with quadratic growth. A high standard deviation compensates by a quadratic lowering of this term. However, increasing the standard deviation increases the second term with a logarithmic growth. With a small error, the second term of equation (3.9) or (3.12) is predominant. Thus, in the same way, when the forecast error is small, the estimated standard deviation should also be small to produce a lower NLL.

The NLL face value - A low NLL is a good indicator that the forecast error standard deviation is well estimated on each sample. However, its interpretation is not intuitive because it mixes the forecast accuracy and the error covariance estimation. The NLL is lowered either with a better covariance estimation or a better forecast accuracy (reducing $|z|$). Moreover, there is no reference value as with FDE and RMSE that reach 0 for a perfect forecast. With the NLL, if the forecast error is exactly 0, the first term of equation (3.9) or (3.12) is null for any covariance value. However, the second term is unbounded. It may tend to $-\infty$ as the covariance tends to 0 or the correlation to 1 or -1. To assess the forecast error covariance estimation separately from the forecast accuracy and to give an intuitive representation of it, we perform an error covariance assessment.

3.3.4 Covariance Assessment

One observed mode - The method presented here cannot assess the whole mixture of Gaussian but only the mode that fits the observed outcome. Therefore, the covariance assessment is more meaningful for the uni-modal case than with multiple modes. With multiple modes, the forecasting model outputs a covariance matrix at each time step for each component and we consider only the component for which the observation has the highest likelihood.

Sample aggregates - For each sample, a distribution is forecasted, but only one true observation is made. Thus, we must consider the distribution of the error over multiple samples. For each sample at the k^{th} time step, the forecasting model produces an error

covariance matrix $\Sigma_k^{(i)}$ associated with the considered component. The average, over N samples, of the estimated error covariance matrices after k time steps is an estimation of the global error covariance matrix and is noted $\overline{\Sigma_k^{(\text{pred})}}_N$. The mean covariance estimation $\overline{\Sigma_k^{(\text{pred})}}_N$ can be compared with the empirical error covariance of the forecast error computed on the dataset $\overline{\text{var}}_N(\mathbf{dZ}_k)$. We note the error $\mathbf{dZ}_k = \tilde{\mathbf{Z}}_k - \hat{\mathbf{Z}}_k$.

$$\overline{\text{var}}_N(\mathbf{dZ}_k) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \mathbf{dZ}_k \mathbf{dZ}_k^T \tag{3.14}$$

In the case where every covariance estimation is dependent on the input observation, this does not allow individual comparison between estimations made by the forecasting model and an empirical covariance estimation. However, in all cases, the average of the individual covariance estimations should be close to the empirical covariance, *ie:* $\overline{\Sigma_k^{(\text{pred})}}_N \approx \overline{\text{var}}_N(\mathbf{dZ}_k)$.

Covariance visualization - If the forecast error bias is small, the average of the individual error covariance estimations $\overline{\Sigma_k^{(\text{pred})}}_N$ should match the empirical covariance of the error $\overline{\text{var}}_N(\mathbf{x}_k)$. The comparison may be performed in a table of values, as it was done in chapter 1. We showed that the bias is small, and we compared the forecasted standard deviation with the RMSE. However, in the case of correlated x and y values, the values are not easy to compare. Thus, the covariances may also be represented with ellipses describing the contour of one standard deviation. The unit dispersion ellipses comparison is easily made visually and allow the covariance estimation assessment.

We have now defined all the indicators used in this work for a generic forecasting model evaluation. In the next section, we use the results from chapter 1 to compute the indicators values and evaluate our first model.

3.4 Application to the Baseline Evaluation

In this chapter, we defined the tools to evaluate any forecasting model that outputs Gaussian mixtures. This forms an evaluation procedure that we apply to evaluate the constant velocity forecasting model defined in chapter 1. Its interface is compatible with the one defined for the evaluation process. The sequence of Gaussian it produces can be seen as a special case of Gaussian mixture with only one component.

Error estimation and RMSE - Table 3.1 presents all the performance indicators defined in section 3.3 at different time horizons (except for the similarity indicator because this model is unimodal). The line "STD" presents the averaged standard deviation distance, $\frac{1}{N}\sum_{i=1}^{N}\sqrt{{\sigma_x^{(i)}}^2 + {\sigma_y^{(i)}}^2}$ with σ_x and σ_y the forecast standard deviations in the eigen directions. If the standard deviations are well estimated, it should be comparable to the RMSE values. The x and y standard deviations are compared with the x and y RMSE in figure 3.5.

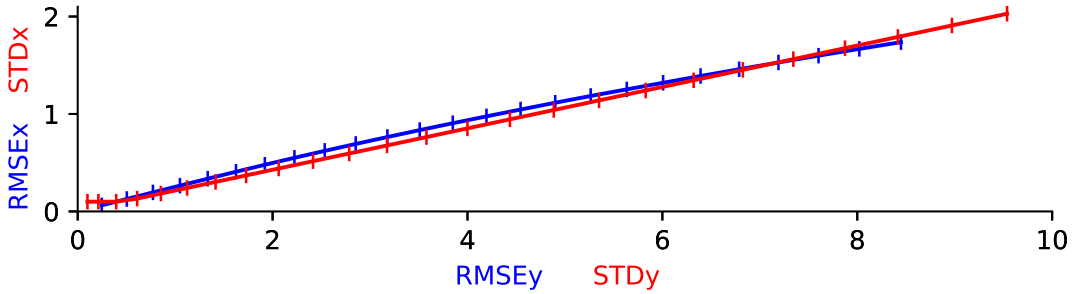


Figure 3.5: Parametric curves of RMSE in blue and standard deviation, $\sigma^{(\text{model})}$ in red as functions in meters of the forecasting time.

Table 3.1: Performance indicators for the constant velocity model on the NGSIM dataset.

Time horizon	1s	2s	3s	4s	5s
RMSE (m)	1.38	2.94	4.67	6.59	8.75
STD (m)	0.87	2.47	4.53	6.98	9.75
bias (%RMSE)	1.08	1.86	2.56	3.09	3.55
FDE (m)	0.65	1.61	2.79	4.19	5.77
NLL	2.88	3.66	4.41	5.04	5.57
MR	0.05	0.26	0.49	0.65	0.75

Performance at time horizons - In the typical situations from the NGSIM dataset, the constant velocity model gives reasonable results. The dataset is composed of highway trajectories that are almost straight line road segments. At one second in the future, the results of the first column of table 3.1 are satisfactory. The miss rate is very low, and the predicted position is on average at a distance under 1 meter from the truth. As the time horizon increases, in the next columns, the forecasts are more difficult to make, and indeed the error grows.

Reading NLL values - The NLL values are to be interpreted as lines of the contour plot in figure 3.3. The FDE indicator shows the mean error value giving the average position on the x axis of the NLL contour plot 3.3. This is on average around 0.65 meters at 1s. The RMSE is much higher at 1.38 meters. For this FDE, the NLL value could be almost as low as 1 with a perfect covariance estimation. Thus, the NLL value of 2.88 shows a poor covariance estimation. This number is still challenging to interpret. Thus, making a visual covariance validation is necessary. For the average individual covariance estimation $\overline{\Sigma_k^{(\text{pred})}}_N$ to match the empirical covariance $\overline{\text{var}}_N(\mathbf{dZ}_k)$, the error bias should be small compared to the RMSE value. The bias value is reported in the second line of table 3.1 with biases under 4% of the RMSE value at all forecasting time. However, in figure 3.5, that was already presented in chapter 1, the standard deviation estimation in red is overestimating the RMSE in blue, especially on the horizontal axis. The red curve is slightly under the blue one on the vertical axis, for the first points, showing that the standard deviation estimations are slightly underestimated in this direction.

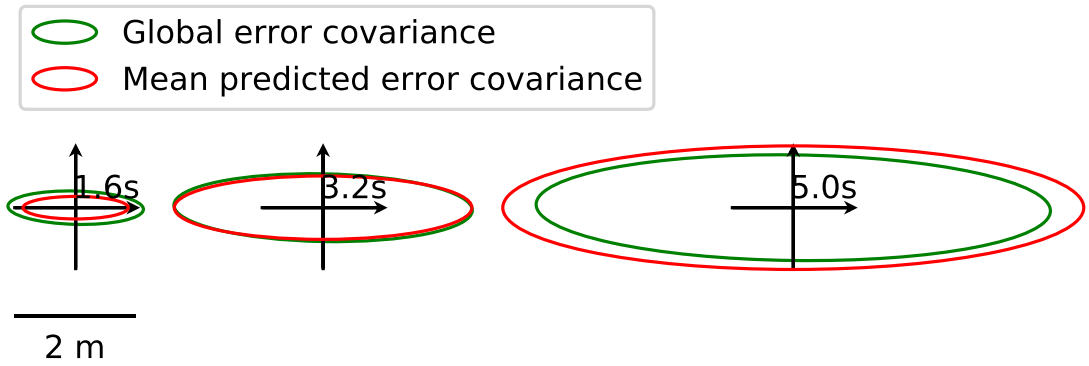


Figure 3.6: Comparison between the error covariance estimation and the global empirical error covariance.

Covariance visual assessment - Figure 3.6 shows a comparison of the empirical error covariance and the average of the estimated error covariances. The covariance matrices are represented with their unit dispersion ellipses. The ellipses are oriented along the first eigenvector of the covariance matrix with the radius given by the first eigenvalue and width by the second eigenvalue. In two dimensions, they represent a 39% probability domain. We represent them at 3 instants of the 5s forecast. As expected from the previous graph, it shows an under-estimation of the covariance at a short horizon and an over-estimation at a longer horizon. This representation is more intuitive than the graph to evaluate the quality of the covariance estimation, and it adds information about the correlation between the directions x and y.

We are unable to define a unique value judging for the quality of a forecast. However, using the multiple indicators defined in this chapter, we can evaluate and compare many aspects of the forecasting models. Computing each indicator and using their interpretation as discussed in this chapter, forms an evaluation process. We used this process on the results of the constant velocity forecasts from chapter 1.

Overall, the evaluation made in this section shows acceptable results in terms of distance for short-term forecasts (under one second). However, the covariance estimation is not satisfactory. The Kalman filter parameters are computed using the dataset, as described in section 1.2.3. However, the parameters are computed using noisy data and simplistic approximations. Moreover, the data might not entirely follow the model assumptions such as the discrete time Gaussian noise assumption. Therefore, it is possible to find a better set of parameters to define the forecasting model. However, even with better parameters, the constant velocity model remains very limited. To go further, in the next chapter, we review the approaches that model human behavior instead of pure vehicle kinematics.

Chapter 4

Modeling the human behavior

Figure 4.1: A very crowded road scene. (Photo: *A. Roy Chowdhury*)

Figure 4.1 shows an overwhelmingly complex road scene. This example shows how far the forecasting model proposed in chapter 1 is from a viable solution. In this situation, driving is a negotiation between humans. The vehicle kinematics is not the main factor to consider for the forecasting model. In this section, we give a broad overview on the different methods used for trajectory forecasting. This guides us toward the best-performing method of this past few years: neural networks.

The application makes the model - As shown in chapter 2, motion forecasting is only a module in a broader system. This implies that forecasting is a means to accomplish other goals. Thus, the best motion forecasting model might not be the one that produces a description of the most likely future. In fact, the notion of trajectory itself could be replaced. The forecasting model might instead be more useful if it predicted the intentions of the surrounding road users. The forecasting module objective could be to ensure safety. Then, a model describing the worst-case scenarios could be preferred to a model describing the most probable future. For this task, the objective could also be to decide if a maneuver should be performed. A model describing relative semantic actions such as {a follows b, a yields to b, a overtakes b} could be preferred over lists of Cartesian coordinates.

Multiple approaches for multiple applications - Because of this variety of objectives, there is also a wide variety of solutions in the literature. In their survey, [LVL14] separate the vehicle evolution models: *dynamic* or *kinematic*, the maneuver intention models: *classification*, and the trajectory prediction using them: *single trajectory simulation*, *Gaussian noise simulation* and *Monte Carlo simulation*. Another section is dedicated to risk assessment. The semantic prediction, such as [HZT18] can be regrouped with maneuver prediction methods because they both formalize the problem as a classification task. Finally, decision-making models are also able to produce forecasts. They are not centered on the forecasting task, but forecasting is an inherent part of their process. Producing forecasts from a decision model involves simulating the recommended action over the next time steps to define the future trajectory. These models can be formalized in game theory as a Markov decision process. An excellent review of the more recent trajectory forecasting models is written on the Standford AI Lab blog [1] and is based on their paper [Sal+20]. It separates ontological and phenomenological approaches that they define as follows:

> *"Ontological approaches (sometimes referred to as theory of mind) generally postulate (assume) some structure about the problem, whether that be a set of rules that agents follow or rough formulations of agents' internal decision-making schemes. Phenomenological approaches do not make such assumptions, instead relying on a wealth of data to gleam agent behaviors without reasoning about underlying motivations."*

In the next sections, we separate the forecasting models that predict the human behavior in four classes of approach: *behavior heuristics*, *game theory*, *maneuver classification*, and *statistical forecast*. By considering the human behavior, these models extend the forecasting capabilities to longer time horizons or produce a different formalism.

[1] `http://ai.stanford.edu/blog/trajectory-forecasting/`

4.1 Behavior Heuristics

Models based on heuristics are defined with a few parameters that affect the forecasts in a way that is easy to explain. For example, the model detailed below depends on an aggressiveness level, a comfortable maximum acceleration, and a reaction time.

IDM-MOBIL - One well-known combination of heuristics to describe the average vehicle behavior is the Intelligent Driver Model (IDM) [THH00] with the Minimizing Overall Braking Induced by Lane changes (MOBIL) [KTH07] model. IDM is a car-following model that describes the longitudinal behavior of a car, knowing the distance to the vehicle in front of it. It tends to converge to a hand-defined desired velocity when the space between the cars is wide. It smoothly adjusts the velocity to adapt it to the vehicle inter-space. The model converges to a hand-defined time interval between cars when the traffic is dense. MOBIL is a lane change decision model (and not a lateral behavior model) that outputs the decision to change lane left or right or remain in the current lane. The lane change is decided if it leads to a situation where the IDM model would accelerate more than a given threshold value. However, this lane change might force a deceleration to other vehicles. A trade-off is set between the ego acceleration gain and the deceleration of the other vehicles. It uses two parameters called politeness factor and maximum safe deceleration. This model can be set to incorporate symmetric or asymmetric rules. For asymmetric passing rules such as the European ones, the right lane is the default one, and only the left lane is used to take over. IDM/MOBIL is rarely used as a motion forecasting model and often as a traffic flow simulator that can be set and compared using empirical observations such as the ones in the NGSIM [CH07] dataset. It can reproduce emerging characteristics of the traffic such as traffic waves, phase diagram, and lane changing rate but not the precise local position of each car.

Social forces - Another heuristic for social interactions used in pedestrian motion forecasting is the social force model [HM95]. It defines a force pulling each agent toward its desired position. This force produces a smooth acceleration in a straight line. A second force accounts for obstacle avoidance along the way. It acts as a repulsive effect that accounts for the agent velocities. Finally, a third force accounts for convergent behaviors, such as agents following each other with an attraction. The social forces model had some success in pedestrian motion forecasting, but it cannot predict complicated behaviors. Moreover, it would be difficult to adapt to vehicle motion forecasting, especially if it cannot rely on an HD-map to consider lane-following behaviors.

Real-world self-driving approaches - The DARPA Urban Challenge (DUC) [BIS09] has set the first milestone in urban self-driving. The different participants used various motion forecasting strategies. In zones without markings, the teams Boss and Odin use simple kinematic predictions such as the one we described in chapter 1. In zones with markings, they include heuristic optimization procedures with obstacle avoidance and lane following strategies. When several lanes could be followed, multiple forecast hypotheses are produced. The Junior team uses another strategy. It unifies its forecast strategy in all cases with a particle filter. Several particles represent the same object with small state variations. A kinematic model applied to each particle gives the forecast, and its distribution. These forecasts allow the participants to avoid collisions with the moving objects, but they lead to overly conservative behaviors. Finer behavior forecasts would allow less conservative models while keeping a solid security level.

The key points missing from the methods developed for the DUC are data-driven statistics and interaction patterns. The statistics allow lower error margins and a better selection of hypotheses. Modeling interactions enable longer-term forecasts and may allow strategies that involve influencing others.

4.2 Decision Making

In contrast with heuristics, decision making using game theory or reinforcement learning approach attempts to approximate the human decision process with interactions. It does not only forecast the vehicle trajectories but it makes driving decisions. Depending on the model, the forecast can be done explicitly as in [Fis+19] or implicitly as in [Din+18]. This approach uses an approximation of the human objective, and it works under the assumption that the human will choose its actions to achieve that objective. Either the human optimizes the objective achievement, or its actions are seen as samples from a distribution that is biased toward the objective resolution, as in [Fis+19]. These models try to find a Nash equilibrium for the driving strategies. To our knowledge, they are the only models that can do strategic planning. It means that they can explicitly account for the influence the self-driven vehicle has on the other agents' behaviors. However, these models are trained within simulations and are extremely challenging to transfer to the real-world. In [Gon+17; Xu+19], inverse reinforcement learning methods learn the cost functions optimized by the drivers using real data. Thus, minimizing these cost functions should mimic the human behavior as it is represented in the dataset. However, it tends to overfit and does not generalize well to new situations.

4.3 Maneuver Classification

At the time of the survey [LVL14], the maneuver-based models were prevalent because they allow long-term forecasts (a few seconds) without the need to define the precise trajectory. It is also a way to force the forecasting model to consider several hypotheses. The maneuver definitions can also match predefined use-case scenarios. This is appreciated in the industry that often measures progress on the number of use-cases that can be solved. With this approach, the vehicle motions are seen as a succession of maneuvers. Then, the motion prediction task becomes an early identification of the driver's intentions. An intention is described using a discrete dictionary of maneuvers.

Maneuver classification in the literature - A motion forecasting solution predicting trajectories based on maneuver identification is developed by [Hou14] and [DT18]. In [Hou14], trajectories are computed as an optimal path that achieves a maneuver. The optimality criterion developed in [Hou14] is hand-defined heuristics that sets a trade-off between lateral acceleration minimization and the time needed to accomplish the maneuver. In [Wan+18], a hierarchical maneuver dictionary is used with a first layer deciding between *Salient situation* and *Lane Keeping*. If the situation is Salient, the maneuver may be *Lane Keeping, Change Preparing, Left Changing*, or *Right Changing*.

Semantic forecast - A semantic forecast is proposed in [SSS17]. It uses maneuver definitions that relate to the driving scene, such as following another agent. However, we did not find any proof of concept using semantic maneuvers.

Prototype trajectories - A maneuver execution often follows a specific motion pattern. Thus, it is also possible to define the maneuvers as prototype trajectory. A dictionary of prototype trajectories defines the set of known maneuvers. This method as well as the maneuver classification described in the previous paragraph are both called maneuver-based in the survey [LVL14]. This prototype trajectory approach is made by [CH06] with two steps. First, a set of latent trajectory segments and their variance are learned from a dataset. Then a model forecasting the future trajectory as a sequence of prototype trajectory segments is learned.

Maneuver classifiers - The methods for maneuver prediction involve either support vector machines, hidden Markov models, or neural networks. These three classification approaches learn a set of parameters on a dataset to define a classification function. Once the parameters are learned, the models predict the next maneuver or the distribution over possible future maneuvers knowing the vehicle current state. An optimization method determines the parameters by finding the ones that minimize a given loss function on a dataset. In most cases, the parameters reaching the global minimum cannot be found, but a satisfactory local minimum is obtained. The neural networks form a set of functions that is well adapted to this optimization process. They quickly became the go-to method for the classification task. We introduce neural networks in the next chapter.

4.4 Statistical Forecast

A statistical forecast expresses the probability distribution of the future driving sequence knowing the context, including the past observations. However, this is not a known distribution, and it is challenging to describe.

HMMs - The Kalman filter that we used in chapter 1 considers a Gaussian distribution over an action that modifies the current state. It is a simple statistical forecast that can be described as a Hidden Markov Model (HMM) with a continuous state. An HMM is a statistical model that assumes the existence of a hidden state. In the case of the constant velocity Kalman filter, the observed state is the position and the hidden state is the velocity. The whole state is modeled as a Markov process. This means that the next state only depends on the previous state. Instead of directly considering a sequence of positions, HMMs have been used at a coarser scale in [OP00] where the state describes the maneuver intention instead of the vehicle kinematic state. It also couples the ego HMM instance with HMM instances attached to other road users to account for the influence of other vehicles on the ego vehicle. This model can predict the next maneuver using the vehicle state observations (steering angle, acceleration, brake, and gear position) and combinations of other observations: driver gaze and lane positioning. Passing, starting, and stopping seems to be well recognized with high accuracy. These maneuvers are predicted on average more than one second in advance. However, the lateral maneuvers, such as lane changes and turns, are not very well anticipated. In [AK07], a two-layer HMM is used for vehicle maneuver prediction. The first layer identifies "gestems" which are intermediary representations between the raw observations and the high-level maneuver representation of the second layer. This layered architecture is shown to be robust to noise and to miss-classifications of gestems.

Bayesian networks - In [SWA14], an expert Bayesian network is defined to describe not only the maneuver intention but the whole probability density for the next actions knowing the observations. Expert Bayesian networks are oriented acyclic graphs with nodes representing random variables and oriented edges representing dependencies. These networks can be efficiently learned to represent the probability density of the data under statistical prior assumptions. These assumptions are characterized by hand-defined distributions with learnable parameters and fixed dependencies between them.

Neural networks - In the recent literature, neural networks have become the primary long-term motion forecasting method. [Ort+11] used a "multi-layer perceptron" for behavior classification. This is the old name for fully connected neural networks. In the more recent work [AL17], a model forecasts the ego vehicle longitudinal velocity and lateral position up to 10 seconds in the future using the LSTM [HS97] neural network architecture. For the past three years, all the State-Of-The-Art (SOTA) results on the motion forecasting datasets use neural networks. Neural networks do not need a layer-wise prior distribution or the Markov assumption. They offer more flexibility than the HMM while being also compatible with a hierarchical representation. Thus, neural networks can learn dependencies over longer sequences and generalize well to new situations thanks to the abstractions learned in the depth of their non-linear layers.

Three learning methods making use of neural networks are employed to identify predictive models: reinforcement learning, supervised learning, and unsupervised learning. Direct reinforcement learning methods are only used in simulated environments. Inverse reinforcement learning methods are used in forecasting using real data, but they do not generalize well. The supervised and unsupervised learning methods have become the favored approaches in the motion forecasting literature. The distinction between supervised and unsupervised is not clear in the forecasting task because different parts of the same data sequence are used as input and supervision. Since their regain of attraction around 2013, the use of neural networks in forecasting applications has dramatically augmented.

Our goal is to find methods to produce a motion forecasting model that would eventually be used as inspiration for the real-world applications. Thus, the best performance results as defined in chapter 3 should be obtained with a reasonable computation time and a good generalization to various road scene situations. The most promising methods to meet these objectives involve neural networks. The forecasting application does not necessarily require labeled data, and open datasets are freely available. This makes the neural network models both promising and accessible. For these reasons, we chose to orient this work toward the applications of neural networks for trajectory forecasting. The next chapter builds a link between the machine learning methods and the constant velocity model from chapter 1 by applying the optimization procedure that is ubiquitous in neural networks learning. We use the gradient descent algorithm to find better parameters for the constant velocity model.

Chapter 5

A First Gradient Descent Application

The evaluation criteria and their interpretation made in the previous chapter is used as a validation procedure. Based on this evaluation, we compare different models and different versions of the same model. In this short chapter, we use the gradient descent method to find a set of parameters that improves the results from the constant velocity model produced in chapter 1. This is a widely used and generic iterative optimization process to find the parameters that minimize a scalar loss. The gradient descent method is a core component in most of the modern machine learning applications. The reader looking for an introduction to the gradient descent method and specifically in the type of applications made in the rest of this work should refer to sections 4.3 and 6.2 of [GBC16].

5.1 The Gradient Descent Method

Settings - We consider a model function f_θ depending on the set of parameters θ. The function produces an output $\hat{\mathbf{y}} \in \mathbb{R}^{\dim_y}$ for each input $\mathbf{x} \in \mathbb{R}^{\dim_x}$: $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$. A loss function $\mathcal{L}$ produces a scalar value for a given pair $(\mathbf{y}, \hat{\mathbf{y}})$. This value is low if $\hat{\mathbf{y}}$ is a good approximation of $\mathbf{y}$ and high otherwise. The loss may also depend on a set of parameters θ and on intermediary outputs to regulate their values.

Parameter update - The gradient of the loss with respect to each parameter in the set θ is used to update its value. Thus, either the gradient is well defined, or a substitute parameter update rule must be given. The gradient with respect to the i^{th} parameter is noted $\frac{\partial \mathcal{L}(\mathbf{y}, f_\theta(\mathbf{x}))}{\partial \theta_i}$. This gradient is the loss variation rate of the parameter θ_i around its current value. In the case of a substitute definition of the gradient, it should be chosen to approximate this variation rate. If the variation of the loss is known for all variations of the parameters, finding the parameters that minimize it seems straight forward. At each iteration, the gradient descent algorithm computes the gradient and slightly moves the parameters in the direction that lowers the loss.

Limitations - There are several problems with the gradient descent algorithm:

- The loss function must be smooth enough with respect to the parameters for a finite variation of the parameters to lower the loss value.

  ↪ It is not guarantied to lower the loss in the general case even with differentiable functions.

- To be computed, the loss would need to be evaluated for all values of $\mathbf{x}$, knowing all expected $\mathbf{y}$.

  ↪ The gradient can only be approximated.

- The gradient only gives a local indication in the parameter space.

  ↪ Only a local minimum of the loss can be obtained with this method.

The machine learning research community is actively studying each limitation. Even without guaranty that the optimal solution is found, it is possible to verify that a given solution is satisfactory using the evaluation criteria defined in chapter 3.

In the next section, we apply the gradient descent algorithm to optimize the parameters of the constant velocity model defined in chapter 1.

5.2 Application on the Constant Velocity Model

As described in section 1.3, in each tested sequence, the forecast position sequences and the estimated error covariance are compared with the future observations. We chose to minimize the average Negative Log-Likelihood (NLL) to improve both the covariance estimation and accuracy.

The Kalman filter parameters described in section 1.2.3 are used to initialize the parameters: The process acceleration covariance with the parameters $(\tilde{q}_x, \tilde{q}_y)$, and the measurement noise covariance with the parameters $(\tilde{r}_x, \tilde{r}_y)$. The process noise covariance matrix Q is now written as follow:

$$Q_s = \begin{pmatrix} \frac{dt^2}{2}\alpha_x q_x & 0 \\ dt\alpha_{v_x} q_x & 0 \\ 0 & \frac{dt^2}{2}\alpha_y q_y \\ 0 & dt\alpha_{v_y} q_y \end{pmatrix}$$
$$Q = Q_s Q_s^T$$

With q_x, q_y and $\alpha_x, \alpha_{v_x}, \alpha_y, \alpha_{v_y}$ the parameters to optimize in the definition of Q. q_x and q_y stand for the acceleration standard deviation and are initialized with the values computed in chapter 1. The different noise models such as continuous time Gaussian noise, or the one we considered, discrete time Gaussian noise give different coefficients in the matrix Q. We compute it in the case of continuous time Gaussian noise in appendix A.1. Introducing $\alpha_x, \alpha_{v_x}, \alpha_y, \alpha_{v_y}$ allows the optimization to relax this hypothesis made in our noise model. They are initialized with value 1 to match the hand-defined discrete time Gaussian noise model. The observation noise R is defined with three parameters r_{11}, r_{22}, r_{12} as:

$$R = \begin{pmatrix} r_{11}^2 & \tanh(r_{12}) r_{11} r_{22} \\ \tanh(r_{12}) r_{11} r_{22} & r_{22}^2 \end{pmatrix}$$

The initial position and velocity are still computed from the two first observations. However, the initial state covariance defined with $\sigma_x, \sigma_{v_x}, \sigma_y, \sigma_{v_y}$ and approximated from the dataset can also be optimized. Thus, the parameters to be optimized are:

$$\text{args} = ((q_x, q_y), (\alpha_x, \alpha_{v_x}, \alpha_y, \alpha_{v_y}), (r_{11}, r_{22}, r_{12}), (\sigma_x, \sigma_{v_x}, \sigma_y, \sigma_{v_y}) \in (\mathbb{R}_+^2, \mathbb{R}^4, \mathbb{R}_+^3, \mathbb{R}_+^4)$$

The past perceptions are written $\tilde{Z}_h = \{\tilde{z}_k\}_{k=-N_H,0}$. The forecast sequence computed using the algorithm 1 is written $\text{Kalman}_{\text{pred}}(\tilde{Z}_h, \text{args})$. The future observations are noted $\tilde{Z}_f = \{\tilde{z}_k\}_{k=1,N_F}$. The minimization performed to learn the parameters is:

$$\underset{\text{args}}{\text{argmin}} \Big(\text{loss}(\text{Kalman}_{\text{pred}}(\tilde{Z}_h, \text{args}), \tilde{Z}_f) \Big)$$

We used the average of the NLL over time and over a subset of samples for the loss function. A gradient descent algorithm is used to optimize the parameters. The model is implemented with the Pytorch library. The parameters args are set with the computed values from chapter 1, then fitted to the training set using the Stochastic Gradient Descent (SGD) optimizer, or the Adam optimizer [KW14]. Our code for data preprocessing and model training is accessible on Github[1].

Table 5.1: Comparison of the constant velocity model parameters before and after the optimization. They are associated with a discretized time step $dt = 0.2$s.

Parameter	σ_x	σ_y	σ_{v_x}	σ_{v_y}
hand-defined	0.04	0.57	0.20	4.01
Optimized	0.58	11.1	0.13	13.0

Parameter	r_{11}	r_{22}	r_{12}	$\alpha_x q_x$	$\alpha_y q_y$	$\alpha_{v_x} q_x$	$\alpha_{v_y} q_y$
hand-defined	0.01	0.01	0	0.93	2.82	0.93	2.82
Optimized	0	0	0	-1.64	0.62	-0.04	-2.42

Interpretation made impossible - Table 5.1 presents the Kalman parameters before and after the optimization procedure. The parameters αq are very different from the initialization and even take negative values. With such values, we cannot interpret the model with kinematics. The positive $\alpha_y q_y$ value and negative $\alpha_{v_y} q_y$ value would mean that a positive y acceleration would decrease the position variance due to the velocity while increasing the velocity variance. This seems absurd because an uncertainty about the velocity over time should cause an uncertainty about the position. We could enforce positive values for the α parameter and add other constraints such as $\sigma_{v_x v_x} = \frac{2\sigma_{xx}}{dt^2}$ to keep the physical properties of the model. However, we show in the next section that the model using the unconstrained parameters leads to better results than those of the model using the parameters defined in chapter 1 and we do not aim to find the best physical parameter but to get the best results in the evaluation.

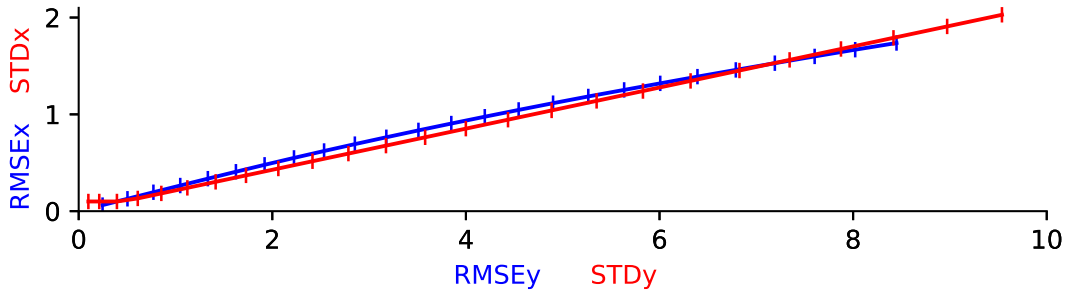
[1] `https://github.com/jmercat/KalmanBaseline`

5.3 Optimized Model Evaluation and Comparison

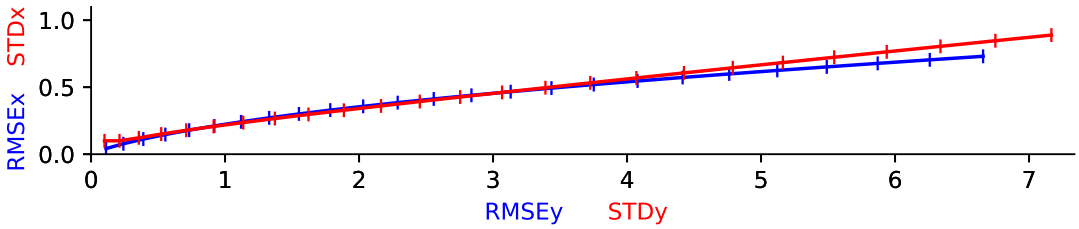
Some published work produced similar models using the NGSIM dataset. Thus, we can make a meaningful comparison with the results from articles [DT18; Ju+19; Mes+19; Xu+20].

Bugs in NLL function - However, it is unclear if the NLL computations are made using the correct definition. The NGSIM dataset uses feet to record the positions, and we have changed this to meters. This factor could produce an offset of -2.4 on the NLL value. In our result comparison, the equation (3.11) using the NLL formulation from equation (3.9) is used and is reported using metric inputs. The NLL result from [DT18] might be computed with a bug in their code. However, we reporte it because the authors published their data processing function, making it reproducible. Their published code is available on github[2]. However, the constant velocity model is not a part of their published code, and we cannot reproduce their baseline. We have only used their released preprocessing function, so our results are computed using exactly the same training and testing datasets.

Results from the computation of the performance indicators over the test set are reported in table 5.2. The RMSE and Final Displacement Error (FDE) values are similar for our optimized model and the ones from the literature. They are much improved over our previous unoptimized model. However, we obtain different NLL values (even if we considered the -2.4 offset). This means that all the forecasts make approximately the same average error but have different performance for the covariance estimation that only depends on the identified parameters.



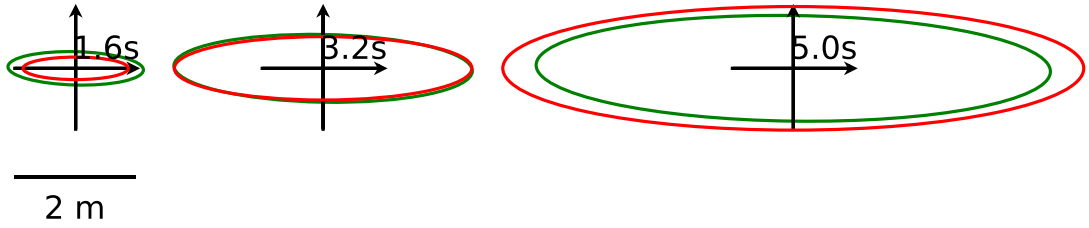
(a) Hand-defined parameters.



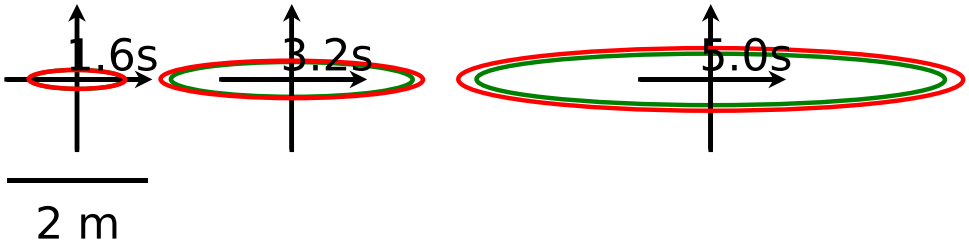
(b) Optimized parameters.

Figure 5.1: Parametric curves of the RMSE and the model covariance in meters for the constant velocity predictor with hand-defined and optimized parameters from 0 to 5s in the future at 5Hz.

[2] https://github.com/nachiket92/conv-social-pooling



(a) hand-defined parameters.



(b) Optimized parameters.

Figure 5.2: Unit dispersion ellipse representation of the test set error covariance in green and the covariance predicted by the constant velocity model in red with hand-defined (a) parameters and with optimized parameters (b).

The representation of the covariance in figure 5.2 shows a good match between estimated error dispersion ellipses and the empirical error dispersion ellipses with only a slight overestimation. The 39% probability domains that unit dispersion ellipses define are much smaller with the optimized parameters than with the hand-defined parameters. This shows that our optimization process has both reduced the average error and the quality of the covariance estimation of our model. The low NLL values in table 5.2 show that the individual error covariance estimations are also satisfactory and where much improved over the non-optimal model.

Table 5.2: Comparison of RMSE, NLL and FDE results for **constant velocity models** with the NGSIM test set preprocessed with the code published by [DT18] (In [Xu+20] a different set from NGSIM and 10Hz observations instead of 5Hz are used). Our results are reported as "Kalman with learned parameters." FDE and miss rate at two meters are not given in the comparable work.

Time horizon		1s	2s	3s	4s	5s
RMSE	From [Xu+20]	0.48	1.50	2.91	4.72	NA
	From [DT18]	0.73	1.78	3.13	4.78	6.68
	Kalman with set parameters	1.37	2.92	4.65	6.57	8.71
	Kalman with learned parameters	0.75	1.81	3.16	4.80	6.69
NLL	From [DT18]	3.72	5.37	6.40	7.16	7.76
	Kalman with set parameters	2.98	3.76	4.53	5.18	5.72
	Kalman with learned parameters	0.81	2.31	3.22	3.91	4.46
FDE	Kalman with set parameters	0.65	1.61	2.80	4.19	5.78
	Kalman with learned parameters	0.46	1.24	2.27	3.53	4.99
MR	Kalman with set parameters	0.05	0.26	0.49	0.65	0.75
	Kalman with learned parameters	0.02	0.20	0.44	0.61	0.71

With these improvements, the model might be reliable for up to 2 seconds in the future on the highway but is still very insufficient for longer-term forecasts and in a more complex context.

In this chapter, we introduced the gradient descent algorithm that is heavily used in the following chapters. We applied it to improve the constant velocity model by simply optimizing its parameters. The results could be compared with those of similar models from the literature. Different evaluation criteria showed that our optimized model is acceptable for forecasts up to 1 or 2 seconds but is clearly not satisfactory for forecasts extending from 3 to 5 seconds in the future with FDEs greater than two meters and large miss rates above 30%.

The uncertainty estimation of the model for its forecast is reliable, but it needs a careful setting of its parameters. To build a forecasting model that makes forecasts extending to longer horizons with an acceptable error, less restrictive hypotheses allowing control actions, such as accelerating and turning, must be made. These actions might follow common patterns that can be forecasted, but they surely also depend on human driver behaviors and other observations of the road scene.

The method overview made in the previous chapter has convinced us to pursue our research with neural networks. Neural networks are almost always trained with the gradient descent algorithm. We used this algorithm to optimize the parameters of our first motion forecasting model and we were able to improve its results. In the next chapter, we define the usual blocks used in the neural network architectures and apply the gradient descent algorithm to train them to learn the task of motion forecasting.

Chapter 6

Forecasting Neural Networks and Applications

In the rest of this work, the mathematical properties and learning capacities of neural networks are used to build road scene forecasting models. The neural network vocabulary is influenced by biological terms and anthropomorphic interpretations. However, the use of this vocabulary does not mean that our approach is in any way bio-mimetic. In this chapter, we introduce some of the standard tools that we will use in the next chapters. The book [GBC16] presents all these concepts and many more. In its sixth chapter, it introduces the neural networks in detail. The readers that desire an in-depth approach should refer to it.

6.1 Neural Networks

Neural networks form a class of multi-dimensional functions that associate an input vector $\mathbf{x} \in \mathbb{R}^n$ to an output vector $\hat{\mathbf{y}} \in \mathbb{R}^m$. In their classical form, they are composed of a succession of layers. The network depth is its number of layers, whereas the width is the number of neurons or units in each layer.

Each layer computes a matrix multiplication, addition of a constant vector, and application of a function called activation. The matrix coefficients are called the weights. Each row of the matrix represents the weights of one artificial neuron or unit. The corresponding constant value that is added is the bias of that unit.

The activation function is *most of the time* a real non-linear function applied to each element of the vector. However, the only restriction to the activation function definition is its compatibility with the learning algorithm. We note with an index (l) the parameters and output of the l^{th} layer. With this notation, $\mathbf{x}^{(l)}$ is the output feature vector of the l^{th} layer. The output $\mathbf{x}^{(l)}$ of the l^{th} layer with activation function $f^{(l)}$, weights $W^{(l)}$, and bias $b^{(l)}$ is computed as follows:

$$\mathbf{x}^{(l)} = f^{(l)}(W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)}) \tag{6.1}$$

For the first layer, the input vector is used thus $\mathbf{x}^{(0)} = \mathbf{x}$. Each subsequent layer is computed in the same fashion using the output of the previous layer. The output value of the last layer is noted $\hat{\mathbf{y}}$. Its value is deterministic and is determined by the input, the neural network architecture, and the weight values.

Architecture - The architecture is the description of the layers, their number, size, type, the inputs and outputs, and a set of constraints and variations modifying to the network. Some variations cannot be expressed in the classical form of equation 6.1. An example of such a variation from the usual layer definition is the following, using $\odot$ as the element-wise multiplication:

$$\begin{aligned} \mathbf{x}_a^{(l)} &= f_a^{(l)}(W_a^{(l)}\mathbf{x}^{(l-1)} + b_a^{(l)}) \\ \mathbf{x}_b^{(l)} &= f_b^{(l)}(W_a^{(l)}\mathbf{x}^{(l-1)} + b_b^{(l)}) \\ \mathbf{x}^{(l)} &= \mathbf{x}_a^{(l)} \odot \mathbf{x}_b^{(l)} \end{aligned} \tag{6.2}$$

This kind of variation is used to build specific mechanisms such as gating as in the LSTM introduced in the next section. Some constraints, such as weight sharing, can be expressed in the usual fully connected form, but their expression is often made more explicit and their computation faster by writing them differently.

Weight sharing - Weight sharing forces some weights and biases of the network to be equal. It is the same thing to share weights or to use the same layer several times. We use the expression "weight sharing" for either implementation. The widely used convolutional layer is an example of such a weight sharing constraint that forces a very sparse weight matrix and bias. These shared weights are part of the same layer; thus, the weight-sharing happens within the width. It is also possible to share weights in depth by using the same weights in different layers or even using the same layer twice.

Expressivity - Once the architecture is fixed, the function is defined by the values of its parameters. The same neural network architecture may perform very different computations with different parameters. The expressivity of a neural network is the range of functions that it can approximate. This notion is studied in [Rag+17]. The whole point of neural networks is that they are well suited for *efficient methods to find the parameters* that makes them behave approximately in the desired way expressed through examples. Their expressivity ensures that this approximation exists.

Supervised learning - Finding a set of parameters that makes the neural network behave similarly to a set of examples of $\mathbf{x}$ and $\mathbf{y}$ is called supervised learning. A *loss function* is used to evaluate the difference between the dataset examples $(\mathbf{x}, \mathbf{y})$ and the outputs $(\mathbf{x}, \hat{\mathbf{y}} \stackrel{\text{def}}{=} h_\theta(\mathbf{x}))$ of the neural network instance h_θ. The loss function returns a scalar and is usually defined on $\mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ because it compares $\mathbf{y} \in \mathbb{R}^m$ and $\hat{\mathbf{y}} \in \mathbb{R}^m$. For regularisation, the loss often considers the weights θ. It may also use the input $\mathbf{x}$, and intermediate values. An optimization algorithm called back-propagation of the gradient is used as the learning method in almost all the neural network applications. It is a procedure that iteratively modifies the neural network parameters in a way that lowers the given loss function for a set of input and output examples. At each iteration, it computes the gradient of the loss with respect to each parameter. With the chain rule differentiation, it can make a very efficient use of the neural network layered expression.

The learning process is the minimization of a loss for the set of parameters θ:

$$\underset{\theta}{\operatorname{argmin}} \left(\text{loss}\left(\mathbf{x}, \theta, \mathbf{y}, h_\theta(\mathbf{x})\right)\right) \tag{6.3}$$

Learning is not optimizing - However, learning a neural network is not precisely an optimization procedure. The first reason is that the learning process only aims at finding a local minimum of the loss because the considered functions are highly non-linear. Another reason is that learning is often used in cases where the goal to be achieved is not well defined. This means that it is often possible to find a function behaving in an undesired way that produces a lower loss than another behaving in a preferred way. This and other differences between optimization and learning are discussed in the chapter 8 of [GBC16].

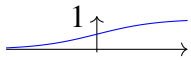
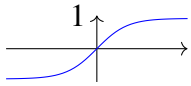
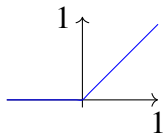
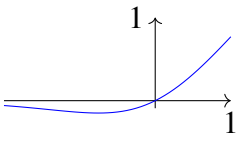
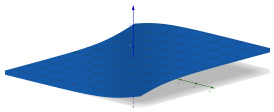
6.2 Neural Network Architectures for Time Series

In this section, we define the neural network layers that are used in the next chapters. They can all be applied to the time-series formed with vehicle state sequences. We write the most common activation functions before defining three common types of architectures needed in our applications. They are assembled to form larger networks.

6.2.1 Activation Functions

There are a few very common activation functions that are used in almost every architecture: sigmoid, tanh, relu, softmax. Their Wikipedia page[1] defines many activation functions. We report the ones used in this work in table 6.1.

Table 6.1: Neural network activation functions used in this work.

Name	Plot	Equation	Derivative
Sigmoid		$\frac{1}{1+e^{-x}}$	$\sigma'(x) = \sigma(x)(1-\sigma(x))$
Tanh		$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$\tanh'(x) = 1-\tanh^2(x)$
ReLU		$\max(0,x)$	$\mathrm{ReLU}'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$
GELU		$\alpha = 1.702$ $\approx x\sigma(\alpha x)$	$\mathrm{GELU}'(x) \approx$ $\sigma(\alpha x)(\alpha x(1-\sigma(\alpha x))+1)$
Softmax		$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}}$	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$

[1] https://en.wikipedia.org/wiki/Activation_function

Choosing an activation - The sigmoid function is the one that was historically used because it somewhat mimics biological neurons activations, and the expression of its derivative allows very efficient learning. The tanh activation is used in the same way but produces a symmetric output centered around 0. The Rectified Linear Unit (ReLU) activation is the most common one in recent work and has become the default activation function. Its derivative is the simplest, and its non-linearity is sufficient for the neural network to be expressive. The Gaussian Error Linear Unit (GELU) [HG16] is similar to the ReLU activation with a soft bump around -1. The function is increasing in either directions, causing a regulating effect during the learning process. The softmax activation is used differently. It is the only one in table 6.1 that is applied on the whole input vector and not separately on each element. As its name suggests, it acts as a smoothed argmax function. Its output is normalized, such that it sums up to 1. Therefore, it is often interpreted as a probability distribution over a number of classes. It is used as the last layer activation function for classifications and can also be found in intermediate layers in attention mechanisms.

6.2.2 Fully Connected

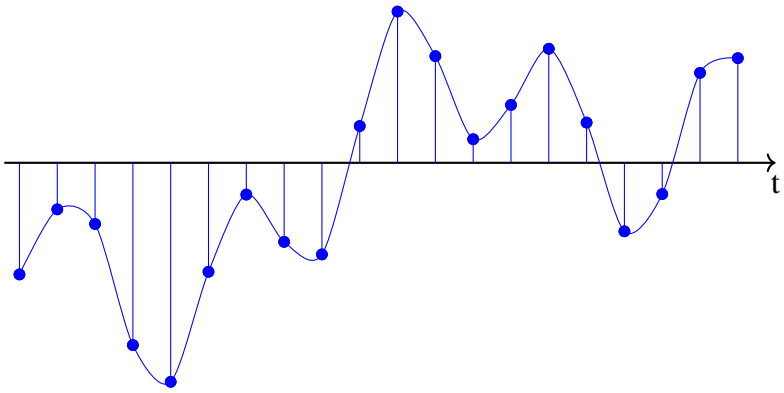


Figure 6.1: A time series used as an illustrative example of input.

The fully connected layer is the most basic piece of neural network architectures. It exactly follows the classic layer equation given in the introduction without any additional constraint: $\mathbf{x}^{(l)} = f^{(l)}(W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)})$. All the parameters of the matrix W_l are used. Thus, it is a dense matrix and the fully connected layer may also be called dense layer. Even if it is affine and with an activation function, it can also be called linear layer.

Dimensions - $W^{(l)}$ must have the same number of columns N_j as the dimension of $\mathbf{x}^{(l-1)}$, but it may be defined with any number of rows N_i. Therefore, the output vector $\mathbf{x}^{(l)}$ is of size N_i. Expanding and reducing the feature vector dimensions can lead to interesting properties such as the information bottleneck discussed in [TPB00] and used in applications called auto-encoders [RHW86; HS06]. For the illustrative input represented in figure 6.1, the sequence is composed of 20 points; thus, each unit (a unit is one row in the weight matrix and one element in the bias vector) of the first layer must count 20 parameters plus the bias.

Too many parameters - This layer definition is the simplest, but a few fully connected layers with ReLU activation functions make a very expressive neural network. Its dense structure contains many parameters that are difficult to train for some specific

tasks. It is often used in the last layers of pre-trained convolutional networks. In such architectures, the fully connected layers represent a small proportion of the number of layers but a large proportion of the number of parameters. Thus, we should look for restrictions about the task such as invariances and symmetries and adapt the architecture to avoid the fully connected layer. For example, the convolutional layers offer a very efficient solution when the data is invariant by local translation in some of its dimensions such as the time in a time-series.

6.2.3 Convolutional

Convolutional Neural Networks (CNNs) are introduced in chapter 9 of [GBC16]. A convolutional layer may be seen as a sparse fully-connected layer with shared weights. However, it is simpler to see it as a smaller fully-connected layer computed multiple times on different sub-parts of the input. The classical use of CNNs is two-dimensional image processing. In this work, we only use and present the one-dimensional CNNs for time series.

There are two hyper-parameters to define a convolutional unit: the size of the sub-parts called the kernel size and the number of steps separating two sub-sequences called the stride. In figure 6.2, the input sequence is decomposed in sub-parts of size 3 separated by a stride of one. This layer is composed of units counting 3 weights and one bias.

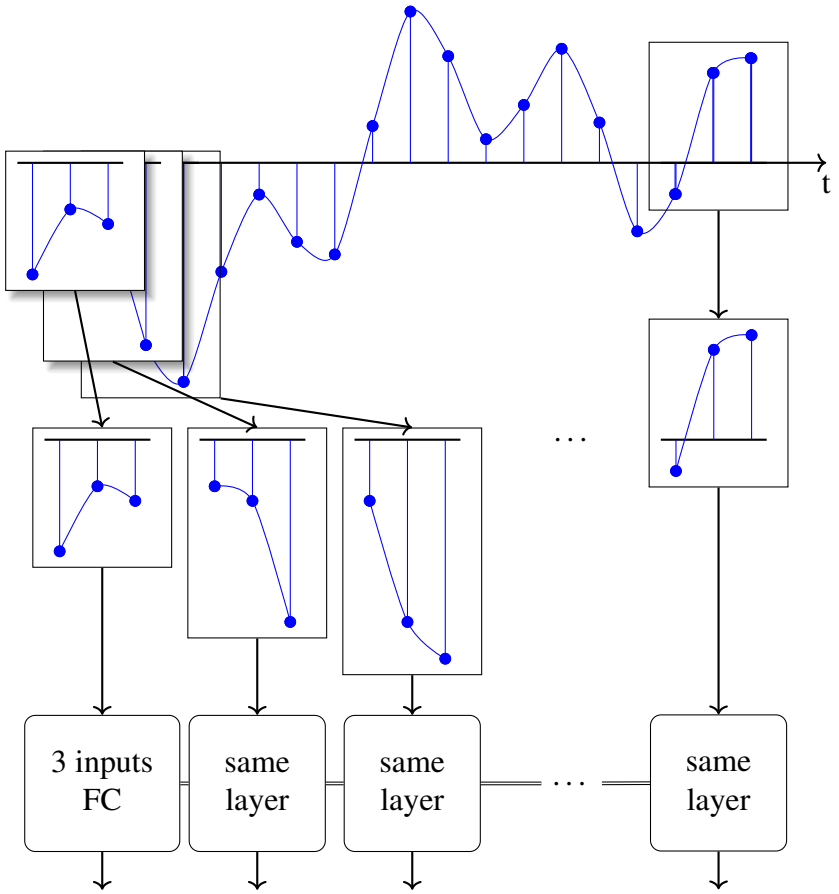


Figure 6.2: Illustration of the convolutional layer seen as a fully connected layer duplicated and applied on parts of the input.

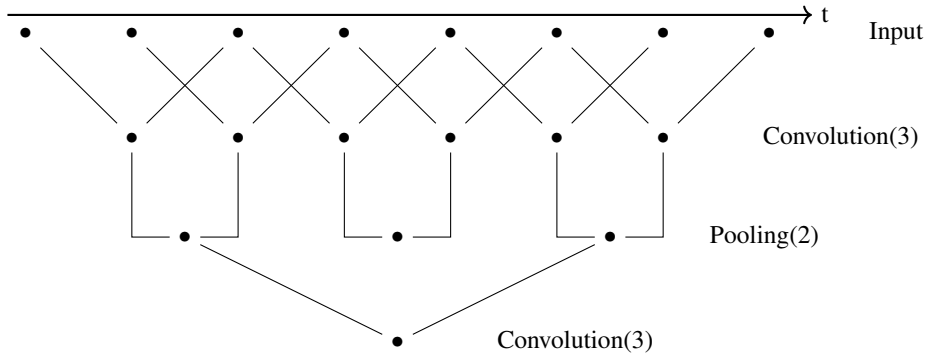


Figure 6.3: Representation of the receptive field of a convolutional neural network using two convolutional layers and one pooling layer.

Advantages over fully connected layers - A fully connected layer counting as many units requires many more weights to process the whole sequence. Moreover, the convolutional layer can be computed for different sequence lengths, whereas a fully connected layer requires a fixed-length input. Of course, the reduction of the number of parameters also reduces the neural network expressivity because the convolution makes the same computations at every time step. However, in some applications, this is a desired behavior. If the input sequence is statistically invariant in time, a fully connected layer could over-fit longer time dependencies. In those cases, the convolutional layer is preventing this source of over-fitting by forcing time invariance in its computation. Moreover, each part of the input sequence is used to train the same weights. This helps the learning process produce meaningful filters.

Receptive field and pooling - An essential aspect of convolutional neural networks is their receptive field. It is the maximum length of the patterns that can be recognized. With only one layer, the receptive field is the size of the kernel. However, pooling mechanisms between multiple convolutional layers can extend the receptive field of the multi-layer neural networks. Pooling mechanisms are explained in section 9.3 of [GBC16]. The pooling reduces the time dimension of a feature sequence. One way to do this is to increase the stride of the convolution. Using a stride 2 boils down to computing every other output of the convolutional layer. Another common way is to use a max-pooling or an average-pooling layer. It computes the maximum, respectively the average, of the feature over a sub-sequence of features to return only one value. These layers usually use a stride of the same size than the sub-sequence length. The most common choice for this size is 2, which halves the input time sequence size. An architecture using a size 3 kernel with stride 1 convolution, followed by a size 2 pooling and a second convolution of size 3, stride 1 has a receptive field of 8 time steps as illustrated by figure 6.3. The size of the receptive field is the maximum length of pattern that can be recognized.

Time invariance - The max-pooling layer discards a part of the input at different times within the sub-sequence. Thus, the exact time of occurrence of the pooled maximum feature is lost. This may be a desired feature to obtain a local time invariance.

6.2.4 Recurrent

The recurrent architecture allows low-frequency pattern recognition with variable input sequence size and much fewer parameters than the fully-connected one. Recurrent Neural Networks (RNNs) are the subject of chapter 10 of [GBC16]. Two main differences characterize this architecture. Firstly, the weights of the network are shared in the depth (instead of the width as in CNNs). Secondly, the input observations are not all fed at once to the first layer but one by one to each layer along the depth.

Variable depth - This implies that a recurrent architecture does not have a fixed depth. The figure 6.4 shows a dynamic depth equal to the length of the input sequence. Each cell is a new layer of the network (with the same parameters), but RNNs are improperly designated and treated as unique layers called recurrent layers.

Recurrent cell - The recurrent cell considers two inputs: the new observation and the previous output. Feeding the previous output to the current cell allows it to keep a memory of the past computations. An initial recurrent input must be defined with the first input. The function computed by the cell must be differentiable for its parameters to be trained. A neural network, fully connected or convolutional, is the usual choice for the cell definition. The recurrent cell is a function $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$. At each time step, its inputs are both a vector of $\mathbb{R}^m$ and the recurrent input of $\mathbb{R}^n$.

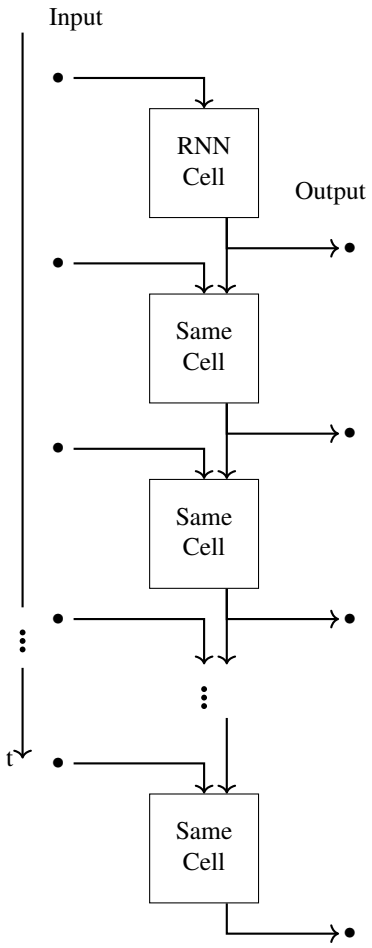


Figure 6.4: Illustration of a Recurrent neural network (RNN)

Gradient problems - If a sequence of $k+1$ elements is fed to the recurrent layer, the last output is a composition of $k+1$ times the same function:

$$\text{output}^{(k)} = f(\text{input}^{(k)}, f(\text{input}^{(k-1)}, f(\text{input}^{(k-2)}, \dots f(\text{input}^{(0)}, \text{init}) \dots)))$$

The network is trained with back-propagation of the gradient. Thus, the gradient of this composition of the same function must be computed. With RNN, the depth and time are linked, and this is called back-propagation through time. A common problem arising from the chain rule derivation applied to a composition of the same function is the exponential evolution of the gradient through the layers. The gradients larger than 1 tend to explode, and gradients lower than 1 tend to vanish. This makes it challenging to learn RNNs with long time dependencies because the learning process is made unstable by the gradients vanishing and exploding.

LSTM - One of the most used recurrent cell architecture is the Long Short-Term Memory (LSTM) [HS97]. The LSTM architecture is introduced in section 10.10 of [GBC16]. The recurrent input is considered as short-term memory because it is contained in the feature vector as opposed to the long term memory that would be present in the learned weights of the network. The LSTM architecture allows the RNN to retain this short-term memory for many steps. Its computation is written in equation (6.4). The symbol "$\odot$" denotes the elementwise multiplication. The sigmoid σ, and $\tanh$ are applied elementwise. The LSTM cell depends on the weight matrices $W_i, W_f, W_o, W_g, U_i, U_f, U_o, U_g$ and bias vectors b_i, b_f, b_g, b_o.

$$\begin{aligned}
\mathbf{f}^{(k)} &= \sigma\big(U_f\mathbf{x}_{k-1} + W_f\mathbf{h}^{(k-1)} + b_f\big) \\
\mathbf{i}^{(k)} &= \sigma\big(U_i\mathbf{x}_{k-1} + W_i\mathbf{h}^{(k-1)} + b_i\big) \\
\mathbf{g}^{(k)} &= \tanh\big(U_g\mathbf{x}_{k-1} + W_g\mathbf{h}^{(k-1)} + b_g\big) \\
\mathbf{o}^{(k)} &= \sigma\big(U_o\mathbf{x}_{k-1} + W_o\mathbf{h}^{(k-1)} + b_o\big) \\
\mathbf{c}^{(k)} &= \mathbf{f}^{(k)} \odot \mathbf{c}^{(k-1)} + \mathbf{i}^{(k)} \odot \mathbf{g}^{(k)} \\
\mathbf{h}^{(k)} &= \mathbf{o}^{(k)} \odot \tanh(\mathbf{c}^{(k)})
\end{aligned} \tag{6.4}$$

Figure 6.5 graphically represents the computation of the equations (6.4). The recurrent input/output is split in two vectors, $\mathbf{h}$ and $\mathbf{c}$. $\mathbf{c}$ is purely a recurrent input/output whereas $\mathbf{h}$ is both the cell output and the recurrent input/output. These computations are interpreted as a gating and updating mechanism of the memory $\mathbf{c}$. The sigmoid function σ produces values between 0 and 1 that are used as a mask. The first mask is the forget gate $\mathbf{f}$. Its elementwise product with $\mathbf{c}$ allows to "forget" some elements by multiplying them with a value close to 0. Then, the memory $\mathbf{c}$ is updated. The update vector $\mathbf{g}$ is itself masked with $\mathbf{i}$ before being added to the memory. Finally, the output $\mathbf{h}$ is a copy of the memory passed through the $\tanh$ activation and masked with the output gate $\mathbf{o}$. This gating architecture allows the back-propagation of the gradient through time to remain meaningful on long sequences. A gate keeps a value by multiplying it by 1 and thus does not affect the gradient. On the contrary, a gate erasing memory by multiplying elements by 0 produces a null gradient. The forgotten value does not affect the result; thus, the gradient of the loss after that point is independent of the forgotten value. This gating mechanism allows the gradient to flow through time almost unaffected until the memory update that changed the memory value is reached. Therefore, this architecture allows the RNN to be trained on long sequences.

In this section, we presented the common layers, also called vanilla layers: fully connected, convolutional, and recurrent. They are the building blocks of the neural network architectures that we use in the next chapters. The next two sections introduce their use for supervised learning in the most straightforward settings.

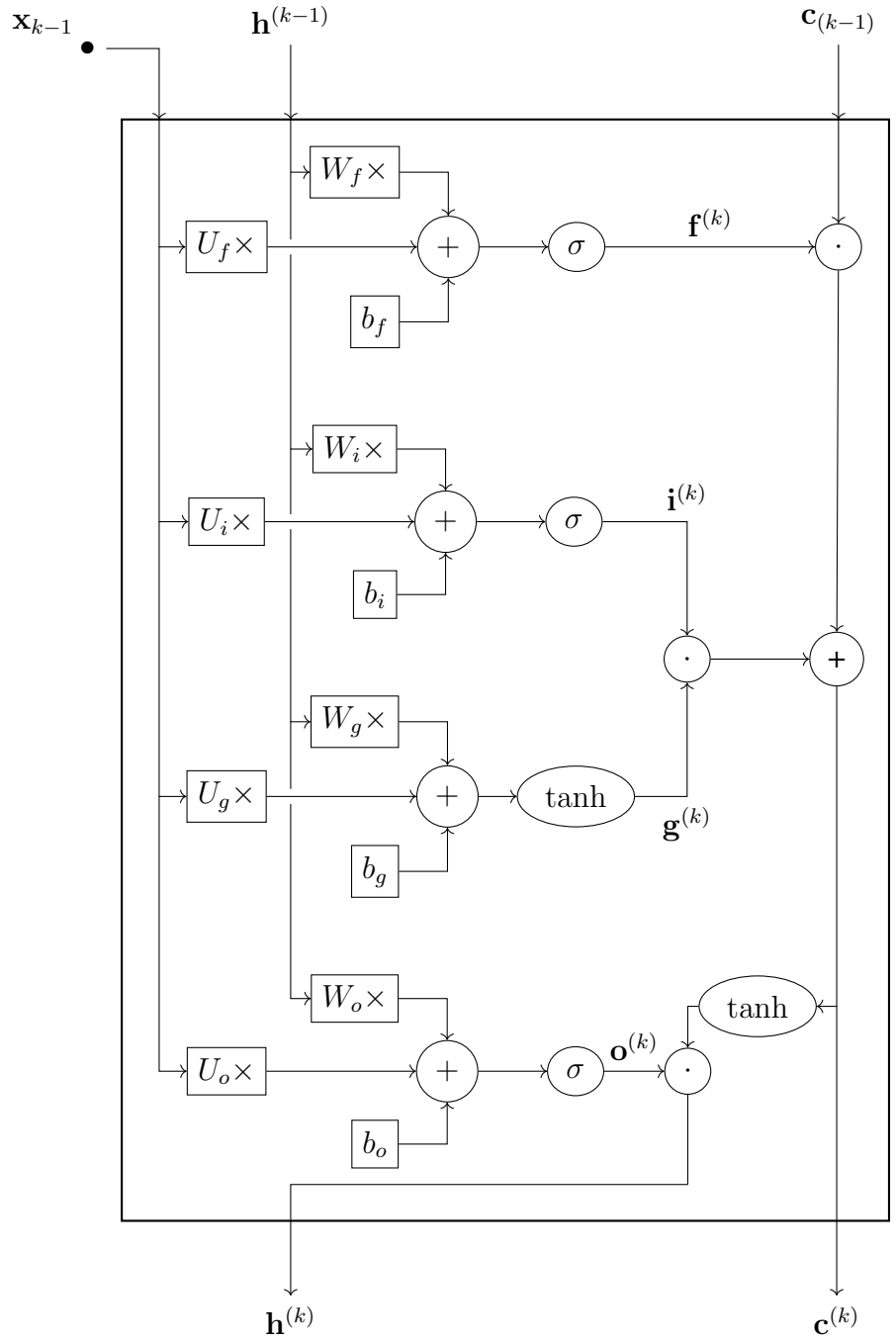


Figure 6.5: Representation of the LSTM cell computation.

6.3 Applications with Trivial Examples

For supervised learning applications, the dataset is composed of two types of entries: the input data X and the labels Y. The input data is a set of samples from an observation process $\mathcal{O}$. The labels are paired with matching data. They are either unobservable or too expensive to retrieve in the desired applications. Each pair $(\mathbf{x}_i, \mathbf{y}_i)$ from the dataset is an element of $\mathbb{R}^n \times \mathbb{R}^m$.

Two differentiable functions must be defined for the learning process: A function $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ depending on a set of parameters θ to learn, and a loss function $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ that is minimal when both its input vectors are equal. The learning process aims at finding a particular set of parameters θ that minimize the loss on the dataset: $\sum_{(\mathbf{x},\mathbf{y})\in(X,Y)} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$.

In our forecasting application, the input data X is the past road scene sequence and the label Y is the future road scene sequence. The mean squared error is an appropriate loss function (the mean is computed both in time and among several data samples.) In the same fashion as the application made in chapter 1, the data is split in 8 second sequences. It is used to learn a forecasting model that should associate the first three seconds with the five following seconds. In this application, the three architectures described in the previous section are used to define three models to train.

The input is a three-seconds sequence at 5Hz; thus, $N_H = 15$. It is composed of the x, y vehicle positions and forms an input vector $\mathbf{x} \in \mathbb{R}^{30}$. The forecast is a five-seconds sequence at 5Hz; thus, $N_F = 25$. It is also composed of the x, y vehicle positions, forming an output vector $\mathbf{y} \in \mathbb{R}^{50}$.

Fully connected - As a first application, we define a two layer fully connected network with 2 units in the first layer and a ReLU activation function. This architecture uses 212 parameters.

$$
\begin{aligned}
N_{\text{param}}(W^{(1)}) &= 30 \times 2 \\
N_{\text{param}}(b^{(1)}) &= 2 \\
N_{\text{param}}(W^{(2)}) &= 2 \times 50 \\
N_{\text{param}}(b^{(2)}) &= 50
\end{aligned}
$$

It is represented in figure 6.6. Using only 2 units is very restricting. To include a more expressive architecture, a fully connected model with three layers using 16 units is also produced. It is defined with 1618 parameters.

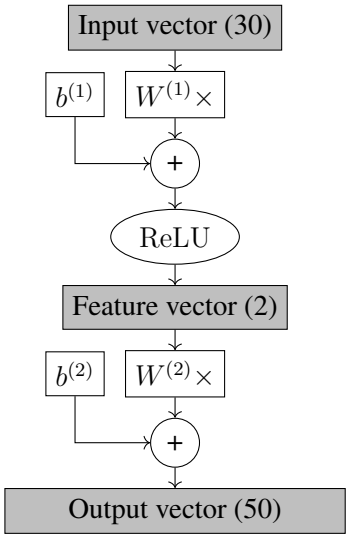


Figure 6.6: Representation of a 2 layer fully connected neural network.

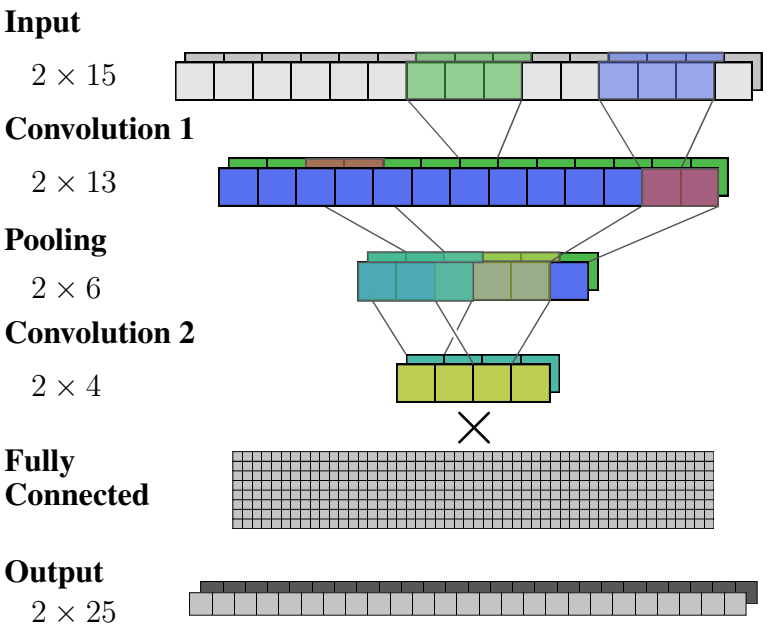


Figure 6.7: Convolutional neural network with a fully connected output layer. Feature sizes after each layer are reported.

Convolutional - The input is kept as a 2 by 15 sequence $\mathbf{x} \in \mathbb{R}^{2\times 15}$, and the output a 2 by 25 sequence $\mathbf{y} \in \mathbb{R}^{2\times 25}$. It is composed of a convolutional layer, a max-pooling layer, a second convolutional layer and an output fully connected layer. Both convolutions use two kernels of size 3, resulting in a model using 478 parameters. This is more than the fully connected architecture because of the last fully connected layer that is defined using 450 parameters. Before reaching this last layer, the time dimension is reduced to 4 with 2 channels. The number of channels is the feature vector dimension at each time step. It is equal to the number of units in the previous layer. The details about the parameter count is given below:

$$
\begin{aligned}
N_{\text{param}}(W^{(1)}) &= 3 \times 2 \times 2 && \text{Kernel size} \times \text{Input feature size} \times \text{Number of units} \\
N_{\text{param}}(b^{(1)}) &= 2 \times 2 && \text{Feature size} \times \text{Number of units} \\
& && \text{Pooling has no parameter} \\
N_{\text{param}}(W^{(2)}) &= 3 \times 2 \times 2 && \text{Kernel size} \times \text{Feature size} \times \text{Number of units} \\
N_{\text{param}}(b^{(2)}) &= 2 \times 2 && \text{Feature size} \times \text{Number of units} \\
N_{\text{param}}(W^{(3)}) &= 4 \times 2 \times 50 && \text{Sequence size} \times \text{Feature size} \times \text{Number of units} \\
N_{\text{param}}(b^{(3)}) &= 50 && \text{Number of units}
\end{aligned}
$$

As shown in figure 6.3, the receptive field of the two convolutions with pooling is 8 time-steps long. We could consider that it is sufficient and only use the last output of the second convolution instead of all 4. Then, the input size of the last fully connected layer is divided by 4, and this new reduced network only requires 178 parameters.

Recurrent - The input is also a sequence $\mathbf{x} \in \mathbb{R}^{2\times 15}$. This network is defined with two LSTM layers, following a common encoder-decoder architecture. The first LSTM layer encodes the input sequence and only the last recurrent feature vectors $(\mathbf{h}_e^{(15)}, \mathbf{c}_e^{(15)})$ are kept. The second LSTM layer is not fed with new observations at each step but only updates the feature vectors to produce a sequence of output $(\mathbf{h}_d^{(0)}, ..., \mathbf{h}_d^{(24)})$. Thus, the matrices called U in figure 6.5 are not used. Both LSTM layers use 2 hidden units. The second LSTM is run for 25 steps, producing the desired 5-second sequence. This sequence is re-scaled using a convolutional layer with two kernels of size 1. Overall, this architecture requires 74 parameters.

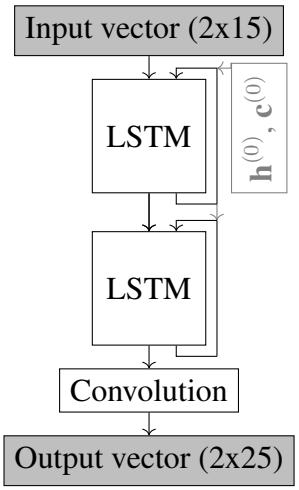


Figure 6.8: Representation of the recurrent neural network.

Application - These models are implemented in Python using the Pytorch library. They are trained on the same data as used in chapter 1 using the Adam optimizer. The Adam optimizer [KW14] is based on the gradient descent algorithm to update the parameters iteratively. It computes the gradient of the loss, averaged on a batch of samples, with respect to the parameters. We chose a batch size of 128. The parameters are updated using a rescaled gradient multiplied by a learning rate. We chose a learning rate of 0.0003. Once all the samples from the dataset have been used, one epoch is completed. We use 4 times the same data to continue the training until 4 epochs have been made. With about $6 \cdot 10^6$ data samples, this makes for approximately 180,000 steps of gradient update ($\approx \frac{6\cdot 10^6 \times 4}{128}$). In our experiments using an Nvidia tesla V100 GPU, this is done in about 30 minutes. These computations were made possible by the granted access to the HPC resources of IDRIS under the allocation 2019-39282 made by GENCI. Since we stop the iterations after a fixed number of steps, it favors the architectures that converge faster (in number of steps, not time). Figure 6.9 shows the training curve of the three models with a feature size of 2. Table 6.2 shows the average results on the test set.

Theory and practice - The only difference between the presented models and the implemented ones are in the recurrent network. In Pytorch, the LSTM cell is defined with two sets of biases in each sum resulting in 8 additional parameters in the first LSTM layer. The second LSTM layer may be defined in the same way to benefit from the cuDNN LSTM implementation, and instead of removing the U matrices, the $\mathbf{x}$ input are set to 0. Otherwise, it is also possible to re-implement the LSTM cell in Pytorch. The results are reported in table 6.2. The recurrent model without the U matrix should be exactly equivalent to the one using the cuDNN implementation because it only simplifies the addition of zeros in the computation. However, it does not seem to be the case in the application with the latter giving slightly better results.

Hidden size - Increasing the hidden sizes of the fully connected network from 2 to 16 requires more parameters and brings only a small improvement. Thus, only the feature size of 2 was kept for the other architectures.

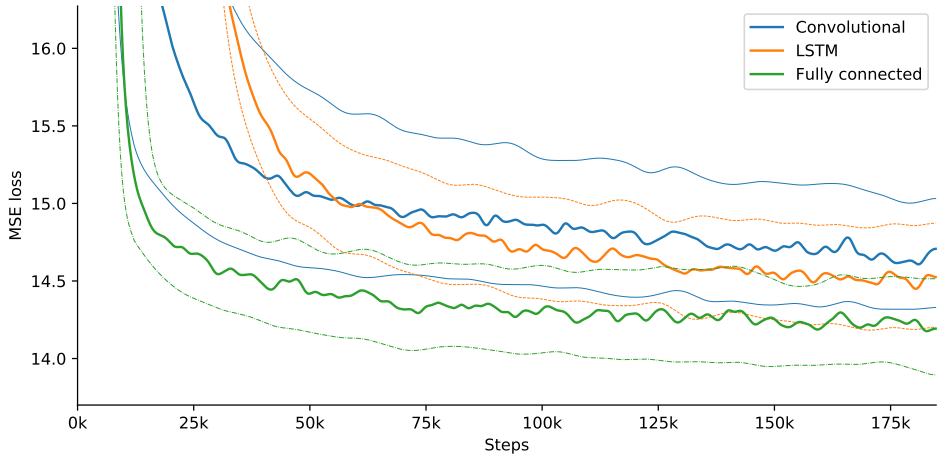


Figure 6.9: Mean Squared Error (MSE) loss during training of the fully connected, convolutional and LSTM vanilla models with hidden size 2. Envelope curves are smoothed maximum and minimum values over three trainings.

Table 6.2: Compared results of vanilla neural networks defined in this section

Time horizon		1s	2s	3s	4s	5s
RMSE (m)	Constant velocity	0.75	1.81	3.16	4.80	6.69
	Fully connected [size 2]	0.76	1.77	3.06	4.64	6.47
	Fully connected [size 16]	0.72	1.69	2.98	4.55	6.37
	Convolutional [only last]	0.77	1.82	3.15	4.77	6.64
	Convolutional [all]	0.77	1.80	3.12	4.71	6.54
	Recurrent [no U]	0.97	1.79	3.17	4.73	6.80
	Recurrent [cuDNN]	0.81	1.80	3.11	4.69	6.62
FDE (m)	Constant velocity	0.46	1.24	2.27	3.53	4.99
	Fully connected [size 2]	0.52	1.28	2.27	3.49	4.90
	Fully connected [size 16]	0.49	1.23	2.24	3.47	4.90
	Convolutional [only last]	0.51	1.31	2.35	3.61	5.07
	Convolutional [all]	0.52	1.30	2.32	3.55	4.98
	Recurrent [no U]	0.76	1.29	2.36	3.58	5.26
	Recurrent [cuDNN]	0.57	1.28	2.29	3.54	5.05
MR	Constant velocity	0.02	0.20	0.44	0.61	0.71
	Fully connected [size 2]	0.02	0.20	0.44	0.61	0.72
	Fully connected [size 16]	0.02	0.19	0.44	0.62	0.73
	Convolutional [only last]	0.02	0.22	0.46	0.63	0.73
	Convolutional [all]	0.02	0.21	0.46	0.62	0.73
	Recurrent [no U]	0.03	0.22	0.46	0.62	0.76
	Recurrent [cuDNN]	0.02	0.21	0.44	0.62	0.74

Learning dynamic - The recurrent network seems to converge more slowly than the other models, and thus it shows slightly worse performances at first. It is likely to benefit more than other models from more training epochs. A criterion based on the lowering of the validation loss such as early stopping (see 7.8 in [GBC16]) should be preferred to a set number of iterations. However, in many case the limiting factor deciding the end of the training is the computational time.

Non-convex overparametrized optimization - The training procedure finds a set of parameters reaching a local minimum of the loss. However, as studied in [FHL19], the model performances for the different local minimums are all equivalent. This is believed to be resulting from the high dimension of the parameter search space. This topic is actively being studied in the neural network research community. In the applications produced here, we use feature sizes of 2. Thus, we expect more variety of results for the learned models from different initializations than what is usually obtained with high dimensional applications. For this reason, when small feature sizes are used, three identical trainings are performed with different random initializations. The colored envelopes in figure 6.9 show the range of results. This range is due to two factors: the variance of the loss between the iterations and the variance of the loss between the different trainings. In one test, the convolutional model that only uses the last encoded feature did not converge at all. After four epochs, its RMSE at one second was still at 4.12m.

6.4 Forecasting Gaussian Probability Parameters

In the previous section, we defined the usual neural network architectures, and as an application, they were trained to produce trajectory forecasts. The comparison with the constant velocity model showed that the neural network models only match the baseline results. Moreover, the constant velocity model is able to produce the error covariance matrix that indicates the expected forecast error covariance. However, with the model from chapter 1, this covariance matrix is independent of the input data. The same error covariance matrix is given for all samples. Even if this global estimation fits well the overall error covariance, it is more informative to evaluate the specific error covariance associated to each forecast. The global error covariance estimation can still be assessed with the evaluation procedure from chapter 3. If a better individual error covariance estimation is obtained, an improvement is expected on the NLL evaluation. In this section, we adapt our forecasting neural networks to predict the error covariance matrices.

Gaussian parameters - The probabilistic forecast of the different possible futures can estimate the prediction error covariance. This can be seen as an uncertainty estimation or as a forecast of the future probability density function. As introduced in chapter 3, we consider a Gaussian output $\mathcal{N}((\hat{x}, \hat{y}), \Sigma)$ defined with the expected position forecast $(\hat{x}, \hat{y})$ and the additional output $(\sigma_x, \sigma_y, \rho)$ that defines the covariance matrix around the forecasted position:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

Output activation function - We follow the recommendations of [HR18] to produce outputs with the correct range. This is expressed as a specific activation function applied on the output of the neural network. Let o_i be the i^{th} coordinate of the output tensor before the activation function. To constraint the model outputs in the desired range, the following function is applied on each coordinate at every time steps:

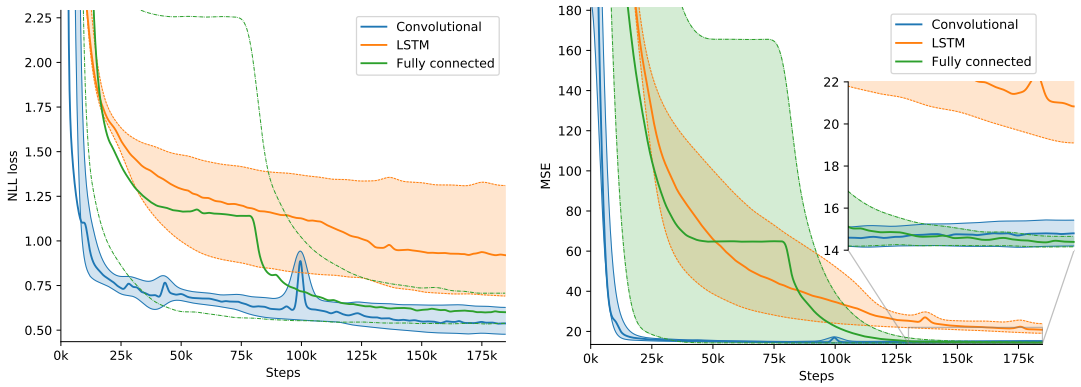
$$(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho) = \text{activation}(\{o_1, o_2, o_3, o_4, o_5) = (o_1, o_2, e^{\frac{o_3}{2}}, e^{\frac{o_4}{2}}, \tanh(o_5))$$

Thus, the only modification brought to the previously defined neural network architectures is the number of outputs and the use of this output function.

Partial supervision - The true error covariance cannot be estimated on each sample from the dataset. As a consequence, the covariance prediction cannot be supervised. Thankfully, minimizing the NLL loss pushes the model to maximize the likelihood of the observed future for the predicted distribution. This is the same loss function as used in the application of chapter 5 that allowed the learning of the Kalman parameters to forecast as a Gaussian distribution.

Hyperparameters - We tested the modified models with feature sizes of 8 and 16. The NLL loss allows a higher learning rate than the MSE. Thus, for these experiments, we train the neural networks with the Adam optimizer with a learning rate of 0.001 for 4 epochs.

Learning curves - The learning curves of the models with a hidden size of 2 are represented in figure 6.10. As expected, the LSTM loss is decreasing slower than the others and would benefit from more iterations. The fully connected layer seems to have reached its local minimum quickly, but it does not give the best performances. The convolutional architecture seems to work best and to converge faster. This is probably due to the higher number of parameters.



(a) NLL loss training curve.

(b) MSE training curve (m^2).

Figure 6.10: Fully connected, convolutional and LSTM vanilla models with hidden size 2. Envelope curves are smoothed maximum and minimum values over three trainings.

Table 6.3: Compared results of basic neural networks with uncertainty estimation

Time horizon		1s	2s	3s	4s	5s
RMSE (m)	Constant velocity	0.75	1.81	3.16	4.80	6.69
	Fully connected [size8]	0.71	1.73	3.05	4.65	6.49
	Fully connected [size16]	0.72	1.73	3.02	4.60	6.43
	Convolutional [size8]	0.73	1.76	3.08	4.68	6.53
	Convolutional [size16]	0.71	1.73	3.03	4.61	6.45
	Recurrent [size8]	0.75	1.80	3.11	4.70	6.60
	Recurrent [size16]	0.70	1.71	3.00	4.60	6.49
FDE (m)	Constant velocity	0.46	1.24	2.27	3.53	4.99
	Fully connected [size8]	0.46	1.24	2.27	3.52	4.97
	Fully connected [size16]	0.47	1.24	2.26	3.51	4.95
	Convolutional [size8]	0.47	1.25	2.29	3.54	5.00
	Convolutional [size16]	0.46	1.23	2.25	3.50	4.96
	Recurrent [size8]	0.48	1.26	2.30	3.57	5.07
	Recurrent [size16]	0.47	1.25	2.28	3.55	5.05
NLL	Constant velocity	0.81	2.31	3.22	3.91	4.46
	Fully connected [size8]	0.40	2.04	3.01	3.71	4.27
	Fully connected [size16]	0.37	2.02	2.98	3.68	4.24
	Convolutional [size8]	0.38	2.04	3.01	3.71	4.27
	Convolutional [size16]	0.32	2.00	2.97	3.67	4.22
	Recurrent [size8]	0.48	2.09	3.05	3.74	4.30
	Recurrent [size16]	0.33	2.01	2.99	3.70	4.26
MR	Constant velocity	0.02	0.20	0.44	0.61	0.71
	Fully connected [size8]	0.02	0.19	0.43	0.62	0.75
	Fully connected [size16]	0.02	0.19	0.44	0.62	0.74
	Convolutional [size8]	0.02	0.20	0.44	0.63	0.74
	Convolutional [size16]	0.02	0.19	0.43	0.62	0.74
	Recurrent [size8]	0.02	0.20	0.45	0.63	0.75
	Recurrent [size16]	0.02	0.19	0.44	0.64	0.75

Performance indicators - The evaluation procedure from chapter 3 allows a unified result comparison. Table 6.3 presents the results obtained after training for the different architectures. The results that only depends on the distance between the forecast and the observation, RMSE, FDE, and MR, have about the same values as the constant velocity results and as our previous models. This could be surprising because the loss that is minimized is not the MSE anymore.

The NLL values are slightly improved over the constant velocity baseline. This is probably because the error covariance values are adjusted for each input. Figure 6.11 shows that the averaged covariance estimations match the empirical error covariance even better than the optimized constant velocity model.

Figure 6.11: Ellipses representation of the validation set error dispersion and the dispersion predicted by the fully connected model with feature size 16.

Covariance outliers - With a few samples (around 200 samples among the 1.5 million sequences) from the validation set, in some of our tests, the standard deviation can be hugely overestimated. This has an important impact on the mean value that shows a much better fit if these values are saturated at 100m. With a huge standard deviation and a low correlation ρ, the NLL can be approximated with $\ln(\sigma_x \sigma_y)$. Therefore, the loss is almost independent of the prediction error and the logarithm keeps a small impact on overall loss. When looking at these samples, it appeared that they match the most noisy situations where the model is not able to make a forecast.

This chapter defined simple neural network architectures that are suitable for sequential trajectory data. They were directly applied to the motion forecasting task of individual tracks. The results have not shown improvement over the constant velocity model developed in chapter 1. However, the forecast error covariance can be estimated more accurately and is adapted to each sample. This is the only improvement that our neural networks made over the Kalman constant velocity baseline. Augmenting the models capacity with wider and deeper neural networks did not improve these results. Thus, it seems that the factor limiting the performances at longer time horizons is not the model size or its architecture but the limited information in its input data. In fact, in the data that were given to these neural networks, all the context information had been discarded. Only single tracks were used without considering neither the road network nor the surrounding vehicles. In the NGSIM dataset that we used, the road network is mainly composed of straight lines. Thus, the most likely solution to improve our results is to consider the interactions with the surrounding vehicles. Using multiple tracks as input is the subject of the next chapter.

Chapter 7

Multiple Agent Interactions

In the previous chapters, the motion forecasting models used a single track as input. Forecasting the interdependent behavior of human drivers is a necessary step to improve our simplistic models. Chapter 4 has shown some of the approaches used in the literature to consider human behaviors in the motion forecasting models. In that context, we consider that we cannot model human cognition. The interactive models IDM/MOBIL, social forces, hand-crafted objectives with optimization procedures, and game theory approaches were not able to capture the intricate real-world interaction patterns to a satisfactory level. Considering this, the most promising way to forecast human behavior is with statistical learning. Thus, we define forecasting as the estimation of the outcome statistics conditioned on the available context observations. Neural networks are well suited to learn such tangled statistics from observed data. However, in our applications in chapter 6, the only improvement neural networks brought over the constant velocity model was a slightly improved covariance estimation. This poor performance follows from the lack of necessary information in the input data.

7.1 Multiple Inputs cause Multiple Problems

Independent interactions - In this chapter, the observations are extended to many vehicles in the road scene. The tracks of the surrounding vehicles become part of the models input data. The dependency of the future track distribution on the context is a one-way interaction. The mutual inter-dependency between the context and the future track is a two-way interaction. In this work, the two-way interactions are not modeled explicitly. The distribution of the vehicles future positions are considered independent of the future of the others but not independent of their past.

Neural Networks with Multiple Inputs - The different types of input representations lead to different neural network architectures. In this chapter, we consider:

- Static-size ordered lists of features
- Rasterized images
- Coarse grids
- Graphs
- Dynamic-size unordered lists of features

Theses different input types go along with specific neural network architectures that handle the input data.

Description of a road scene - In all the following, the observation area is a neighborhood centered around a reference vehicle that is considered to be the ego car. Within that area and throughout the sequence, the observed agents are tracked. In the NGSIM dataset all these agents are other vehicles. The surrounding vehicles may not be observed during the whole sequence. Their tracks might contain some missing pieces in the input data and in the supervision. The coordinates reference point is the ego position at time t=0 exactly in the same fashion as it was done in chapter 1. However, the input is now a list of tracks containing a variable number of elements.

A problem of size - The input of the neural networks defined in chapter 6 must be a fixed size vector. It may have a variable time sequence length with the convolutional and recurrent architectures, but it must have a fixed number of features. Thus, the variable number of observed vehicles is not well suited for deep learning methods. Several ways to solve this difficulty are employed in the literature.

Egoistic or joint - Depending on the model architecture and on the objective, it is possible to make a joint forecast for all the vehicles in the scene or to forecast only the ego vehicle trajectory. When only the ego trajectory is forecasted, the other vehicle tracks are only used to describe the context around the ego vehicle. It is way more difficult to interpret the results of joint forecasts because it averages out errors in very different contexts with partial observations, uneven perception noise, and sometimes different types of observed road users (cars, bikes, trucks, pedestrians...) This is why, almost all the reported vehicle forecasting results are about the ego vehicle; even when the method is able to make a joint forecast. This tendency is changing thanks to normalized dataset used on platforms such as Kaggle and EvalAI that make a common comparison possible. However, the joint forecasts are still considered independent of one another for the reasons discussed in the paragraph "Independent forecasts" of section [**sec:independent**].

7.2 Static-Size Inputs

This section presents various strategies to form and use static-size input to describe the multiple agent tracks that co-exist in the road scenes.

7.2.1 List of Features

Fixed-size unordered list - The most straightforward solution to build a fixed number of features in the input vector is to define a maximum number of vehicles. If fewer vehicles are observed, the rest of the input is set to 0 and is masked out. If more vehicles are observed, the furthest ones are discarded and not fed to the neural network. This fixed-size vector can be fed to the vanilla neural networks, such as the ones defined in the previous chapter. All the vehicle tracks are concatenated into a unique input vector. For this example, the expected output is the forecast for the ego vehicle only.

We choose the maximum number of vehicles to be 18. Thus, the input feature dimension is now 36 instead of 2 at each time step. Otherwise, the neural network architectures that we use are the same as in the previous chapter. With an unordered list of the surrounding vehicles, the results are not improved. This is because the neural networks layers are sensitive to the order in which the vehicles are listed in the feature vector. Thus, adopting a meaningful ordering might be enough to improve the results.

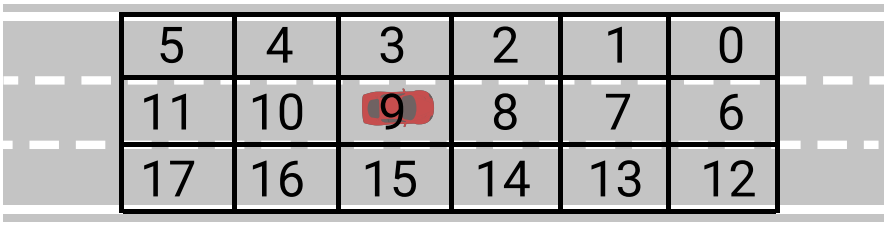


Figure 7.1: Representation of the coarse grid input ordering used in our tests.

Fixed-size ordered list - We take inspiration from [DT18] using the code they provide[1] to order the list of vehicle features depending on their position in a coarse grid as represented in figure 7.1. We simplify the representation by using a 3×6 grid instead of the 3×13 used in their work.

The simple neural network architectures defined in the previous chapter are used with this new input and a feature size of 16. The results are reported in table 7.1.

Table 7.1: Compared results of basic neural networks with uncertainty estimation based on an ordered list of context vehicles.

Time horizon		1s	2s	3s	4s	5s
RMSE (m)	Constant velocity	0.75	1.81	3.16	4.80	6.69
	Fully connected	0.69	1.59	2.72	4.14	5.82
	Convolutional	0.69	1.58	2.72	4.12	5.78
	Recurrent	0.68	1.57	2.72	4.66	8.05
FDE (m)	Constant velocity	0.46	1.24	2.27	3.53	4.99
	Fully connected	0.47	1.14	2.00	3.08	4.36
	Convolutional	0.45	1.12	1.97	3.04	4.32
	Recurrent	0.47	1.15	2.03	3.34	5.49
NLL	Constant velocity	0.81	2.31	3.22	3.91	4.46
	Fully connected	0.28	1.71	2.57	3.22	3.73
	Convolutional	0.24	1.67	2.55	3.19	3.71
	Recurrent	0.25	1.69	2.55	3.21	3.80
MR	Constant velocity	0.02	0.20	0.44	0.61	0.71
	Fully connected	0.01	0.15	0.38	0.56	0.68
	Convolutional	0.01	0.15	0.37	0.56	0.68
	Recurrent	0.01	0.15	0.39	0.59	0.72

Not good enough - Except for the recurrent model at a 5 seconds horizon, the results are better than those of the constant velocity on every indicator. This confirms our hypothesis that the performance would be improved by merely considering the interactions. The NLL is much better than in our previous models. However, the RMSE at 5s is only improved by 13% and there is a difference of only 3% in miss rate at 5s.

The ordered list of features representation is subject to important limitations:

- It uses 0 padding.
- It limits the number of vehicles.
- It limits the field of view.
- It suffers from discontinuities.

[1] `https://github.com/nachiket92/conv-social-pooling`

7.2.2 Rasterized Image

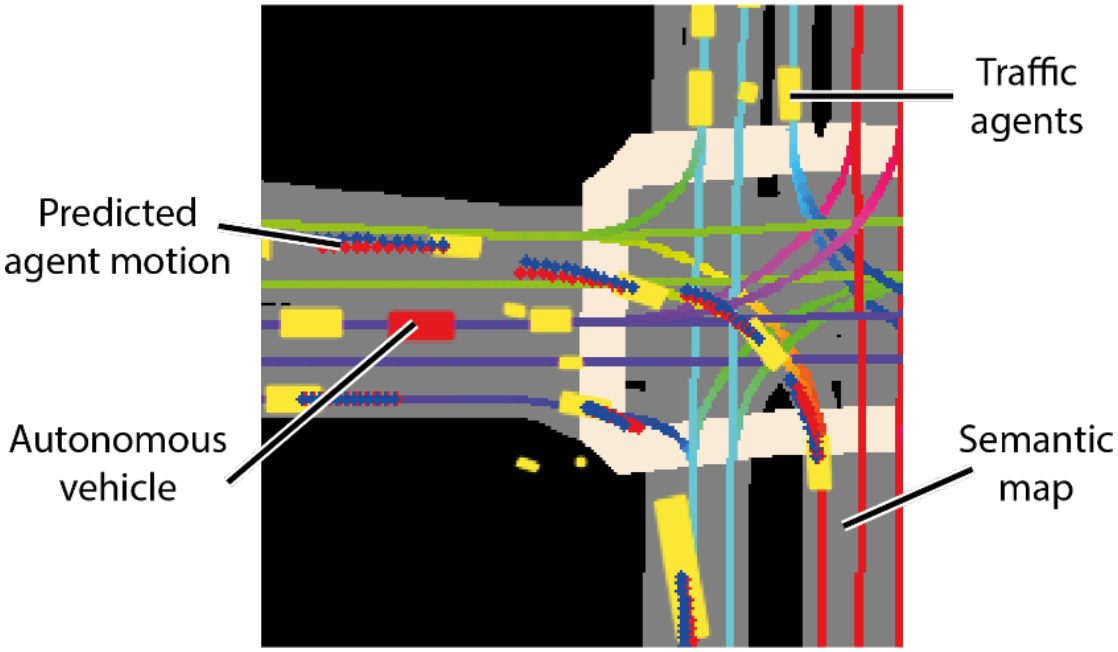


Figure 7.2: A rasterized road scene. (source: Lyft)

Image processing is all you need - One of the reasons for the recent success of the neural networks is their excellent image classification results. These results were made possible with the use of two dimensional Convolutional Neural Networks (CNNs) over the image pixels. The CNNs are built with a hierarchical architecture that allows the learning of high-level abstract features in the input data. They work so well that some tasks that do not involve images are re-framed as image processing to employ CNNs. This is the case for the audio speech recognition task for example [ADY13]. In motion forecasting, [Dju+18; Cha+19a] use rasterized images representing the road scenes as the input of a CNN. Such a representation is shown in figure 7.2.

Image processing is expensive - The bird-eye view representation and the convolutional architecture force the sense of spatial relations between the objects of the scenes. The rasterized image fixes the size of the field of view and the spatial precision instead of the number of input. Moreover, it can be enriched with any information from the HD-map or other perceptions such as static objects, lanes, and markings. However, rasterized images represent the road scene with much redundancy. The redundancy makes the model computationally expensive and memory intensive. The authors from [Dju+18] report training their model on 16 Nvidia Titan Xs for 24 hours. Using rasterized images is bound to remain inefficient unless a method to handle a sparse input image is used. We do not know of any forecasting method successfully using sparse road scene images as input. Because of the considerable amount of computational resources required, and because it does not seem to bring an edge to the model, we do not reproduce these results.

No compromise - Our results are compared with those of image based methods on a standardized test in an open competition presented in chapter 9. It shows that image-based models perform well but do not produce the best results. Moreover, the superiority of vector-based method over image-based methods is demonstrated very thoroughly in [Gao+20]. This means that not using rasterized image is not a compromise between computational efficiency and good results. The best results can be achieved at a lower computational cost.

The rasterized image representation is a powerful one that produces good results but has a number of drawbacks:

- It is expensive to train and to compute.
- It limits the field of view.
- It is not compatible with joint forecasting.

7.2.3 Coarse Grid

The coarse grid representation is intermediary between the list of features and the rasterized image. As the image representation does, it considers a 2D grid map but it uses cells instead of pixels. A coarse grid representation is shown in figure 7.3. The coarse grid is composed of only a few cells to describe the scene and each one contains more features than the number of channels per pixel in a rasterized image. It differs with the ordered list of input along a grid only in the way it is handled. With coarse grids, the relative positions of the cells is used while list of inputs do not use this information.

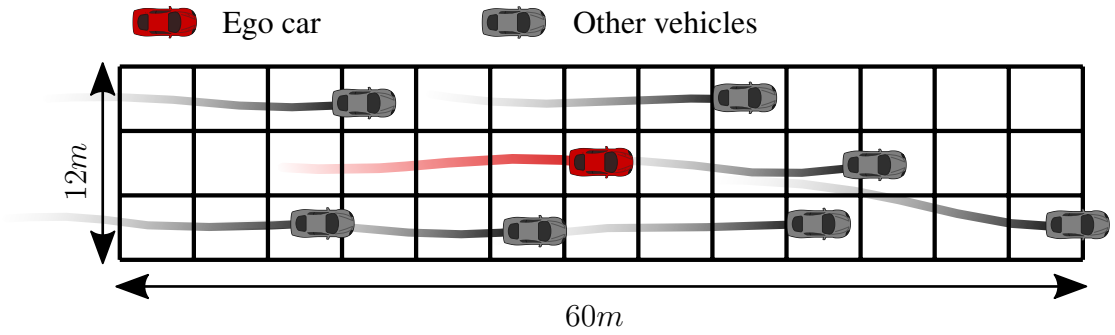


Figure 7.3: Representation of the coarse grid input used in [DT18]

Social pooling - The social pooling model (S-LSTM) is invented in [Ala+16]. It uses a coarse grid representation to order the surrounding agents in a constant size input vector. The coarse grid is a discretization of the space surrounding an agent of interest. Each grid cell either contains a list of 0 or the encoding of an agent state if its position matches the cell position. Each agent and the list of cells defining its social context is represented as a fixed size input grid. Instead of directly feeding this input to the neural network, each track is encoded using the same LSTM. The resulting encoded values are used to populate the coarse grid cells. The social pooling layer extracts a context vector from the coarse grid around the agent of interest. It produces a fixed size context vector and the encoding of the ego trajectory that can easily be used to make a forecast. Two variants of this process are represented in figures 7.4 for the social pooling and 7.5 for the convolutional social pooling.

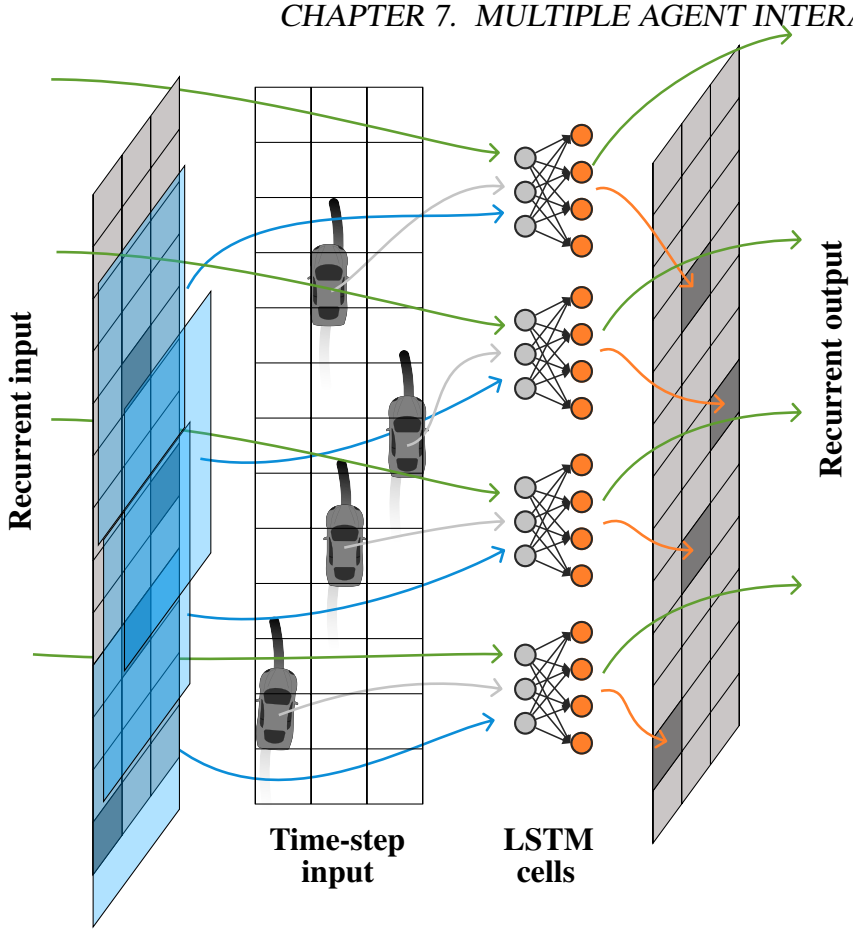


Figure 7.4: Illustration of the recurrent cell used in the S-LSTM model. The colored arrows represent different kind of inputs: in green the recurrent LSTM cell input and output, in gray the current postition, in blue the sub-part of the recurrent context around the considered region, finally in orange the features to be placed in the coarse grid.

S-LSTM - For the pedestrian motion forecasting application of the S-LSTM, a square grid of 32×32 cells is used to describe the whole scene. Each agent in the scene is matched with an 8×8 sub-grid of its surroundings as its social context. In the S-LSTM model, the coarse grid is updated at each time step to account for agents moving from one cell of the grid to another. We illustrated this in the context of road scene motion forecasting in figure 7.4.

In the same article [Ala+16], the authors of the S-LSTM also produce results with a version called O-LSTM where the coarse grid is used as a binary occupancy map instead of a feature pooling mask. The O-LSTM is a coarse version of the image rasterization. The authors show that the O-LSTM is enough to outperform hand-defined interaction models such as social forces.

The dynamic cell assignment of the agents in the coarse grid, made at each recurrent step, is not easily implemented. It requires to build a custom operation at each recurrent call. This prevents the use of an optimized LSTM implementations. In our attempt to reproduce this work we obtained a model that was so slow that we could not make a complete training. A better implementation than ours would surely solve this problem.

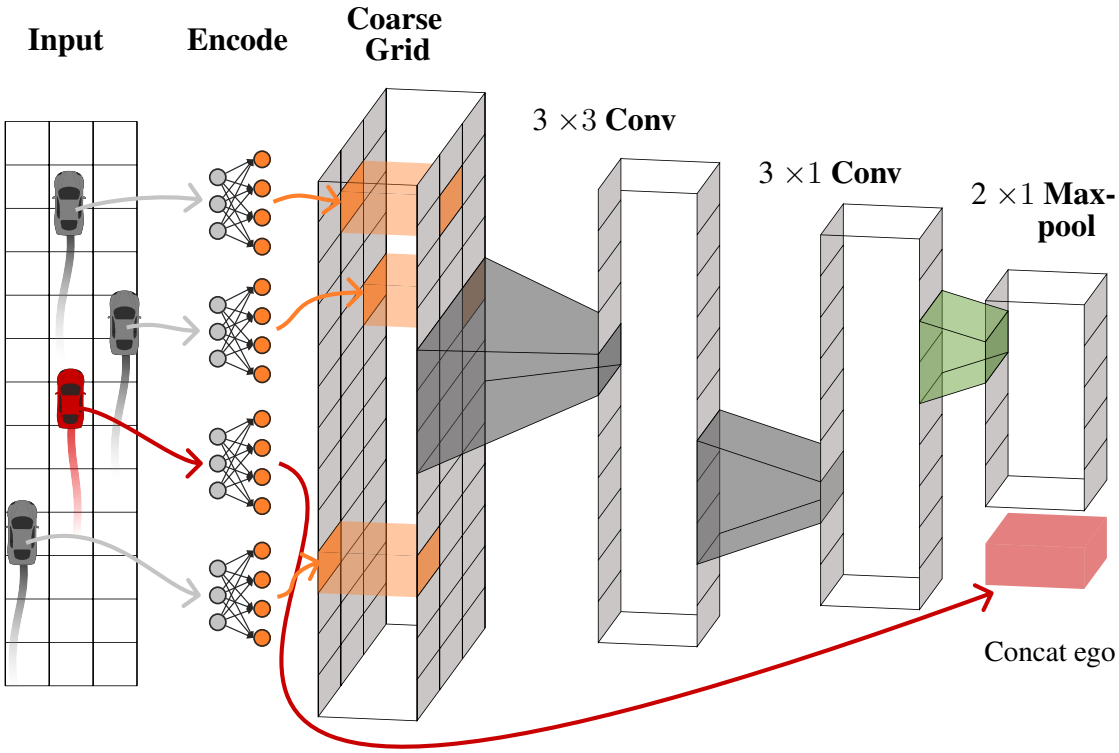


Figure 7.5: Illustration of the CS-LSTM encoder from the coarse grid input to the fixed size representation the context and the ego trajectory.

CS-LSTM - The CS-LSTM architecture, illustrated in figure 7.5, decouples space and time. This model also use LSTM encoders for the input tracks. However, each track is encoded separately for the whole past sequence. The resulting encoded tracks are placed in the coarse grid accordingly to the present time vehicle positions. Once this static coarse grid is formed, instead of being fed directly to an LSTM decoder, a convolutional pooling is performed. This consists of two convolutional operations over the grid and a max-pooling. The output is a representation of the context around the ego vehicle. This context and the encoded past trajectory of the ego vehicle are used by the rest of the network to make a forecast for the ego trajectory.

False comparison - The authors of the CS-LSTM compare their results with their own re-implementation of the S-LSTM and show that their method brings some improvement. Since the code was released, it is easy to reproduce and obtain the same results as published. However, from the study of the code, it appears that several errors were made. In their re-implementation of the S-LSTM, they kept a static social context instead of the dynamic one described in [Ala+16]. The only remaining difference between their implementation of the two models is the convolutional pooling of the CS-LSTM instead of a fully connected layer used for the S-LSTM. The static grid can only allow one-way interactions of the context on the ego vehicle. In [Ala+16] the coarse grid evolves dynamically in time allowing two-ways interactions between the vehicles and their context. Therefore, we cannot compare the two methods unless the S-LSTM is re-implemented.

Some minor bugs - The NGSIM dataset is recorded in imperial measurements, whereas the vast majority of the literature uses the metric system. The NLL values reported in [DT18] are computed with the imperial system while the RMSE is in meters. This creates an offset in the NLL results. Moreover, the authors did not add the constant value of the NLL, creating an additional offset. Finally, one of the terms in the NLL missed a multiplicative constant. This is not an offset, and it requires to recompute the results. This does not have a substantial incidence on the model performance. However, several other published works used this implementation to improve the method and reported results containing the same mistakes. At the pace of publication in the field, it seems probable that other results might contain errors. Thus, in our experiments, we should either reproduce the results or use a public benchmark. All these observations are made possible by the open-sourcing of the code from the authors of [DT18]. We welcome open-sourcing or the use of public benchmarks as the best practices in our field of research.

An unsatisfactory trade-off - The social pooling method is a trade-off between the list of inputs and the rasterized image. It compensates the grid coarseness by adding more features in each cell and keeping the state encoding of the vehicle whose trajectory is forecasted. However, it still suffers from some of the problems of the rasterized image. Most social grid cells are empty and it cannot fully benefit from the local translation invariance as the convolutional architecture would with a rasterized image. Therefore, two very similar situations could lead to different cell patterns. This means that this representation is discontinuous. The network is not fully invariant to these local discontinuities in its input because the spatial convolution is over a coarse grid that does not allow a deep hierarchical architecture. In any case, it still suffers from the discontinuities provoked by vehicles entering or leaving the grid boundaries. Finally, the separation between the context and the ego vehicle does not allow a joint motion forecasting for the whole scene.

7.3 Failed Attempt: Coarse Free-Space Representation

This section is the result of our first attempt to solve the data representation problem when the scene contains multiple tracks. We tried to build a motion forecasting of the surrounding agents specifically adapted to one purpose: smoothly avoiding collisions. For this objective, the useful information that must be forecasted is the evolution of the free-space around the ego vehicle instead of the future positions of all the agents. We restrict this study to the highway environment and choose a specific coarse grid representation that uses the discontinuities to define important events that modify the free-space. We call this representation Free-Space Obstacle Pack (FOP) because the surrounding vehicles are seen as a pack of obstacles tracked as a single pack that defines the free-space around the ego vehicle.

A fixed-size road scene state - In our applications, each obstacle is a part of the whole pack state. An obstacle is defined with a position, a velocity, and an acceleration. There is one obstacle in each of the six zones surrounding the ego vehicle in the center. When a vehicle enters one of the zones for the first time, if it is closer to the ego than the current obstacle in the zone, it becomes the tracked obstacle. Consequently, the state is updated with this vehicle position, velocity, and acceleration. This update may cause a discontinuity. All possible discontinuities are listed in section 7.3.2.

The FOP represents the free space around the ego vehicle using the nearest obstacles in 6 zones of interest. An essential difference with frameworks that represent the whole pack, such as the representation used in [AL17], is that the six zones of interest are updated dynamically as the vehicles are moving and in the forecasted sequence. The second difference is that there is not a global frame of reference. Each obstacle in each zone is tracked using a different frame of reference. Section 7.3.3 shows that computing the distance in the FOP space emphasizes the events such as lane changes in the error evaluation. This penalizes the models that do not capture them and allows machine learning models to be biased toward fitting these important events better.

7.3.1 Free-Space Representation

The FOP representation relies on the local center lines of the lanes. It uses them to express the surrounding vehicles kinematic states in 2D Frenet coordinates.

Euler spirals - Each of the six centerline segments is described with an Euler spiral. It forms six zones, each defined with six parameters: the origin coordinates, the angle at the origin, the curvature at the origin, the curvature variation, and the arc length. The arc length is arbitrarily set to 50m. The other parameters are produced from the ego vehicle localization, and the lane properties as illustrated in figure 7.6. The origin of each Euler spiral is placed at the ego position projected on the respective centerlines. The angle at origin, curvature at origin and curvature variation are computed using the lane centerline properties at its origin. Appendix B gives details about Euler spirals.

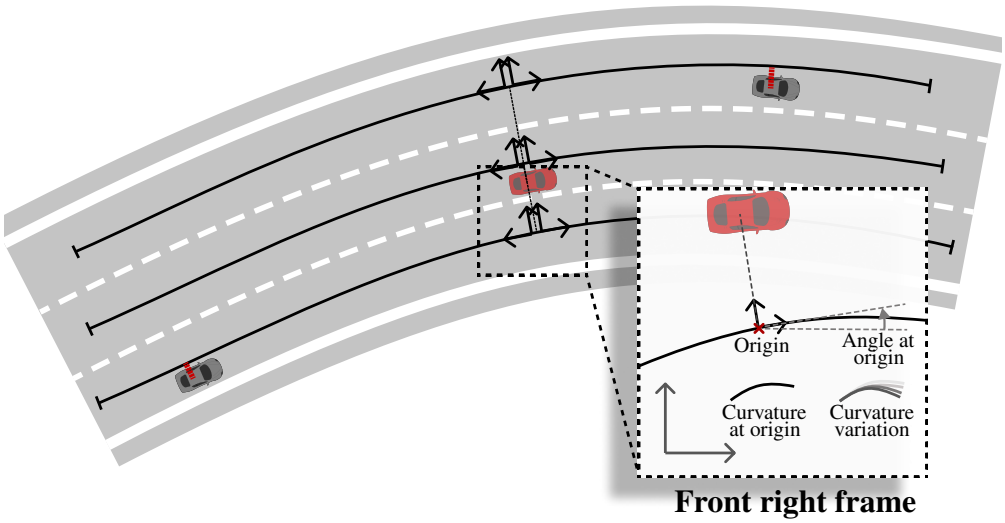


Figure 7.6: Representation of the Euler spirals used to track the obstacles.

Frenet coordinates - The FOP state is the global state vector of the road scene. It is the concatenation of the closest obstacle state in each zone. A local Frenet coordinate system is associated with each Euler spiral with a positive longitudinal direction going away from the ego vehicle. The coordinate system is moving and accelerating with the ego vehicle. The obstacle states are lateral and longitudinal positions, lateral and longitudinal velocities, lateral and longitudinal accelerations. The lateral parts of the state are relative to the lanes centerlines. The longitudinal parts of the state are relative to the moving ego position projected onto the middle centerline.

Edge cases - When there is no left or right lane, no obstacle can be present in the corresponding zones, and their states are set to zero. When there is no obstacle in a zone, the state is saturated. The saturated state is set at the maximum arc length distance of the Euler spiral with zero lateral distance, relative velocities, and accelerations. Relying only on relative states discards a part of the information such as the absolute velocity. However, it also allows the abstraction from absolute values that would otherwise be specific to each sequence. This could help our models to generalize.

Free-space representation - The preferred way to see the FOP representation is represented by the blue zone in figure 7.7. It is a free space representation defined with a polyhedron whose vertices are the observed obstacles positions. The FOP representation encloses the ordering of the obstacles and states that define the free space polyhedron over time.

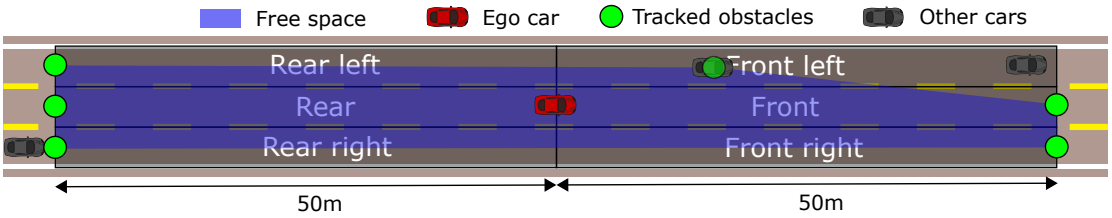


Figure 7.7: A driving scene top view representation with the six observed zones around the ego vehicle in the center. In each zone, the tracked obstacle is the closest observed vehicle or the furthest point of the zone. The six obstacles are the vertices of the free space polygon surrounding the ego vehicle.

Zoning - In the literature, the surroundings are sometimes described with eight zones, the two additional zones being in the right and left lanes, forming an intermediate zone between front and rear. In the case of [AL17], nine zones are used with an additional zone reaching further in front of the ego vehicle. The six zones that we use allow a notion of maneuver to be expressed directly with a zone change and smooth out some of the discontinuities. For example, the saturated values allow the longitudinal distance to change continuously when a vehicle enters the zone from its extremity (as it usually happens when no lane change occurs).

7.3.2 Discontinuities

Zone changes cause discontinuities in the obstacle states sequence. There are 34 types of discontinuities, 17 in one direction and 17 in the opposite direction. Some occur within the observed pack and others with the unobserved environment. In the list below, a vehicle from the UnObserved environment is written UO. Other zones are written with the first letters (Front Right FR, Front Left FL, Front F, Rear R, Rear Right RR, Rear Left RL).

A. 8 lane changes within the pack: FL $\leftrightarrow$ F, FR $\leftrightarrow$ F, RL $\leftrightarrow$ R, RR $\leftrightarrow$ R

B. 4 zone changes within the pack: FL $\leftrightarrow$ RL, FR $\leftrightarrow$ RR

C. 12 disappearances/appearances from unobserved zones through longitudinal boundaries: UO $\leftrightarrow$ FL, UO $\leftrightarrow$ RL, UO $\leftrightarrow$ F, UO $\leftrightarrow$ R, UO $\leftrightarrow$ FR, UO $\leftrightarrow$ RR

D. 8 disappearances/appearances from unobserved zones through lateral boundaries: UO $\leftrightarrow$ FL, UO $\leftrightarrow$ RL, UO $\leftrightarrow$ FR, UO $\leftrightarrow$ RR

E. 2 ego lane changes

The groups **A.** and **B.** cause two discontinuities; one is a transfer of state, the other is the observation of a new obstacle, or saturation in the freed zone. The groups **C.** and **D.** cause one discontinuity. For **C.** longitudinal distance is continuous. The ego lane changes **E.** cause discontinuities over the whole pack. Since our representation is relative to the ego, the ego lane change is similar to all vehicles changing lane in the opposite direction; this is 4 state transfers, 2 appearances, and 2 disappearances all happening simultaneously.

It is possible to make a short exhaustive list of the discontinuities. Thus, detecting and managing these discontinuities is possible with an engineered solution or with a learned classifier model. Discontinuities are at the root of the desirable properties of our data representation. They emphasize important events in the environment, which helps the unsupervised learning of such events. It is a more complex data distribution to fit, but it embeds the complexity of maneuvers that are often treated separately in the literature.

7.3.3 Maneuver free framework

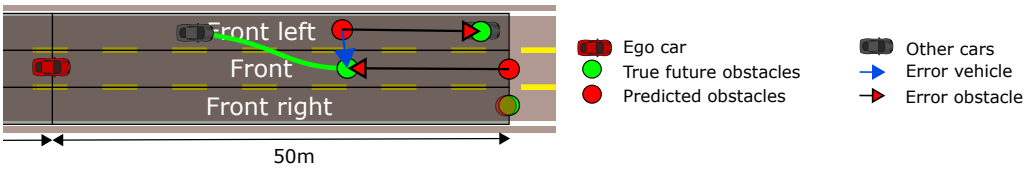


Figure 7.8: The same driving scene as figure 7.7 with an evolution and corresponding forecast. At time $t = 0$ the front left vehicle starts a lane change along the green path. At time $t = t_f$ this vehicle is in the front lane. During this time, there is a switch of obstacles, the front left obstacle represents a new car and the front obstacle becomes this car. If the lane change were not predicted, two kinds of error could be measured, the two red arrows or the blue arrow.

Error penalization - Figure 7.8 shows a possible evolution of the scene from figure 7.7. In this scenario, the front left vehicle changes lane to the front lane. A possible forecast for the final obstacles positions in this scenario is represented with red circles. For the example, we suppose that the forecast missed the lane change. The error is calculated by comparing the forecasted FOP state with the true future FOP state. The two red arrows in figure 7.8 represent the resulting error in the FOP representation. By contrast, the blue arrow represents the resulting error in the usual, vehicle centered, representation. The two red arrows express a large error while the blue arrow only conveys a small error. This example shows how lane changes are emphasized by the FOP representation.

Event forecasting - In the learning process of a model that uses this representation, missing a maneuver prediction is strongly penalized and is being pushed away from these large errors. Thus, the methods such as labeled maneuvers are not needed to capture these events. In the application, section 7.3.4.2, we illustrate this fact with a simple neural network that classifies the zone changes while being supervised only by the FOP error and no label.

7.3.4 Applications

This section presents predictive deep learning algorithms that exhibit some characteristics of the FOP representation. For these experiments, the forecasting model is partitioned in two parts: a continuous forecasting model and a zone classification model. Figure 7.9 represents both parts of the model and the operation that uses the results from both parts to make the FOP forecast.

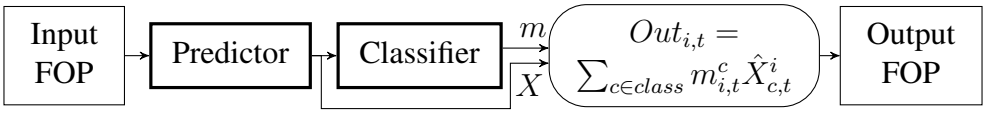


Figure 7.9: Predictor with classifier structure. The continuous forecasts and classifications are combined to produce the output in the FOP representation. The element $m^c_{i,t}$ is the confidence that the obstacle in the zone i at time t is of class c. $m^c_{i,t}$ should be either very close to 0 or to 1. The class of the obstacle in zone i at time t is $m^j_{i,t}$ if, at time t, it contains a vehicle that was originally in the zone j. It can also be of class saturated if no vehicle is present in the zone or of class None if the zone is not drivable. $\hat{X}^i_{c,t}$ is the forecasted state of class c at time t converted to the coordinate system associated with the zone i.

Tests evaluating the FOP characteristics - In our first test, the continuous forecasting model is set with the constant velocity model and the zone classifier is a neural network to be trained. In our second test, both parts are defined with neural networks and are trained in an end-to-end manner.

We use highway scenarios from the HighD dataset [Kra+18] to learn from real-world traffic scenes. It is similar to the NGSIM dataset that we used in our previous experiments. We pre-process this dataset to obtain ego-centered observations in the FOP representation before using it with usual deep-learning methods. Finally, forecast errors and zone classification accuracy are presented.

Considering 25 measurements per second, the 3 second FOP sequences used in this work are arrays of 75 time measurements of the 6 states (longitudinal and lateral positions, velocities, and accelerations) for the 6 obstacles. The predictive models we developed use an FOP sequence of the past 3 seconds to output another FOP sequence of the future 3 seconds. The first element of the forecasted FOP sequence follows the last element of the input FOP sequence. The neural networks are implemented and trained using Python with Keras and Tensorflow. The data pre-processing and analysis are made with the Pandas and Numpy library in Python.

7.3.4.1 Pre-processing simulating ego-centered data

The HighD dataset used for this application, records the positions of all vehicles on a highway section at each time step. For the needs of this work, exactly as we did with the NGSIM dataset, an embedded ego perception is simulated from this data. First, a vehicle is chosen to be the ego. Then, the relative kinematic states of the surrounding vehicles are extracted to obtain the FOP representation described in section 7.3.1. Local center lines are expressed as Euler spirals in each scene. We computed the global lane center lines as a polynomial fit of all the vehicle positions assigned to each lane. The local Euler spirals may be seen as inputs from a map or as the real-time perception of local lane geometry. Every car from the database is alternatively chosen as the ego and its observation time is split into 6 second sequences with 3 seconds of overlap. This process produces a transformed database that we use for the training of machine learning models.

7.3.4.2 Deep learning networks

The FOP representation forms a constant size input vector that can easily be fed to a neural network. It allows the neural network to account for the relation between the ego car and the surrounding vehicles. The relative positioning is crucial because it determines the type of interactions taking place. The whole pack of obstacles being considered at once also allows the model to find relations among the surrounding vehicles.

First test: zone classification Using the structure represented in figure 7.9, we produced a model with a CV predictor and a classifier network that matches the surrounding vehicles forecasts with their respective zones. The CV predictor is defined directly in the FOP representation. It sets the acceleration to 0 and uses constant longitudinal and lateral velocities to update the longitudinal and lateral positions along the Euler spiral. The ability to train it using the forecast error shows the FOP representation capacity to be maneuver free.

Classifier - A classifier neural network, with the architecture shown in figure 7.9, is built to prove that it can be trained with the FOP loss. It outputs, m, a matrix that encodes a smooth indicator function (the hard indicator is used at test time) to match each forecasted obstacle with one of the eight classes (six relative zones, none position, and saturated position). An output FOP forecasting is produced with the combination of the continuous forecast and the zone classification. This combination, represented by the rounded corner box in figure 7.9, is made with a differentiable function for the gradient descent training to be applicable. It uses the following sum:

$$Out_{i,t} = \sum_{c \in class} m^c_{i,t} \hat{X}^i_{c,t} \tag{7.1}$$

Changes of coordinate system - The computation of this sum requires for the forecast in any zone i at time t, X^i_t, to be expressed in any axis system c. This forecast expressed in the other axis system is noted $\hat{X}^i_{c,t}$. The classification coefficients $m^c_{i,t}$ act as a mask that selects the zone with its axis system in which the output should be expressed. If i is the front left zone and c the front zone, $\hat{X}^i_{c,t}$ has almost the same longitudinal coordinate and the same velocity and acceleration as X^i_t. However, its lateral coordinate

is changed with an offset of the lateral distance between the lanes. For $c = i$ no axis system change is required.

The L1 distance between the true FOP representation and the forecasted FOP is used to train the classifier. It is is the sum of the absolute FOP error over the state features.

A classifier trained for regression - This classifier model is built and trained only to attest that the FOP loss makes the model sensitive to zone changes. Thus, unlike standard classifiers, this one is not trained with a label of the correct class but with the loss computation of the FOP error. However, the classification prediction accuracy cannot be determined exactly and several classifications could lead to the same result. To estimate the classification accuracy, an auxiliary classification that minimizes the FOP loss for a given forecast and corresponding truth is computed. If the classifier prediction produces a position error within a small margin of the minimal one, the classification is judged as correct. We arbitrarily chose the margin at $10^{-4}m$ because it is small enough to be an insignificant error and it is sufficient to validate equivalent classifications. After training a 3-layer fully convolutional neural network classifier, we obtain an accuracy of more than 70% for each time of the forecasted sequence. This shows that the FOP representation allows maneuver learning while being supervised only for regression.

Second test: End-to-end training For this test, we do not use a constant velocity forecast but another fully convolutional neural network. We train both the maneuver classification module and a neural network trajectory forecasting at the same time. It still follows the global architecture represented in figure 7.9. The model is trained end to end with the FOP L1 loss as supervision. It is expected to both classify maneuvers and make joint trajectory forecasts of the surrounding obstacle trajectories.

Hard noisy softmax - The classifier tends to average the forecasts from several zones. We want to force it to make a harder classification while keeping it smooth enough for the gradient descent algorithm to work properly. Thus, the softmax is hardened with an intensity factor, λ, and a Gaussian noise is added on the logits (the values before softmax). The parameters of the noise and the intensity factor are hyperparameters of the model that we set with trial and error. The classifier produces the logits h that are turned into the classification matrix m with: $m = softmax(\lambda(h + noise))$. As expected, the classification, especially with a large intensity multiplier, produces large gradient values when computing the back-propagation of the loss gradient with respect to the network weights. Thankfully, lowering the learning rate is enough to keep the learning process stable.

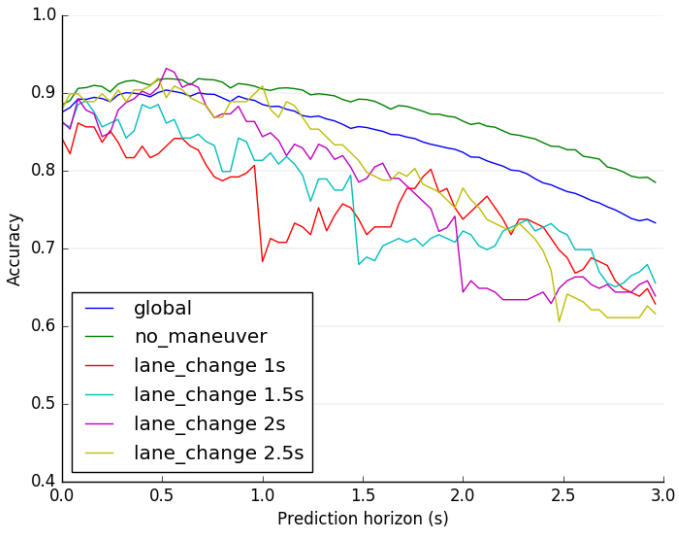


Figure 7.10: Accuracy graph of obstacle classification over time for our second model. The accuracy is the percentage of correct classification for the whole scene at specific future times. Three second forecasted sequences with a lane change occurring at the second 1, 1.5, 2 and 2.5 are tested. The accuracy drops visible at these times are partially compensated afterward meaning that in most cases, the lane change that is not predicted at the correct time is predicted correctly before or after that time.

7.3.5 Result analysis

The FOP representation is designed to produce significant errors when a maneuver is not caught. The Root Mean Squared Error (RMSE) computed in the FOP representation cannot be compared directly to the vehicle state forecast RMSE that is usually reported. To evaluate our results, we establish more specific comparisons in this section.

Evaluating classification and regression separately - The raw obstacle distance between the actual future and the forecast is a good performance indicator since it relates directly to the minimized values. However, this is hard to interpret because substantial errors are produced every time a maneuver is not predicted at the exact time it occurs in the future. Thus, we introduce two other indicators: the correct maneuver classification rate and the forecast error with perfect maneuver prediction. The results presented in this section are produced with the two models described in section 7.3.4.2.

7.3.5.1 Zone classification

Maneuver prediction based on trajectory forecasting is explored in [Woo+17] with a maneuver prediction concerning one vehicle knowing its surroundings. With our scene observation, only partial information is known about each observed vehicle. Our representation of the road scene is well suited to an embedded perception system that would track the surrounding vehicles but not always the next ones that could be occluded by the closest ones. Moreover, zone changes for the whole pack are to be predicted instead of a maneuver for a unique vehicle.

Classification accuracy - Using our zone classification model as a maneuver predictor allows us to express a zone classification accuracy. Figure 7.10 shows the zone

classification prediction accuracy over the time horizon for all obstacles considering global or specific data subsets. We consider that a lane change occurs at the time a vehicle comes closer to another centerline than the one it was associated with. We have selected specific sequences with lane changes occurring at 1, 1.5, 2, and 2.5 seconds in the future. A small drop of accuracy is expected at that prediction time, even for a perfect model. Indeed, in some cases, a lane change would lead the FOP representation to track a new unobserved obstacle (noted UO in section 7.3.2). Another kind of accuracy drop occurs when the lane change is predicted slightly before or after the correct maneuver time in the future. This is visible in figure 7.10 with drops of accuracy at each lane change occurrence time. In those cases, the accuracy drops are local and the accuracy raises again when predicting further. We observed that our model tends to predict lane changes slightly later than it should and seldom before. This is probably because the lateral distances distribution of the data pushes the predictor toward more centered predictions.

7.3.5.2 FOP forecasting

In this section, we evaluate free-space forecast error and trajectory forecast error of the surrounding obstacles. To this end, we compute the Root Mean Squared Errors (RMSE) of the forecasts compared with the true observations.

FOP forecasts are not agent forecasts - Since our model forecasts an FOP representation, recovering the corresponding vehicle trajectories from it is not possible. Indeed, the vehicle identities are lost and only the obstacles in each relative zones are forecasted. We can approximately recover the vehicle positions using an optimistic classification of the zone. The optimistic classification is the one that minimizes the FOP loss for a given continuous forecast. Using this classification, we can recover the relative trajectories of the surrounding vehicles. The absolute trajectories are defined by incorporating the true future ego motion.

For the obstacle avoidance task the absolute trajectories of the surrounding agents are not needed. If we recovered them, it is only to obtain evaluation metrics that are comparable to the one published in the literature.

Result comparisons - We compare the error we computed in the absolute coordinate system with the one produced by a constant velocity model similarly to what is done in [DT18]. The RMSE of the position forecasts is shown in table 7.2. In the first column, the constant velocity prediction errors are given as a comparison baseline. The data marked with a '*' are errors in the absolute coordinate system involving the optimistic classification. In our free-space representation, many obstacles are only the saturated value of the surrounding space. They are expected to be forecasted too but since they do not describe vehicle positions, we also compute the RMSE when they are excluded. Errors are marked as 'no sat' when saturated values are masked out of the error computation. Errors marked as 'FOP' are raw errors using the FOP representation. The values from [DT18] in gray correspond to forecasts about the ego vehicle with knowledge of its surroundings and with the NGSIM dataset. The values 'RMSE* no sat' refer to obstacles matching existing vehicles in the absolute coordinate system and is the most comparable result.

Table 7.2: RMSE, in meters, of the road scene forecasts.

Predictor	Constant velocity	Neural network
CSP RMSE [DT18]	3.13	2.09
RMSE* no sat	3.78	3.18
RMSE*	2.31	1.95
RMSE FOP	4.99	4.70
RMSE FOP no sat	7.06	6.50

Poor performance gain - Table 7.2 shows that this model brings only a 16% improvement over the constant velocity model, whereas the paper [DT18] brings a 33% improvement. This is judged to be a poor performance and is partly why this whole section is named "a failed attempt."

No fair evaluation - Our model is focused on free-space and this result comparison is not fair to our model. The RMSE metric favors the usual models that directly minimize the RMSE and our observation restrictions are harder than the one used in most work in the literature. We were not able to produce a metric that would make a fair comparison between our free-space centered approach and usual forecasts. It would be possible to use our model within an autonomous driving system in a simulator. Then, the number of accidents or the level of risk could be adequate indicator to compare the different predicting approaches results but would not test specifically the forecasting module. Our incapacity to evaluate the result is a second reason why the use of the FOP representation is a failed attempt.

Limited to highway - The FOP representation is tightly linked to the centerlines. In highway scenarios, it is easy to define the zones with which it operates. However, it is difficult to extend to more complex road networks such as crossroads. This is the third and final reason why we did not pursue this idea.

7.4 Dynamic-size List of Features

Forcing a dynamic number of observed agents into a fixed-sized input vector is bound to produce redundant or sparse representations or to discard some of the information. In the previous sections, we also met the discontinuity problem provoked by some objects being included, excluded, or reordered in the scene representation. This section tackles these challenges using specific neural network architectures that allow the forecast computation with a dynamic number of unsorted objects. We present three closely related solutions. They produce models that exhibit the desired properties with a similar global architecture following three steps: Each observed object track is encoded with the same function h. An equivariant function defined for any number of input objects is applied to all the encoded tracks. The outputs of this function are decoded to produce the forecast.

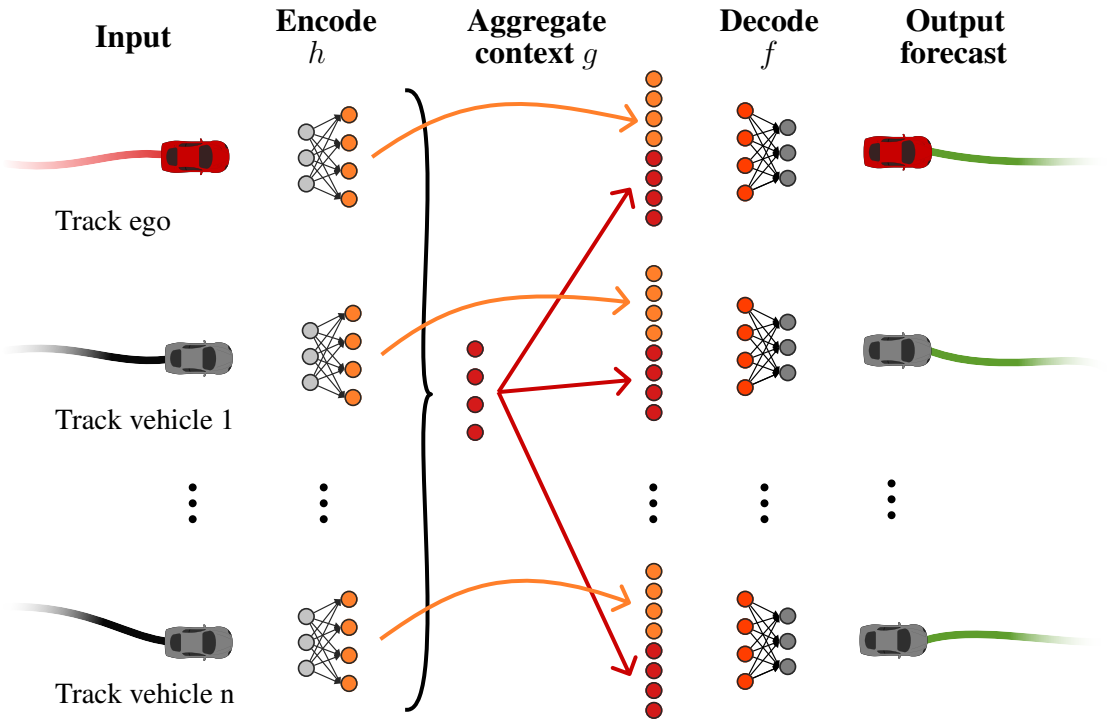


Figure 7.11: Forecasting neural network with global interaction pooling and aggregation.

7.4.1 Symmetric Functions and Dynamic Dimensions

Symmetric function - An n-dimensional function g is symmetric if for any permutation p of the indices, $g(x_1, ..., x_n) = g(x_{p(1)}, ..., x_{p(n)})$. The element-wise product and the sum are examples of symmetric functions and are defined for all n. Such symmetric functions defined for any number of input vector n are good candidates to extract a global representation of a road scene described with an unsorted list of n objects. This global representation should play the same role as the social tensor produced by the social pooling [Ala+16] that we described in section 7.2.3.

PointNet - The model PointNet [Qi+17] makes use of a symmetric function for point cloud data segmentation. The symmetric function used in their work is an element-wise maximum operation. This has the good property of not being impacted too much by the number of input data or by small data variations. However, the maximum operation must be performed using a large feature dimension of its input. Thus, before applying the aggregation $g = \text{element\_max}$, an embedding function h maps the input in a high dimensional space that is well suited for the maximum aggregation not to discard too much information.

PointNet trajectory forecasting - An adaptation of this idea for trajectory forecasting would follows the architecture presented in the introduction. The function h embeds each vehicle track in the scene. Then, the social context tensor is computed with $\mathbf{c} = \text{element\_max}(h(x_1), ..., h(x_n))$. Finally, a forecasting function f produces a forecast for each track using the track embedding and its context $\text{forecast}(x_i) = f(h(x_i), \mathbf{c})$. The embedding function h, and the forecasting function f are defined as vanilla neural networks. The complete forecasting function involving them is learned on a dataset. This architecture is represented in figure 7.11.

Adaption in StarNet - A similar idea is produced for pedestrian motion forecasting in a model called StarNet [Zhu+19]. StarNet does not only concatenate the embedded trajectory with the context tensor before computing the forecast, but it mixes the two feature tensor using an attention mechanism. The authors did not propose any interpretation or justification for this choice. The aggregated context is probably too generic and high-dimensional. Thus, selecting the relevant information for each agent might be preferable than using the whole tensor.

7.4.2 Graph Neural Networks

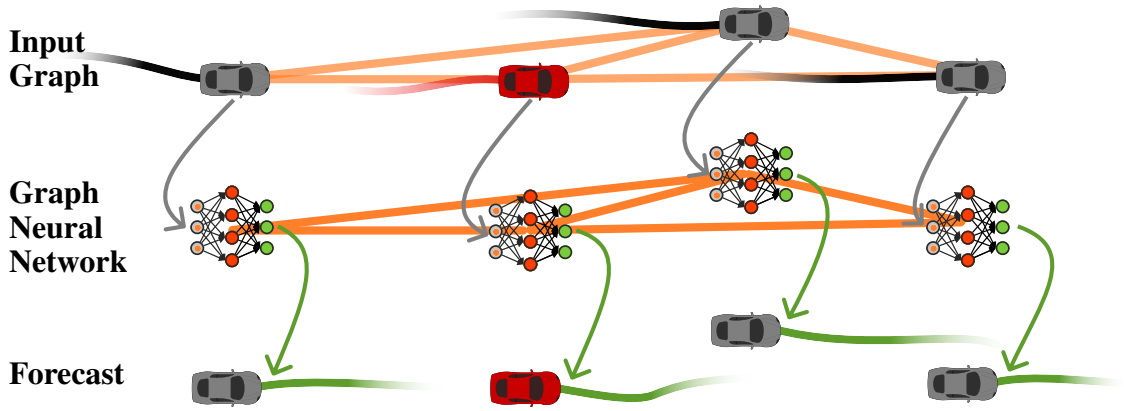


Figure 7.12: Illustration of a Forecasting Graph Neural Network.

Graph - A graph is described as a set of edges and nodes (or vertices). Each node is associated with a vector of values describing its state and can be in relation to any subset of nodes. An edge connects two nodes. It can be oriented with one parent node and one child node, or it can be undirected with two equal nodes. The edges describe the relations between the nodes. Depending on the type of graph, the edges can also be associated with a state vector or a scalar weight that characterize the relation between the two nodes.

Road-scene graph - Our representation of the road scene as a list of object tracks can be re-framed as a graph. Each object in the road scene is a node of the graph. The interactions between the objects are represented with the edges of the graph. There are several ways to use this representation to solve the fixed-sized input vector and input ordering problems.

Spatial or spectral - In the literature of the last decade, many forms of Graph Neural Networks (GNNs) were developed. In [Bru+13], the authors present an extension of the convolution operation on graphs. Defferrard *et al.* [DBV16] describe two strategies to define convolution filters on graphs: spatial approaches and spectral approaches. In [Bru+13], the spectral approach is used.

7.4.2.1 Spectral Graph Operations

Graph Laplacian - The graph Laplacian L is a square matrix describing the graph and is defined as $L = D - A$. D is a diagonal matrix with d_{ii} the number of connections to the i^{th} node, it is the degree of the node. A is the adjacency matrix. If an edge exists between the nodes i and j, $a_{ij} = 1$ otherwise, $a_{ij} = 0$.

Spectral operation - The parameters of a graph convolution filter are defined as the diagonal terms of a matrix F. This matrix is used as a factor of the graph Laplacian eigenvectors to compute the graph convolution operation. We note V the matrix of eigenvectors and Λ the eigenvalues in the decomposition $L = V\Lambda V^T$, ordered by eigenvalues. The l^{th} layer of graph convolution operation at the node i defined with a state s_i is written:

$$s_{l,i} = \text{activation}(V \sum_{j=1}^{n_l} F_{l,i,j} V^T s_{l-1,j})$$

Where $F_{l,i,j}$ is a diagonal matrix and n_l the number of eigenvalues kept by the operation.

Fixed number of inputs - The F matrix is formed with the learnable weights of the network but its dimension depends on the number of nodes. This means that such a network could only be used with a fixed number of nodes in the graph. This is not compatible with the varying number of objects in the road scenes. Thus, this operation must be approximated to rely only on a fixed number of parameters. This is achieved in [Bru+13] using cubic splines and in [DBV16] with Chebyshev polynomials. However, these function basis are defined for a fixed number of nodes. In conclusion, the spectral approaches to graph convolution solves the problem of ordering in our data representation and they can be used to define hierarchical operations but they are not compatible with a variable number of graph nodes. Thus, a fixed-size input must be used to apply this technique. Even if this can easily be done with zero padding, we favor the spatial approach.

7.4.2.2 Spatial Graph Operations

Spatial operation - The spatial graph convolution defines a simple update rule of the graph nodes:

$$s_l = \text{activation}(GW\mathbf{s}_{l-1})$$

Where W is the matrix of edge values such that the edge value between the nodes i and j is $w_{i,j}$. And $G = \text{norm}(D + A)$, the normalized sum of the adjacency matrix and the number of connections. With this formulation, the matrix of learnable weights W is of size $n \times n$ with n nodes. A model based on this idea is used in vehicle trajectory forecasting in [LYC20]. The authors show good performances on the NGSIM dataset and reached third place on the ApolloScape trajectory prediction competition[2]. This graph operation does not allow a variable number of input. However, the update rule can be reformulated.

Message passing - The graph edges connect two nodes. Therefore, the graph operation can be applied on every pair of nodes instead of globally:

$$s_{l,i} = \text{activation}\left(\sum_j \zeta_{\vartheta_l}(s_{l-1,i}, s_{l-1,j})\right)$$

With ζ_{ϑ_l} a function defined with the parameters ϑ_l that defines the relation between the nodes. It is used to define the update rule locally. This formulation allows to update the nodes states for any number of nodes. It can even define its own adjacency matrix if the nodes states contain the necessary information.

Convex sum - A classic definition of ζ_{ϑ_l} is:

$$\zeta_\vartheta(s_{l,i}, s_{l,j}) = \text{sim}(W_1 s_{l,i} + b_1, W_2 s_{l,j} + b_2) s_{l,j}$$

sim defines a similarity measure such as $x, y \longrightarrow \frac{1}{||x-y||+1}$ and $\vartheta = \{W_1, W_2, b_1, b_2\}$ are learnable parameters. Then, the update is a weighted sum of the neighboring nodes states. In those cases, this weighted sum is often normalized and becomes a weighted average. The operations are defined at the nodes and the same graph convolution parameters can be used with graphs composed of any number of nodes. Thus, this method is adapted to road scene representations with a dynamic number of observed vehicles. We use this kind of graph operation in our model but we formulate it as a self-attention mechanism.

[2] http://apolloscape.auto/leader_board.html

7.4.3 Self-Attention

In this section, we define the architecture that we settled on: the dot-product self-attention, also called transformer. It was popularized by [Vas+17] and used for natural language text translation. It defines an equivariant function for a dynamic number of input vectors. It can also be presented as a binary operation on every pair of nodes that computes the edge weights of a complete directed graph. With this point of view, it is a particular case of ζ_ϑ function and of normalization in the spatial graph operation described in the previous section.

7.4.3.1 Social Attention

Figure 7.13 illustrates how the dot-product self-attention mechanism can be used to replace the global aggregation function defined in section 7.4.1. It computes interaction representations from an unsorted list of encoded vectors.

Dot-product self-attention - The dot-product self-attention is made with four steps:

1. Harvesting a subset of specific value features
2. Tagging these values with a key
3. Querying among the keys
4. Gathering the queried values

Value - A first fully connected layer produces a selection of features $\mathbf{v}$ from each vehicle encoded vector. The results are concatenated into the value tensor V.

Key - A key vector $\mathbf{k}$ is associated with each value to identify them. It is produced with a different fully connected layer computed on the same input. The overall concatenated key tensor is noted K.

Query - Then, the interactions between the vehicles are computed by querying the features. For that purpose, a query tensor Q is produced with a third fully connected layer computed on the same input with the same feature dimension as the keys. The match score between a key $\mathbf{k}$ and a query $\mathbf{q}$ is their *dot-product*.

Dot-product attention - Therefore, the matrix product QK^T is composed of all the combinations of dot products between the keys and the queries. It is the correlation matrix between the queries and the keys. This matrix is scaled with the square root of the key dimension $\sqrt{d}$. Its lines are normalized with a $\mathrm{softmax}$ such that the sum of all the columns of this matrix is a vector of ones.

Attention matrix - The resulting matrix coefficients are interpreted as the relation between the pairs of vehicles; we call it the attention matrix.

Graph operation - The value vectors are the states of the nodes. This defines a dense directed graph on which only a simple operation is performed: the sum of the parent nodes values, weighted by the edges. Thus, with $\mathbf{v}_i$, $\mathbf{q}_i$, $\mathbf{k}_i$ row vectors stacked to form the matrices V, Q, K, the self-attention computation is written:

$$V = \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{n_{\text{veh}}} \end{pmatrix} \quad Q = \begin{pmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{n_{\text{veh}}} \end{pmatrix} \quad K = \begin{pmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_{n_{\text{veh}}} \end{pmatrix} \tag{7.2}$$

$$\text{aggregation} = \underset{\text{rows}}{\mathrm{Softmax}}\left(\frac{QK^T}{\sqrt{d}}\right) V$$

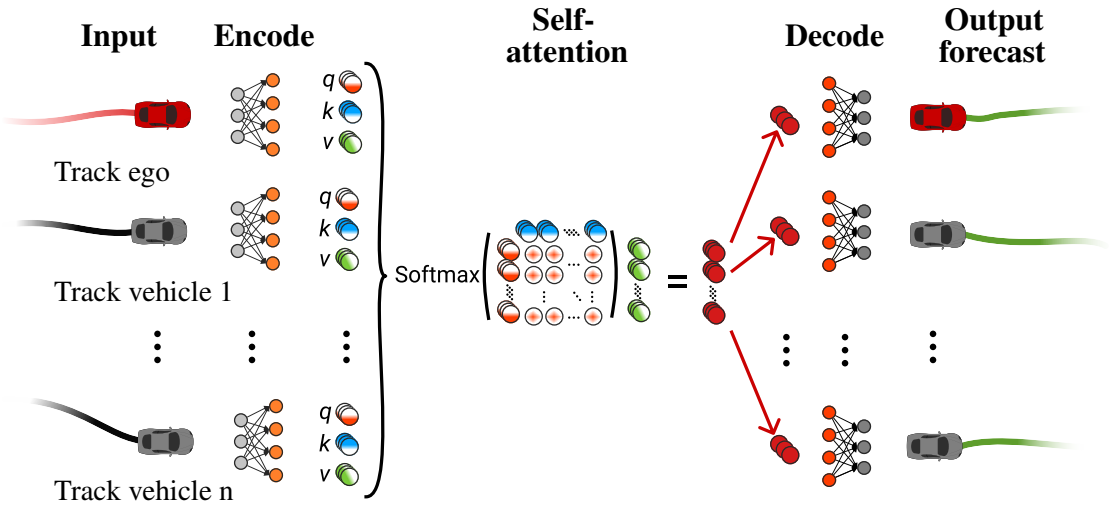


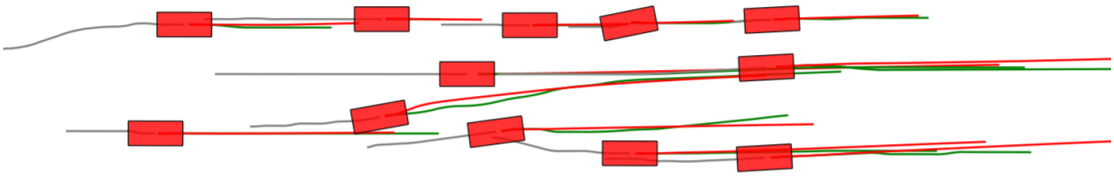
Figure 7.13: Forecasting neural network with self-attentive interaction pooling.

Self-attention for trajectory prediction - Figure 7.13 shows a complete neural network architecture using self-attention as context aggregation for trajectory forecasting. Each trajectory is encoded with the same neural network. The interactions between the inputs are computed by the self-attention operation. Then, each value is decoded with a neural network to produce the forecast.

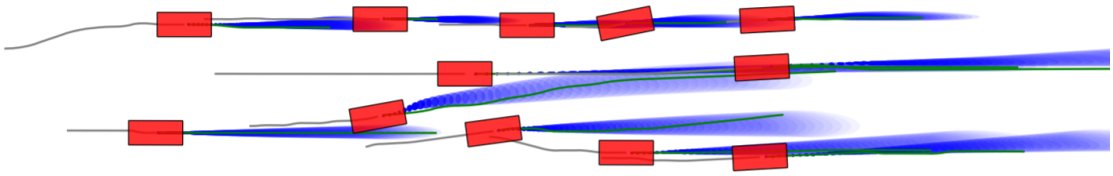
Hidden graph operation - This representation does not explicitly show the relation to graphs. However, the attention matrix (computed as the softmax of the sum of the outer products of keys and queries) defines an adjacency matrix. Its coefficients are between 0 and 1 instead of binary values but it can be interpreted in the same way. Therefore, this model can be seen as a combination of the two approaches described in the previous sections. It defines an equivariant aggregation function that consists of a graph construction and a graph operation.

Multi-head attention - In [Vas+17], the dot-product self-attention is extended with multiple attention heads. It consists of multiple parallel computations of the graph construction and graph operations in lower dimensions. Each low dimension graph operation is called an attention head. The outputs from all the heads are combined with an additional fully connected layer.

Skip connection - After the self-attention is computed, the input is added to the aggregated values so that the input ordering is preserved in the attention output. Moreover, it makes the identity function (no interaction) easy to learn for the model.



(a) Representation with the forecasts maximum likelihood in red.



(b) Representation with the forecast distributions in blue shades.

Figure 7.14: Bird eye view representation of a road scene from the NGSIM dataset. The past trajectories are represented in gray, the true future trajectories are in green.

7.4.3.2 Application

Structure - The definition and implementation of a variant of this model is the subject of our publication [Mer+20]. A convolutional layer followed by a sequence to one LSTM layer defines an encoder. Then, the aggregation is performed with the multi-head attention. The decoding is made with a one to many LSTM layer that produces an encoded sequence. Finally, the output is decoded with three convolutional layers of kernel size 1 to produce Gaussian trajectory forecasts simultaneously for all the vehicles.

Hyperparameters - We apply the model described above on the NGSIM dataset with a feature size that we have set at 60 for each vehicle of the scene. We chose to use six heads for the dot-product self-attention layer.

Training - The model is trained to minimize the NLL loss for 10 epochs with the Adam optimizer. This is takes about 10h using an Nvidia Tesla V100 provided by IDRIS under the allocation 2019-39282 made by GENCI. A longer training time would likely lead to better performances. However, the progress is logarithmic and 10 epochs are enough to reach good performances and interpret the results.

Results - Figure 7.14 shows an example of a road scene with the forecasts for each agent. A forecast is made for each agent in the scene but to be comparable to the other models, the performance indicators are evaluated for the ego forecast only.

The results presented in table 7.3 show that the self-attention mechanism allows the model to make much better forecast on every metric. This attests of a much better interaction mechanism than the sorted list with 0 padding that was restrictive and redundant. We gain about one second of forecast for the same errors.

RMSE plot - Following the evaluation procedure defined in chapter 3, we plot the RMSE parametric curve and the estimated covariance in figure 7.15. It shows a good fit between the predicted error and the empirical error.

Table 7.3: Compared results for the basic neural networks with uncertainty estimation using input context vehicles and the self-attention

Time horizon		1s	2s	3s	4s	5s
RMSE (m)	Constant velocity	0.75	1.81	3.16	4.80	6.69
	List	0.69	1.58	2.72	4.12	5.78
	Self-attention	0.48	1.10	1.85	2.81	4.00
FDE (m)	Constant velocity	0.46	1.24	2.27	3.53	4.99
	List	0.45	1.12	1.97	3.04	4.32
	Self-attention	0.31	0.78	1.36	2.07	2.95
NLL	Constant velocity	0.81	2.31	3.22	3.91	4.46
	List	0.24	1.67	2.55	3.19	3.71
	Self-attention	-0.57	0.99	1.90	2.56	3.09
MR	Constant velocity	0.02	0.20	0.44	0.61	0.71
	List	0.01	0.15	0.37	0.56	0.68
	Self-attention	0.01	0.06	0.22	0.39	0.54

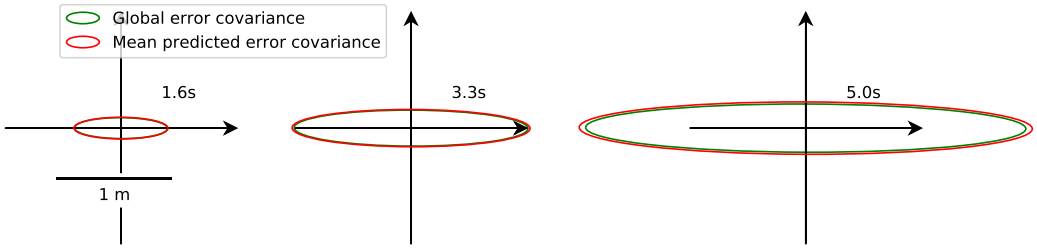


Figure 7.16: Comparison between the error covariance estimation and the global empirical error covariance.

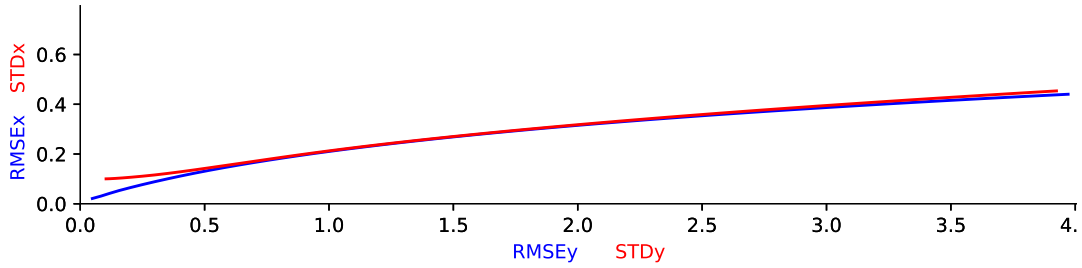
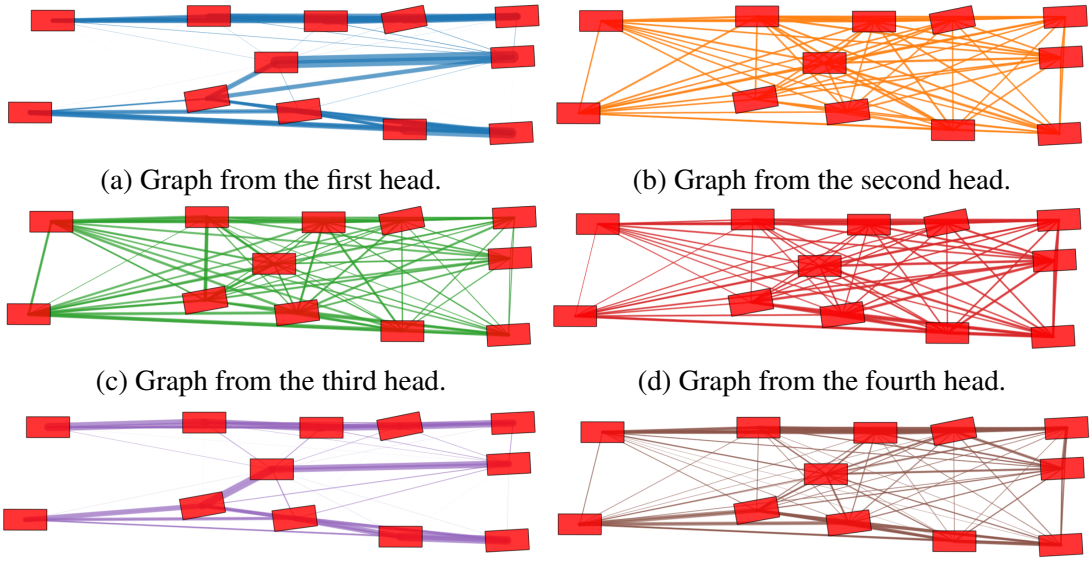


Figure 7.15: Parametric curves of RMSE in blue and standard deviation, $\sigma^{(\text{model})}$ in red as functions of the forecasting time.

Covariance assessment - The unit dispersion ellipses characterizing the mean predicted error covariance and the empirical error covariance are compared in figure 7.16. It shows a much smaller uncertainty than previous methods. The good fit between the ellipses shows a satisfactory global error covariance prediction. Individual error covariance estimations cannot be directly assessed but the low NLL values are a good indication that the estimated variances are close to the squared errors.

Attention graphs - The attention matrix defined by each attention head can be interpreted as the matrix of a complete directed graph. Therefore, we can draw the corresponding graphs in each scene of our dataset. Figure 7.17 represents all the attention graphs from all the heads in the same road scene. It shows that the head specialize in different attention patterns. Because this is not directly supervised, these patterns cannot all be interpreted. However, the first head in blue is attending all the vehicles aligned in front. The fifth head in purple is attending mainly the closest vehicle in front. Except maybe the third one in green, all heads disregard the vehicles that are close laterally and mainly focuses their attention on the ones in front. The fourth head in red is somewhat sensitive to the distance while the second one in orange is broadly distributed. The sixth head in brown seems to be more dependant on the context than the others, its patterns do not look similar from one example to the next.



(a) Graph from the first head.

(b) Graph from the second head.

(c) Graph from the third head.

(d) Graph from the fourth head.

(e) Graph from the fith head.

(f) Graph from the sixth head.

Figure 7.17: Graph representation of the interactions in a road scene from the NGSIM dataset. The attention graphs are represented as segments between the vehicles. The attention coefficients scale the widths of these segments.

Something is missing - The miss rate in table 7.3 shows that the forecasts at five seconds into the future miss the true future position by more than 2m in 54% of the scenes. This is a substantial improvement over the method that we produced up to this point but it is still not satisfactory. We believe that we can improve this point by forecasting several propositions instead of one. It might be impossible to predict the intentions of the drivers from the past trajectory observation. Thus, making several guesses would help cover each possible intention. This is the subject of the next chapter that introduces multi-modal distribution fitting and multi-modal forecasting.

Chapter 8

Multi-Modal Forecasts

In the previous chapter, we have presented neural network architectures capable of modeling the interactions between multiple objects in the road-scene. Considering the interactions between agents brought much improvement to our neural network models. However, it adds more choices. A vehicle could pass or yield; it could take over or slow down to get behind another vehicle or any choice depending on the situation and on the driver's intent. We introduced in chapter 4, the idea of forecast modes. They are a number of possible trajectories that are distant from each other but that could all be the outcome of the same situation. The different discrete choices made by the agents lead to different forecast modes.

Nothing in between - Figure 8.1 shows a simple situation where two choices could be made in the future. Because of the uncertainty between these choices, the forecasting model must guess between the two at the risk of making a large error. Another solution that it tends to fall into is to forecast the average of the different possibilities. This produces the path drawn with crosses shown in figure 8.1. As we can see, the average of two likely trajectories is not a likely trajectory.

One scene, several forecasts - The past observations are insufficient to consistently predict the choice that an agent will make. Thus, to avoid a wilde guess or a wrongful average, the forecast must represent several possibilities among the ones that seem likely.

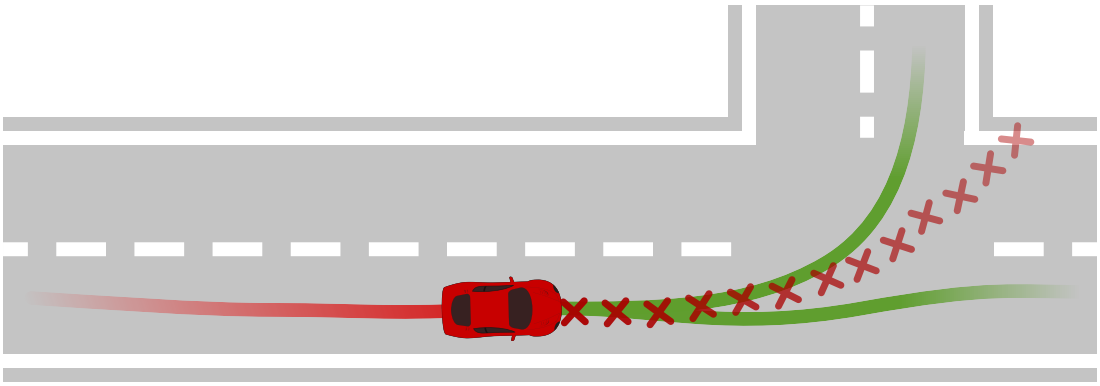


Figure 8.1: Illustration of a situation with two likely outcomes that are distant from each other and an average trajectory that is very unlikely.

Maneuvers as modes - The first idea is to use the concept of maneuver. It tries to generically defines all the possible choices. The maneuver-based forecast is made in two steps: maneuver classification and trajectory forecasting conditioned on the likely maneuvers. We have introduced this concept in the literature review of maneuver classification in section 4.3.

Multi-modal probability density - Another way to represent the discrete choices is to directly produce a model able to fit a complex distribution with local probability density maxima: the modes. It can be done explicitly by forecasting the parameters of a Gaussian mixture, as suggested in chapter 3. Otherwise, the distribution can be simulated by a generative model.

Generative model - A generative model is a function that transforms samples from a given distribution into samples following a desired distribution. It does so without defining explicitly the probability density of the desired distribution. We review the generative models that give promising results. Studying generative models helps us understanding a large part of the literature. It is also an excellent source of inspiration for neural network architectures. However, we do not pursue these approaches because they all share a common issue rooted in their definition: they need to be sampled. Chai *et al.* [Cha+19a], precisely state the issue of sampling:

> *"There are a number of downsides to sample-based methods when it comes to real-world applications such as self-driving vehicles:*
> *(1) non-determinism in a safety critical system,*
> *(2) a poor handle on approximation error (e.g,. "how many samples must I draw to know the chance the pedestrian will jaywalk?"),*
> *(3) no easy way to perform probabilistic inference for relevant queries, such as computing expectations over a spacetime region."*

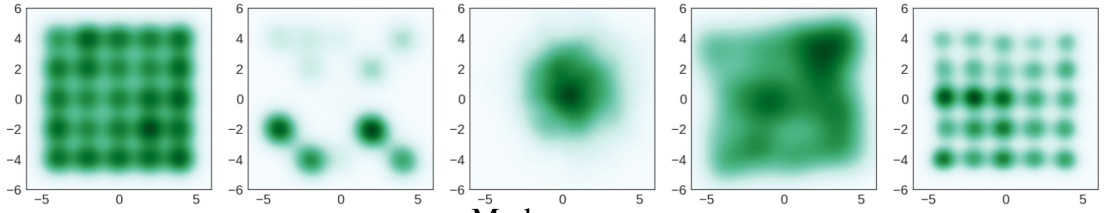


Figure 8.2: Density plots of the true data and generated distributions from different methods trained to fit a mixture of Gaussians arranged in a grid. *Source*: [Sri+17]

Mode collapse and mode average - There are two dominant issues in the fit of multi-modal distributions: mode collapse and mode average. They are illustrated in figure 8.2. Mode collapse results from the learning of some of the modes in the data distribution but missing some others. Mode averaging is the characteristic of a model that generates unrepresentative samples that are intermediary between several modes.

8.1 Maneuver Based Forecasting

Maneuvers make modes - A multinomial distribution over a set of maneuvers describes the probabilities of the different possible modes of the trajectories. Combining the maneuver probabilities with the uni-modal trajectory forecasting associated to each maneuver defines a multi-modal distribution.

Maneuvers make incorrect modes - Maneuvers are challenging to define. They must be generic because the set of maneuvers should not contain too many elements, but this set should fit all likely road scene situations so that the forecasted distribution does not miss too many modes. Using a large number of maneuvers makes the classification difficult. For example, the trajectory segment classification approach from [CH06] can be considered as a maneuver classification with many maneuvers represented by the trajectory segments. However, this approach does not produce competitive results. On the other end, defining a small number of possible maneuvers that properly define all the likely situations is very difficult.

Well-defined modes do not fit the data - [SWA14] and [DT18] precisely define a set of maneuvers for their model. In [SWA14], the applications are only made in simulations. Therefore, its ability to generalize to unforeseen situations that arise in real road-scenes is not tested. In [DT18], the maneuvers are defined with a hand-made procedure that uses the past and future trajectory. The forecasting model learns to fit this maneuver classification using only the past trajectory. It forecasts the future trajectory distribution as a weighted sum of the trajectories achieving each possible maneuver. It uses its maneuver classification output as weights for this sum. However, the chosen set of trajectories does not cover every likely situation, even on the NGSIM dataset that only contains highway scenes. We expose this problem in section 8.4.1 that builds a multi-modal constant velocity model for baseline comparisons.

Fuzzy modes are not tested - Another possibility is to use fuzzy concepts as the semantic forecast described in [SSS17]. However, because the semantic maneuver is relative to the situation, it does not define distribution modes in the trajectory space and thus is not applicable in the same way. For example, the semantic maneuver "follow the vehicle in front" is not a mode in the trajectory space because it could be performed with different velocities. This could be overcome, but to our knowledge, there is no well-performing models using semantic maneuvers in the literature. We suspect that it is due to the difficulties to adequately define semantic maneuvers with the same constraints: they should cover all possibilities without too many classes.

Learning maneuvers - We want to find a well-defined set of maneuver definitions that match the data distribution. To this end, the maneuvers can be learned using the data with an unsupervised clustering method of the trajectories [AMP10]. Once the maneuvers are defined, either by hand or using a clustering method, they can be used in a maneuver-based forecast. However, this forces an offline maneuver definition that cannot be relative to each specific scene described by the input data.

Forgetting maneuvers - We conclude that trying to fit the modes of the road scene distribution with a set of maneuver is probably not the best option. Therefore, we explore other models from the literature that are able to fit multi-modal distributions.

8.2 Generative models

In this section, we explore some models capable of fitting a given distribution from a set of samples Ξ. More precisely, the generative model should produce samples $\hat{\xi}_s$ distributed in the same way as the samples ξ_s from a given dataset. The extension from distribution fitting to forecasting is made in a second step by fitting a conditional distribution with the past observation conditioning the future trajectory distribution.

Distribution fitting - There are a wide variety of solutions to learn complex distributions. The recent approaches that encountered the best success are Variational Auto-Encoders (VAE) (and their variants), Normalizing Flows (NF), and Generative Adversarial Networks (GANs). These models are generative in the sense that they are capable of generating data following a complex distribution $\hat{\mathcal{D}}_{\text{data}}$ from the sampling of a simple known distribution $\mathcal{D}_{\text{latent}}$ called latent distribution. For a latent variable $l \sim \mathcal{D}_{\text{latent}}$ the generative model p_θ is applied to produce the posterior samples $\hat{\xi}_s = p_\theta(l_s)$. Thus, for a learned generative function, the random variable $\hat{\xi}$ follows the distribution $\hat{\mathcal{D}}_{\text{data}} \approx \mathcal{D}_{\text{data}}$.

VAEs - In the case of VAEs, the latent distribution is fixed, and the mapping is learned using an encoder-decoder architecture composed of two quasi-reciprocal functions. The encoder maps the data ξ_s to a latent variable $l|_{\xi_s}$, and the decoder maps back the latent sample l_s to $\hat{\xi}|_{l_s}$ that should be close to ξ_s. The decoder is trained to invert the encoder while enforcing the distribution in the latent space.

Normalizing flows - Normalizing Flows (NF) are a similar type of model as VAEs. They use the same architecture but apply an invertible mapping function such that the encoder and decoder are true inverses. Then only the distribution mapping has to be learned, which simplifies the learning process at the cost of a restricted expressivity of the model.

GANs - GANs do not require an encoder, and instead, use a discriminator that differentiates the real data from the generated data. The decoder is not "inverting" an encoder and is thus called generator. This architecture is represented in figure 8.3. The discriminator is trained to classify its input returning $p(\text{real}) \approx 1$ if its input is a real data ξ_s and $p(\text{real}) \approx 0$ if its input is a generated data $\hat{\xi}_s$. This output is interpreted as a probability estimation that the given input is a true sample from the dataset. The generator is trained to fool the discriminator while the discriminator is simultaneously trained to discriminate between generated data and real data, hence the adversarial nature.

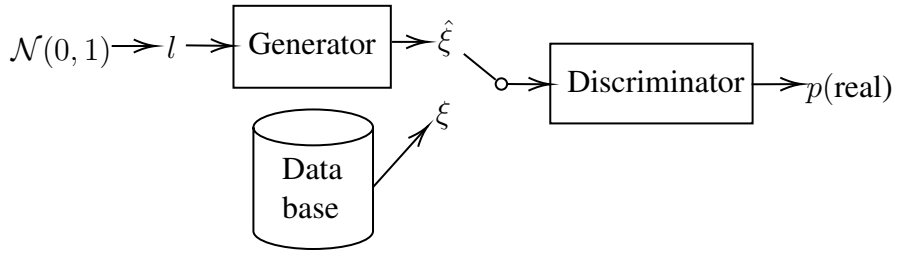


Figure 8.3: Illustration of a generic GAN architecture.

Why GANs? - The generator is trained to maximize the probability estimation for the generated samples $\hat{\xi}_s$ to be a real data sample. Why not merely supervise the generator with ξ since it is available? There are two reasons: The supervision forces the mapping between the latent space samples and the data samples, while the discriminator only requires similar global distributions. Secondly, we are unable to produce a satisfactory loss function that is needed for the supevision.

Difficult to supervise - A loss function is needed to minimize the difference between ξ_s and $\hat{\xi}_s$. This loss could be very difficult to express in the data space. For example, the distance between two trajectories could be the average distance between the trajectory waypoints. However, when two very different modes are likely, as illustrated with the two green path in figure 8.1, the loss should be low for all generated trajectory that is close to either one. This desired property is not compatible with the distance that we described. For a given situation, only one mode is known from the dataset and may be used in the supervision. Therefore, a loss measuring distances is unable to differentiate a good trajectory falling in a different mode from a bad trajectory that is very unlikely. GANs solve this issue by using a discriminator that avoids the supervision in the data space. The only supervision that GANs need is a simple classification error metric.

8.2.1 Generative Adversarial Networks

Srivastava *et al.* [Sri+17] state that the main issue with GANs is mode collapse and reproduce some examples like the one we showed in the introduction of this chapter, figure 8.2. Some solutions have emerged, such as unrolled GAN [Met+16] that solved this issue but produced mode averaging.

VEEGAN The solution proposed by Srivastava *et al.* that they call VEEGAN uses an idea from the VAE to force the latent space to encode for all images of the dataset without defining a loss in the data space. In VAE, the decoder is trained toward the reconstruction of the data from the latent encoding. This means that a loss must be defined in the data space to evaluate the reconstruction. This often leads to mode averaging because the common loss is a distance and the model minimizes the average distance between two modes leading to an intermediary output instead of choosing one or the other mode. In the GAN architecture, the loss is not expressed in the data space which helps to solve the mode averaging issue but trade it for mode collapse. The authors of VEEGAN claim that the best of both solutions can be reached, avoiding both mode average and mode collapse. Their idea can be explained from two view points a modified auto-encoder or a modified GAN.

Modified auto-encoder - An auto-encoder matches a data ξ_s to a latent vector l and decodes a latent sample l_s back into an approximate data $\hat{\xi}_s$. To do so, it must be supervised in the data space. The VEEGAN reverses this architecture as represented in figure 8.4.

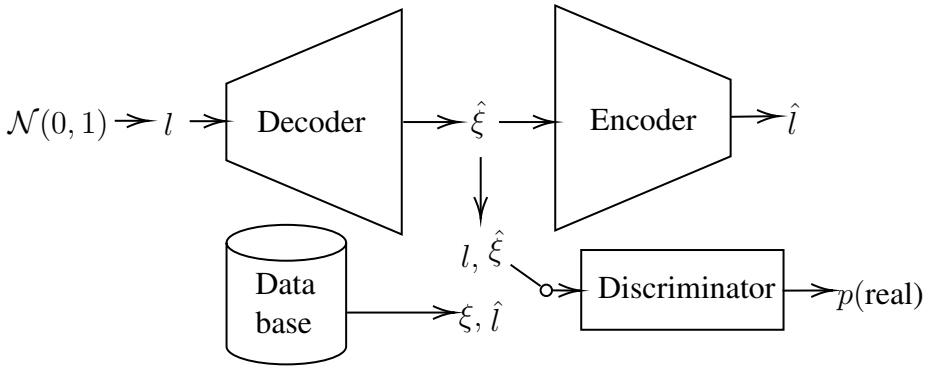


Figure 8.4: Illustration of the VEEGAN model seen as a reversed auto-encoder with a discriminator that forces the distribution in the decoded space.

It samples a given latent distribution, producing a latent sample l_s that it "decodes" into an approximate data $\hat{\xi}_s$. The data $\hat{\xi}_s$ is encoded back into the latent space, producing $\hat{l}_s$. The supervision is made in the latent space that can be chosen to be Gaussian. The L_2 loss is well adapted for a Gaussian latent distribution. However, the generated data distribution could be anything and must be forced to match the distribution of real data. Here an adversarial discriminator is used. It is trained to estimate the probability that a sample comes from the real dataset while the encoder is trained to fool it. However, its input is not only the data sample but also the latent sample, either the one drawn from the distribution to generate the data, either the encoding of the real data.

Modified GAN - Another way to understand the VEEGAN model is to see it as a modified GAN represented in figure 8.5.

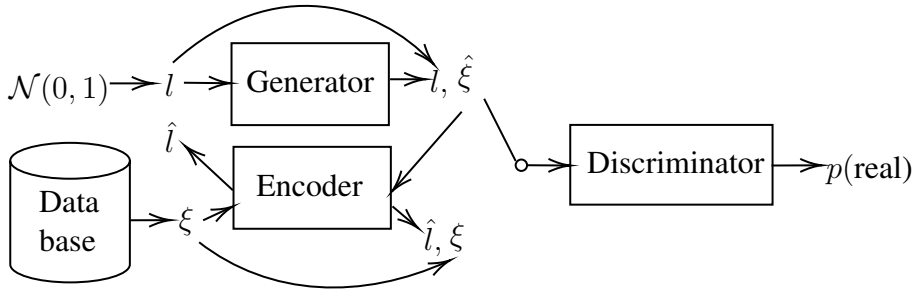


Figure 8.5: Illustration of the VEEGAN model seen as a modified GAN with an encoder that invert the generator.

The generator maps l following the prior distribution of the latent space, to the data space, producing $\hat{\xi}$. As in GANs, the discriminator is trained to discriminate the real samples ξ_s from the generated samples $\hat{\xi}_s$, but here it also uses the latent sample. The discriminator input is either $\hat{\xi}_s|_{l_s}$ and l_s either ξ_s and $l_s|_{\xi_s}$. Unlike VAEs decoders (VAEs

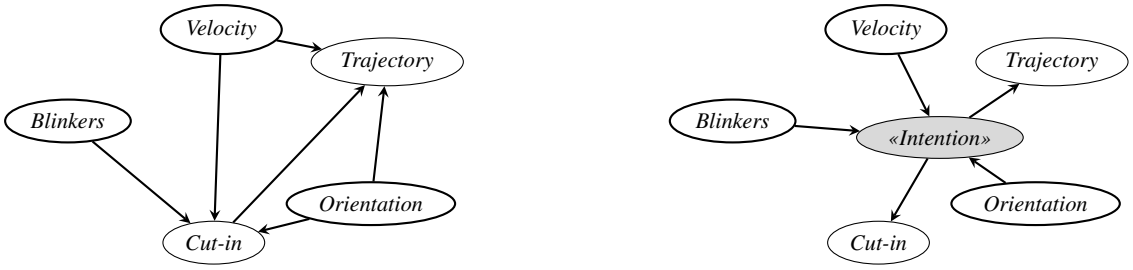
are the subject of the next section), the generator is *not* trained to reconstruct ξ_s from the encoded variable $l|_{\xi_s}$ because this would require a loss in the data space. Instead, it has two objectives: Firstly, as usual in GANs, it must fool the discriminator by maximizing the probability that the discriminator associates with $\hat{\xi}_s|_{l_s}$ for being a real sample from the dataset. Secondly, the latent variable $l|_{\hat{\xi}_s}$ that the encoder associates with the generated sample $\hat{\xi}_s|_{l_s}$ must be close to the original sample l_s that was fed to the generator. In other words, the encoder is trained to be an approximate inverse of the generator. The generator collaborates with it on this task. All three networks are trained simultaneously for all their objectives. The generator is defined for any sample of l following the latent distribution. The encoder is defined for all ξ_s in the dataset. Thus, if the encoder and the generator, noted g^{-1} and g, were the exact inverse of each other, mode collapse would be impossible. Indeed, missing a mode around the data ξ_{miss} would mean that $\xi_{\text{miss}} \neq g(g^{-1}(\xi_{\text{miss}}))$ which contradicts the invertibility. Since g is differentiable, this property should hold in a local neighborhood of ξ_{miss}. The neighborhood notion in the dataspace is not correctly defined because it would require a distance metric that we do not have. However, if we suppose the existence of such a distance d, we can define $\hat{g}^{-1}$ as a quasi-inverse of g with the existence of ϵ such that for any ξ_s in the data space $d(g(\hat{g}^{-1}(\xi_s)), \xi_s) < \epsilon$. Quasi-invertibility is sufficient to prevent mode collapse because it guaranties that for all ξ_s from the dataset, $d(g(g^{-1}(\xi_s)), \xi_s) < \epsilon$ and thus ξ_s is represented in the generated distribution. This motivates the two objectives of the generator of both fooling the discriminator and being invertible.

Fake noise - GANs seem promising to generate the full distribution of future states. However, with noisy input data, GANs will simulate the noise because it is needed to fool the discriminator. Thus, the generated distribution mixes the forecast uncertainty and the perception noise of the supervision. The confusion between uncertainty and noise prevents the network from learning to filter out the noise. As a result, the variance of the generated samples is artificially increased.

8.2.2 Variational Auto-Encoders

The Variational Auto-Encoder (VAE) is a generative model defined in [KW14]. Its purpose is to model the probability distribution $p(\xi)$ of the input data ξ. However, $p(\xi)$ is a complicated distribution with local maxima in its probability density. We do not know any reasonable prior that could describe this distribution.

Latent variables may simplify the distribution description - Suppose that we observe a vehicle in the next lane and that we want to estimate both the probability that it will make a cut-in and the probability density of its future trajectory. For this example, the observations used to estimate these probabilities are the velocity, the orientation, and the use of blinkers. In such case, a common approach to fit the distribution is to define a Bayesian network. A Bayesian network is defined as a causality graph and a table of probabilities for the random variables conditioned with their causes. We illustrate this example in figure 8.6a with a causality graph representing the situation that we described.



(a) Without latent variable

(b) With latent variable

Figure 8.6: Illustrative causality graph representing the inter-dependencies between random variables with and without a latent variable.

The causality graph 8.6a represents the joint distribution as:
p(Cut-in, Blinkers, Velocity, Trajectory, Orientation) =
$p(\text{Trajectory}|_{\text{Velocity, Cut-in, Orientation}})p(\text{Cut-in}|_{\text{Blinkers, Velocity, Orientation}})p(\text{Blinkers})\ p(\text{Orientation})p(\text{Velocity})$.

Marginal variable - The figure 8.6 illustrates that it is sometimes possible to simplify the distribution descriptions by introducing a marginal random variable l and considering the distribution $p(\xi|_l)$. In figure 8.6b, the latent variable is arbitrarily called "Intention." The Trajectory and the Cut-in only depend on this Intention. Moreover, the Intention only depends on the observations. We make the assumption that the distribution of l is simple. A common choice is $l \sim q(l) = \mathcal{N}(0, I)$ with a chosen dimension depending on the expected complexity of the variable. This allows the use of a simple prior for each sample l_s from $q(l)$ without restricting the model too much: $p(\xi|_{l_s}) = \mathcal{N}\left(\mu(l_s), \text{diag}(\sigma(l_s))\right)$. Contrarily to what the name "Intention" suggests in figure 8.6b, the meaning of the latent variable is not chosen. We only set its prior distribution.

8.2.2.1 Construction of the VAE

We model the data distribution using two parts: a function q, called encoder, and a function p, called decoder.

Encoder - The encoder q maps a data sample ξ_s to the parameters of a Gaussian distribution. This defines the posterior marginal distribution $q(l|_{\xi_s})$ in the latent space. The random variable $l|_{\xi_s}$ follows this distribution.

Decoder - The decoder p maps a latent sample l_s to the parameters of a posterior distribution $p(\xi|_{l_s})$ expressed in the data space.

Variational auto-encoder - The VAE is a neural network in two parts, enc_ϕ and dec_θ approximating the two functions q and p. We illustrate this model in figure 8.7.

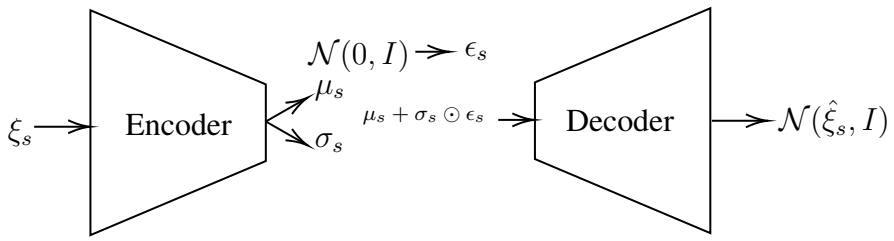


Figure 8.7: Illustration of the VAE model.

Working assumptions - We suppose that the latent marginal posterior distribution $q(l|_{\xi_s})$ is a multivariate Gaussian with a diagonal covariance matrix. We consider that the overall latent distribution is a standard normal distribution, thus $l \sim \mathcal{N}(0, I)$. Finally, we assume that the marginal posterior distribution $p(\xi|_{l_s})$ has a unit covariance: $\xi|_{l_s} \sim \mathcal{N}(\hat{\xi}_s, I)$. Under these priors, the encoder must estimate the means $\mu = (\mu_1, ..., \mu_D)$ and the standard deviations $\sigma = (\sigma_1, ..., \sigma_D)$. The decoder must estimate the means $\hat{\xi}_s$.

Encoder notations - We write $q_\phi(l|_{\xi_s})$ the Probability Density Function (PDF) of the distribution $\mathcal{N}(\mu, \text{diag}(\sigma))$ defined from the output of the encoder neural network enc_ϕ with parameters ϕ applied on the data ξ_s that approximates the probability density of the posterior distribution $q(l|_{\xi_s})$. The marginal likelihood of a latent sample l_s knowing $\xi = \xi_s$ is given by the PDF $q_\phi(l|_{\xi_s})$ applied at l_s that we write $q_\phi(l_s|_{\xi_s}) \overset{\text{def}}{=} q_\phi(l|_{\xi_s})(l_s)$.

Decoder notations - In the same fashion, $p_\theta(\xi|_{l_s})$ is the probability density function defined with the result of the decoder neural network dec_θ with parameters θ applied on the latent variable l_s that approximates the probability density of the distribution $p(\xi|_{l_s})$. Table 8.1 recapitulates our notations.

	Data	Latent
Prior distribution	Unknown	$\mathcal{N}(0, 1)$
Random variable	ξ	l
Sample	ξ_s	l_s
Marginal Posterior Distribution	$\mathcal{N}(\hat{\xi}_s, I)$	$\mathcal{N}(\mu_s, \text{diag}(\sigma_s))$
Marginal random variable	$\xi\|_{l_s}$	$l\|_{\xi_s}$
Marginal PDF	$p_\theta(\xi\|_{l_s})$	$q_\phi(l\|_{\xi_s})$
Joint PDF	$p_\theta(\xi, l)$	$q_\phi(l, \xi)$
Joint likelihood	$p_\theta(\xi_s, l_s)$	$q_\phi(l_s, \xi_s)$
Marginal likelihood	$p_\theta(\xi_s\|_{l_s})$	$q_\phi(l_s\|_{\xi_s})$

Table 8.1: The notations used in this section.

8.2.2.2 Maximizing a differentiable expression

Learning the the neural network parameters θ and ϕ is an optimization process aiming at maximizing the likelihood to observe the dataset of samples ξ_s given the priors and the neural network. This likelihood is written $L(\text{dataset}|_{\theta,\phi})$. Equations (8.1) reformulate the expression to maximize with respect to the parameters θ and ϕ.

$$\begin{aligned}
L(\text{dataset}|_{\theta,\phi}) &= \prod_{\xi_s \in \text{dataset}} L(\xi_s|_{\theta,\phi}) \quad \text{Input data are independent} \\
L(\xi_s|_{\theta,\phi}) &= \int p_\theta(\xi_s, l_s) dl_s \quad \text{Integration over marginal } l \\
\int p_\theta(\xi_s, l_s) dl_s &= \int p_\theta(\xi_s, l_s) \frac{q_\phi(l_s|_{\xi_s})}{q_\phi(l_s|_{\xi_s})} dl_s \quad \text{Multiplication by 1 trick} \\
&= \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right] \quad \text{The expectation to maximize}
\end{aligned} \tag{8.1}$$

The learning process is written:

$$\theta^*, \phi^* = \underset{\theta,\phi}{\operatorname{argmax}} \prod_{\xi_s \in \text{dataset}} \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right] \tag{8.2}$$

Lower bound maximization - For the maximization process, we want to compute the derivatives of the expression (8.2) with respect to the parameters θ and ϕ. However, it cannot be done analytically. Thus, we want to maximize a lower bound of the logarithm of this expression.

Lower bound expression - A lower bound is computed using twice the concavity of the log. First in equation (8.3) then in the Jensen inequality (8.4).

$$\underset{\theta,\phi}{\operatorname{argmax}} \prod_{\xi_s \in \text{dataset}} \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right] = \underset{\theta,\phi}{\operatorname{argmax}} \sum_{\xi_s \in \text{dataset}} \ln \left(\mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right] \right) \tag{8.3}$$

$$\ln \left(\mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right] \right) > \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\ln \left(\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right) \right] \stackrel{\text{def}}{=} \mathcal{L}(\theta, \phi; \xi_s) \tag{8.4}$$

This define the lower bound $\mathcal{L}(\theta, \phi; \xi_s)$.

New goal - Instead of maximizing the expression (8.2) that is intractable, the lower bound is maximised:

$$\hat{\theta}^*, \hat{\phi}^* = \underset{\theta,\phi}{\operatorname{argmax}} \sum_{\xi_s \in \text{dataset}} \mathcal{L}(\theta, \phi; \xi_s) \tag{8.5}$$

Hope for a tight bound - The lower bound maximization is considered to be an approximation of the true maximum, thus the function represented by the neural networks with these sets of parameters should be similar: $\text{dec}_{\hat{\theta}^*}, \text{enc}_{\hat{\phi}^*} \approx \text{dec}_{\theta^*}, \text{enc}_{\phi^*}$.

Reformulation - Kingma and Welling [KW14] write the lower bound using the Kullback-Leibler divergence:

$$\mathcal{L}(\theta, \phi; \xi_s) = \underbrace{\mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\ln p_\theta(\xi_s|_l) \right]}_{\text{Reconstruction}} - \underbrace{D_{KL} \left(q_\phi(l|_{\xi_s}) || q(l) \right)}_{\text{Marginal regularization}} \tag{8.6}$$

This equality is established in the next sections.

Interpretation - The first term is interpreted as the reconstruction quality and the second term as a regularization pushing the posterior PDF $q_\phi(l|_{\xi_s})$ toward the prior $q(l)$ that is the PDF of $\mathcal{N}(0, I)$.

Computing the lower bound - The reconstruction quality can be computed with a Monte-Carlo method, by sampling $l|_{\xi_s}$ using its posterior PDF $q_\phi(l|_{\xi_s})$. Using these samples the likelihood $p_\theta(\xi_s|_{l_s})$ can be computed and their accumulation is used to estimate the expectation. However, in practice with gradient-based optimization techniques, only one sample is sufficient at each forward propagation. The number of forward propagation during the optimization process is enough to ensure a dense sampling of the latent space. The KL divergence is computed analytically.

8.2.2.3 From the lower bound to interpretable losses

In this section, we prove the equality between the two lower bound formulations in (8.4) and (8.6).

$$\mathcal{L}(\theta, \phi; \xi_s) \stackrel{\text{def}}{=} \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\ln \left(\frac{p_\theta(\xi_s, l)}{q_\phi(l|_{\xi_s})} \right) \right]$$

First, $p_\theta(\xi_s, l) = p_\theta(\xi_s|_l) q(l)$ is injected in the definition. Then, it is reformulated using the linearity of the expectation and the properties of the log:

$$\mathcal{L}(\theta, \phi; x_s) = \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\ln \left(p_\theta(\xi_s|_l) \right) \right] - \mathbb{E}_{q_\phi(l|_{\xi_s})} \left[\ln \left(\frac{q_\phi(l|_{\xi_s})}{q(l)} \right) \right]$$

Using the definition of the KL divergence from [KL51] (noted I(1:2) in the original paper):

$$D_{KL}(q||p) \stackrel{\text{def}}{=} \int q(x) \ln \frac{q(x)}{p(x)} dx = \mathbb{E}_q \left[\ln \left(\frac{q}{p} \right) \right]$$

We obtain the result (8.6).

8.2.2.4 Computation of the KL-divergence loss with Gaussian encoder prior

We use the expression of the prior $l \sim \mathcal{N}(0, I)$ and the posterior $l|_{\xi_s} \sim \mathcal{N}(\mu(\xi_s), \text{diag}(\sigma(\xi_s))$. The dimension of l is noted D. Then, we use the equalities established in appendix C:

$$\int q_\phi(l_s|_{\xi_s}) \ln q(l_s) dl_s = -\frac{D}{2} \ln(2\pi) - \frac{1}{2} \sum_{d=1}^{D} \mu_d^2(\xi_s) + \sigma_d^2(\xi_s)$$

$$\int q_\phi(l_s|_{\xi_s}) \ln q_\phi(l_s|_{\xi_s}) dl_s = -\frac{D}{2} \ln(2\pi) - \frac{1}{2} \sum_{d=1}^{D} 1 + \ln \sigma_d^2(\xi_s)$$

This gives the value of the KL divergence in our case:

$$D_{KL}(q_\phi(l|_{\xi_s})||q(l)) = \frac{1}{2} \sum_{d=1}^{D} \mu_d^2(\xi_s) + \sigma_d^2(\xi_s) - 1 - \ln(\sigma_d^2(\xi_s))$$

8.2.2.5 Computation of the reconstruction loss with Gaussian decoder prior

The appendix C.2 of [KW14] considers that the decoder produces the parameters of a multivariate Gaussian with diagonal variance. We go further and consider a unit-variance posterior $\xi|_{l_s} \sim \mathcal{N}(\hat{\xi}_s, I)$. This is the most common choice in the applications of the VAE. Then the reconstruction loss is estimated by Monte Carlo sampling:

$$\frac{1}{L} \sum_{n=1}^{L} \frac{1}{2} ||\xi_s - \hat{\xi}_n||^2 + \frac{D_\xi}{2} \ln(2\pi) \tag{8.7}$$

With $\hat{\xi}_n$ the output of the decoder for the n^{th} sample of the latent variable $l|_{\xi_s}$, L the number of samples, and D_ξ the dimension of the data space. The constant $\frac{D_\xi}{2} \ln(2\pi)$ is discarded because it does not affect the gradient. What remains is half of the samples mean squared error.

8.2.2.6 Reparameterization trick

For the backward propagation to be computed, the encoder, the decoder and the latent sampling should be differentiable with respect to θ and ϕ. This is not a problem for either the encoder $\mathrm{enc}_\phi(l_s|_{\xi_s})$ nor the decoder $\mathrm{dec}_\theta(\xi_s|_{l_s})$ since they are neural networks. However it is a problem for the sampling of $l|_{\xi_s}$. This sampling is used between the two steps in the Monte-Carlo evaluation of the reconstruction term. Thankfully this problem is solved with the reparameterization trick from [KW14]. The trick is to externalize the sampling with an independent distribution and combine it with the encoder output to simulate the desired distribution. In our case this is $l_s = \mu(\xi_s) + \sigma(\xi_s) \odot \epsilon_s$ with the operator $\odot$ the element-wise product. $\mu(\xi_s)$, $\sigma(\xi_s)$ are the output of the encoder and ϵ_s is a sample from $\mathcal{N}(0, I)$. This is differentiable with respect to θ and ϕ and is a correct sampling from $\mathcal{N}(\mu, \mathrm{diag}(\sigma(\xi_s))$.

12pt In [YK19], Y. Yuan and K. Kitani clearly express some drawbacks of the VAE:

> *"VAE model with Gaussian latent codes are not guaranteed to be diverse for two reasons. First, the sampling procedure is stochastic, and the VAE samples can fail to cover some minor modes, even with a large number of samples. Second, since VAE sampling is based on the implicit likelihood function encoded in the training data, if most of the training data is centered around a specific mode while other modes have less data, the VAE samples will reflect this bias and concentrate around the major mode."*

8.2.3 Discrete representations

8.2.3.1 Supervised maneuvers

We have excluded the use of supervised maneuvers because it forces a global definition of the maneuvers and forbids the network to adapt the number of modes and their meaning to the current situation. However, maneuvers could be adapted to work with generative models. With VAEs and normalizing flows, if the maneuvers are supervised, the encoder must produce two outputs from the input ξ: it must find the latent distribution continuous parameters and learn to classify the current maneuver. With GANs, a separate network is used for maneuver classification. The generator uses three inputs: ξ, l, and the maneuver m. During learning, the maneuver is given by the supervision. Thus, it does not need to be sampled. During inference, a classifier makes an estimation of the maneuvers distribution and samples from that distribution are used. The generative models are expected to fit multi-modal distributions without this restriction and we still do not believe that using supervised maneuvers would produce better results.

8.2.3.2 Unsupervised discrete representation

We may want to enforce the multi-modality without supervising the model with some maneuvers. To this end, the model architecture can be adapted to produce a discrete internal representation explicitly. There are two reasons why this would be preferred over a supervised maneuver classification. Firstly, it may learn a discrete representation that is more suited to the dataset modes than the hand-defined maneuvers. Secondly, it allows

the model to express the discrete representation in a well suited encoded space. However, the gradient computation used for training is not well adapted to discrete representations.

Back propagation through argmax - The gradient computation is made difficult when a discrete feature vector is used and, in the case of generative models, when sampling from a discrete distribution conditioned on some input. In the second case, it is solved with the reparameterization trick. It was invented with Gaussian distributions by Kingma *et al.* [KW14] and extended to categorical distributions by Jang *et al.* [JGP16]. This was applied for human activity forecasting in [Gua+20]. The gradient computation through a discrete feature vector is addressed in two ways: Either the discrete representation is smoothed. This is usually done using a softmax. Either the gradient is redefined through the discrete operation. A "soft-hard" quantization is produced in [Agu+17] to allow a sound definition of the gradient in the quantization procedure. A similar method called VQ-VAE [VV+17] uses a hard vector quantization with a redefinition of the gradient called "straight-through estimator" discussed in [BLC13]. The soft-hard quantization relies on a softmax with a multiplicative coefficient accounting for the quantization "hardness." This trick is similar to the one used in the classification model developed in section 7.3.4.2. The vector quantization method uses a hard quantization and neglects the gradient through this operation. As long as the quantized value is close to the continuous one, this is a valid approximation.

Discrete space - Both [VV+17] and [Agu+17] apply a quantization procedure that minimizes the L_2 distance between the quantized space and the samples. This quantization procedure is similar to the one described by Lloyd *et al.* [Llo82] leading to the Lloyd's algorithm. This leads to similar results as a k-means algorithm but using only one sample at a time. The authors of [Agu+17] do not make an explicit connection with the VAE but have developed a very similar approach to the VQ-VAE [VV+17].

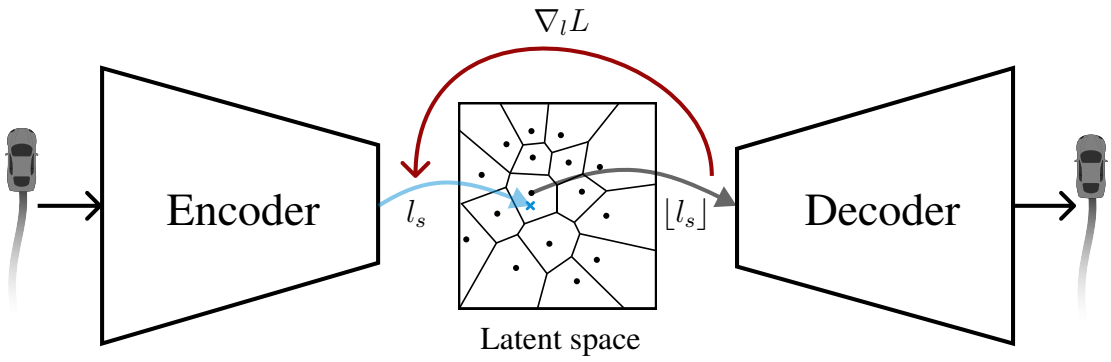


Figure 8.8: Illustration of the VQ-VAE model where l_s is the generated latent sample and $\lfloor l_s \rfloor$ is the nearest latent vector from the learned dictionary of latent vector.

VQ-VAE vs VAE - The main difference distancing the VQ-VAE from the classical VAE is its use of a categorical prior distribution in the latent space. The soft-hard procedure from [Agu+17] also use a categorical prior but enforces it in a way that produces instabilities in the learning process, making the VQ-VAE the preferred solution. Because the distance between the encoded vector to the quantized space must remain small, the VQ-VAE method must use a large number of quantized vectors.

The VQ-VAE diminishes the sampling coverage problem of VAEs and provides a sharper fit, but it often fails to represent the minor modes.

8.3 Generative Forecasting Models

Conditional latent sampling - With forecasting models, the mapping that the generative model should learn is not between a latent variable l and the data ξ but between the past trajectory $\mathbf{x}$ and the future trajectory $\mathbf{y}$. However, the past trajectory distribution $\mathbf{x}$ is only known from the dataset and cannot be sampled further. Thus, a prior latent distribution is chosen and samples $l|_{\mathbf{x}}$, conditioned on the past trajectory, are drawn. A particular case is to use an unconditional sample from a Gaussian distribution and concatenate it with $\mathbf{x}$. This means the decoder or generator must map a sample $l|_{\mathbf{x}}$ to the future $\mathbf{y}$.

Forecasting with VAEs and NFs - With VAEs, the only change is to the encoder that must map $\mathbf{x}$ instead of ξ to the latent variable l and to decode $\hat{\mathbf{y}}$ instead of $\hat{\xi}$. For the normalizing flows, since the encoder and decoder are true inverses, this cannot be applied. However, in [Bha+18], the authors used the normalizing flows to modify the latent distribution of a VAE. They apply their model, called CF-VAE, for multi-modal trajectory forecasting of agents and show that the modified latent distribution is multi-modal.

Forecasting with GANs - In the case of GANS, the latent variable l follows a known distribution, that is usually a standard Gaussian. Then, we define $l|_{\mathbf{x}}$ as the concatenation of l and $\mathbf{x}$. Thus, the generator is a function of the past observation $\mathbf{x}$ and a latent sample l that produces a trajectory $\hat{\mathbf{y}}|_{\mathbf{x},l}$. $\hat{\mathbf{y}}|_{\mathbf{x},l}$ should be difficult to differentiate from the actual future $\mathbf{y}$. The discriminator is a function of $\mathbf{x}$ and $\mathbf{y}$ or $\mathbf{x}$ and $\hat{\mathbf{y}}$ that outputs an estimated probability that the second input (either $\mathbf{y}$ or $\hat{\mathbf{y}}$) is the true future. This architecture is illustrated in figure 8.9.

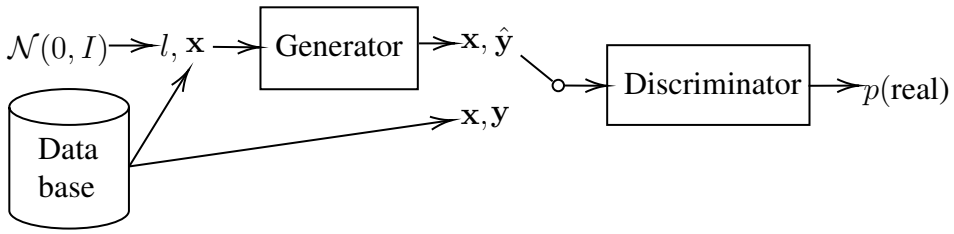


Figure 8.9: Illustration of a generative forecasting model.

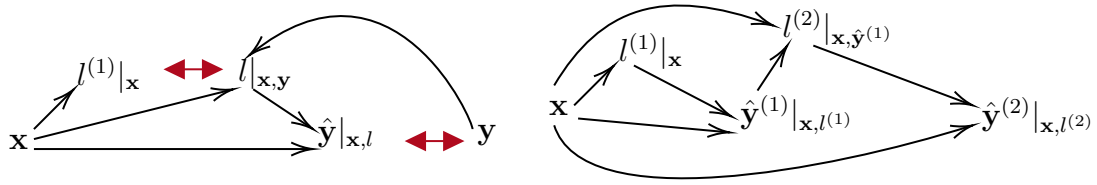
Applications in the literature - [Gup+18] and [Sad+18] make pedestrian trajectory forecasting using GANs. The generator module encodes the interactions with a max-pooling in [Gup+18] and with an attention mechanism in [Sad+18]. The discriminator produces a loss for the generator that helps it to learn a diversified forecast distribution. However, it must not be sufficient because in [Gup+18], the authors have used a technique to produce more diverse forecasts. They trained their model using the best of n_s samples at each iteration. This allows some of the samples to make errors as long as one of the n_s generated trajectory is close to the true future.

8.3.1 Forecasting with Multi-Modal Generative Models

In this section, we discuss the CVAE, CF-VAE, MT-VAE, and DSF-VAE models that are three variations of the VAE developed to avoid mode averaging and mode collapse. They consider an input $\mathbf{x}$ that is different from the expected output $\mathbf{y}$ and are especially well adapted to the context of trajectory forecasting.

8.3.1.1 CVAE

The Conditional-VAE model (CVAE) [SLY15] generates diversified samples and avoids mode averaging when the output is not a reconstruction of the input. It is precisely the situation in the forecasting task: the output $\hat{\mathbf{y}}$ is not a reconstruction of the input $\mathbf{x}$. In this case, generating $\hat{\mathbf{y}}$, usually made with $\hat{\mathbf{y}}_s = \text{dec}_\theta(l_s)$, uses the direct knowledge of the input $\mathbf{x}$ in addition to the latent sample l_s. Thus, the generation is now written $\hat{\mathbf{y}}_s = \text{dec}_\theta(l_s, \mathbf{x}_s)$.



a) Learning b) Inference

Figure 8.10: Illustration of the CVAE [SLY15] learning and inference process. The loss is computed using comparisons of the values linked with red arrows.

Using the future to forecast the future - The decoder could ignore the latent sample l_s altogether because it is just a noisy encoding of x_s. The model would become a direct forecasting model $\mathbf{y}_s = \text{dec}_\theta(\mathbf{x}_s)$ and would fail to represent the posterior distribution. Thus, some useful information is encoded in the latent variable. Both $\mathbf{x}$ and the output $\mathbf{y}$ are encoded into l using a second encoder rec_ψ called the recognition network. Thus, two latent values are produced by the CVAE: $l|_{\mathbf{x}}$ and $l|_{\mathbf{x},\mathbf{y}}$. During training, the whole trajectory $\mathbf{x}, \mathbf{y}$ is encoded to define the latent distribution that is sampled, forming $l|_{\mathbf{x},\mathbf{y}}$. Then, the decoder produces a forecast $\hat{\mathbf{y}}_s = \text{dec}_\theta(l_s|_{\mathbf{x},\mathbf{y}}, \mathbf{x}_s)$. If the future trajectory is multi-modal and noisy, the real future mode and the expected variance are encoded in the distribution, producing $l|_{\mathbf{x},\mathbf{y}}$. The decoder must rely on this value to produce a sample in the correct mode $\hat{\mathbf{y}}_s$.

When the future is not available - During inference, the true future $\mathbf{y}$ is not accessible. Thus, the sample l must also be estimated without the need for $\mathbf{y}$. This is why the encoder defines the distribution producing $l|_{\mathbf{x}}$. It is trained to match the distribution of $l|_{\mathbf{x},\mathbf{y}}$. A sample $l^{(1)}|_{\mathbf{x}}$ can be produced using the encoder during inference, when the future $\mathbf{y}$ is not available. This sample is used by the decoder to produce $\hat{\mathbf{y}}_s^{(1)} = \text{dec}_\theta(l^{(1)}, \mathbf{x})$.

Using a forecast to forecast - The forecast $\hat{\mathbf{y}}_s^{(1)} = \text{dec}_\theta(l^{(1)}, \mathbf{x})$ will likely fall into mode averaging because it is produced almost in the same way as a normal VAE. The difference is the use of $\mathbf{x}$ in the decoder. This mode averaging can be avoided. The

recognition network can produce the sample $l_s^{(2)}|_{\mathbf{x}_s, \hat{\mathbf{y}}_s^{(1)}}$ using the forecast in place of the true future. The recognition network should recognize and encode the mode that $\hat{\mathbf{y}}^{(1)}$ falls into and correct an eventual mode average. Finally, the decoder is used again to produce $\hat{\mathbf{y}}_s^{(2)} = \mathrm{dec}_\theta(l^{(2)}, \mathbf{x})$. It could be repeated, but Sohn *et al.* report that one iteration is enough. This inference process is represented in figure 8.10. The CVAEs avoids mode averaging, but some modes can still be missing from the generated data.

8.3.1.2 CF-VAE

Bhattacharyya *et al.* [Bha+19] learn conditional normalizing flows in the latent space of a CVAE to transform the Gaussian latent prior into a multi-modal prior that depends on the input $\mathbf{x}$. This conditional prior is multi-modal and helps the CVAE avoiding mode collapse. In their work, both the decoder of the CVAE and the prior distribution of the latent space are conditioned on the input $\mathbf{x}$. Optionally, they also use an unconditional VAE with only the prior distribution of the latent space conditioned on the input. Figure 8.11, illustrates this second version.

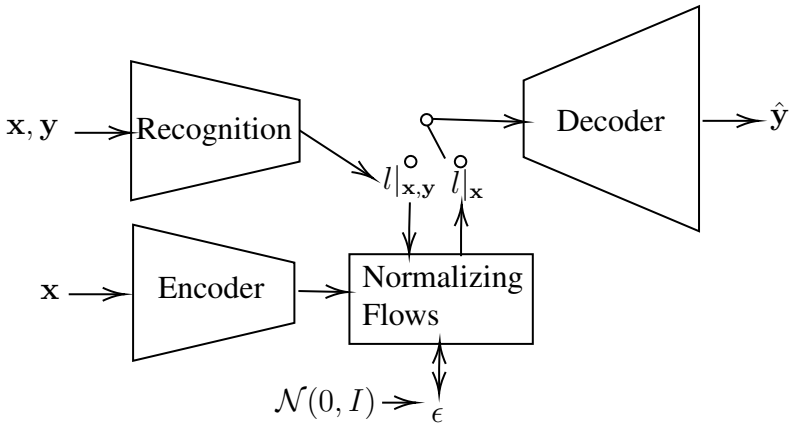


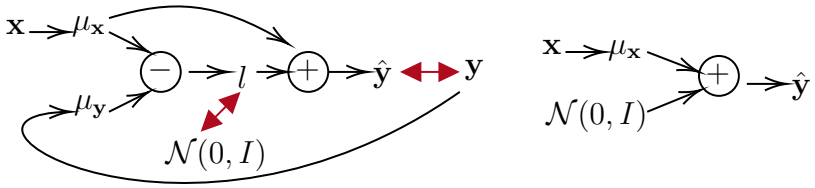
Figure 8.11: CF-VAE [Bha+19]. At inference, the Gaussian prior is sampled and the Normalizing Flows transform it into the multi-modal prior knowing the input.

Computing the lower bound - As shown in the section 8.2.2, the maximization of the dataset likelihood lower bound with VAEs relies on a reconstruction term and a KL divergence. The reconstruction term is unchanged by this new method. However, the KL divergence of the conditional latent distribution and the prior latent distribution must be computed with the modified prior. Thankfully, the authors of [Bha+19] simply express the KL divergence of the modified prior using the Gaussian prior and the Jacobian of the normalizing flows transformation. Thus, the lower bound to maximize can be computed.

The learned prior is multi-modal - The conditional prior (the distribution of $l|_{\mathbf{x}}$) is defined as the transformation of a Gaussian prior through a normalizing flow. This means that the inverse of the normalizing flow $\mathrm{NF}^{-1}_{\psi,\mathbf{x}}$ should transform the encoded latent distribution into a Gaussian distribution. Reciprocally, the direct normalizing flow $\mathrm{NF}_{\psi,\mathbf{x}}$ should transform a Gaussian distribution into the conditional latent distribution. Because the normalizing flow function itself depends on the input $\mathbf{x}$, it is able to produce a prior distribution conditioned on $\mathbf{x}$ from an unconditional Gaussian distribution. The authors call this model Conditional Flow Variational Auto-Encoder CF-VAE and apply it to trajectory forecasting on real data and show that the normalizing flows learn to map the Gaussian prior into a multi-modal latent distribution.

8.3.1.3 MT-VAE

Yan *et al.* [Yan+18] produce the Motion Transformation VAE, a variant of the VAE idea in the case where the expected output $\mathbf{y}$ is sampled from the same distribution as the input $\mathbf{x}$. It is precisely the case in trajectory forecasting with both $\mathbf{x}$ and $\mathbf{y}$ being trajectories from the same distribution. Two encoders (or twice the same) are used during learning: one to encode $\mathbf{x}$ into $\mu_\mathbf{x}$ and one to encode $\mathbf{y}$ into $\mu_\mathbf{y}$. The difference $\mu_\mathbf{y} - \mu_\mathbf{x}$, called transformation vector, is used to condition the latent variable sampling. The overall latent distribution is a Gaussian. The latent variable sample $l_s|_{\mu_\mathbf{y}-\mu_\mathbf{x}}$ is mapped back to a transformation vector that is added to $\mu_\mathbf{x}$. This sum is decoded into $\hat{\mathbf{y}}$.



a) Learning b) Inference

Figure 8.12: Illustration of the MT-VAE [Yan+18] learning and inference process. The loss is computed using comparisons of the values linked with red arrows.

Self-correction - Exactly as proposed with CVAEs [SLY15], at inference, it is possible to use the forecast to cycle through this procedure. $\hat{\mathbf{y}}$ is used instead of $\mathbf{y}$ to produce $\mu_{\hat{\mathbf{y}}}$. The resulting transformation vector $\mu_{\hat{\mathbf{y}}} - \mu_\mathbf{x}$ is used to produce a new forecast. The authors of [Yan+18] use this cycle during learning to enforce consistency.

MT-VQ-VAE - The MT-VAE could be simplified with the use of a quantized latent space. Both the Gaussian latent space and the mapping used to sample the transformation vectors can be replaced by simply using a quantized latent space. The quantized latent space would simply be a transformation vector dictionary without the need for mapping functions. Since both MT-VAE and VQ-VAE were published 3 months apart, the authors of [Yan+18] likely produced their work before the VQ-VAE existed. However, we did not pursue this idea further to focus our efforts on methods that do not require sampling.

8.3.1.4 DSF-VAE

Yuan *et al.* [YK19] provide a way to forecast diversified trajectories using a CVAE. The CVAE defines a latent distribution that is sampled a fixed number times N. The different modes should be identified with only N samples as long as N is larger than the number of modes. In addition to the encoder that produces the latent distribution parameters, the authors propose a Diversity Sampling Function (DSF) to sample N diversified latent vectors $l_s|_\mathbf{x}$ from the encoding of the past trajectory $\mathbf{x}$.

Rank of the similarity matrix - The authors define an $N \times N$ similarity matrix between the N trajectories decoded from the proposed latent samples. It is composed of the trajectory distances weighted with their quality. If the i and j trajectories are identical, the rows i and j are equal. This equality would lower the rank of the similarity matrix. Since there might be less than N modes associated with a given $\mathbf{x}$, the objective cannot be computed using the similarity matrix determinant that could be ill-conditioned. Instead, the objective is to maximize the expected rank of this matrix. Thus, the model is trained to find all the different trajectory modes that have a high estimated quality.

Balancing diversity and quality - Favoring diversified trajectory sets could lead the model to produce very unlikely outputs. Therefore, the diversity is balanced with a quality factor. The quality is a smoothed binary threshold indicating if a given latent sample is within the bounds of the smallest 90% probability latent sub-space. This high-probability latent sub-space is defined using the prior distribution of the latent variable.

Application in trajectory forecasting - The authors apply this method to trajectory forecasting with synthetic data and show a wide forecast diversity capturing all the modes. It is shown to produce more diversity than the other methods. The downside of the DSF-VAE is that the model requires to be sampled many times to estimate the variance around each mode. Moreover, the diversity objective that we described might not allow a correct estimation of the modes relative probabilities.

8.4 Gaussian mixture parametric forecasting

Baseline producing SOTA results - Following the recent literature, we have considered highly complex non-linear fit of the forecast distribution. However, the careful production of baselines by Schöller *et al.* [Sch+20] shows that a trivial pedestrian trajectory forecasting baseline actually outperforms the State-Of-The-Art (SOTA) results from the generative models such as Social-GAN [Gup+18] and SoPhie [Sad+18]. This simple baseline is a constant velocity model estimated from the last two positions and sampled by varying the initial velocity angle. The authors arbitrarily use a centered Gaussian with a standard deviation of 25° for the distribution of initial angle variations.

Questioning the results - The surprising performance of this simple model is not only due to the difficulty of evaluating multi-modal forecasts. It remains better than SOTA models when only the most probable trajectory is evaluated. Presumably, it results from both an over-estimation of the neural networks performances and the difficulty of evaluating multi-modal or sample-based models. This work regards pedestrian motion forecasting but its conclusions remain untested with road scene forecasting. Thus, as our first multi-modal baseline, we produce a multi-modal constant velocity model using a sampling of the initial state distribution as estimated by the Kalman filter defined in chapter 1.

8.4.1 Multi-Modal Constant Velocity Forecasting

Constant velocity generative model - Following the lead of Schöller *et al.* [Sch+20], we want to produce a multi-modal baseline model for vehicle trajectory forecasting. To produce different trajectories, the constant velocity forecast baseline described in chapter 1 is slightly modified. We want to explore several possible forecasts by sampling the estimation of the current state. To this end, we estimate the current state t_0 from the track history using the Kalman filter described in chapter 1. Then, we define two modifications of the current state: a heading angle variation θ and a velocity variation α_v. We use the dataset to compute the values of these parameters that would have modified the forecast such that the observed final position would match the forecasted final position. The distribution of the parameters that would have corrected the forecast defines a centered bivariate Gaussian that we call the exploration distribution. We have represented this distribution computed on a subset of the NGSIM dataset in figure 8.13.

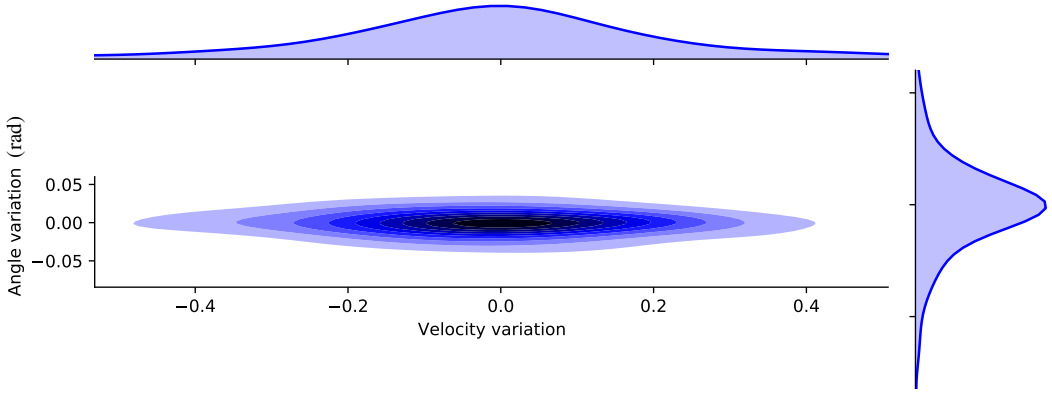


Figure 8.13: Distribution of velocity and heading angle variations from the estimated values on a random subset of NGSIM.

Angle and velocity factor are on the same scale - Because the angles are small, the variation of heading angle is proportional to the lateral displacement that it causes after a given time. The proportionality coefficient is the velocity. By construction, the longitudinal displacement caused by the velocity factor is also proportional to the velocity. Thus, there is no distortion from the exploration distribution to the displacement distribution that it produces, only re-scaling.

Sampling constant velocity forecasts - We define a different constant velocity forecast for each exploration sample (θ, α_v) from this distribution. The modified constant velocity forecast follows these three steps: Firstly, the initial state of the vehicle is estimated using the Kalman filter. Then, this state is modified using an exploration sample. The initial velocity is rotated by θ and multiplied with a factor $1 + \alpha_v$. Finally, the constant velocity forecast is produced with the modified initial state.

With this simple procedure, the constant velocity model can be sampled in the same fashion as in [Sch+20]. We modify the velocity modulus and not only the heading angle because with vehicle trajectories, the velocity varies a lot more than with pedestrian trajectories. Moreover, the heading angle is also less likely to change.

Exploration anchors - We want to explore diverse possibilities using only n_{mix} exploration anchors. Instead of sampling the exploration distribution, we quantize it with n_{mix} centroids. To do so, we use a K-means algorithm on a large amount of samples from the exploration distribution. These samples are used only during preprocessing to define the n_{mix} exploration anchors. No sampling is necessary at inference. The n_{mix} resulting centroids $c_1, \dots, c_{n_{\text{mix}}}$ are used as modes of the forecast distribution.

Anchor probabilities - With n_{mix} modes, the j^{th} centroid probability estimation p_j is the probability of the Voronoi cell associated to c_j for the exploration distribution represented with the centered Gaussian PDF G_{PDF} with covariance Σ:

$$V_{\text{cell}}(c_j) = \{z \in \mathbb{R}^2 \ / \ \forall i \neq j \in [\![1, n_{\text{mix}}]\!] \ \|z - c_j\| \leq \|z - c_i\|\}$$
$$p_j = p(V_{\text{cell}}(c_j)) = \int_{V_{\text{cell}}(c_j)} G_{\text{PDF}}(z, \Sigma) dr$$

We estimate these probabilities as the proportion of samples associated with each centroid. This is equivalent to a Monte Carlo estimation. Since the Voronoi cells are a partition of $\mathbb{R}^2$, $\sum_{i=1}^{n_{\text{mix}}} p_s = 1$.

Table 8.2: Evaluation of the multi-modal constant velocity forecast with $n_{\text{mix}} = 6$, $\sigma_\theta = 0$ and $\sigma_{\alpha_v} = 10\%$

Time horizon	1s	2s	3s	4s	5s
pRMSE (m)	0.79	1.85	3.21	4.83	6.71
RMSE (m)	1.73	3.58	5.59	7.77	10.08
minRMSE (m)	0.88	1.54	2.15	2.84	3.82
pFDE (m)	0.47	1.24	2.27	3.53	5.00
FDE (m)	1.40	2.87	4.43	6.11	7.89
minFDE (m)	0.64	1.08	1.42	1.74	2.21
NLL	2.21	3.08	3.89	4.59	5.22
MR	0.02	0.13	0.21	0.25	0.29
Sim	5.63	1.65	0.92	0.71	0.57

Anchor covariance contributions - The constant velocity forecast predicts an error covariance matrix. When several modes are considered, each mode represents a part of the distribution described by this error covariance. Thus, the error covariance around each mode is expected to be lower than the error covariance for one central mode. We compute a covariance coefficient for each mode as the fraction of the standard deviation inside the Voronoi cell and the exploration standard deviation:

$$s_j = \sqrt{\frac{\mathbb{E}_{V_{\text{cell}}(c_j)}\left[\left(X - \mathbb{E}_{V_{\text{cell}}(c_j)}[X]\right)^2\right]}{\mathbb{E}\left[X^2\right]}}$$

We estimate these coefficients by Monte Carlo, using the preprocessing samples.

Multi-modal constant velocity forecast - The constant velocity forecast described in chapter 1 with optimized parameters can be used directly to produce multi-modal forecasts. For a forecast $f(\mathbf{x}) = (\mathbf{y}, \Sigma)$, the associated multi-modal forecast is $f(\{\mathbf{x}_m\}_{m\in[\![1,n_{\text{mix}}]\!]}) = \{(\mathbf{y}_m, s_m^2\Sigma, p_m)\}_{m\in[\![1,n_{\text{mix}}]\!]}$. It uses the same forecasting function f that leads to the same error covariance Σ independently of $\mathbf{x}$. Therefore, this fully defines a multi-modal constant velocity forecast in the form of a Gaussian mixture, and the evaluation procedure defined in chapter 3 can be applied to evaluate it. In table 8.2, we present the results for a constant velocity forecast with $n_{\text{mix}} = 6$ modes exploring a distribution defined with $\sigma_\theta = 0$ and $\sigma_{\alpha_v} = 10\%$. No training is necessary to compute the results with a new set of parameters. Thus, we obtained these parameters with a simple grid search over σ_θ, and σ_{α_v} using a fixed number of modes $n_{\text{mix}} = 6$.

Evaluation - The results show a sharp decrease in the miss rate showing that the modes explore a state-space that fits the data. After a parameter grid search, the best exploration angle that we could find is 0. It shows how little effect the lateral maneuvers have on the NGSIM dataset. The similarity values are high. It means that this model is elongating the velocity distribution more than it is producing new modes. The NLL at 5s is 4.46 for one mode and 5.22 with the 6 modes. This could probably be improved with a better estimation of the covariance coefficients s. These parameters could also be optimized using the training data. However, besides the NLL, this is a satisfying baseline for multi-modal forecasting. Remarkably, the miss rate at 5 seconds is 30%, while the same number of modes with the Convolutional Social Pooling from [DT18] shows a 44% miss rate. This result is a strong evidence against the use of hand-defined

maneuvers as modes.

8.4.2 Stable Gaussian Mixture Forecasts

In [Bis94] Bishop describes Mixture Density Networks (MDN). It is a neural network that outputs a Gaussian mixture. A Gaussian mixture probability density function is defined as a convex sum of Gaussian Probability Density Functions (PDF). With a Gaussian PDF expressed as:

$$G_{\text{PDF}}(\mu, \Sigma)(z) = \frac{1}{(2\pi)^{D/2} \left|\Sigma\right|^{1/2}} \; e^{-\frac{1}{2}(z-\mu)^\top \Sigma^{-1}(z-\mu)} \tag{8.8}$$

The Gaussian mixture PDF is written, with $p_m > 0$ for all m and $\sum_{m=1}^{n_{\text{mix}}} p_m = 1$:

$$\text{GM}_{\text{PDF}}(\{\mu_m, \Sigma_m, p_m\}_{m\in[\![1,n_{\text{mix}}]\!]}) = \sum_{m=1}^{n_{\text{mix}}} p_m G_{\text{PDF}}(\mu_m, \Sigma_m) \tag{8.9}$$

MDN activation - In his article, Bishop defines the MDN model, its output activation function, the use of the NLL loss in the learning process, and computes its derivative with respect to the network outputs. For a neural network producing a mixture of n_{mix} Gaussians with $n_{\text{mix}} \times 6$ output neurons. We note o_i the i^{th} coordinate of the output value for one of the mixture components. The activation function defining the parameters of a bivariate Gaussian mixture is written as follow:

$$\begin{aligned}
&\{(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho, p)\}_{m\in[\![1,n_{\text{mix}}]\!]} \\
&= \text{activation}(\{o_1, o_2, o_3, o_4, o_5, o_6\}_{m\in[\![1,n_{\text{mix}}]\!]}) \\
&= \left\{(o_1, o_2, e^{\frac{o_3}{2}}, e^{\frac{o_4}{2}}, \tanh(o_5), \underset{m\in[\![1,n_{\text{mix}}]\!]}{\text{Softmax}}(o_6))\right\}_{m\in[\![1,n_{\text{mix}}]\!]}
\end{aligned}$$

Instability - Since its description in [Bis94], the MDN has become a standard technology and is used in various applications. In some applications [HN99; Gra13; Rup+17; CR18], the optimization procedure of MDNs becomes unstable. This might be why MDN does not seem to be the favored approach in trajectory forecasting applications. However, we can avoid the instabilities at low cost with the bivariate Gaussian that we use. Let us recall the NLL loss computation from chapter 3:

$$\begin{aligned}
\text{NLL}_k^{(i)}(dx, dy, \Sigma) = &\frac{1}{2} \underbrace{\frac{1}{(1-\rho^2)}\left(\frac{d_x^2}{\sigma_x^2} + \frac{d_y^2}{\sigma_y^2} - 2\rho\frac{d_x d_y}{\sigma_x\sigma_y}\right)}_{(z_k-\hat{z}_k)^T\Sigma_k^{-1}(z_k-\hat{z}_k)} \\
&+ \underbrace{\ln\left(\sigma_x\sigma_y\sqrt{1-\rho^2}\right)}_{\ln(\sqrt{|\Sigma_k|})} \\
&+ \ln(2\pi)
\end{aligned} \tag{8.10}$$

The estimation of this expression is numerically unstable for very low values of σ_x or σ_y or if ρ is too close to 1 or -1. The minimal values of σ_x or σ_y mean the model has maximal confidence in its current forecast.

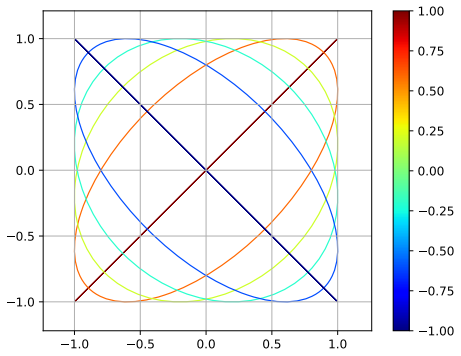


Figure 8.14: Representation of the covariance ellipses for a Gaussian distribution with standard deviations $\sigma_x = \sigma_y = 1$ and varying correlation coefficient $\rho \in [-1, 1]$

The lowest standard deviation - When not only the fitting problem but also the underlying physical world is considered, it becomes apparent that the numerical instabilities arise with values that are not realistic in the physical application. In the physical problem, a precision under a few centimeters for a vehicle position is useless and unrealistic because the perception accuracy (even filtered) is more limiting. For long-time forecast horizons (more than one second), with these conditions, making forecasts with a standard deviation below a few centimeters cannot be achieved and would not be useful. Thus, we can set the minimum value for σ_x and σ_y at 1 cm without any loss of generality. This prevents both numerical instabilities and overfitting.

The extreme correlations - The same reasoning goes for the correlation coefficient ρ. Figure 8.14 shows the ellipse delimiting the one sigma surface of a bivariate Gaussian with $\sigma_x = 1$, $\sigma_y = 1$ and $\rho \in [-1, 1]$. The smallest radius of the ellipse is given by the square-root of the smallest eigenvalue of the covariance matrix. For $\rho = -1$ or $\rho = 1$, this value is 0. For the same reason we set a 1cm minimum standard deviation, we want to limit the minimum radius to 1cm. The minimum covariance eigenvalue is :

$$\lambda_{\text{min}} = \frac{1}{2}\left(\sigma_x^2 + \sigma_y^2 - \sqrt{(\sigma_x^2 - \sigma_y^2)^2 + 4\rho^2\sigma_x^2\sigma_y^2}\right)$$

If we set a threshold ϵ=1cm, $\lambda_{\text{min}} \geqslant \epsilon^2$ becomes:

$$1 - \rho^2 \geqslant \epsilon^2 \left(\frac{\sigma_x^2 + \sigma_y^2 - \epsilon^2}{\sigma_x^2\sigma_y^2}\right) \tag{8.11}$$

However, because there is the expression $\frac{1}{(1-\rho^2)}$ in the equation (8.10), the numerical instabilities may occur for small values of $1 - \rho^2$. The inequality (8.11) is not enough to guaranty stability for large values of σ_x and σ_y. This can also be solved by considering the meaning of large values of σ_x and σ_y.

The highest standard deviation - An estimated standard deviation value greater than 100m means that there is more than 60% probability that the forecast is off by more than 50m. 50m is about ten times the width of a lane. This means that the system is unable to make a reasonable forecast. However, setting a 100m threshold on the standard deviation estimation prevents the system from escaping the data that it is unable to fit. In our experiments, we have indeed noticed that setting a maximum standard deviation threshold during training of the models limits their performance. Producing a very

large standard deviation is a viable option to almost set to 0 the loss that is due to the forecast error. This only produces a logarithmic loss associated with the overestimated standard deviation. Thus, we only use the 100m maximum standard deviation in the equation (8.11) computing the threshold for ρ.

Stability - Using the thresholds we defined, we do not observe the instabilities described in [Gra13; Rup+17; CR18; Mak+19]. The interpretation of the results remains unchanged because the values out of the chosen bounds are either over-confident values or interpreted as too uncertain to be meaningful. These conditions can be forced simply by imposing bounds in the loss function.

MDN with pre-defined modes - As with the generative models, the Gaussian mixture models can be used with predefined maneuvers, unsupervised learning of maneuvers, or direct regression. In [DT18], predefined maneuvers are used to train an MDN with a "Winner Takes All" (WTA) loss. This means that only the forecast matching the true maneuver is used for the gradient update. In [Cha+19a], an MDN model is produced to forecast vehicle trajectories. The multi-modality is obtained via sampling of anchor trajectories. Anchor trajectories are predefined normalized trajectories that are not trained with the model. They are produced with a k-means algorithm performed on the normalized future trajectories of the training dataset. The model produces a transformation and a probability estimation for each sampled anchor trajectory. The transformed trajectories constitute the modes of the forecast. It allows this method to sample only relevant items and achieve good performances with only a few samples. Covariance matrices describing the model uncertainty are associated with each forecast position such that the output is a Gaussian mixture. This model incorporates the context using a rasterized image representation of the scene.

MDN with map dependent modes - In [KS20], the same core idea is used but replacing the anchor trajectories with the centerlines of the surrounding lanes. This allows the model to use the map information without a rasterized image input. However, this model does not account for the interactions between agents. This is not a limitation of the method that could certainly be modified to account for the interactions. Kawasaki *et al.* use a hybrid model-based forecast that relies on a neural network forecast to feed virtual measurements to a Kalman filter with a bicycle model. This method produces Gaussian mixture outputs using a hybrid estimation of the error covariance. It uses the Bayesian update through the Kalman filter to incorporate the estimations from the MDN. This is not in the same fashion as our work in [Mer+19] that used the neural network to feed actions to the Kalman filter instead of virtual measurements.

In [Gao+20], the anchor trajectories are suggested to produce a multi-modal output, but only a uni-modal forecast is evaluated. This model uses graph representations for the map features and a self-attention mechanism accounting for the interactions between the agents and with the map. This allows this method to avoid the map rasterization without relying explicitly on predefined anchors or map anchors. The map anchors may miss some modes if the map is not perfect or if a vehicle is not following the road network correctly. These cases occur in the Argoverse dataset [Cha+19b] where some vehicles make U-turns or simultaneously turn and change lane, following paths that are not connected on the HD-map road network. In the article [Gao+20], an extensive comparison with convolutional networks using rasterized inputs is performed and shows that the attention-based mechanism outperforms these methods.

From the literature review that we have made with the MDN models, several facts are striking:

- The first one raised in [Sch+20] is that no thorough multi-modal baseline has been produced, and only comparisons between complex models are used in the current literature. The original study of pedestrian trajectory forecasting showed that constant velocity models actually perform better than SOTA models. We have reproduced this approach with vehicle trajectory forecasting and showed that on the NGSIM dataset, the constant velocity results are largely underestimated when compared with multi-modal SOTA results.
- The second one is the lack of immediate solutions to the MDN instabilities. The literature presents many solutions to avoid mode collapse, mode averaging, and instabilities in MDNs without the experiments showing that these difficulties arise when the MDN is merely restricted to the physical problem at hand. It seems that there is no instability with vehicle trajectory forecasting, and therefore, the simplest solution should be adopted: not using anchors, not using maneuvers, not using Winner Takes All (WTA) loss, and directly training the MDN with the NLL loss as suggested 26 years ago in [Bis94].
- A third striking fact is the lack of consideration for the practicability of the proposed solutions. The development of a new idea may not be efficient enough to be embedded in the vehicle at first. However, the methods that need to sample many times a complex model will always be computationally expensive, especially when very unlikely events must be forecasted. Some paper involving such methods present their work as part of the effort toward autonomous driving while disregarding completely the number of samples necessary to reach a satisfying certainty level allowing a human life to be involved. While safety has not been the primary concern in our work, the forecasting methods should at least lead to future models that could be remotely compatible with a safety validation system. This means that even when safety should not restrict too much the present research, keeping a slim chance of being compatible with a safe system should remain a concern.

The next chapter presents a technical description of our approach involving everything that has been discussed in the previous chapters: human driven vehicle motion as a multi-modal forecast with agent interactions jointly for all vehicles in the road scene involving the knowledge of a map. Our model is applied to several datasets and is compared to other solutions.

Chapter 9

A complete forecasting model

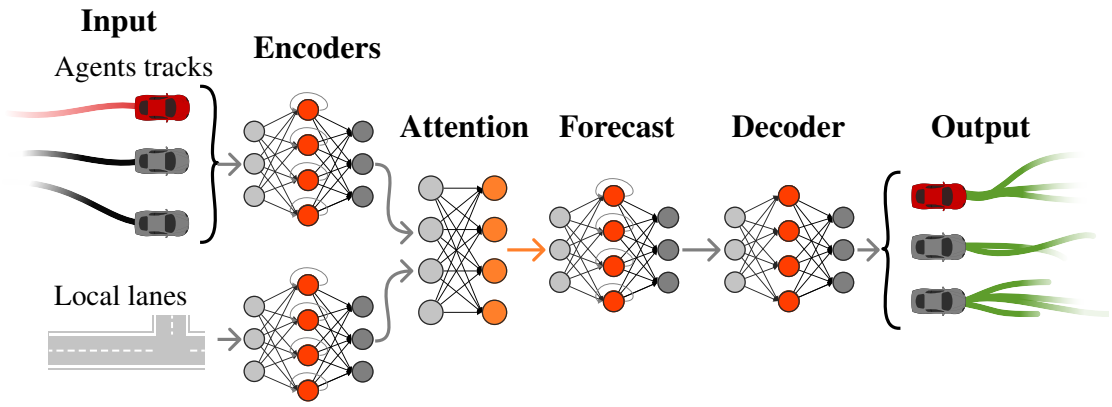


Figure 9.1: Illustration of our neural network architecture.

This chapter presents our proposition for a model implementing the forecasting capabilities presented in the previous chapters. It is based on our paper [Mer+20] with some later additions. It uses convolutional and LSTM layers to encode the trajectories and multi-head attention layers to account for the interactions. Another LSTM layer produces the sequence that three layers of 1×1 convolutions decode into the output. This is broadly illustrated in figure 9.1.

Before the decoder, we add a re-encoding block. It encodes again the sequence and applies a second attention layer. Then, the forecasting LSTM layer is used again to form back the sequence.

The output is a sequence of Gaussian mixtures that account for the multi-modal aspect of the future trajectories distribution.

We evaluate the results with the procedure described in chapter 3. We also used this model to participate in the Argoverse [Cha+19b] motion forecasting competition and won it twice, at NeurIPS 2019 then at CVPR 2020. We study some aspects of this network with ablations and result interpretations.

9.1 Building the Forecasting Model

In this section, we give a technical presentation of the model to allow its reproduction. The goal is to define the computation it performs. We discuss our intuitions and give our interpretations in the next sections.

9.1.1 The Forecasting Function

The inputs are road scene sequences. They are composed of the sequences of all vehicle $z = (x, y)$ positions and centerline pieces of the surrounding lanes. At each time t_0, we consider an observation history with a fixed observation frequency and a fixed maximum number of observations n_{hist}. The past trajectory is written $\{z_{k,\text{veh}}\}_{k=-n_{\text{hist}}+1\ldots0,\text{veh}=1\ldots n_{\text{veh}}}$. We center the coordinate system on the ego vehicle position at t_0, with the x axis aligned with the ego velocity. The lane pieces are also sequences of coordinates with n_{discr} 2D points describing centerline pieces with (x, y) coordinates in the same coordinate system. There are dynamic numbers of vehicle tracks n_{veh} and lane pieces n_{lane}.

During training, batches of road scene sequences are regrouped in a single tensor. However, the number of vehicles and lanes may vary between the different road scene sequences. Moreover, some of the vehicles are only observed partially during the sequence. In the batched data tensor, the maximum sequence length and the maximum number of objects must be used for the tensor dimensions. The tensor is padded with zeros to fill the missing pieces. A binary mask indicating the actual data is also given as input to exclude the padding. The mask is used in the supervision and in the normalization layers.

The outputs are n_{pred} sequences of Gaussians mixtures for each vehicle. They are expressed with sextuplets $(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho, p)_{\text{veh},k,m}$ for each vehicle veh, at each forecast step k and for each mixture component m. Each component is defined by a Gaussian law and a probability coefficient $(\mathcal{N}((\hat{x}, \hat{y}), \Sigma), p)$ with

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

defining the covariance matrix, and p the mixture weight such that for n_{mix} components, $\sum_{m=1}^{n_{mix}} p_m = 1$.

The forecasting model is a set of functions pred_θ : inputs $\rightarrow$ outputs. The inputs and outputs sets are defined with the cartesian products:

$$\begin{aligned}
\text{inputs}_{\text{veh}} &\in \left(\mathbb{R}^2\right)^{n_{\text{hist}} \times n_{\text{veh}}} \\
\text{inputs}_{\text{lane}} &\in \left(\mathbb{R}^2\right)^{n_{\text{discr}} \times n_{\text{lane}}} \\
\text{outputs} &\in \left(\left(\underbrace{\mathbb{R}^2}_{\hat{x},\hat{y}} \times \underbrace{\mathbb{R}_+^2}_{\sigma_x,\sigma_y} \times \underbrace{[-1,1]}_{\rho} \right)^{n_{\text{mix}}} \times \underbrace{\Delta^{n_{\text{mix}}}}_{p} \right)^{n_{\text{pred}} \times n_{\text{veh}}}
\end{aligned}$$

$\Delta^{n_{\text{mix}}}$ is the n_{mix} element probability simplex:

$$\Delta^{n_{\text{mix}}} = \left\{ (p_1, \ldots, p_{n_{\text{mix}}}) \in [0,1]^{n_{\text{mix}}} \;\middle|\; \sum_{m=1}^{n_{\text{mix}}} p_m = 1 \right\}$$

The pred_θ function set is defined as a neural network with weights θ. pred_θ is equivariant with permutations in the ordering of the vehicle list and is defined for any numbers of vehicles n_{veh} and any number of time steps n_{pred}.

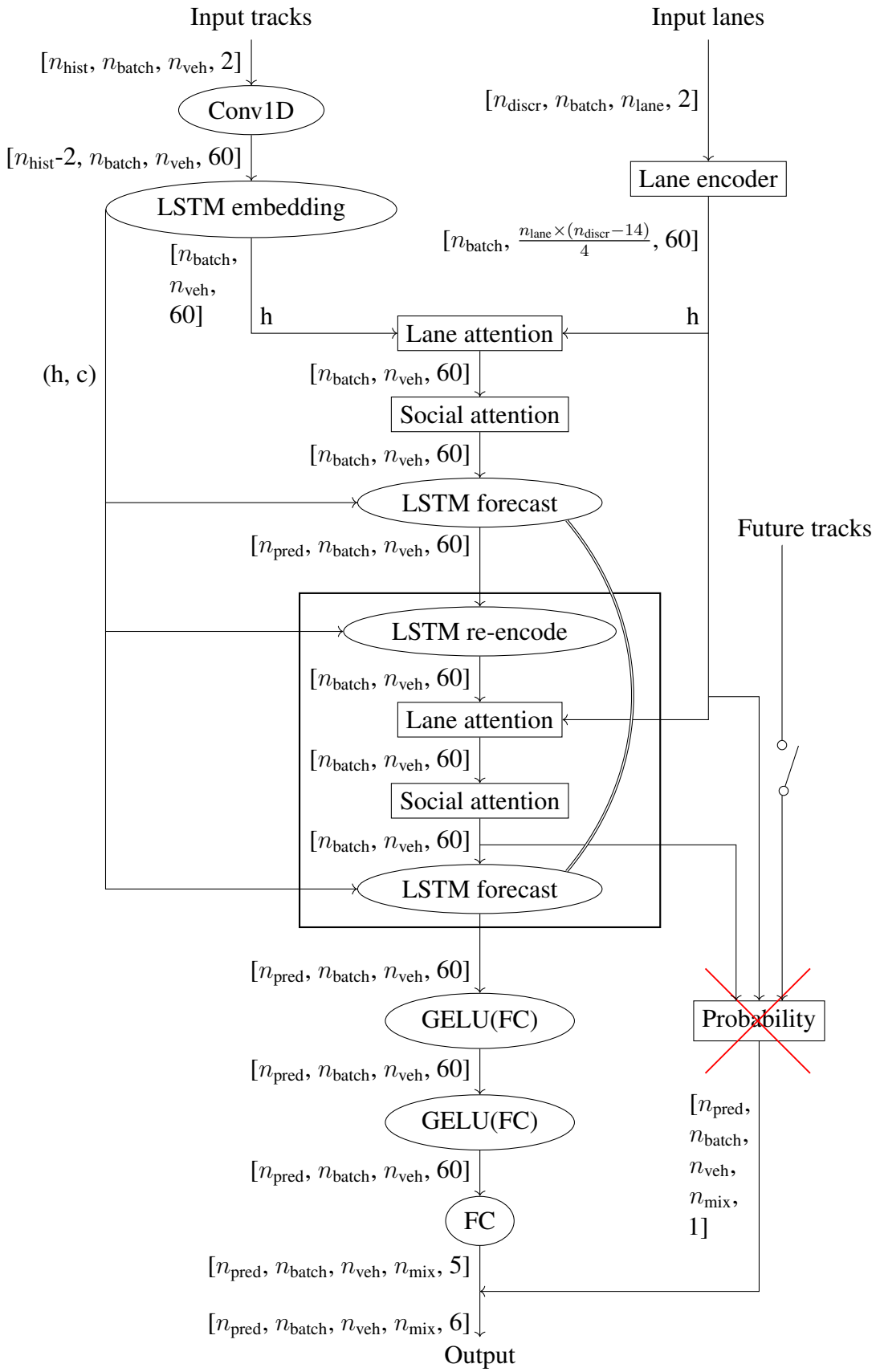


Figure 9.2: Motion forecasting model, the "LSTM forecast" is used twice with the same weights. The tensor sizes are written in brackets. FC stands for Fully Connected.

9.1.2 The Neural Network Architecture

The architecture is represented from top to bottom in figure 9.2. The size of the tensor appears between brackets. Since we use mini-batches during training, we also write the batch dimension.

The network encodes the lanes with an encoder architecture detailed in section 9.1.2.1. On this graph, using the tensor shape, we can read that the first convolutional layer uses 60 filters without padding. We chose 60 filters with a kernel size of 3 with bias. These hyper-parameters values are hand-chosen after some tests. An LSTM layer, used as many to one, embeds the input track sequences into feature vectors without a time dimension. The encoded lane and track features interactions are computed in the attention blocks detailed in section 11.8.1.3. A one to many LSTM layer forms the forecast sequence in the encoded feature space. An optional re-encoding block, detailed in section 9.1.2.4 is used on the sequence of feature. Then, the sequence is decoded with 3 fully connected layers. Either the mixture components probabilities are computed with a probability block detailed in section 9.1.2.6, either it is produced by the main branch along with the other Gaussian mixture parameters.

9.1.2.1 Lane Encoder

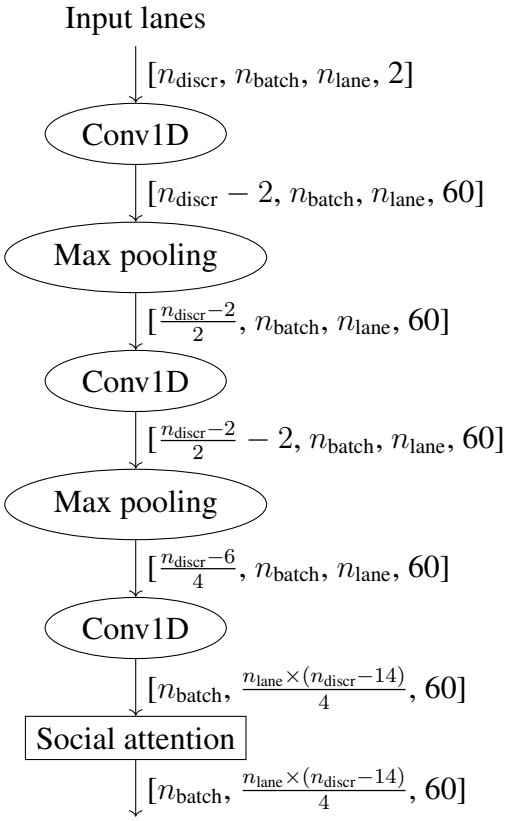


Figure 9.3: Schematic representation of the lane encoder block. The convolutions are all using a kernel size 3 without padding. The pooling layers use a kernel size 2.

The lane encoder is composed of three 1D convolutions with size 3 kernels and no padding. A max-pooling layer of size 2 follows each of the two first convolutions. We

merge the lane pieces dimension and the list of coordinates of each lane piece into a single list of features. A multi-head self-attention layer links these coordinate features with each other. It uses the "Social attention" architecture.

9.1.2.2 Attention Block

The attention block performs the multi-head attention that is central to our model. It is illustrated in figure 9.4. Each attention head mixes together the features relating to all vehicles. Otherwise, in the case of the lane attention, it mixes the features relating to the lanes with those of the vehicles. The computations involved in these attention heads are described in the next section. Each head makes a different mix of its inputs and produces an output tensor with a feature size divided by the number of heads. A fully connected layer combines the concatenated results from all the heads. The input vehicle feature tensor is added to the resulting tensor as in residual networks [He+16]. An optional layer normalization [BKH16] may be used to normalize the output of this block. To compute the layer statistics needed in the normalization, we use the input mask to exclude the padded objects.

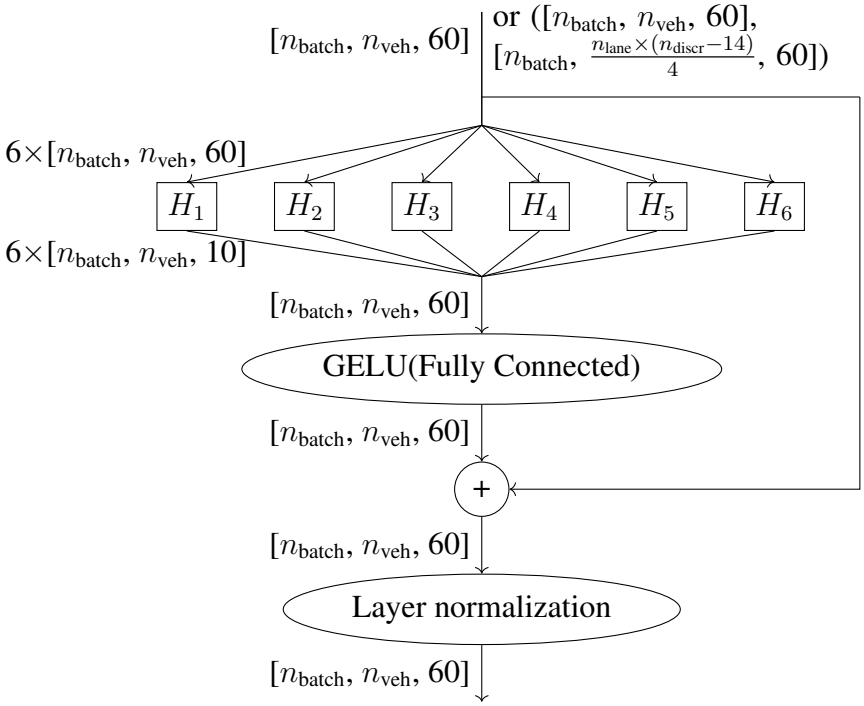


Figure 9.4: Multi-head attention block. The blocks H_1 to H_6 are attention heads.

9.1.2.3 Attention Heads

The attention heads are represented in the figures 9.5 and 9.6. They incorporate a relation encoding that is not present in the usual self-attention mechanism from [Vas+17]. This relation encoding is proposed in [Sch+19], and we experiment with and without it in our ablation study. The L blocks are fully connected layers without activation functions counting $\frac{60}{n_{\text{heads}}}$ units. This means that the feature dimension must be a multiple of the number of heads and that the number of parameters used for the multi-head attention is independent of the number of heads. There are n_{heads} attention blocks, each using its own set of weights for the L matrices. The implicit multiplications are matrix multiplications. We note the element-wise multiplication with the symbol $\odot$.

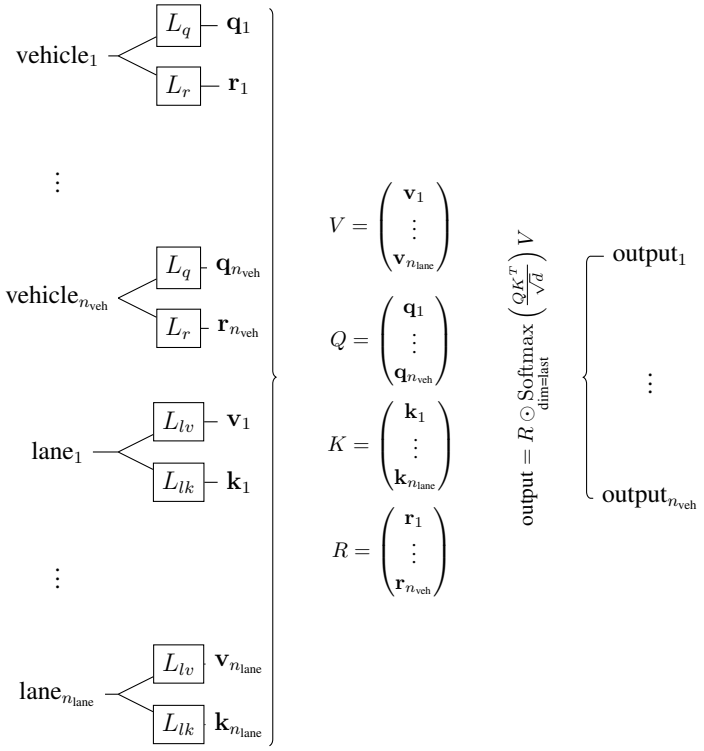


Figure 9.5: Lane attention representation for one head. Blocks L_q, L_r, L_{lv}, L_{lk} are fully connected layers with bias.

d is the dimension of the key vectors. In our case with 6 heads and a feature size of 60, we chose to split the dimension evenly among the heads thus $d = 10$. The self-attention computation for each head is written:

$$\text{output} = R \odot \underbrace{\underset{\text{dim=last}}{\text{Softmax}}\left(\frac{QK^T}{\sqrt{d}}\right)}_{\text{attention matrix}} V \tag{9.1}$$

The "Lane attention" that computes the relation between vehicles and lanes and "Social attention" that computes relations among vehicles both perform the same computation with the same architecture (each with their own parameters). However, the Lane attention block computes the value v and key k using the lane feature vectors and use the vehicle feature vectors to compute the query q and relation r. In contrast, the social attention block computes everything using the same vehicle feature vectors.

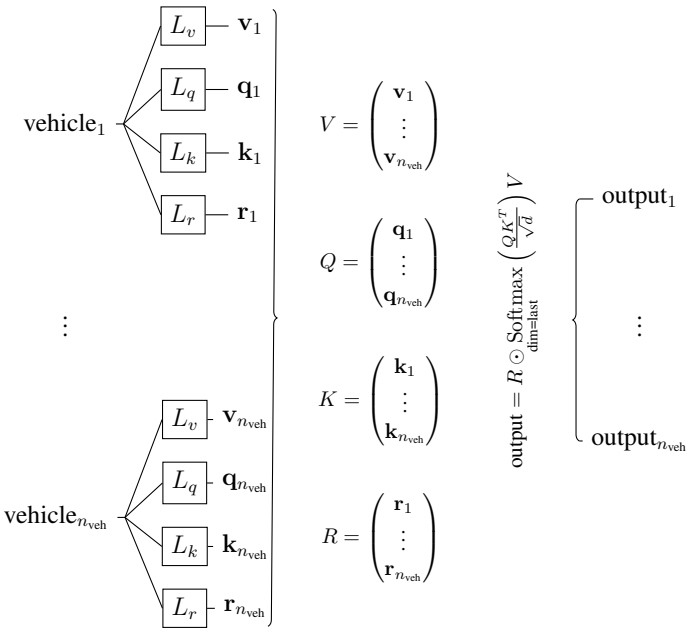


Figure 9.6: Social attention representation for one head. Blocks L_q, L_v, L_k, L_r are fully connected layers with bias.

After the attention block, the feature tensor is duplicated n_{pred} times. This forms a sequence that is fed to the LSTM layer named "LSTM forecast" in figure 9.2. The initial state of this LSTM layer (h, c) is the final state of the "LSTM embedding" layer.

9.1.2.4 Re-Encoding Block

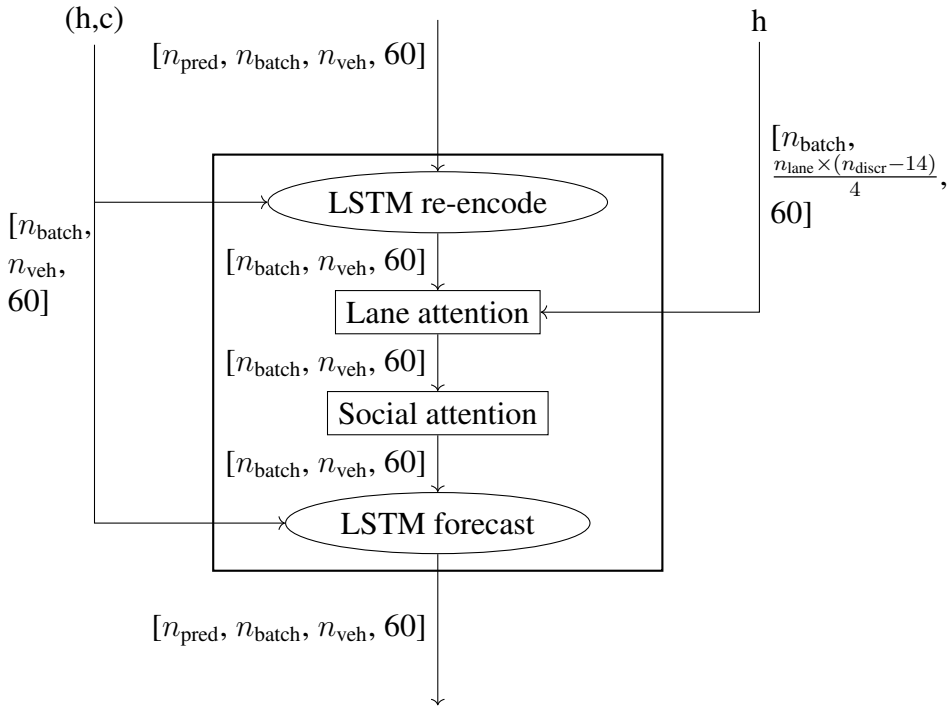


Figure 9.7: Illustration of the re-encoding block.

The model could directly decode the feature sequence into the output forecast, ignoring the re-encoding block that we framed in a rectangle and reported on figure 9.7. However, if it uses the re-encoding block, the input sequence is re-encoded with an LSTM, much like during the embedding but using different parameters. The "LSTM re-encode" block is a sequence to one encoder. A lane attention layer and a social attention layer compute relations between the resulting tensors. The two attention layers are similar to the two first ones described in the previous section but using different weights. Finally, in the same way as before re-encoding, the feature tensor is replicated into a sequence. The same "LSTM forecast" layer with shared weights is applied on this sequence.

9.1.2.5 Decoding

Decoding consists in two fully connected layers with GELU [HG16] activations and a third fully connected layer with the MDN activations. They can also be expressed as 1×1 convolutions in the space of vehicle and time, which is strictly equivalent. These layers are defined with 60×60 matrices plus 60 bias. They transform the features identically for each vehicle and each time step. Finally, the last fully connected layer produces $n_{\text{mix}} \times 5$ features (or $n_{\text{mix}} \times 6$ features if the probability block is not used) that are reshaped in n_{mix} sets of 5 features. The Gaussian activation, described in section 6.4, is applied on each set of 5 features to produce $(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho)$. If the probability block is used, only a set of probabilities is missing to use this as a Gaussian mixture with n_{mix} components.

9.1.2.6 Probability

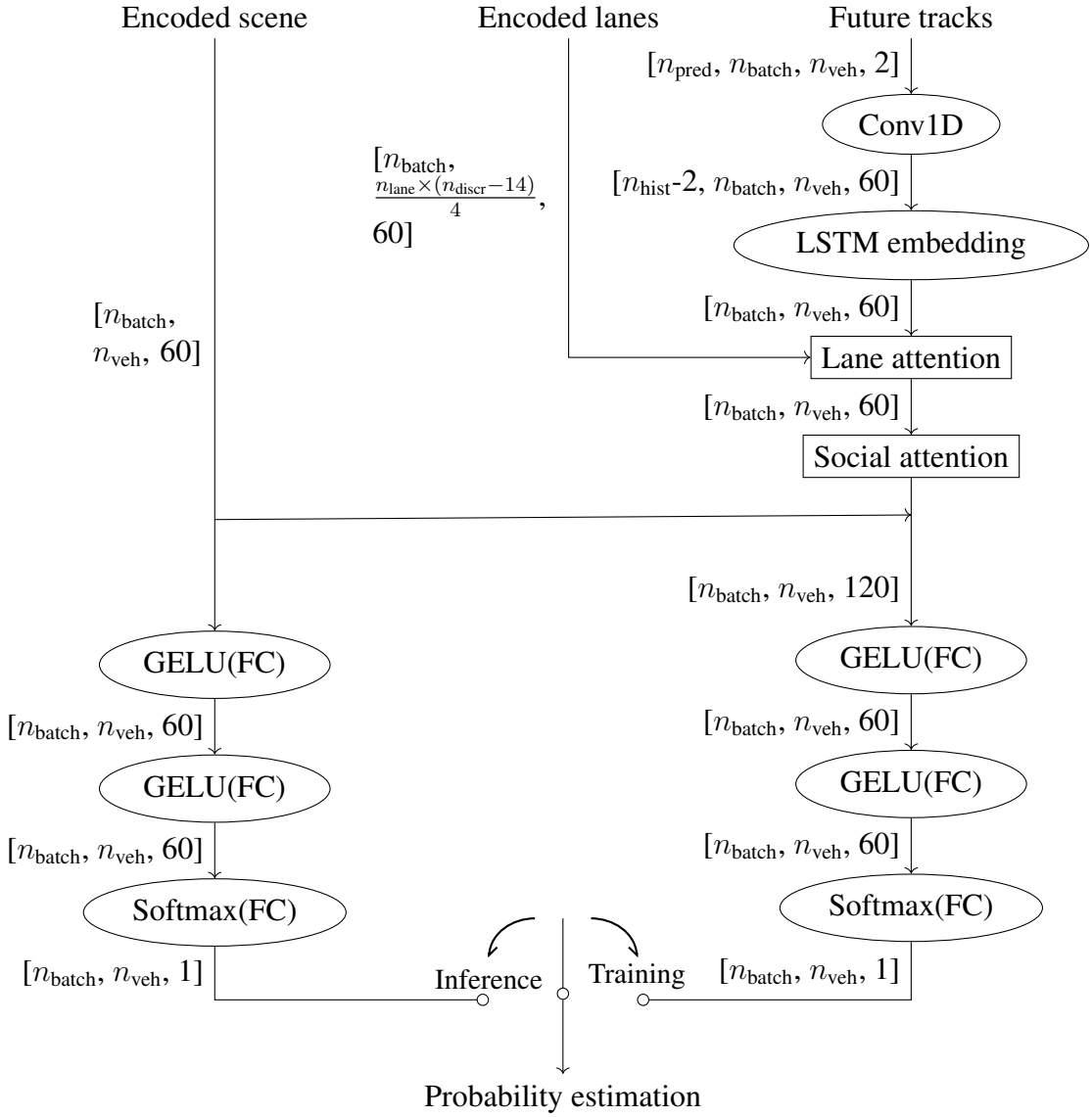


Figure 9.8: Representation of the probability block.

The probability block is composed of two parallel paths that are both trained to produce the Gaussian mixture components probabilities. During training, both paths are computed. The left path is trained to produce outputs that match those of the right path. The right path uses an additional input that is not accessible at inference: the future tracks that the whole model is trained to predict. The future tracks are embedded in the same fashion as the past tracks in the early stage of the model: a 1D convolutional layer with kernel size 3 and no padding followed by a sequence to one LSTM embedding, then two attention layers, one over the lanes the other among vehicle tracks. The output is an encoding of the future scene. The past and future scene encodings are concatenated before being fed to three fully connected layers. The two first use GELU activations, and the last one uses a softmax to normalize the output into a simplex. Only the left path, composed of three fully connected layers, is used during inference when the future tracks are not known.

During training, the distance between the branches outputs is computed before the softmax normalization and is added to the loss to be minimized. The right branch additionally receives its gradient update from the overall NLL loss directly. During the gradient descent, the weight update is not back-propagated through the inputs of this block. This means that the encoders producing the encoded scene and encoded lanes are not trained to minimize the part of the loss that is due to this block.

9.1.2.7 Loss Function and Dropout

During training, if only the ego trajectory is being forecasted, 20% of the lanes and 20% of the objects in the road scene are dropped. All the input values concerning the dropped objects are set to 0 but not masked from the data. For the lanes, chunks of 10 consecutive points describing a lane piece are dropped out, and for the road users, the whole past trajectory is dropped out.

The model is trained to minimize a loss computed with a function of three arguments: the output, the future tracks, and the network weights. It is defined using three terms: the NLL, the miss loss, and the weight decay.

NLL loss - The NLL loss is the average of the timely Gaussian mixture NLL defined in chapter 3, equation (3.11).

Miss loss - The miss loss is the average of a saturated L_1 loss over the error distance $||\tilde{\mathbf{y}} - \hat{\mathbf{y}}||_2$ affecting the forecasts that produce an absolute error between 1 and 3 meters. The miss loss is only applied on the forecasted trajectory that is the closest from the truth, noted $\hat{\mathbf{y}}_{m^*}$. The figure 9.9 represents its graph.

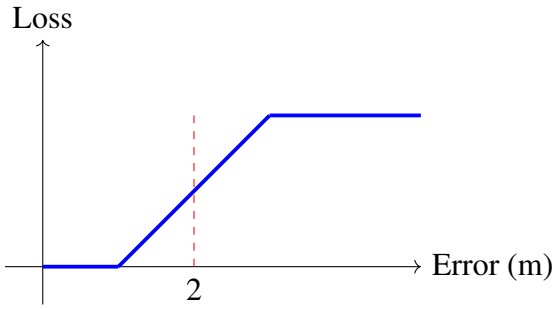


Figure 9.9: Graph of the miss loss.

Weight decay - The weight decay is induced by an L_2 loss over the network weights Θ, scaled with an arbitrary value of 10^{-5}.
The loss function is written:

$$\begin{aligned}
&\text{Loss}(\tilde{\mathbf{y}}, \{\hat{\mathbf{y}}, \Sigma, p\}_{m\in[\![1,n_{\text{mix}}]\!]}, \Theta) = \\
&\quad - \ln\left(\sum_{m=1}^{n_{mix}} \exp\left(-\text{NLL}(\tilde{\mathbf{y}} - \hat{\mathbf{y}}_m, \Sigma_m)\right) p_m\right) \\
&\quad + \text{miss_loss}(\tilde{\mathbf{y}} - \hat{\mathbf{y}_{m^*}}) \\
&\quad + 10^{-5}||\Theta||^2
\end{aligned} \tag{9.2}$$

The average of this loss over the time sequence and the batch is used to perform the gradient descent method at each iteration.

9.1.2.8 Pre-Interpretation

The thought process used to produce the complete architecture described in this section involves a pre-interpretation of what each block is expected to compute. This interpretation is already included in the names given to the different parts and in the arbitrary hyper parameter choices. In this section, we describe the reasoning behind the architecture defined above. The post-interpretation using the results, the trained weights, and ablations is given in the next sections.

A manifesto for fallibility - The researcher's or engineer's intuition and experience is a form of art that is difficult to include in a scientific publication with the current standards. The need for conciseness in the article format and the strict rigor demanded by some reviewers often discourage the author to report their thought process. I believe that neglecting this part of the work is a significant missing piece in the scientific publications. Of course, replacing the intuitive process with a proper scientific method is desirable to transform what is currently an art into science. However, when no alternative to the intuition is found, as it is often the case in applied machine learning, an effort should be made to exhibit the author's thoughts. The introspection it requires should not be avoided even when it is so difficult to exhibit a rough draft. The authors should give a chance to the reader to spot every mistake, factual or intuitive, instead of hiding our shame of being fallible.

Input features - The first consideration we had in building this forecasting model is that preprocessing operations should be minimal and replaced with learned layers. This is an argument against rasterization and also against creating unnecessary features. Adding features such as velocities and locally averaged input can be learned by the first convolutional layer. Therefore, our inputs are simply the lane centerlines position sequences and the tracks in the local coordinate system.

Convolutions - The first layer is a 1D convolution that is able to compute the kinematics in the input tracks. The lane connectivity, shape, direction and curvature can also be computed with layers of 1D convolutions.

No more time - The LSTM layer is able to encode the salient features observed in the given tracks and it can produce an abstract representation such that the rest of the network does not need to consider the time dimension. This was judged to be a good way to encode the present state for each vehicle.

Encoding for attention - Before using attention layers, the computations are made independently for each vehicle and each lane piece. Independent copies of the same module should encode each input into a feature representation with the desired properties. The attention layer makes linear projections of its input and computes a convex combination of the results. Our hope is that the learned encoding would produces a feature space that makes this combination meaningful.

Road network - The last layer of the lane encoding block is an attention layer. The lane segments are encoded with convolutions that respect their local connectivity. However, in the road network, some lanes can cross, separate, or merge. Thus, after encoding, they are all considered as an unordered bag of local lane features that should be in relation to one another. The lane relations can be computed with a self-attention layer.

Attention - The attention layers have the desired mathematical properties described in chapter 7: they are equivariant and defined for any number of inputs. They are computing specific correlation patterns between their inputs that should account for the interactions. This formulation of interactions with the lanes is very permissive. Many other models force a restriction on their model to follow the lanes. This could count as a security measure for planning, but in the case of forecasting, relying too heavily on the lanes produces two problems:

-If the lane perception is noisy, relying on them adds noise to the forecasts.

-If a vehicle makes a forbidden maneuver, such as a U-turn, since it crosses the lanes in an unexpected way, it could not be predicted in a framework relying too heavily on the lanes. This is why our model keeps the lanes as an optional input and does not force a lane following behavior.

We put the lane attention layer right before the vehicle attention layer because it seems to make more sense to first be aware of the lanes and only then interact with other agents on the road network.

Time again - Once a feature vector that includes interactions is produced for each vehicle track, the forecast can be made in the encoded space. Here, an LSTM layer can include the sense of causality using its recurrent computations. It also brings back the time dimension but in the future this time... We use a sequence to sequence LSTM network instead of a one to many architecture. This results from practical computational performance considerations.

Back to the future - The interaction that we presented lies in the past encoding. However, when a forecast is made for each vehicle we want these forecasts to consider the interactions with each other and with the lanes in the future. We encode back the future sequence with a third LSTM layer used as a sequence to one encoder. Then, a lane attention and a vehicle attention layer allow the network to consider the interactions in the future. This could very well be unnecessary: all the information was already contained in the past encoding. However, if the recurrent nature of the forecasting LSTM is indeed able to include a sense of causality in its sequence of feature, this is now considered in the interactions. To force this causality, the same forecasting LSTM layer is used again to bring back the future time dimension. This whole process was described as the re-encoding block. It should be a stable function converging to a coherent forecast. For this reason, we experiment with two loops of re-encoding with the same weights.

The re-encoding was not used at first, and some tests are performed without it to test its usefulness. In [Mer+20], we had tried another method that performed a time-wise interaction. However, this could only correct the trajectory locally in time. This is not sufficient because a trajectory could be different from the start to avoid a collision several seconds in the future.

Adverse probability - The probability block was our latest addition. The thought process is analog to the one of adversarial training. We consider that the forecasts are generated trajectories and that the probability block is a discriminator for these trajectories. If a discriminator had to find the forecast proposition that is closest to the actual future, it would produce a similar output as the probability block. However, in adversarial training, the rest of the network would be trained to generate forecasts that would fool the discriminator. Thus, our intuition is that training the network to both generate the best possible forecasts and estimate their probabilities is an "anti-GAN"

or self-adversarial architecture and could be counter-productive. Another way to see this is by considering that the network should produce all the most probable trajectories but at the same time it should give a low probability to some of them. We want to avoid a situation where the network would produce a wrong trajectory just to make the probability estimation easy: very low probability.

To avoid this, we separate the probability estimation from the forecast generation using the probability block. Moreover, separate probability estimations can be supervised using the true future. The two branch network that we described is made for that purpose. The left branch, not using the future must match the probability estimations of the right branch that cheats by looking at the future. During training, the backward propagation is stopped at the probability block inputs to isolate the probability estimation from the forecast generation.

We go ahead in the experiments and tell the reader that the probability block does not improve the forecasts and even degrades the learning process. There might be an error in our intuition or other effects taking place. We ended up not using this block and producing both the forecasts and their probability estimates in the main branch. However, because we could not find why this intuition would be wrong, this might still be something to investigate.

Dropout - The model should be robust to the noise and should be able to work properly if some of the objects in the scene are not perceived. The 20% dropout is used during training to force it to fill in the missing pieces. We believed that it could force the network to find interactions between the vehicles and make the model more robust to the existence of unperceived objects.

Loss or losses - The NLL loss is what should be minimized to maximize the likelihood of the data for our model. We added a very small weight decay factor to regularize the solutions and avoid numerical problems. This way it pushes the model toward a local minimum with lower weights without modifying much the objective. As we will see in the application, a competition ranked the results with the miss rate. To help the model have a lower miss rate, we added the miss loss. It should push the forecasts that are missing the 2 meter threshold of the miss loss by a small amount into the threshold. We kept it because we think it can help to avoid mode averaging.

Hyperparameters - We have made many arbitrary choices of hyperparameters: the feature dimension, the number of attention heads, the number of layers, the activation functions, the type of layers, the normalizations, the skip connections, the convolution kernel sizes, the optimizer type, the learning rate, the batch size, the preprocessing normalizations or lack thereof...

In the following we "justify" these choices but from our experiments, we know that many different parameters lead to the same performances. For most parameters, we chose a value because one had to be set more than because it was the best choice of value. Even worse, sometimes choices are made implicitly and we might not be aware that we made a choice.

To keep things simple, we only experiment with the same *number of attention heads* in every attention layer. The attention heads split the number of features. Using many heads would lower the dimension in which the interaction is computed. We chose to use *a features dimension of 60* after some trial and errors. For simplicity, the feature dimension is kept identical through the whole network but it could probably be lowered when there

is a time dimension. Using 6 heads with a feature size of 60 makes a split dimension of 10. We judged that it is a good compromise to have several head specializations and a high enough dimension for the interaction matrix to be selective. The input of the attention layer is added to its output and normalized as in the transformer paper [Vas+17]. The network is not very deep and could probably use many *more layers* of self-attention to fit the complex human interactions. However, the transformer model interaction patterns study [Vig19] lead us to believe that more layers would complexify the attention interpretation.

The *GELU activation* might not be the best choice but it usually shows good performances at some computational cost. Since our feature dimension is relatively small, it is not computed too many times which keeps the computational cost in bounds.

The *LSTM layers* work well in our case, changing them for GRU layers is a possibility but it is not expected to change anything significantly.

Now that we have used an intuitive process to define the forecasting model, our intuitions and the model itself must both be tested. The NGSIM dataset that we have used until now was great to find and test our first ideas, but because it is only recorded in two circumscribed highway sections, it only offers a limited set of interactions between vehicles and with the road network. It does not contain a good diversity of road users: mostly cars. It does not contain diverse road configurations such as road crossing and turns. Thus, in the next section, we introduce the Argoverse dataset. It was, at the time of our work, the largest open dataset for trajectory forecasting. Its urban scenarios and the provided HD-map make it a very good fit to train and test our model with more interactions between the road users and the road network.

9.2 The Argoverse Forecasting Dataset

In this section, we present the Argoverse dataset. It was the largest vehicle trajectory open dataset at the time of our study. It improves many aspects of the NGSIM dataset that we used in our applications until this point.

9.2.1 Content of the Dataset

The complete documentation about the Argoverse dataset is accessible on GitHub[1], and in the published paper [Cha+19b]. The data is collected from a fleet of autonomous vehicles equipped with sensors traversing nearly 300km of mapped road lanes in Pittsburgh, and Miami. The sensors count two LiDAR, and two front-facing cameras. A combination of GPS and sensors localize the autonomous vehicle. An HD-map completes the data with the lane centerlines recorded as a set of "polylines." That is a set of lane segments represented as an ordered sequence of (x, y) positions.

The dataset is composed of 5 seconds road-scene clips in which many road users are tracked. Instead of using the whole 1006 driving hours recorded, the authors have mined interesting scenarios and have kept 320 driving hours that make for 323,557 clips of 5 seconds (which is about 450h, thus there are probably redundancies between several scenes of the 320 hours). In each scene, one of the observed vehicle tracks is considered more interesting and observed during the whole 5 seconds. This vehicle is called the

[1] `https://github.com/argoai/argoverse-api`

agent. We will call it the ego vehicle, even if it is not the vehicle observing the scene to unify our vocabulary with the one used for the NGSIM dataset. The other tracked objects may be partially observed during the sequence. They can be vehicles, pedestrians, or bicycles, but they are not identified; only their trajectories are available. The dataset is split into 205,942 training clips, 39,472 validation clips, and 78,143 testing clips. These three sets are from geographically disjoint parts of the cities.

Only the two first seconds of the testing clips are released. This has been the basis of two motion forecasting competitions. The goal of the competitions was to produce the best forecast among six future trajectorie propositions for the agent vehicle in each test set clip. The proposed evaluation system is similar to what we presented in chapter 3. The leaderboard showing the results is accessible online[2].

For the first competition, held at the autonomous driving workshop at NeurIPS in 2019, the goal was to achieve the minimum Final Displacement Error (FDE). Only the best trajectory among the six propositions is used, and only its final position counts. The average of this error on all the 78,143 clips is used to sort the propositions. Our solution was the winning entry along with the method developed in [Cui+19]. At this point, the re-encoding block and the miss loss were not used. For the second competition held at CVPR in 2020, the goal was to achieve the minimum miss rate. Our solution, including the re-encoding block and the miss loss, was again the winning entry, achieving better results than the researchers' solutions from the companies Alibaba, Argo, Uber, and Waymo.

The Argoverse dataset is a good way to evaluate motion forecasting models capabilities in complex scenarios. Compared to the NGSIM dataset, it brings much more interaction in complex settings. The diversity of locations and abundance of crossroads bring the need to incorporate the road network in the model. Moreover, the online competition leaderboard allows for a fair comparison of the different methods. However, since the clips are selected, the dataset is not statistically representative of complete trajectories in Pittsburgh or Miami and even further from being representative of driving scenarios in general.

9.2.2 Constant Velocity Baseline

In the previous chapters, we have established baselines using the NGSIM dataset. We reproduce the uni-modal and multi-modal constant velocity baselines with the Argoverse dataset to compare our model with the same baselines.

9.2.2.1 Uni-modal Constant Velocity

Table 9.1: Evaluation of the uni-modal constant velocity forecast on the Argoverse dataset

Time horizon	1s	2s	3s
RMSE (m)	1.90	4.41	7.72
FDE (m)	1.38	3.20	5.65
NLL	3.58	5.12	6.20
MR	0.23	0.55	0.72

[2] https://evalai.cloudcv.org/web/challenges/challenge-page/454/leaderboard/1279

We have used the optimization procedure described in chapter 6 to optimize the parameters of the constant velocity model for the Argoverse dataset. Table 9.1 presents the results. The time horizon is 3 seconds instead of the 5 seconds used with the NGSIM dataset. However, after only 3 seconds, the RMSE is already at 7.72m. This is more than twice the error produced on the NGSIM dataset at the same time horizon. At 5 seconds in the future, the constant velocity forecast on NGSIM produces an RMSE of 6.69m. This shows that it is more challenging to make 3 seconds forecasts on Argoverse than 5 seconds forecasts on NGSIM. In that context, the 3 seconds time horizon is enough to see the forecasting models limits. At two seconds in the future, more than half of the constant velocity forecasts are off by more than 2 meters.

The neural network forecasting model that we described produces multiple predictions as a multi-modal Gaussian mixture. Therefore, to define a comparable baseline, we must also build a multi-modal constant velocity model.

9.2.2.2 Multi-Modal Constant Velocity Baseline

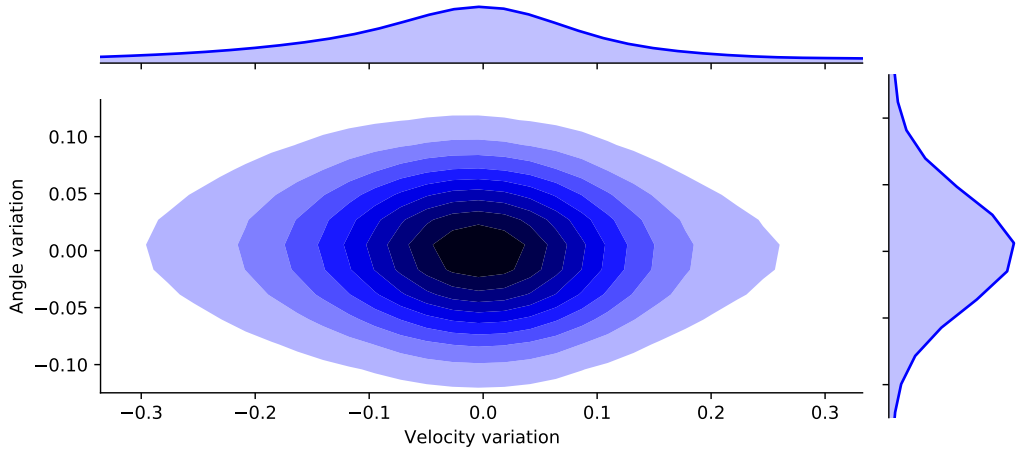


Figure 9.10: Distribution of velocity variation and heading angle variation (in radians) from the estimated values on the Argoverse dataset.

We follow the same procedure as with the NGSIM dataset to produce a multi-modal constant velocity baseline. First, we evaluate the distribution of heading angle variation and velocity variations on the dataset. This produces the figure 9.10. Unsurprisingly, the angle variation is much more pronounced than in the NGSIM dataset. It represents about a third of the velocity variation. Therefore, the exploration distribution from which the modes are drawn must account for both angle and velocity variations.

The tables 9.2 present our multi-modal results with different exploration distributions and 6 predicted trajectories. The two first tables are produced by varying only the velocity while the third table below explores both the heading angle and velocity. Exploring only 10% of velocity standard deviation improves the miss rate by 30%. The angle variation produces very large pRMSE and pFDE values. This shows that most trajectories from the dataset are almost straights lines with only velocity variations. This is also visible on the compared miss rates. However, the minRMSE and minFDE values are low for all exploration distributions. The lowest minRMSE at 5 seconds is achieved with the 20°

Table 9.2: Evaluation of the multi-modal constant velocity forecast with $n_{\text{mix}} = 6$, and different values of σ_θ and σ_{α_v} on the Argoverse dataset

(a) $\sigma_\theta = 0°$, $\sigma_{\alpha_v} = 10\%$

Time horizon	1s	2s	3s
pRMSE (m)	2.11	4.78	8.20
minRMSE (m)	1.40	3.37	6.03
pFDE (m)	1.77	3.95	6.71
minFDE (m)	0.83	2.01	3.67
NLL	3.94	6.44	8.20
MR	0.12	0.28	0.42
Sim	0.54	0.07	0.01

(b) $\sigma_\theta = 0°$, $\sigma_{\alpha_v} = 20\%$

Time horizon	1s	2s	3s
pRMSE (m)	2.63	5.72	9.46
minRMSE (m)	1.30	3.10	5.48
pFDE (m)	2.39	5.15	8.43
minFDE (m)	0.83	1.92	3.35
NLL	3.85	6.25	7.98
MR	0.10	0.24	0.44
Sim	0.42	0.06	0.01

(c) $\sigma_\theta = 20°$, $\sigma_{\alpha_v} = 40\%$

Time horizon	1s	2s	3s
pRMSE (m)	5.61	11.43	17.60
minRMSE (m)	1.36	3.03	5.16
pFDE (m)	5.19	10.64	16.47
minFDE (m)	1.09	2.40	4.03
NLL	4.19	5.83	6.96
MR	0.11	0.49	0.73
Sim	0.01	0.01	0.00

angle standard deviation. This shows that this dataset contains more lateral maneuvers than NGSIM and that simple heading angle variations do not capture them well. We have now established a baseline for the Argoverse dataset. In the next section, we evaluate the trained neural network model.

9.3 Evaluation of the Model

We evaluated our model on the NGSIM dataset and the Argoverse dataset. The first one has been used for a long time in the motion forecasting literature. It constitutes a comparison point for many papers up to 2019. The Argoverse dataset is much closer to the type of applications that we consider. Moreover, its ongoing online competition with special events rewarding the best methods at conference workshops has drawn attention. Therefore, many results from SOTA methods figure in the leaderboard.

9.3.1 NGSIM Evaluation

Table 9.3 reports results using the performance indicators defined in chapter 3. All compared models except GRIP [LYC19] were trained and computed on the same dataset and evaluated with the same functions. Since CSP(M) only forecasts the ego vehicle trajectory, only this vehicle results are compared.

Baselines:

Constant velocity (CV): We used a constant velocity Kalman filter with optimized parameters for forecasting on the same data as described in [Mer+19].

Multi-Modal Constant velocity (MM CV): We used the same constant velocity Kalman filter with the multi-modal velocity variations described in the previous chapter.

Convolutional Social Pooling (CSP(M)): We retrained the model from [DT18]. It uses a maneuver classifier trained with preprocessed data that conditions a predictor for multi-modal forecasts. A forecast of the ego vehicle trajectory is made with information from its social environment using the convolutional social pooling mechanism. In [DT18], the model CSP with unimodal forecast gives better RMSE results than the multi-modal forecast CSP(M).

Graph-based Interaction-aware Trajectory Prediction (GRIP): We took the published results from [LYC19]. It uses a spatial and temporal graph representation of the scene to make a maximum likelihood trajectory prediction simultaneously for all vehicles in the scene. It produces the best RMSE results, but it does not account for forecast error covariance estimation nor multimodality.

Social Attention Multi-Modal Prediction (SAMMP): The model described in this chapter without the separate probability block and with six mixture components to match the CSP(M) model for a fair comparison.

Table 9.3: Comparison of NLL, RMSE, FDE and MR results with baselines using the same dataset. *CSP(M) results were recomputed with some minor modifications for a fair comparison.

Time horizon		1s	2s	3s	4s	5s
NLL	CV	0.82	2.32	3.23	3.91	4.46
	MM CV	2.45	3.35	4.12	4.81	5.39
	CSP(M) [DT18]*	-0.41	1.07	1.93	2.55	3.08
	SAMMP	-0.36	0.70	1.51	2.13	2.64
RMSE (m)	CV	0.76	1.82	3.17	4.80	6.70
	MM CV	0.88	2.04	3.49	5.21	7.15
	CSP(M) [DT18]*	0.59	1.27	2.13	3.22	4.64
	GRIP [LYC19]	0.37	0.86	1.45	2.21	3.16
	SAMMP	0.51	1.13	1.88	2.81	3.98
FDE (m)	CV	0.46	1.24	2.27	3.53	4.99
	MM CV	0.52	1.32	2.39	3.67	5.14
	CSP(M) [DT18]*	0.39	0.91	1.55	2.36	3.39
	SAMMP	0.31	0.78	1.35	2.04	2.90
MR	CV	0.02	0.20	0.44	0.61	0.71
	MM CV	0.01	0.03	0.10	0.20	0.30
	CSP(M) [DT18]*	0.004	0.03	0.12	0.28	0.44
	SAMMP	0.002	0.02	0.08	0.15	0.23

9.3.2 Argoverse Evaluation

Two competitions were held by the company Argo for motion forecasting on the dataset Argoverse. In the first competition, the evaluation criterion was the minFDE after 3 seconds. We won this competition with a tie at a minFDE of 1.55m. The latest Argoverse competition was evaluated on the miss rate criterion but all the top minFDE results were better than those of the previous competition. The other evaluations are given as indicators, but the results may be optimized for miss rate. The Average Displacement Error (ADE) is the average of the FDE over the forecast time sequence.

Table 9.4: Results at 3 seconds on the Argoverse test set. 6 modes are proposed for each sample. The results were copied on Ag. 21 2020 from EvalAI.

Method	ADE	minADE	FDE	minFDE	MR
Our model	**1.68**	0.97	**3.73**	1.42	**0.13**
Waymo Poly [Cha+19a]	1.71	0.89	3.85	1.50	**0.13**
Waymo TNT [Zha+20]	1.78	0.94	3.91	1.54	**0.13**
Alibaba	1.97	0.92	4.35	1.48	0.16
Uber ATG-LaneGCN [Lia+20]	1.71	**0.87**	3.78	**1.36**	0.16
Argo CMU Wimp [Kha+20]	1.82	0.90	4.03	1.42	0.17

All the results in table 9.4 are very similar and are also using similar methods (with uncertainty about Alibaba's method). Besides the one from Alibaba, each method is the object of a published paper. The reader interested in the details should report to them. A paper from Alibaba might come out soon.
The main differences between the published methods are:

- The use of anchor trajectories in TNT [Zha+20].
- The lanes graph representation in TNT [Zha+20] and Uber [Lia+20].
- The rasterized context CNN encoder in Poly [Cha+19a] and Uber [Lia+20].
- The explicit conditioning on hypotheses about the scene in [Kha+20].

The Wimp (What If Motion Predictor) solution organizes in a different way very similar blocks than the one we used in our model. They have shown interpretable attention over the lane key points during the forecast sequence that correctly predicts which lane is being followed.

The self-attention mechanism has been extensively used in these solutions. One recurrent criticism of the self-attention mechanism is the quadratic growth in memory and in the number of computations with respect to the number of considered items. However, compared to the rasterized models that use a 400×400 pixel map in [Cha+19a], it would produce the same memory usage as 3 heads (matching the 3 color channels of the image) and 400 considered objects. It is extremely rare to find scenes with more than 100 objects. The costly computation is an outer product between the keys and queries. It is as expensive as a 1×1 convolution over an image with the same number of channels and the same number of pixels in each dimension as the number of objects. Thus, the methods using rasterized images are more expensive. In the previous chapter, we have introduced the Graph Neural Network approaches and presented the self-attention as a complete directed graph. The sparsity allowed in the GNN approaches lowers the computational cost but it neglects the cost of building the graph.

9.4 Ablation Study

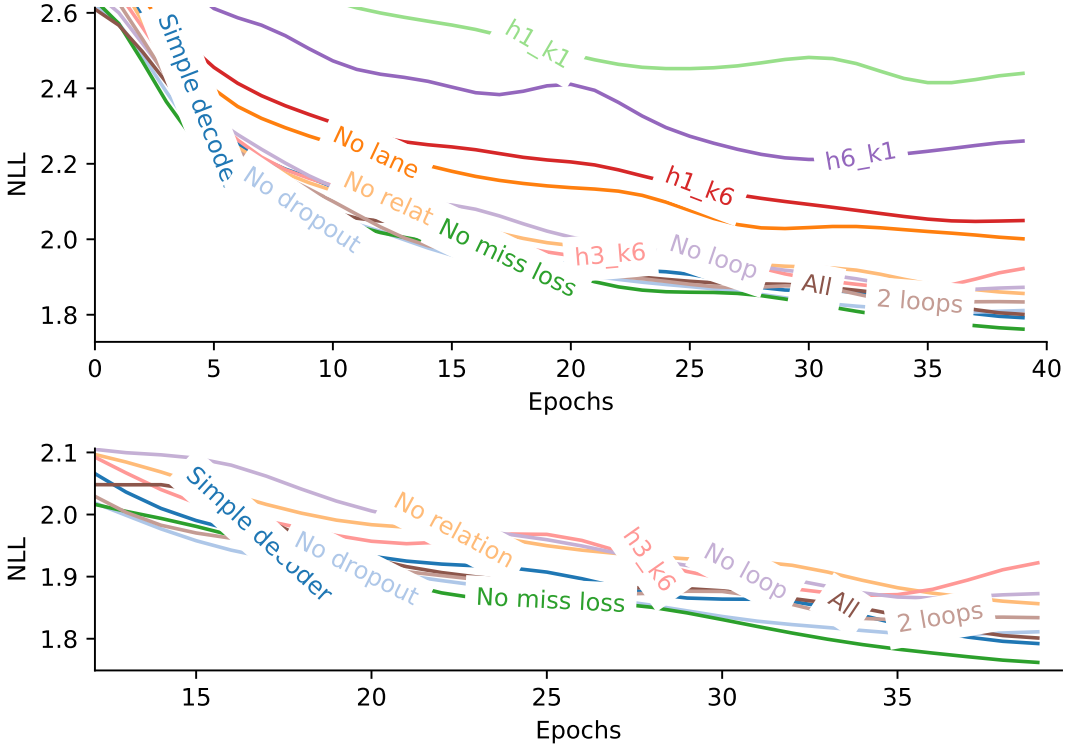
We have defined a complex model composed of many parts. An interpretation has given some reasons that have led us to consider these blocks, but we also want to produce experimental results to evaluate their impact on the forecasts. In this section, the model is trained on the Argoverse dataset with different settings.

With our current implementation, the full training requires about 20 hours on one Nvidia Tesla V100. This represents about 300 epochs of training. Each ablated setting is tested with training made from scratch for 40 epochs. All these computations were made possible by the granted access to the HPC resources of IDRIS under the allocation 2019-39282 made by GENCI. The results are not the best achievable, and some effects that might only occur during the late training phase would be missed. However, we believe that this ablation study helps to understand the effects of the different model parts.

A first result is that the use of the probability block is detrimental to the model performance. A lower similarity value indicates that it might focus too much on the most probable mode and produce secondary modes that are less relevant. Therefore, in the following, the main branch directly outputs the forecasts, the covariances, and the probabilities evaluation at once; the probability branch is not used.

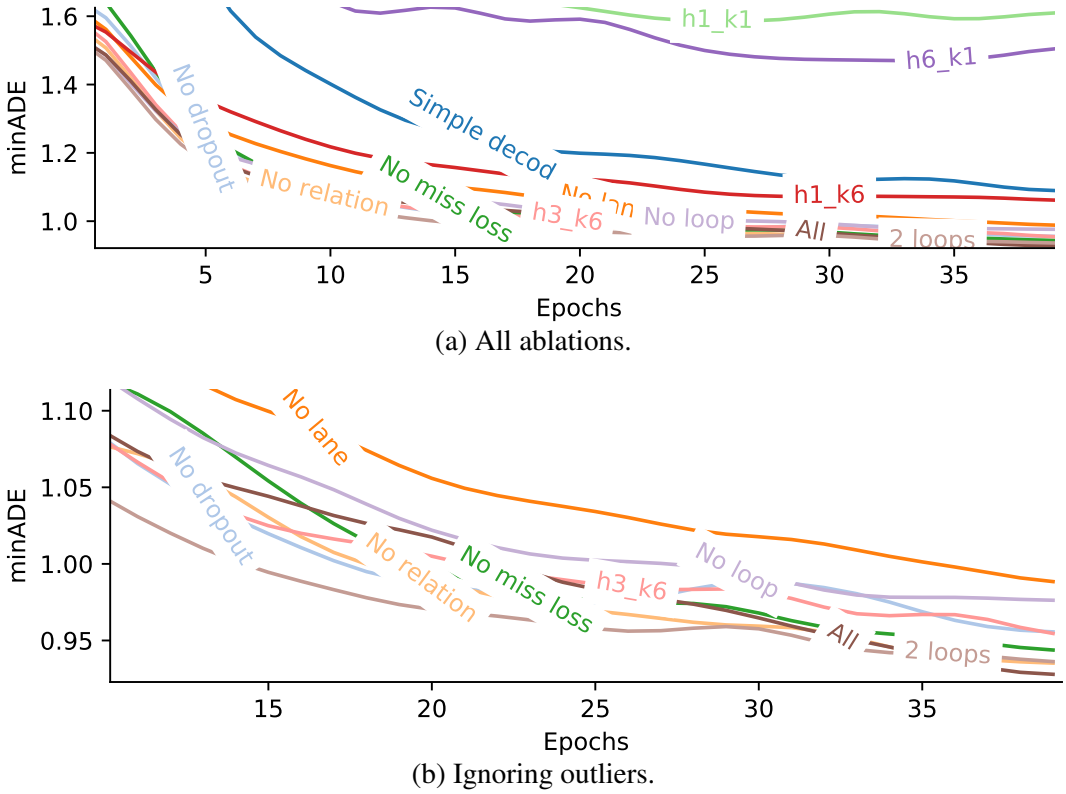
The different ablations that we study in this section are listed below with colors matching the ones used in the figures:

- Simple decoder: Only one layer is applied after the "LSTM forecast" layer instead of three layers with non-linear activations.
- No dropout: The 20% dropout over the road users and the lanes is not applied.
- No lane: No lane observation is fed into the model.
- No relation: The relation encoding from [Sch+19] used in the lane attention and self-attention blocks is not used.
- No miss loss: The miss loss is not added to the loss that is minimized.
- h1_k1: Only one head is used and one Gaussian is produced.
- h6_k1: 6 heads are used and one Gaussian is produced.
- h1_k6: Only one head is used and 6 Gaussian mixture components are produced.
- h3_k6: Three heads are used and 6 Gaussian mixture components are produced.
- No loop: The re-encoding block is not used.
- 2 loops: The re-encoding block is looped through twice.
- All: Everything (but the probability block) is used as described.



(b) Ignoring outliers.

Figure 9.11: Validation convergence curves of the NLL for the ablated models.



(a) All ablations.

(b) Ignoring outliers.

Figure 9.12: Validation convergence curves of the minADE (m) for the ablated models.

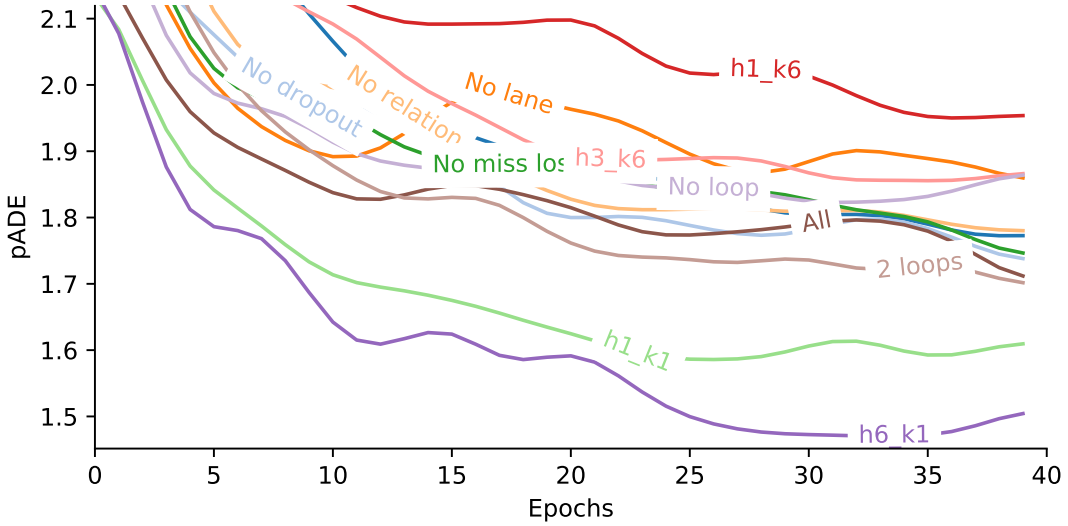
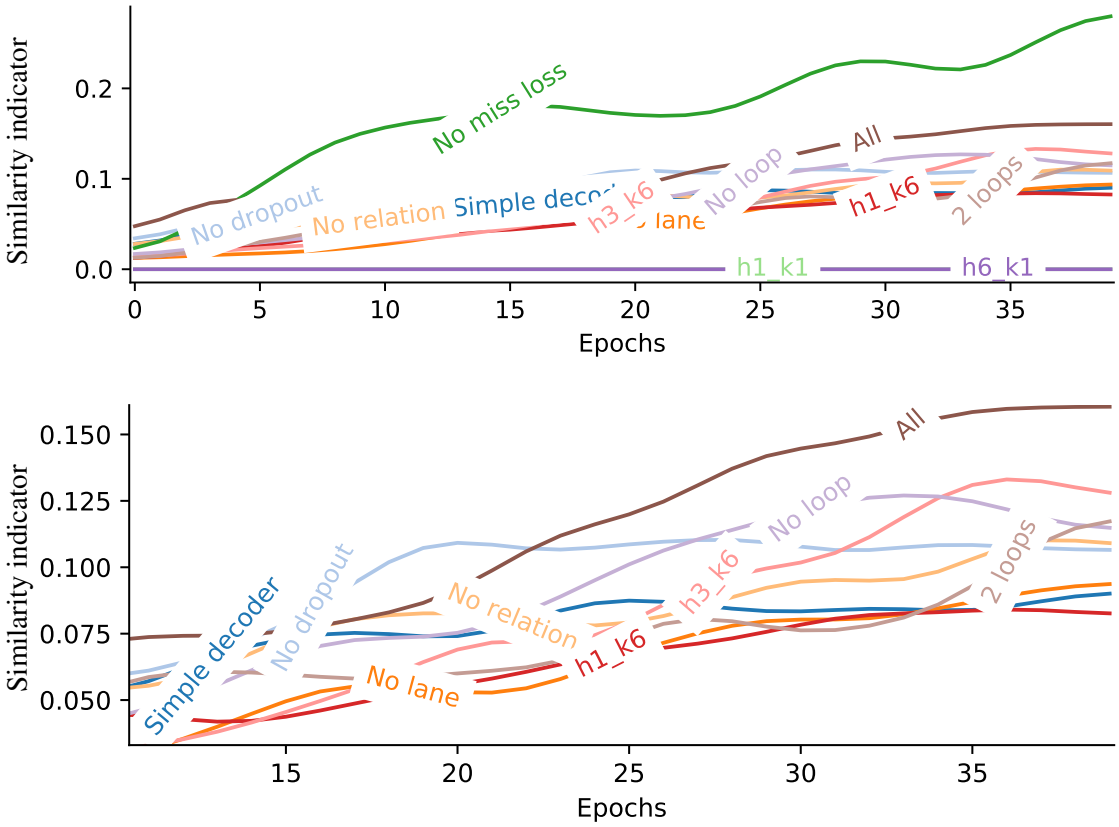


Figure 9.13: Validation convergence curves of the pADE (m) for the ablated models.



(b) Ignoring outliers.

Figure 9.14: Validation convergence curves of the similarity indicator for the ablated models.

- h1_k1 and h6_k1:

figure 9.13 shows the variation of the pADE on the validation set after each epoch of the training process. It shows that the two versions of the model that only produce one mixture component as forecast (i.e. a unimodal forecast) reach a much lower pADE than the weighted average of the 6 mixture components.

However, if we consider the minADE, figure 9.12a shows that the uni-modal forecast lies a lot further from the truth than the best mixture components. On figure 9.11, the NLL value is much smaller for the multi-modal forecasts. We conclude that the multi-modal Gaussian mixture produces better forecasts than the uni-modal ones.

- Simple decoder, No lane, and No relation:

In the minADE graph 9.12a, the simple decoder is not performing well; thus, the two non-linear layers before the output layer should be kept. The same goes for the model that does not use the lane input. Using the lane attention improves the results on all evaluation criteria. Not using the relation encoding in the attention blocks is somewhat degrading the pADE, and NLL.

- No loop and 2 loops:

Not using the re-encoding block (no loop) degrades the ADE and minADE. Thus, the re-encoding block is also improving the results and should be used. It can also be looped through several times. However, the difference between using 2 loops and one loop (All) is small on all evaluation criteria and does not justify the required additional computation. Thus, only one re-encoding block is used and looped through only once.

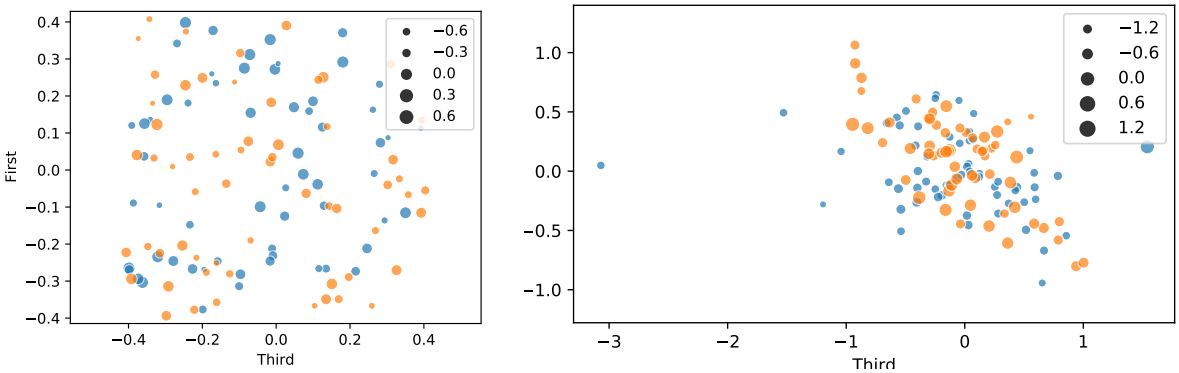
- No miss loss: The training without miss loss produces the lowest NLL. It reaches about the same pADE and minADE values as the "All" model. However, the produced forecasts have a much higher similarity. This means that it does not explore as diverse possible futures as the models trained with the miss loss. Therefore, we use this loss modification during the training of our model.
- No dropout: Using dropout can slow the learning process. However, after 40 epochs, the no_dropout model shows the same performances as the others. The dropout makes the model more robust to input noise and undetected objects. Thus, the model is trained with object dropout.
- h1_k6 and h3_k6:

The model using only one attention head shows the highest pADE and a higher minADE than the model that does not use the lane input. The performance difference between the use of 6 or 3 heads is not as significant but seems in favor of 6 heads. We show in the next section that the different heads specialize in different attention patterns. Using more head might reduce too much the feature dimension in the self-attention layer. Further tests could be performed with a different number of heads in the different attention layers.

9.5 Interpretation of the Model

Once a model is fully trained and achieves satisfactory results, its particular outputs, some intermediary feature vectors, and some weight values can be interpreted. This investigation is made to describe the model computations and confirm or infirm the intuitions used to build it.

9.5.1 Interpretation of the First Layer



(a) At initialization.

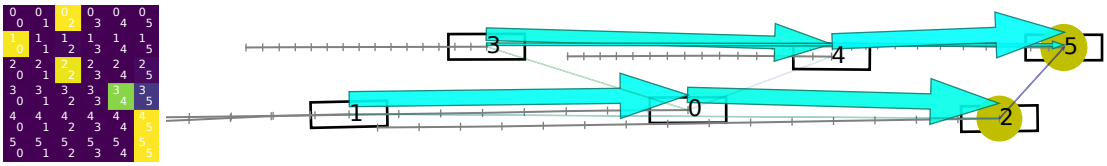
(b) After training.

Figure 9.15: Scatter plot of the convolution kernel values before and after training. The x and y dimensions represent the first and last value of the size 3 kernel. The radius of the circles represent the central value. The filters over the input x dimension are in blue and the ones over the y dimension are in orange.

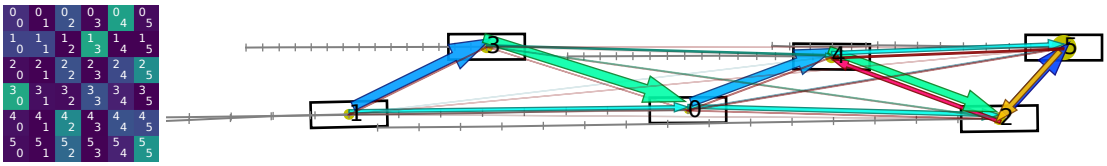
The first layer of the model is a 1D convolution with kernels of size 3. From figure 9.15, we see that the first and third values are mainly of opposite signs, close to the $y = -x$ line. This corresponds to approximates of scaled computations of derivatives. In the annexe D, we plot the Fourier transfom of the convolution kernels to analyze further what they compute. However, we could not make a more convincing interpretation.

9.5.2 Interpretation of the Attention

NGSIM attention - The attention matrices formed by the heads for a given road scene can be represented as a directed graph, as shown in figure 9.16. The model is not specifically trained to give interpretable outputs, but looking at the attention patterns shows that the different heads specialize in specific situation recognition. Moreover, some of them can be understood. In many different tests that we have performed on the NGSIM dataset, nearly all of them learned an attention head that had specialized in front vehicle attention, such as the one in figure 9.16a. Other patterns are also observed. Some are sensitive to the distance, such as 9.16b. Others could not be interpreted.



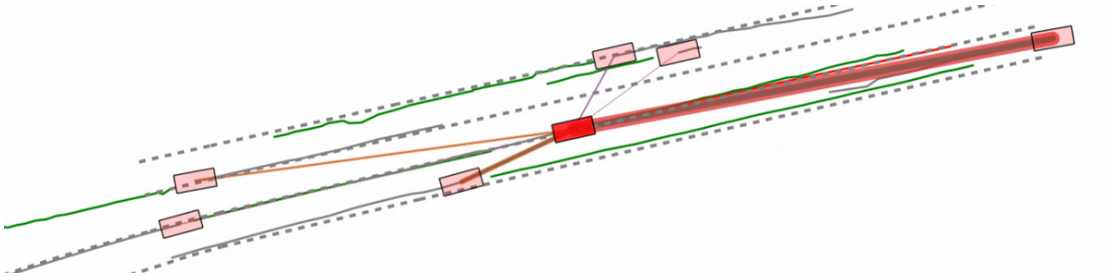
(a) Head specialized in front vehicle attention



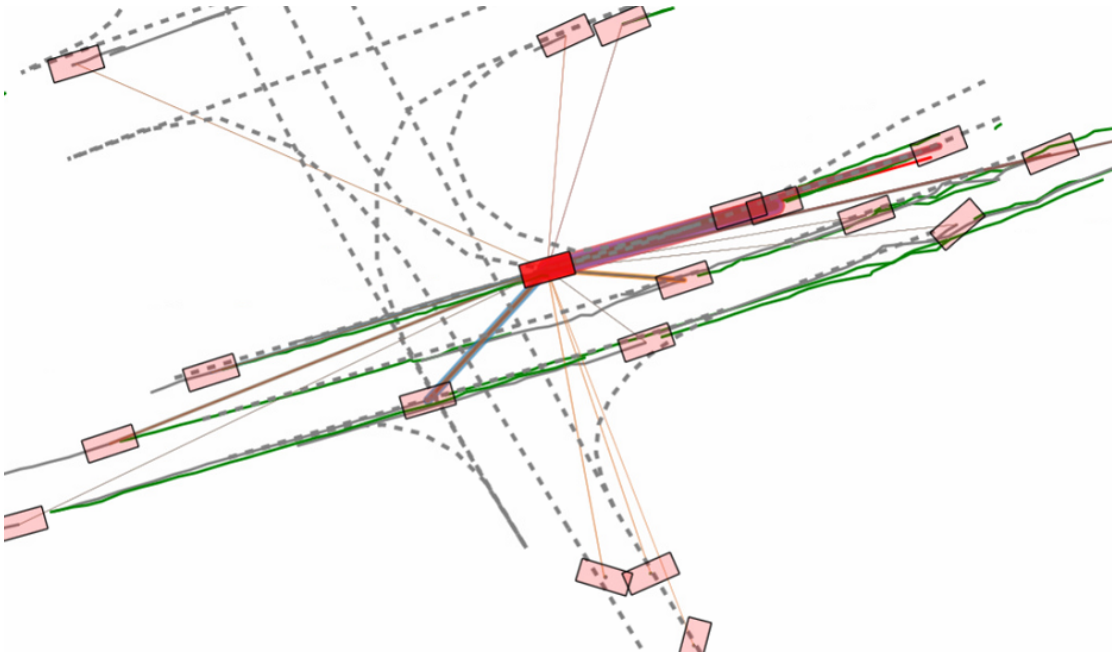
(b) Head mainly specialized in closest vehicle attention

Figure 9.16: An NGSIM scene with all observed vehicles, their past positions in gray and the attention matrix for two heads of the first attention layer. The attention that vehicle i is giving to j is drawn as an arrow from i to j, and a circle when $i = j$ with widths proportional to the attention coefficient and color varying with the arrow angle. Attention is also visible as color from purple to yellow in the i, j coefficient of the left matrices.

Argoverse attention - With the Argoverse dataset, the road network is much more complicated, as we can see on figure 9.17. Thus, the patterns are more difficult to recognize. However, the front vehicle often receives the most attention. The attention patterns are also sensitive to the distance and the velocity of the surrounding vehicles.



(a) A simple scene.

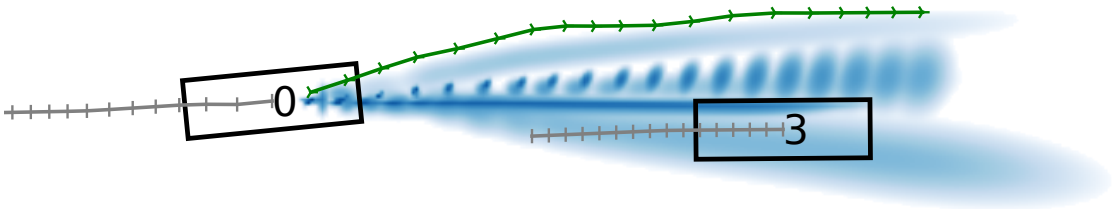


(b) A complex scene.

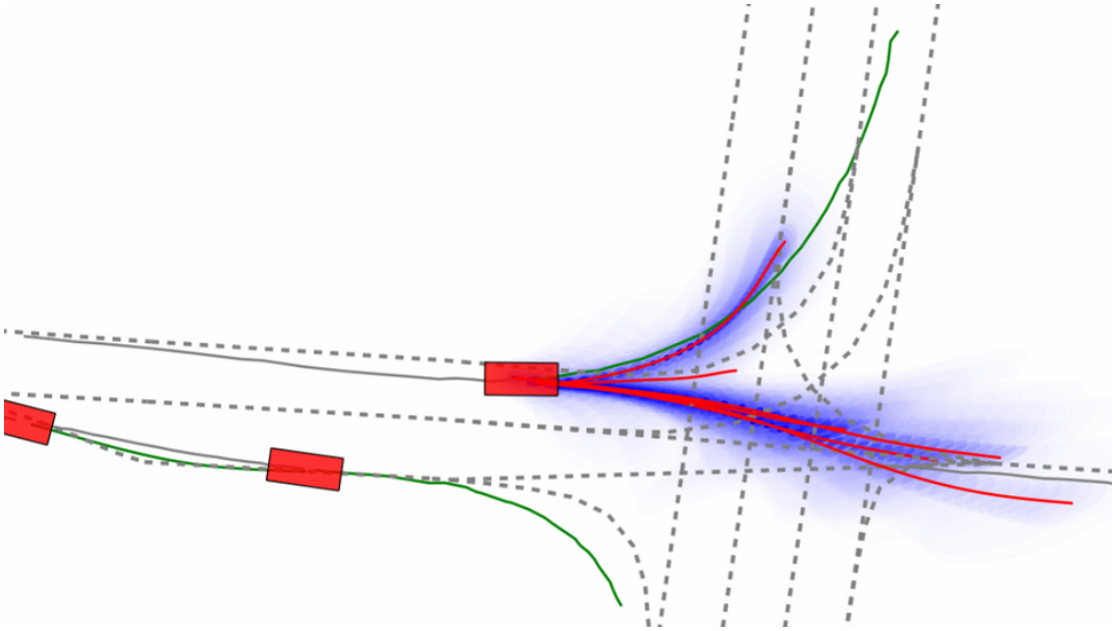
Figure 9.17: Two Argoverse driving scenes top view representation with all observed vehicles. The lanes centerlines are represented in dashed lines. The past trajectories are in gray, the true future trajectory in green and the agent vehicle forecast is in red. The agent vehicle attention on the others is represented in color segments with different colors for different heads. The segments width are proportional to the attention coefficient.

9.5.3 Multi-Modality

Both in NGSIM and Argoverse, the mode diversification is mainly concerning the velocity variation and not lateral maneuvers. This has also been shown with our multi-modal constant velocity model. However, the learned models also use their mixture of Gaussian outputs to describe various lateral maneuvers, as shown in the figures 9.18. Even then, most of the modes describe different velocities for the most represented type of trajectory in the dataset: going straight ahead.



(a) NGSIM.



(b) Argoverse.

Figure 9.18: Driving scenes top view representation. Forecasts are represented in blue shades and red lines. The actual future trajectory is represented with a green line.

In this chapter, we have tested our final model. It produced the best results at the Argoverse competition, performing on par with the solutions developed by Waymo, Uber, Argo/CMU, and Alibaba. The ablation study has shown the benefit of the multi-head attention, the different blocks, the benefit from the road network attention, and the multimodality of the results. It also showed that separating and improving the modes probability estimation is detrimental to the learning of the model.

The field of motion forecasting with neural networks is very active. One of the winning entries in last year competition is now at the 16^{th} position of the Argoverse leaderboard. The next expected evolutions are:

- Advanced usage of graph architectures.
- Consideration for more object types and road features.
- More model invariances such as global scene orientation.
- Definition of a satisfactory multi-modal evaluation criterion.

Another direction of improvement could be centered around safety and embedded solutions. Making the model compatible with a continuous forecast at every step of a long sequence is needed for the real-world application. A different formulation of the output and particular attention to the model computational cost is needed for the continuous forecast. Keeping the computational cost into bounds is a difficult challenge if a dynamic number of inputs is considered. The safety issue is much more challenging and is mainly addressed with extensive data validation requiring enormous datasets that are continuously updated. Adding invariance to the model will help to reduce this validation cost by limiting the number of factors that influence the results and require testing.

Chapter 10

Conclusions and Forecasts

In this work, we have re-established constant velocity baselines in trajectory forecasting, with uni-modal forecasts and a plugin to make them multi-modal. Using the different evaluation criteria from the literature, we defined an evaluation process that gives indications about various aspects of the forecasting models performances. To keep the real application in our thoughts, we presented the perception system with the modular software architecture that we work with. This perception system gives our models all the data that they must rely on. A global literature review showed how the research went from heuristics models for human behavior to learned based statistics or learned reinforced models. We decided to follow the path of statistical learning with neural networks that currently gives the best results in trajectory forecasting for the usual performance measurements. Many neural network architectures are able to produce motion forecasts. Selecting the best ones is still impossible for our limited knowledge of these powerful tools. Thus, we reviewed and applied the most common architectures adding progressively more functionalities: maximum likelihood forecasts, forecasts with error covariance estimations, agent interactions, multi-modal forecasts, and interactions with the road network. Our final model implements all of these at once and was the best model two years in a row at the most significant international trajectory forecasting competition at the time.

10.1 Organizational Observations

During my work, many points could have been improved. Getting up to speed more quickly with neural networks, and their python implementation would have gained much time by avoiding many dead-ends that I followed for too long. On that point, I thank Coursera for the great courses that I could follow for free and that I ended up paying willingly afterward. I regret not having known earlier the book by Aurélien Géron [Gér19] that would have allowed quicker preliminary tests.

Being uncomfortable with a lot of the machine learning tools, I had poor code management and spent a lot of time rewriting the same codes again. For the same reasons, the proper experimental process with a unified validation procedure presented in chapter 3 has been set very late in my work. This has required that I recompute almost all the experiments presented in this work to unify the validation results.

I believe that our evaluation process has made the goal very clear, but it has not been defined so clearly from the start. For example, the work on free-space 7.3 tried to solve a different problem that might be more adapted to some real applications, but that has been a failure as a forecasting model. Unclear objectives are the hindrance slowing down the applied research.

Expressing clear objectives is part of the job. However, it is rare to find research works that clearly state why they have set these objectives for themselves and how they ended up expressing them in this way. We settled with the common objectives in trajectory forecasting: NLL and RMSE. But it was not obvious to me that these objectives, which are poor proxies to measure what we want to do, are the ones we should use. I had to be in a situation where I needed to publish the performance of my models before I realized that unless I optimized the same performance criteria as comparable works, I would not obtain competitive results. I believe that this competition is required to publish applied works and is killing not all but some of the creative lines of research. After settling on the well-defined and comparable objectives of chapter 3, the work that I did was much less creative and much more successful.

I believe that not being perfectly efficient, as stated above, is intrinsic to the research process. During the three years of this work, we explored different approaches and learned how to build neural networks. We were able to apply these new skills and reach state of the art performances. The more creative approaches that we could think of but did not follow are briefly introduced in the next section.

10.2 Approches that Could Work

Along the way, there were many ideas that we could not pursue and many problems that we could not solve:

Model-based approach - We have not followed the model-based forecasts that we briefly introduced in our arXiv publication [Mer+19]. The model-based formulation makes our implementation slower to train. It could be redundant with the trajectory planning module, but it could also help the forecasting model. The model-based approach allows to restrict the forecasting model to an interpretable and controlled environment. It can also help to formulate the model invariants that we want to consider such as a rotational invariant that would not rely on an angular normalization of the scene that sometimes pose problems.

Limited samples - We have not pursued the sampled based approaches because sampling is not reliable to estimate unlikely events in a limited computational time. However, the DSP method [YK19] and the MT-VQ-VAE could be good directions to solve the sampling problems. The generative approaches bound the forecasts to the learned trajectory distribution. They are more likely to detect an out of distribution input while other approaches would give a result that would probably be very wrong.

Free-space centered approaches - I have worked with fluid mechanics where Lagrangian and Eulerian formulations are used. The motion forecasting models seem to either use a fully Lagrangian approach as we did: considering several agents and modeling their behavior. Or a hybrid approach: considering an agent immersed in a context that affects it. Some occupancy grids work might look like fully Eulerian approaches, but they often use Lagrangian particles interacting with the occupancy grid. We stopped our free-space centered work because it was very restrictive and did not produce results that we could easily validate. Our approach can be seen as an Eulerian formulation relying on a coarse grid instead of the fine mesh that comes to mind with such models. This line of research might be more adapted to what an ADAS system or a self-driving vehicle needs.

For instance, we can think of three-dimensional space with a vertical direction being the time and the two other directions, the bird-eye view positions of the surrounding vehicles. The volume is the free space-time. Finding a safe path in this volume is connecting tubes from the bottom frame to the top frame that do not intersect the other tubes. This formulation leads to a social force behavior that considers the whole trajectory sequence instead of forces expressed at each time frame. Therefore, it could incorporate strategic planning.

Differential equations - From the kinematics equations to the equilibrium between safety and velocity, our problem is full of differential equations. For kinematics we can formulate good approximations but for the behavior, we do not know how to formulate them. The mind-blowing work of Ricky T. Q. Chen *et al.* [Che+18; CD19; CAN20a; CAN20b] might be an elegant solution to smoothly solve this problem and produce models with interpretable incentives.

10.3 Tools that Could Help

We did not engage in improving the neural network architectures themselves. However, the literature on this subject is giving us all the tools we have been using. Some ideas might be worth following but are not directly linked to motion forecasting.

RK residual net - In [Che+18], residual networks are expressed as learning the derivative of the neural network function along the depth. We can consider an elementary step in depth dl and approximately integrate the neural network f using: $\mathbf{h}_{l+dl} \approx \mathbf{h}_l + dl\,\mathrm{grad}(f)(\mathbf{h}_l)$ The residual neural networks [He+16] compute exactly this with $dl = 1$. Thus, we can easily imagine a Runge Kutta residual network. For example with the second order approximation: $\mathbf{h}_{l+dl} \approx \mathbf{h}_l + dl\,\mathrm{grad}(f)(\mathbf{h}_l + \frac{dl}{2}\mathrm{grad}(f)(\mathbf{h}_l))$. The approximation order can be variable depending on the step size $\mathrm{grad}(f)(\mathbf{h}_l)$. This is an intermediate solution between ODE neural networks [Che+18] and residual neural networks [He+16] and could be an interesting research direction.

Memory refinements - It is not very clear in neural networks what is learned. There is both the learning of a function and the memorization of the training data. Boltzmann machines [HS86] and other energy-based learning models are able to store information. Learning both data in memory and functions that query and confront memorized data is probably a good way to make an interpretable neural network. Ramsauer *et al.* [Ram+20] exploit this and build a new type of neural network layer. I believe that this line of work could lead to validation systems that would verify the learned data and interpret the "reasoning" that confront these memorized data. This is much finer than a global statistical validation procedure that is currently in use.

Reflexion time - We were able to both make forecasts and estimate the error with our models. This means that we could build internal self-correcting loops such as our re-encoding mechanism that would dynamically choose the necessary number of loops that should be made to forecast a given situation. This could lower the computational cost in simple situations and refine the accuracy in complex ones.

Invariance and equivariance - Finding ways to express neural network layers with designed invariance or equivariance such as convolutions for translation and self-attention for input permutations might be the best way to enrich the neural network toolbox. We were unable to design a layer that would provide a road scene rotation equivariance. However, using such a layer would probably improve the results and accelerate learning. Moreover, having invariants is an excellent way to diminish the cost of validation by reducing the dimension of the data space to validate.

Chapter 11

Résumé long en Français

11.1 Introduction

Un million trois cent cinquante mille vies humaines. C'est le nombre de morts sur la route l'année précédant le début de cette thèse (2016). La plupart de ces morts est due à des erreurs humaines et pourrait être évitée par des systèmes d'aide à la conduite. Le présent travail est une contribution à l'amélioration de ces systèmes.

Pour éviter un accident, un système automatisé d'aide à la conduite ou un système de conduite autonome doit être capable d'anticiper les mouvements des scènes routières. Plusieurs approches permettent cette capacité d'anticipation. La plus employée utilise un découpage en trois modules : Perception, Prédiction, Planification (et contrôle). La prédiction est une estimation de l'évolution du futur qui permet une planification qui anticipe les mouvements futurs dans la prise de décision. La prédiction utilise la perception de la scène ainsi que des connaissances a priori.

Nous utilisons des modèles cinématiques simples pour établir une base de comparaison de prédiction de trajectoire. Une étude de l'état de l'art nous oriente vers les méthodes d'apprentissage statistique et en particulier l'utilisation des réseaux neuronaux. Il existe de nombreuses façons d'utiliser ce type de modèles pour la prédiction de trajectoire et de nombreuses façons d'exprimer les prédictions. Nous étudions les méthodes de l'état de l'art et produisons deux types de prédictions différentes. L'une fondée sur l'espace libre ne donne pas de résultats compétitifs tandis que l'autre employant l'auto-attention produit de très bons résultats.

11.2 Un premier modèle prédictif

Débutons dans le vif du sujet et produisons un premier modèle de prédiction de trajectoire. L'objectif est de produire une base de comparaison et dans le même temps d'identifier les difficultés à surmonter. Dans cette première section, nous définissons un modèle de prédiction à vitesse constante et nous l'appliquons à des trajectoires de la base de données NGSIM.

11.2.1 Prédiction à vitesse constante

Une variété de modèles cinématiques est décrite par Yaakov Bar-Shalom *et al.* [YT13]. Nous appliquons le plus simple d'entre eux : un modèle à vitesse constante qui convient bien aux cas auto-routiers de la base de données NGSIM.

Nous considérons une trajectoire de véhicule formée d'une séquence temporelle d'observations de positions (x, y). Cette séquence s'étend sur 3 secondes dans le passé et contient l'historique des positions, observées toutes les 100 millisecondes, jusqu'au temps présent. Nous utilisons un filtre de Kalman pour estimer la position et la vitesse du véhicule à l'instant présent. L'incertitude de l'état du véhicule à l'instant présent est obtenue par un filtre de Kalman avec l'estimation d'une matrice de covariance.

Nous considérons que les observations de position de véhicule résultent d'un processus cinématique à vitesse constante auquel s'ajoute des accélérations discrètes Gaussiennes centrées réduites constantes entre les les instants t et $t + dt$. La variable d'état qui décrit le véhicule est notée $X = (x, v_x, y, v_y)^T$. L'évolution de l'état entre les pas de temps k et $k + 1$ s'écrit comme suit :

$$X_{k+1} = AX_k + E\tilde{a}_k \tag{11.1}$$

Avec A la matrice de transition qui représente l'évolution du modèle :
$A = \begin{pmatrix} A_x & 0 \\ 0 & A_y \end{pmatrix}$ avec $A_x = A_y$. Pour un pas de temps dt, $A_x = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$.
E est la matrice de poids de l'accélération et $\tilde{a}_k = (\tilde{a}_{xk}, \tilde{a}_{yk})^T$ est l'accélération modélisée par un bruit.
$E = \begin{pmatrix} E_x & 0 \\ 0 & E_y \end{pmatrix}$ et $E_x = E_y$.
Pour un pas de temps dt, $E_x = \left(\frac{dt^2}{2}, dt\right)^T$.

Le filtre de Kalman peut être décomposé en trois temps : prédiction, calcul de l'innovation, mise à jour. La prédiction produit une estimation de l'état futur que l'on s'attend à observer $\hat{X}_{k+1|k}$ en utilisant l'estimation de l'état présent $\hat{X}_{k|k}$.

La matrice de covariance de l'erreur P est mise à jour à l'aide de la matrice de transition A et la matrice de covariance du bruit de processus Q. P est indépendante des observations. Lors du calcul de l'innovation, on associe l'observation $\tilde{Z}_k = (\tilde{x}_k, \tilde{y}_k)^T$ avec l'estimation de l'état X à l'aide de la matrice $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

La covariance d'innovation S est la somme de la covariance du bruit de l'état observé HPH^T et de la covariance du bruit d'observation R. L'étape de mise à jour utilise une nouvelle observation pour faire une mise à jour Bayesienne de l'état prédit.

Prédiction :

$$\begin{aligned} \hat{X}_{k+1|k} &= A\hat{X}_{k|k} \\ P_{k+1|k} &= AP_{k|k}A^T + Q \end{aligned} \tag{11.2}$$

Innovation :

$$\begin{aligned} e_{k+1} &= \tilde{Z}_{k+1} - H\hat{X}_{k+1|k} \\ S_{k+1} &= HP_{k+1|k}H^T + R \end{aligned} \tag{11.3}$$

Mise à jour :

$$\begin{aligned} K_{k+1} &= P_{k+1|k}H^T S_{k+1}^{-1} \\ \hat{X}_{k+1|k+1} &= \hat{X}_{k+1|k} + K_{k+1}e_{k+1} \\ P_{k+1|k+1} &= P_{k+1|k} - K_{k+1}H_{k+1}P_{k+1|k} \end{aligned} \tag{11.4}$$

Les équations (11.2), (11.3), et (11.4) sont utilisées sur la séquence d'observations passées pour atteindre une estimation de l'état présent à t_0. Ensuite, une prédiction cinématique est produite avec l'équation (11.2). Ce procédé de filtrage puis prédiction cinématique est décrit par l'algorithme 2.

Algorithm 2: Prédiction avec Kalman

données : $[\tilde{Z}_{k\in[\![-N_H,N_F]\!]}]^{(i)} \quad i \in [\![1, N]\!]$
entrées : Q, R
for $\tilde{Z}^{(i)}$ *dans les données* **do**
  définir: $\hat{X}^{(i)}_{-N_H}, P_{-N_H}$
  for k ***de*** $-N_H + 1$ ***à*** 0 **do**
    $\hat{X}^{(i)}_k, P_k \leftarrow \text{Kalman}_{\text{filtre}}(\tilde{Z}^{(i)}_k, \hat{X}^{(i)}_{k-1}, P_{k-1}, Q, R)$
  for k ***de 1 à*** N_F **do**
    $\hat{X}^{(i)}_k, P_k \leftarrow \text{Prédiction}_{\text{vc}}(\hat{X}^{(i)}_{k-1}, P_{k-1}, Q)$
    $\hat{Z}^{(i)}_k \leftarrow H\hat{X}^{(i)}_k$
    $\Sigma^{(i)}_k \leftarrow HP_kH^T$
return $(\hat{Z}^{(i)}_k, \Sigma^{(i)}_k)^{i=1,N}_{k=1,N_F}$

Pour chaque séquence de la base de données, l'état $\hat{X}_{-N_H}$ et la matrice de covariance P_{-N_H} sont initialisés avec les approximations (11.5) suivantes. On omet l'indice de la séquence $i \in [\![1, N]\!]$ parmi les N disponibles dans la base de données pour alléger l'écriture.

$$\begin{aligned} \hat{X}_{-N_H} &\leftarrow (\tilde{x}_{-N_H}, \frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt}, \tilde{y}_{-N_H}, \frac{\tilde{y}_{-N_H+1} - \tilde{y}_{-N_H}}{dt})^T \\ P_{-N_H} &\leftarrow \text{diag}(\sigma_{xx}, \sigma_{v_xv_x}, \sigma_{yy}, \sigma_{v_yv_y}) \end{aligned} \tag{11.5}$$

11.2.2 Estimation des paramètres

Il reste à définir les paramètres Q et R. Le bruit de processus représente les changements de vitesse et de position dûs à l'accélération modélisée par une Gaussienne centrée à temps discret. La covariance d'accélération à chaque pas de temps est estimée par $q \approx \overline{\tilde{a}^2}$ en $m^2.s^{-4}$. On écrit alors la matrice de covariance de processus comme suit :

$$Q = \begin{pmatrix} dt^4/4 & dt^3/2 & 0 & 0 \\ dt^3/2 & dt^2 & 0 & 0 \\ 0 & 0 & dt^4/4 & dt^3/2 \\ 0 & 0 & dt^3/2 & dt^2 \end{pmatrix} \text{diag}(q_x, q_x, q_y, q_y)$$

Nous estimons la covariance de l'état initial avec l'égalité $\sigma_{v_xv_x} = \frac{2\sigma_{xx}}{dt^2}$ qui nous permet d'approximer σ_{xx} et l'estimation grossière de $\sigma_{v_xv_x}$ qui suit :

$$\sigma_{v_xv_x} = \overline{(\tilde{v}_x - \overline{\tilde{v}_x})^2} = \overline{\left(\frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt} - \overline{\frac{\tilde{x}_{-N_H+1} - \tilde{x}_{-N_H}}{dt}}\right)^2}$$

Enfin, l'accélération est estimée avec l'équation suivante :

$$\tilde{a}^{(i)}_{k+1} = \frac{x^{(i)}_{k+2} - 2x^{(i)}_{k+1} + x^{(i)}_k}{dt^2}$$

$\overline{\tilde{a}^2}$ est une approximation de la variance car l'accélération moyenne est quasi-nulle. Le bruit d'observation R est établi arbitrairement avec une valeur faible.

11.2.3 La base de données NGSIM

La base de données Next Generation SIMulation (NGSIM) [CH07] rassemble des observations de trajectoires de véhicules sur autoroute. La figure 11.1 représente l'installation d'acquisition et des échantillons d'observation. Nous utilisons le pré-traitement de Deo *et al.* [DT18] pour obtenir la même base de données composée de séquences de 8 secondes contenant les trajectoires d'une sous-partie du tronçon de route observé. La figure 11.2, représente un exemple de séquence avec 3 secondes d'observation passées et 5 secondes dans le futur.

Figure 11.1: Système d'acquisition, et échantillons d'observations brutes et traitées. (source: [CH07])

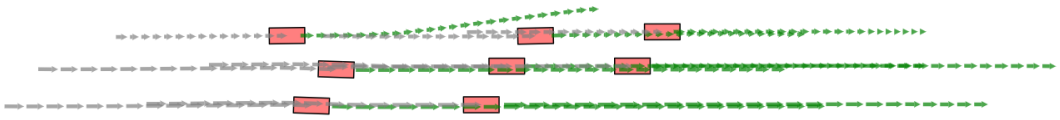


Figure 11.2: Exemple de séquence de la base de données avec 3s de trajectoires passées en gris et 5s de trajectoires futures en vert.

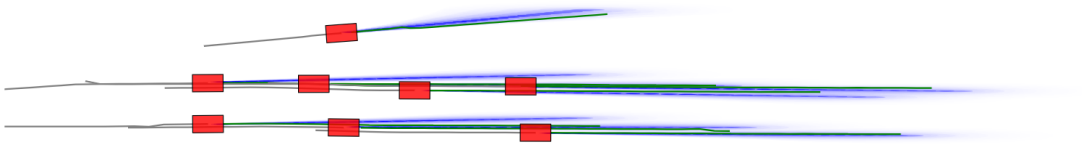


Figure 11.3: Exemple de prédictions à vitesses constantes représentées par des ellipses de dispersion unitaires bleues aux différents temps futurs.

11.2.4 Évaluation

Les paramètres du modèle de Kalman peuvent être calculés en utilisant les données de la base d'entrainement et sont reportés dans le tableau 11.1.

Table 11.1: Paramètres du modèle à vitesse constante.

Paramètre	$\sigma_x(m)$	$\sigma_y(m)$	$\sigma_{v_x}(m.s^{-1})$	$\sigma_{v_y}(m.s^{-1})$
Valeurs calculées	0.04	0.57	0.20	4.01

Paramètre	$q_x(m^2.s^{-4})$	$q_y(m^2.s^{-4})$
Valeurs calculées	0.93	2.82

Paramètre	$r_{11}(m^2)$	$r_{22}(m^2)$	$r_{12}(m^2)$
Valeurs choisies	0.01	0.01	0

Les prédictions Gaussiennes produites par le modèle sont représentées sur la figure 11.3.

11.3 Évaluation des prédictions

Dans cette section, nous considérons que les prédictions sont formulées comme des séquences de mélanges Gaussiens et nous établissons les critères d'évaluation permettant de mesurer la performance de nos modèles et ceux de la littérature. Nous fixons le nombre de composantes du mélange Gaussien à n_{mix}. Chaque composante décrit un choix qui peut être fait par l'agent considéré. À chaque pas de temps elle est décrite par sa position moyenne, sa matrice de covariance et sa probabilité : $((\hat{x}, \hat{y}), \Sigma, p)$. Avec $\mu = (\hat{x}, \hat{y})$ et pour tout $z \in \mathbb{R}^2$, sa fonction de densité de probabilité s'écrit comme suit :

$$G_{\text{PDF}}(\mu, \Sigma)(z) = \frac{1}{2\pi \left|\Sigma\right|^{1/2}} \; e^{-\frac{1}{2}(z-\mu)^\top \Sigma^{-1}(z-\mu)} \tag{11.6}$$

La densité de probabilité du mélange Gaussien s'écrit :

$$GM_{\text{PDF}}(\{\mu_m, \Sigma_m, p_m\}_{m \in [\![1, n_{\text{mix}}]\!]}) = \sum_{m=1}^{n_{\text{mix}}} p_m G_{\text{PDF}}(\mu_m, \Sigma_m) \tag{11.7}$$

11.3.1 Les entrées - sorties

Les entrées sont des séquences de positions (x, y) des véhicules de la scène et des centres de voies alentour. À chaque temps t_0, une séquence passée de longueur maximum fixée n_{hist} notée $\{(x, y)_k\}_{k=-n_{\text{hist}}+1,0}$ est définie pour chaque véhicule observé. Des séquences spatiales des centres de voies peuvent aussi être utilisées. Le repère est centré et aligné sur le véhicule ego au temps t_0.

Les sorties sont des séquences de mélanges Gaussiens pour chaque véhicule de la scène. Elles sont exprimés comme suit : $(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho, p)_{v,k,m}$ pour chaque véhicule v, à chaque pas de temps k et chaque composante m. La matrice de covariance est définie avec ces paramètres :

$$\Sigma_{v,k,m} = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}_{v,k,m}$$

Le modèle de prédiction est un ensemble de fonctions pred_θ : entrées $\rightarrow$ sorties dépendant de l'ensemble de paramètres noté θ.

11.3.2 Indicateurs de performance

Les FDE et RMSE dans le cas multi-modal sont définis de deux façons possibles pour n_{mix} propositions de trajectoires futures. Le FDE (Final Displacement Error) mesure la distance de l'erreur commise après k pas de temps de prédiction. La moyenne de ces erreurs sur N trajectoire est calculée. Le RMSE (Root Mean Squared Error) mesure la racine carré de l'erreur quadratique moyenne. De même, il est calculé à l'étape k sur N trajectoires considérées.

• Seule la proposition jugée la plus probable est considérée (évaluation pessimiste) :

$$\text{RMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} \mathbb{1}_{p_m^{(i)}=p_{\text{max}}^{(i)}} \left((\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2\right)}$$

$$\text{FDE}(k) = \frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{n_{\text{mix}}} \mathbb{1}_{p_m^{(i)}=p_{\text{max}}^{(i)}} \sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,m}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,m}^{(i)})^2}$$

• Seule la trajectoire dont la position finale est la plus proche de la position finale effectivement observée est considérée (évaluation optimiste) :

$$\text{minRMSE}(k) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (\tilde{x}_k^{(i)} - \hat{x}_{k,\text{min}}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,\text{min}}^{(i)})^2}$$

$$\text{minFDE}(k) = \frac{1}{N}\sum_{i=1}^{N} \sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,\text{min}}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,\text{min}}^{(i)})^2}$$

Le taux d'erreur (MR pour "miss rate") est le taux de séquences pour lesquelles aucune des propositions de positions finales n'est à moins de 2 mètres de la position observée.

$$\text{MR}(k) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\sqrt{(\tilde{x}_k^{(i)} - \hat{x}_{k,\text{min}}^{(i)})^2 + (\tilde{y}_k^{(i)} - \hat{y}_{k,\text{min}}^{(i)})^2} > 2} \tag{11.8}$$

11.3.3 Indicateur probabiliste

La log-vraisemblance négative (NLL pour "negative log-likelihood") décrite dans [Bis94] est utilisée à la fois comme indicateur de performance et comme fonction de coût à minimiser pour les modèles entrainés par minimisation d'un coût.

On note $\tilde{\mathbf{Z}}_k$ la position observée au temps t_k, $\tilde{\mathbf{Z}}_h$ l'ensemble des positions précédemment observée d'une séquence, et $\hat{\mathbf{Z}}_k|_{\tilde{\mathbf{Z}}_h}$ la position prédite au temps t_k. La distribution de position prédite est exprimée par un mélange Gaussien. On note $f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}$ l'estimation de densité de probabilité de position au temps t_k. Ainsi la NLL est donnée par :

$$\text{NLL}(k) = -\ln\left(f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_\mathbf{h}}(\tilde{\mathbf{Z}}_k)\right) \tag{11.9}$$

Pour une distribution Gaussienne en deux dimensions, $f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}$ s'écrit comme suit :

$$f_{\hat{\mathbf{Z}}_k|\tilde{\mathbf{Z}}_h}(\tilde{z}_k) = \frac{1}{2\pi\sqrt{|\Sigma_k|}}\exp\left(-\frac{1}{2}(\tilde{z}_k-\hat{z}_k)^T\Sigma_k^{-1}(\tilde{z}_k-\hat{z}_k)\right) \tag{11.10}$$

Les erreurs selon x et y au temps t_k sont notées d_x et d_y (on omet l'indice temporel k pour alléger l'écriture). Alors la NLL_k au pas de temps k est donnée par l'équation (11.11).

$$\begin{aligned}\text{NLL}_k^{(i)}(dx, dy, \Sigma) =& \underbrace{\frac{1}{2}\frac{1}{(1-\rho^2)}\left(\frac{d_x^2}{\sigma_x^2}+\frac{d_y^2}{\sigma_y^2}-2\rho\frac{d_x d_y}{\sigma_x\sigma_y}\right)}_{(\tilde{z}_k-\hat{z}_k)^T\Sigma_k^{-1}(\tilde{z}_k-\hat{z}_k)} \\ &+ \underbrace{\ln\left(\sigma_x\sigma_y\sqrt{1-\rho^2}\right)}_{\ln(\sqrt{|\Sigma_k|})} \\ &+ \ln(2\pi)\end{aligned} \tag{11.11}$$

Pour un mélange Gaussien, la NLL se calcule avec l'équation suivante :

$$\text{NLL}_k^{(i)}\left(\{dx, dy, \Sigma, p\}_{m\in[\![1,n_{\text{mix}}]\!]}\right) = -\ln\left(\sum_{m=1}^{n_{\text{mix}}} p_m e^{-\text{NLL}_k^{(i)}(dx_m, dy_m, \Sigma_m)}\right) \tag{11.12}$$

On utilise la moyenne de la NLL sur les séquences de la base de données pour évaluer la prédiction globalement avec l'équation (11.13)

$$\text{NLL}_k = \frac{1}{N}\sum_{i=1}^{N}\text{NLL}_k^{(i)} \tag{11.13}$$

11.3.4 Validation de la covariance

La prédiction est exprimée comme un mélange Gaussien, cependant, dans chaque séquence, un seul futur est observé. Nous voudrions valider l'estimation de la distribution mais nous ne connaissons pas au cas par cas la distribution réelle des futurs possibles. Dans le cas d'une prédiction Gaussienne simple, il est possible de comparer la moyenne des covariances prédites sur plusieurs séquences avec la covariance des erreurs commises dans ces mêmes séquences. Cela permet de valider l'estimation de la covariance en moyenne en comparant graphiquement les ellipses de dispersion unitaires des covariances prédites et de la covariance d'erreur commise.

11.3.5 Application à l'évaluation de la prédiction à vitesse constante

Dans le cas de la prédiction à vitesse constante appliquée à la base de données NGSIM, nous validons l'estimation de la covariance d'erreur avec la représentation de la figure 11.4.

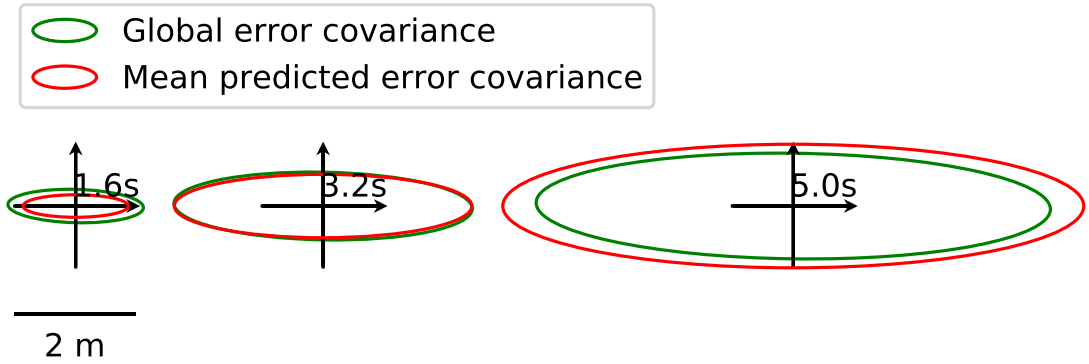


Figure 11.4: Comparaison graphique entre la moyenne des covariances d'erreurs prédites et la covariance de l'erreur commise. Les ellipses de dispersion unitaires représentent la distribution d'erreur observée en vert et la moyenne des distributions d'erreur estimée a priori en rouge.

De même, nous évaluons la prédiction à vitesse constante avec les critères décrits plus haut dans le tableau 11.2. Ces résultats montrent des performances bien insuffisantes avec 75% d'erreur finale supérieure à 2 mètres.

Table 11.2: Évaluation de la prédiction à vitesse constante sur la base de données NGSIM.

Horizon de temps	1s	2s	3s	4s	5s
RMSE (m)	1.38	2.94	4.67	6.59	8.75
FDE (m)	0.65	1.61	2.79	4.19	5.77
NLL	2.88	3.66	4.41	5.04	5.57
MR	0.05	0.26	0.49	0.65	0.75

11.4 Modéliser le comportement humain

L'application fait le modèle - La prédiction de trajectoire n'est pas une fin en soi. Elle sert comme outil dans un système plus large qui peut bénéficier de différentes façons d'anticiper les mouvements de la scène routière. La prédiction peut s'exprimer de manière sémantique, ou en décrivant l'espace libre ou encore en prédisant les positions futures des agents occupant la scène routière. À cette variété d'objectifs, la littérature scientifique répond avec une variété de solutions. Une vue générale est présentée dans [LVL14] qui sépare les modèles en *dynamiques*, *cinématiques*, *classification de manœuvres*, différentes façon de simuler des trajectoires et enfin des modèles d'évaluation des risques. Une autre étude de littérature plus récente du blog de Stanford AI [1] fondée sur le papier [Sal+20] sépare les approches de prédiction en ontologiques (théorie de l'esprit) et phénoménologiques (fondées sur les données). Nous séparons les approches de prédiction de trajectoire en *Lois heuristiques de comportement*, *théorie des jeux*, *classification de manœuvres*, et *prédictions statistiques*.

Lois heuristiques de comportement - Les approches heuristiques définissent un modèle de comportement fondé sur quelques paramètres interprétables. Par exemple, le modèle IDM-MOBIL [THH00] et [KTH07] sépare le comportement longitudinal du comportement latéral en employant des règles de comportement gouvernées par des distances de sécurité souhaitées, un facteur d'agressivité, un niveau d'accélération, et un temps de réaction. On trouve aussi un modèle des forces sociales [HM95] qui modélise le comportement des agents avec des forces attractives et répulsives permettant d'atteindre un objectif en évitant les obstacles.

Approches historiques pour la conduite autonome - Le DARPA Urban Challenge (DUC) [BIS09] a été la première compétition d'envergure visant la conduite autonome. Les prédictions de trajectoires employées par les participants ont suivi des approches cinématiques ou d'optimisation de règles heuristiques. Certains ont utilisé plusieurs modèles sélectionnés selon le contexte.

Prise de décision - Les modèles de prise de décision considèrent que les agents prennent des décisions de conduite qui optimisent un objectif. Cette approche doit trouver un équilibre de Nash pour les stratégies de conduites des différents agents. Elle permet la prise en compte d'interactions complexes avec des négociations. Cependant, cette approche bien que très pertinente dans des environnements simulés est difficile à transférer aux applications réelles.

Classification de manœuvre - Cette approche peut être employée pour deux raisons: s'en tenir à une prédiction sémantique [Wan+18; SSS17; CH06] des manœuvres ou combiner une la classification de manœuvres avec une prédiction de trajectoire pour décrire différentes possibilités [Hou14; DT18].

Prédictions statistiques - Ici, on cherche à décrire la trajectoire future comme une distribution de trajectoires conditionnées aux observations. On apprend cette distribution conditionnelle à l'aide d'une base de données. L'apprentissage de cette distribution peut se faire avec différentes méthodes : modèles de markov cachés (HMM) [OP00; AK07], réseaux Bayésiens [SWA14], ou plus récemment et avec un très large succès, les réseaux neuronaux [AL17].

[1] `http://ai.stanford.edu/blog/trajectory-forecasting/`

11.5 Premières applications

Les réseaux neuronaux forment une classe de fonctions dérivables dépendant d'un ensemble de paramètres θ de grande dimension. Ces paramètres déterminent la forme de la fonction et sont ajustés dans une procédure appelée apprentissage. Pour effectuer l'apprentissage, la méthode communément employée est la minimisation d'une fonction de coût avec la descente du gradient. Une base de données d'exemples d'entrées et de sorties attendues est utilisée de manière à ce que le réseau neuronal produise une sortie proche de celle attendue pour une entrée correspondante.

Nous appliquons trois types de réseaux neuronaux simples à la prédiction de trajectoires individuelles : totalement connecté, convolutif, et récurrent (LSTM). Ils produisent pour chaque instant futur un ensemble de paramètres o_i définissant une Gaussienne avec la fonction d'activation de sortie définie ci-dessous.

$$(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho) = \text{activation}(o_1, o_2, o_3, o_4, o_5) = (o_1, o_2, e^{\frac{o_3}{2}}, e^{\frac{o_4}{2}}, \tanh(o_5))$$

Nous appliquons la descente du gradient pour minimiser la NLL. Cela nous permet d'entrainer ces modèles ainsi que d'optimiser les paramètres du filtre de Kalman pour la prédiction de trajectoire à vitesse constante. Les réseaux neuronaux employés pourraient être modifiés et entrainés plus longuement pour légèrement améliorer les résultats. Cependant, au vu des résultats du tableau 11.3, on ne peut pas espérer de nette amélioration par rapport à la prédiction à vitesse constante. Les trajectoires traitées individuellement ne permettent pas de prendre en compte les interactions.

Table 11.3: Résultats obtenus avec une prédiction à vitesse constante et des réseaux neuronaux simples.

Horizon de temps		1s	2s	3s	4s	5s
RMSE (m)	Vitesse constante	0.75	1.81	3.16	4.80	6.69
	Totalement connecté	0.72	1.73	3.02	4.60	6.43
	Convolutif	0.71	1.73	3.03	4.61	6.45
	Récurrent	0.70	1.71	3.00	4.60	6.49
FDE (m)	Vitesse constante	0.46	1.24	2.27	3.53	4.99
	Totalement connecté	0.47	1.24	2.26	3.51	4.95
	Convolutif	0.46	1.23	2.25	3.50	4.96
	Récurrent	0.47	1.25	2.28	3.55	5.05
NLL	Vitesse constante	0.81	2.31	3.22	3.91	4.46
	Totalement connecté	0.37	2.02	2.98	3.68	4.24
	Convolutif	0.32	2.00	2.97	3.67	4.22
	Récurrent	0.33	2.01	2.99	3.70	4.26
MR	Vitesse constante	0.02	0.20	0.44	0.61	0.71
	Totalement connecté	0.02	0.19	0.44	0.62	0.74
	Convolutif	0.02	0.19	0.43	0.62	0.74
	Récurrent	0.02	0.19	0.44	0.64	0.75

11.6 Interactions entre plusieurs agents

Dans cette section, nous étendons les modèles de prédictions pour qu'ils prennent en compte les interactions entre agents. Les statistiques des comportements d'interactions peuvent être apprises par des réseaux neuronaux. Pour cela il faut modifier les modèles établis précédemment pour que les observations de plusieurs agents soient utilisées simultanément en variable d'entrée. Deux problèmes importants se posent alors : les réseaux neuronaux usuels demandent un nombre fixe d'entrées lors de leur définition; l'ordre des différentes entrées doit avoir un sens. Ceci ne correspond pas à nos données dans lesquelles figurent un nombre variable de véhicules d'une scène à l'autre. De plus la liste des véhicules observés n'est pas ordonnée.

Plusieurs solutions sont employées dans la littérature :

- Liste de dimension fixe avec ordonnancement
- Images rasterisées de la scène
- Grilles larges
- Graphes d'interactions

Les trois premières propositions visent à modifier la représentation des données pour l'adapter aux réseaux neuronaux tandis que la dernière modifie l'architecture des réseaux neuronaux pour s'adapter aux données. Nous tentons une première approche employant une grille large avant de nous orienter vers une méthode employant les graphes d'interaction.

11.6.1 Peloton d'obstacles

Notre première approche représente la scène environnant le véhicule ego sous la forme d'un peloton de 6 obstacles. Chaque obstacle est lié à une zone. Une zone est une portion de voie directement adjacente au véhicule égo comme représenté sur la figure 11.5. L'agent le plus proche dans chaque zone est représenté par un obstacle tandis qu'une zone vide est représentée par un obstacle à une distance de saturation. La représentation peut présenter des discontinuité, notamment quand un agent change de zone : l'obstacle de la zone quittée est défini par un nouvel agent ou l'extrémité de zone, l'obstacle de la zone d'arrivée représente éventuellement cet agent.

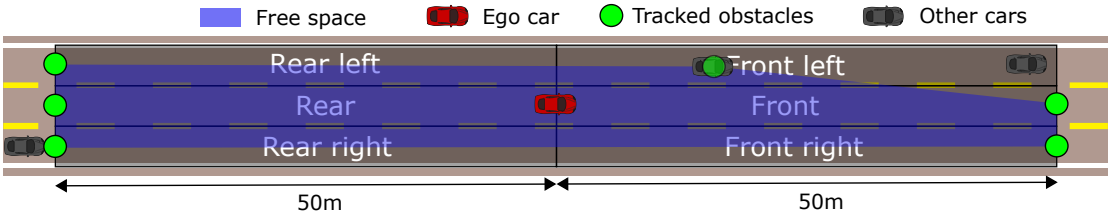


Figure 11.5: Représentation de la scène par peloton. Dans chaque zone, l'agent le plus proche est représenté par un obstacle. Les six obstacles forment un polygone définissant l'espace libre entourant le véhicule ego.

Cette représentation de la scène routière a l'avantage de mettre l'accent sur les évènements importants en associant une manœuvre de changement de zone à une discontinuité. Cependant ces discontinuités causent des difficultés durant l'apprentissage.

De plus, la représentation relative aux voies ne permet pas d'étendre cette représentation à des scènes prenant place sur un réseau routier plus complexe, à une intersection par exemple. Ce manque de flexibilité nous pousse à nous diriger vers d'autres approches de prise en compte des interactions.

11.6.2 Graphes d'interactions avec auto-attention

Il est possible de représenter la scène routière sous forme de graphe en considérant chaque agent comme un nœud du graphe et en connectant les agents interagissant entre eux par des arêtes. Dans notre cas le nombre d'agents est restreint et nous pouvons nous permettre de représenter toutes les interactions possibles avec un graphe complet orienté. L'auto-attention développée dans [Vas+17] définit une méthode permettant d'apprendre à associer un poids scalaire à chaque arête du graphe d'interaction quel que soit le nombre de nœuds. Une opération simple permet aux nœuds du graphe de partager de l'information. C'est une somme pondérée des tenseurs de valeur associés à chaque nœud. La pondération est donnée par les poids des arêtes. Cette opération peut être faite pour tout nombre de nœuds et est invariante pour l'ordre dans lequel les nœuds voisins sont listés. Les poids des arêtes sont vus comme des facteurs d'attention. Une valeur élevée caractérise une interaction importante. Plusieurs instances similaires de définition de poids des arêtes et de moyenne des valeurs des nœuds sont effectuées pour la même entrée; on appelle chaque instance une tête d'attention et on dit que l'on utilise un modèle d'auto-attention à multiples têtes.

11.6.2.1 Réseaux neuronaux avec auto-attention

L'opération décrite ci-dessus peut être effectuée au sein d'un réseau neuronal de sorte qu'à l'apprentissage la définition des graphes d'interaction soit apprise et dépende des données d'entrée. La même fonction d'encodage est utilisée pour chaque trajectoire d'agent en entrée de manière à produire un tenseur encodé représentant cette trajectoire dans un espace latent. C'est entre ces tenseurs encodés que l'interaction est faite. Pour cela, le modèle contient trois matrices de poids qui, multipliées aux tenseurs encodés, associent à chaque agent trois tenseurs : une valeur v, une requête q, et une clé k. Le tenseur de valeur est l'état des nœuds du graphe tandis que les tenseurs q et k permettent de définir les poids des arêtes du graphe. Le poids d'une arête est donné par la normalisation du produit scalaire des vecteurs de requête et de clé des nœud qu'elle relie. C'est donc le produit scalaire de la requête et de la clé normalisé. La normalisation est la fonction softmax. On peut donc réécrire le poids de l'arête : $\frac{e^{q_i \cdot k_j}}{\sum_{j'} e^{q_i \cdot k_{j'}}}$.
Finalement chaque tête d'attention pour chaque nœud fait le calcul suivant :

$$\sum_j \frac{e^{q_i \cdot k_j}}{\sum_{j'} e^{q_i \cdot k_{j'}}} v_j$$

Ce calcul est représenté matriciellement sur la figure 11.6 qui représente un modèle de prédiction complet.

Le modèle encode chaque trajectoire à l'aide d'une couche de convolution avec un noyau de taille 3 et comptant 60 filtres. La sortie de cette première couche est un tenseur de taille 60 pour chaque pas de temps. Un LSTM est utilisé pour encoder la séquence temporelle en un unique tenseur sans dimension temporelle. Ensuite 6

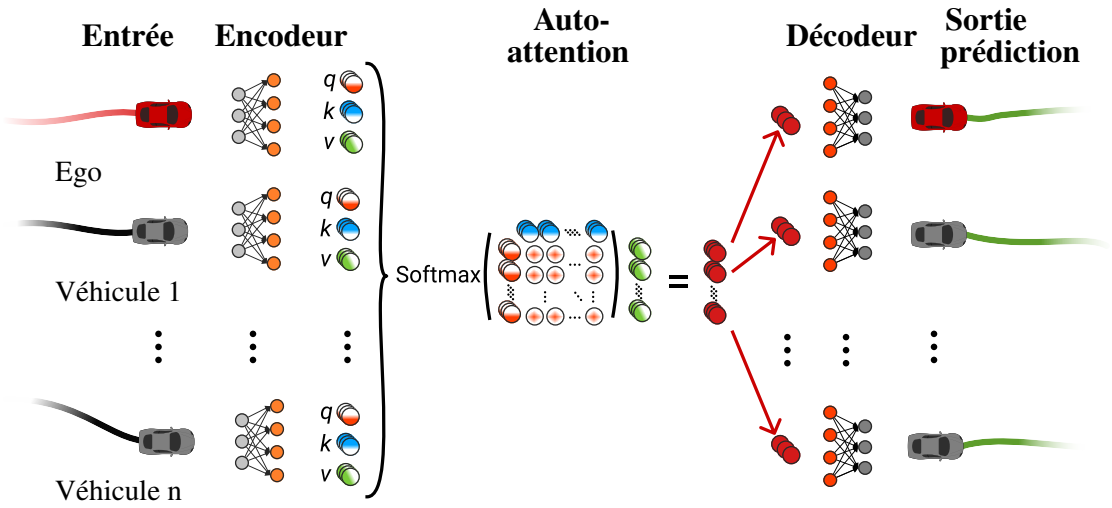
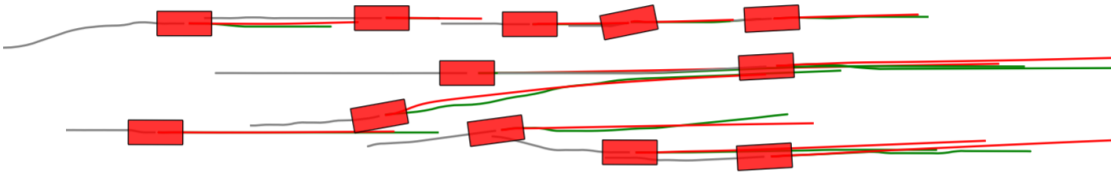
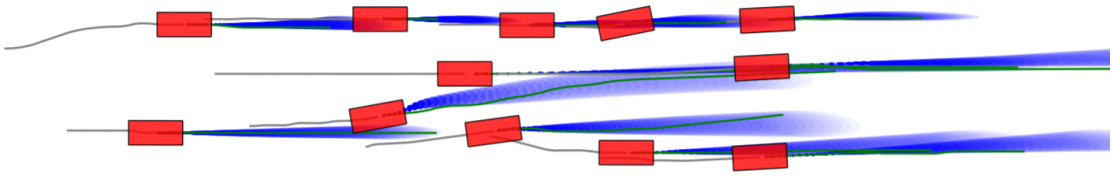


Figure 11.6: Modèle de réseau neuronal avec auto-attention.

têtes d'attention sont employées pour inclure des interactions entre les agents dans le modèle. Les tenseurs résultants sont décodés avec un deuxième LSTM qui reforme une séquence temporelle pour chaque agent puis des couches de sorties forment la prédiction sous forme de Gaussienne pour chaque pas de temps futur. Un exemple de résultat est représenté par la figure 11.7.



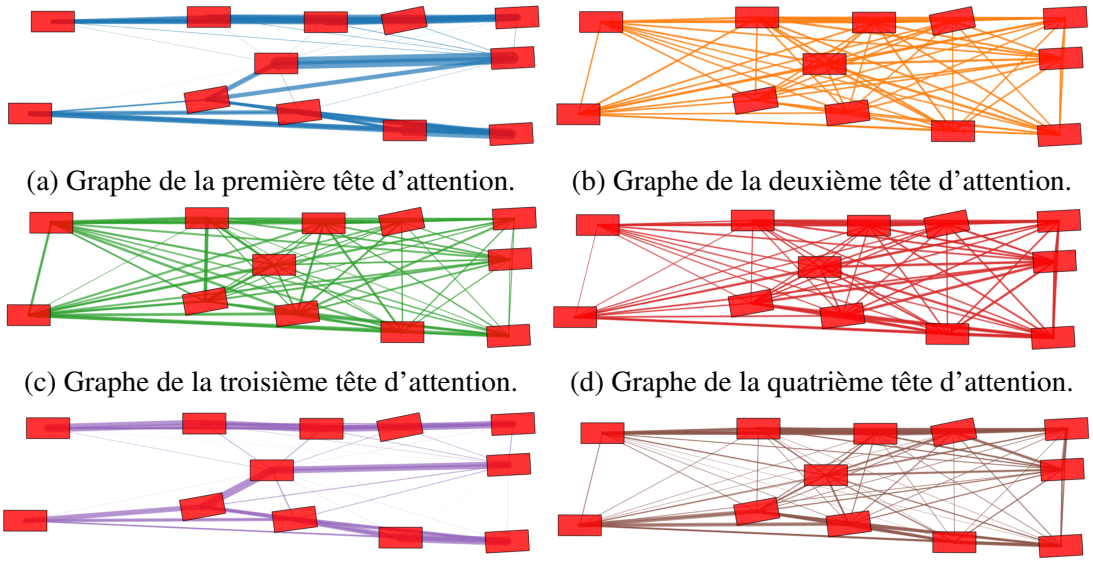
(a) Représentation de la prédiction avec les trajectoires suivant les maximums de vraisemblances prédites représentées en rouge.



(b) Représentation de la prédiction avec des ellipses de dispersion unitaires bleues délimitant des zones de probabilité 39% pour chaque pas de temps de la séquence future.

Figure 11.7: Représentation d'une scène de la base de données NGSIM. Les trajectoires passées sont représentées en gris et les trajectoires futures en vert.

Graphes d'attention - La figure 11.8 représente tous les graphes d'attention que le modèle associe à une situation particulière. On observe une spécialisation des différentes têtes d'attention. Certains graphes peuvent même être interprétés : la première tête en bleu prête attention à tous les véhicules de devant, la cinquième tête en violet prête attention au véhicule de devant le plus proche. Comme l'apprentissage n'est supervisé que par les trajectoires futures, il n'est pas étonnant que d'autres ne soient pas interprétables.



(a) Graphe de la première tête d'attention.

(b) Graphe de la deuxième tête d'attention.

(c) Graphe de la troisième tête d'attention.

(d) Graphe de la quatrième tête d'attention.

(e) Graphe de la cinquième tête d'attention.

(f) Graphe de la sixième tête d'attention.

Figure 11.8: Représentation des graphes d'interaction d'une scène routière de la base de données NGSIM. Les épaisseurs des arêtes liant deux nœuds sont proportionnelles aux coefficients d'attention.

La comparaison des ellipses de la figure 11.9 montre que globalement l'estimation de la covariance d'erreur correspond bien à la covariance d'erreur commise.

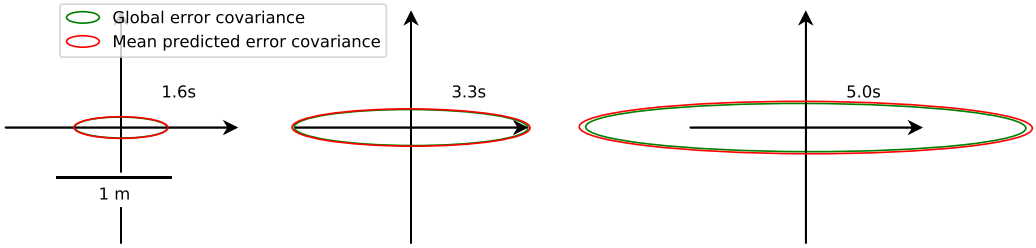


Figure 11.9: Comparaison des ellipses de dispersion unitaires pour la moyenne des covariances d'erreur prédites (en rouge) et pour la covariance de l'erreur commise en vert.

Les résultats obtenus par ce modèle sur la base de validation, présentés dans le tableau 11.4, montrent que le modèle avec auto-attention améliore nettement les performances. La distance d'erreur moyenne (FDE) à 5 secondes passe de 5 mètres à moins de 3 mètres. La log-vraisemblance négative est aussi nettement améliorée. Cependant, le taux d'erreur finale à plus de 2 mètres obtenu, bien que lui aussi largement diminué, reste supérieur à 50% à 5 secondes. Ce n'est pas satisfaisant.

Table 11.4: Résultats comparés du modèle d'auto-attention avec le modèle à vitesse constante et un modèle utilisant une liste ordonnée de taille fixée des véhicules observés.

Horizon de temps		1s	2s	3s	4s	5s
RMSE (m)	Vitesse constante	0.75	1.81	3.16	4.80	6.69
	Liste	0.69	1.58	2.72	4.12	5.78
	Auto-attention	0.48	1.10	1.85	2.81	4.00
FDE (m)	Vitesse constante	0.46	1.24	2.27	3.53	4.99
	Liste	0.45	1.12	1.97	3.04	4.32
	Auto-attention	0.31	0.78	1.36	2.07	2.95
NLL	Vitesse constante	0.81	2.31	3.22	3.91	4.46
	Liste	0.24	1.67	2.55	3.19	3.71
	Auto-attention	-0.57	0.99	1.90	2.56	3.09
MR	Vitesse constante	0.02	0.20	0.44	0.61	0.71
	Liste	0.01	0.15	0.37	0.56	0.68
	Auto-attention	0.01	0.06	0.22	0.39	0.54

11.7 Prédictions multimodales

La raison de l'insuffisance du modèle développé dans la section précédente est décrite par la figure 11.10. Un modèle ne prédisant qu'une seule trajectoire donne de mauvais résultats dans les situations où plusieurs trajectoires futures distantes sont possibles.

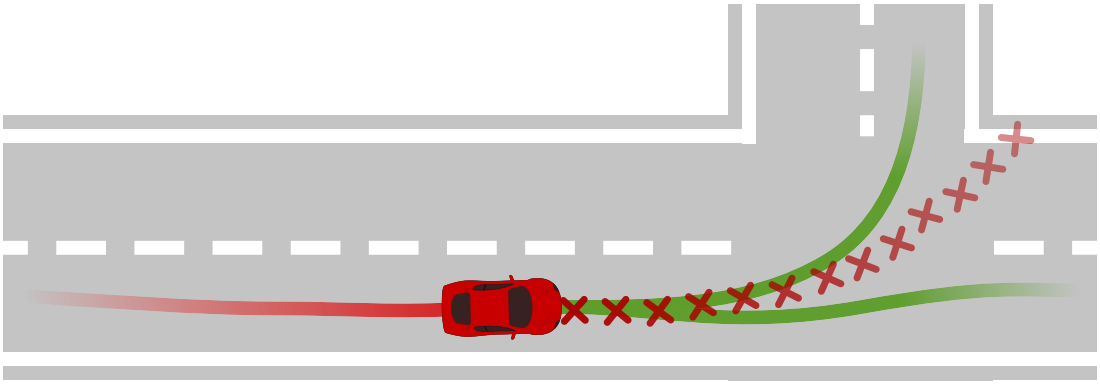


Figure 11.10: Illustration d'une situation menant à deux futurs probables très différents et pour laquelle la moyenne des futurs possible est une trajectoire très improbable.

Les solutions pour prédire plusieurs futurs possibles peuvent être associées à deux catégories : les méthodes par échantillonnage d'une loi de probabilité et les méthodes de régression de distribution.

Les méthodes par échantillonnage dépendent à la fois des observations et d'un échantillon d'une distribution connue. Pour chaque couple d'observation et d'échantillon, une seule trajectoire par véhicule de la scène est prédite. La distribution des différents futurs possibles est estimée en échantillonnant la distribution d'entrée et en exécutant plusieurs fois le modèle de prédiction. Ce type de modèle est dit génératif. Parmi les modèles génératifs, ceux principalement utilisés sont les modèles génératifs adverses (GAN) et les auto-encodeurs variationnels (VAE). Le problème principal lié à ces méthodes est la nécessité d'échantillonner le modèle pour produire les différentes prédictions.

À chaque échantillonnage, des calculs importants doivent être effectués. Ceci limite le nombre d'échantillons que l'on peut effectuer en temps réel et donc la capacité de prédiction des évènements les moins probables. Nous nous dirigeons donc vers des méthodes de régression de distribution.

La régression de distribution vise à décrire la distribution de probabilité de présence des différents agents dans la scène routière en une seule exécution du modèle de prédiction. Les méthodes de régression de distribution utilisent un a priori de distribution des positions futures. Une approche possible est la carte d'occupation qui utilise a priori une loi uniforme discrète d'occupation de la carte considérée. La loi a priori présentée et utilisée dans cette thèse est le mélange Gaussien qui permet d'alléger la représentation et correspond bien aux observations. Un mélange Gaussien est décrit avec deux facteurs : une variation Gaussienne centrée sur chaque mode et une distribution discrète de modes. Cette séparation en deux distributions discrètes peut être explicitement employée en utilisant un modèle de classification de manœuvres couplé avec un modèle de régression Gaussienne de trajectoires réalisant chaque manœuvre. Cependant, cette approche demande à définir a priori les différentes manœuvres possibles ce qui est difficile à faire correspondre aux nombreuses situations observables.

11.7.1 Prédictions paramétriques : mélanges Gaussiens

Schöller *et al.* [Sch+20] ont développé un modèle de prédiction multi-modal à vitesse constante pour les piétons et obtiennent des performances étonnamment bonnes. Ceci démontre que la comparaison fréquente des modèles de prédictions multi-modales avec un simple modèle unimodal est insuffisante. Pour émettre plusieurs modes de prédictions avec un modèle à vitesse constante, Schöller *et al.* estiment la direction initiale et le module de la vitesse de chaque piéton avec seulement les deux dernières observations de position. Des modifications de cette orientation initiale sont opérées en échantillonnant des angles de rotation selon une Gaussienne d'écart-type $25°$. Pour chaque nouvelle direction, une prédiction cinématique à vitesse constante est émise. Ceci définit très simplement un modèle émettant plusieurs prédictions possibles. Nous étendons ce modèle à la prédiction de trajectoire de véhicules. Pour cela, nous utilisons le filtre de Kalman pour estimer la position et la vitesse au lieu d'utiliser simplement les deux dernières mesures. Cela nous permet d'obtenir une estimation plus robuste de l'état initial lorsque les observations sont bruitées. Nous modifions non seulement l'orientation initiale mais aussi le module de la vitesse initiale.

Les variations de l'état initial sont faites en échantillonnant les lois de probabilité d'une modification d'angle θ et d'une modification du module de la vitesse $dv = \frac{v}{v_{\text{obs}}} - 1$ pour un module de vitesse observé v_{obs} et un module de vitesse modifié v. Notons dx et dy les distances entre la prédiction à vitesse constante unimodale à un temps donné et la position observée. Alors, $dx = v_{\text{obs}} \times t \times dv$ et pour des petits angles $dy \approx v_{\text{obs}} \times t \times \theta$. Pour définir la distribution des modifications d'états initiaux qui correspond à la distribution d'erreur de prédiction commise par le modèle unimodal, il nous suffit d'observer la distribution d'erreur normalisée par la vitesse. Une représentation de cette distribution est donnée figure 11.11.

Nous avons ainsi défini un modèle multi-modal de prédiction de trajectoire que l'on peut échantillonner pour produire différentes prédictions. Nous voulons fixer le nombre de prédictions émises. Pour cela, nous discrétisons la distribution de modification

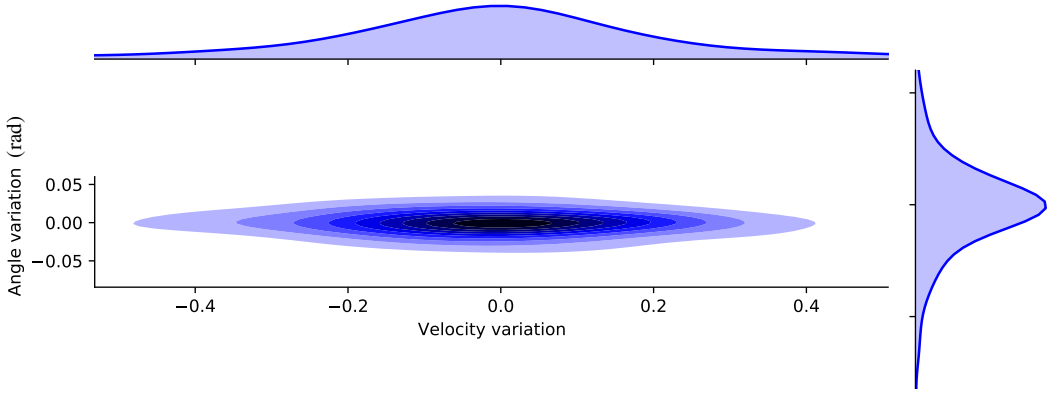


Figure 11.11: Distribution de modification d'état initial calculée avec les données NGSIM.

Table 11.5: Évaluation de la prédiction multi-modale avec $n_{\text{mix}} = 6$ modes

Horizon de temps	1s	2s	3s	4s	5s
RMSE (m)	1.73	3.58	5.59	7.77	10.08
minRMSE (m)	0.88	1.54	2.15	2.84	3.82
FDE (m)	1.40	2.87	4.43	6.11	7.89
minFDE (m)	0.64	1.08	1.42	1.74	2.21
NLL	2.21	3.08	3.89	4.59	5.22
MR	0.02	0.13	0.21	0.25	0.29

d'état initial. Nous proposons de déterminer les probabilités de chaque mode par la probabilité de la cellule de Voronoï associée à chaque modification discrète pour la distribution de modification. La prédiction de covariance est, quant à elle, répartie entre les différents modes. Malheureusement, les résultats présentés dans le tableau 11.5 ne montrent pas une grande pertinence de cette procédure : la NLL passe de 4.46 à 5s pour la prédiction unimodale à 5.22 pour la prédiction multimodale. Cependant, il est remarquable d'observer que le taux d'erreur final à plus de 2 mètre obtenu est inférieur à 30%. Pour le même nombre de modes prédits sur les mêmes données, le modèle [DT18] obtient un taux d'erreur final à plus de 2 mètre de 44%.

11.7.2 Stabiliser les modèles de mélanges Gaussiens

Dans [Bis94], l'auteur décrit un modèle de prédiction de mélange Gaussien (Mixture Density Networks : MDN). C'est un réseau neuronal générique dont la sortie est un mélange Gaussien. Le mélange Gaussien est décrit comme une somme convexe de Gaussiennes exprimées en dimension D comme suit :

$$G_{\text{PDF}}(\mu, \Sigma)(z) = \frac{1}{(2\pi)^{D/2} \left|\Sigma\right|^{1/2}} \; e^{-\frac{1}{2}(z-\mu)^\top \Sigma^{-1}(z-\mu)} \tag{11.14}$$

La densité de probabilité est donnée pour $p_m > 0 \; \forall m$ et $\sum_{m=1}^{n_{\text{mix}}} p_m = 1$ par :

$$\text{GM}_{\text{PDF}}(\{\mu_m, \Sigma_m, p_m\}_{m \in [\![1, n_{\text{mix}}]\!]}) = \sum_{m=1}^{n_{\text{mix}}} p_m G_{\text{PDF}}(\mu_m, \Sigma_m) \tag{11.15}$$

Fonction d'activation d'un MDN - On note o_i la $i^{\text{ème}}$ coordonnée de sortie d'un MDN avant la dernière fonction d'activation. Pour contraindre ces paramètres à décrire un mélange Gaussien, la fonction d'activation de sortie est définie par l'équation suivante:

$$\begin{aligned}
&\{(\hat{x}, \hat{y}, \sigma_x, \sigma_y, \rho, p)\}_{m\in[\![1,n_{\text{mix}}]\!]} \\
&= \text{activation}(\{o_1, o_2, o_3, o_4, o_5, o_6\}_{m\in[\![1,n_{\text{mix}}]\!]}) \\
&= \left\{(o_1, o_2, e^{\frac{o_3}{2}}, e^{\frac{o_4}{2}}, \tanh(o_5), \underset{m\in[\![1,n_{\text{mix}}]\!]}{\text{Softmax}}(o_6))\right\}_{m\in[\![1,n_{\text{mix}}]\!]}
\end{aligned}$$

Instabilité - Depuis sa description dans [Bis94], les MDN sont très largement utilisés. Dans certaines applications [HN99; Gra13; Rup+17; CR18], l'apprentissage des MDN est instable. Cependant, nous pouvons stabiliser ce modèle simplement en définissant des seuils de valeurs. Observons la fonction de coût suivante :

$$\begin{aligned}
\text{NLL}_k^{(i)}(dx, dy, \Sigma) = &\frac{1}{2}\underbrace{\frac{1}{(1-\rho^2)}\left(\frac{d_x^2}{\sigma_x^2} + \frac{d_y^2}{\sigma_y^2} - 2\rho\frac{d_x d_y}{\sigma_x\sigma_y}\right)}_{(z_k-\hat{z}_k)^T\Sigma_k^{-1}(z_k-\hat{z}_k)} \\
&+ \underbrace{\ln\left(\sigma_x\sigma_y\sqrt{1-\rho^2}\right)}_{\ln(\sqrt{|\Sigma_k|})} \\
&+ \ln(2\pi)
\end{aligned} \tag{11.16}$$

Cette fonction prend de très grandes valeurs pour des valeurs faibles de σ_x ou σ_y, ou encore si ρ est très proche de 1 ou -1. Ceci est très simplement résolu en considérant une covariance minimale pour les prédictions. Il est possible de visualiser cela comme une zone minimale limitée par l'ellipse de dispersion unitaire de chaque mode Gaussien. Le calcul montre que cela définit des seuils de valeurs maximales et minimales pour σ_x, σ_y, et ρ. Forcer les valeurs de sortie du MDN entre ces seuils a pour effet de stabiliser l'apprentissage et empêche l'expression de prédictions trop confiantes qui ne peuvent pas correspondre à la réalité.

11.8 Un modèle de prédiction complet

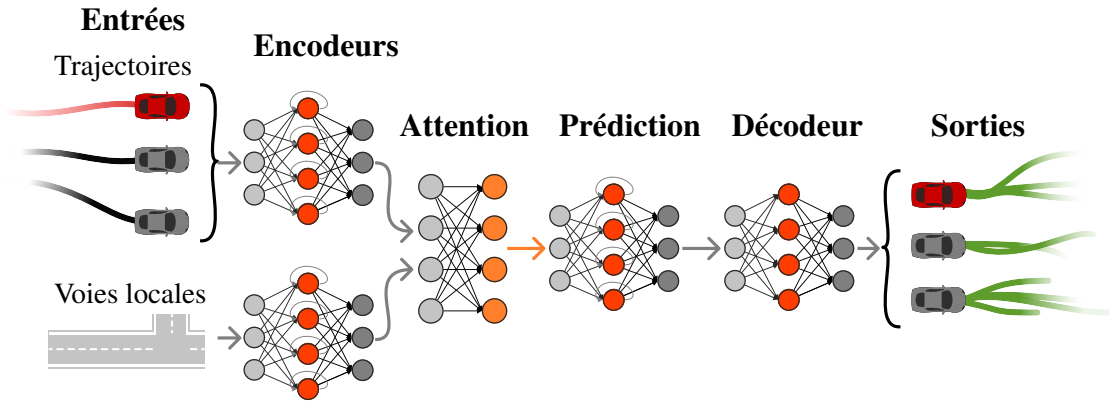


Figure 11.12: Illustration de notre architecture de réseaux neuronaux.

Dans cette section, nous présentons notre proposition la plus aboutie d'architecture de réseau neuronal pour la prédiction de trajectoire. Elle est illustrée par la figure 11.12. On y voit tout d'abord les entrées, comme précédemment, composées des trajectoires des véhicules de la scène et des centres de voies locales. Chaque type d'entrée est encodé par un réseau spécifique composé de couches de convolution et de couches récurrentes de type LSTM. Deux mécanismes d'auto-attention sont employés. L'un entre les centres de voies encodées et les trajectoires encodées, l'autre entre les trajectoires. Suite à cela une couche récurrente LSTM est appliquée de manière à créer une séquence temporelle de la longueur de la prédiction souhaitée. Finalement, les séquences sont décodées en mélanges Gaussiens qui expriment les prédictions. Optionnellement, les séquences peuvent être réencodées, des couches d'attention similaires avec des paramètres différents sont appliquées à ces nouvelles séquences encodées. La même couche LSTM est alors ré-utilisée pour retrouver des séquences temporelles avant le décodage en prédictions.

11.8.1 L'architecture employée

Le graphe 11.13 représente l'architecture employée. Nous utilisons des mini-batches pendant l'apprentissage, la dimension des batches apparait dans les dimensions des tenseurs. Une dimension de 60 a été choisie pour l'espace encodé.

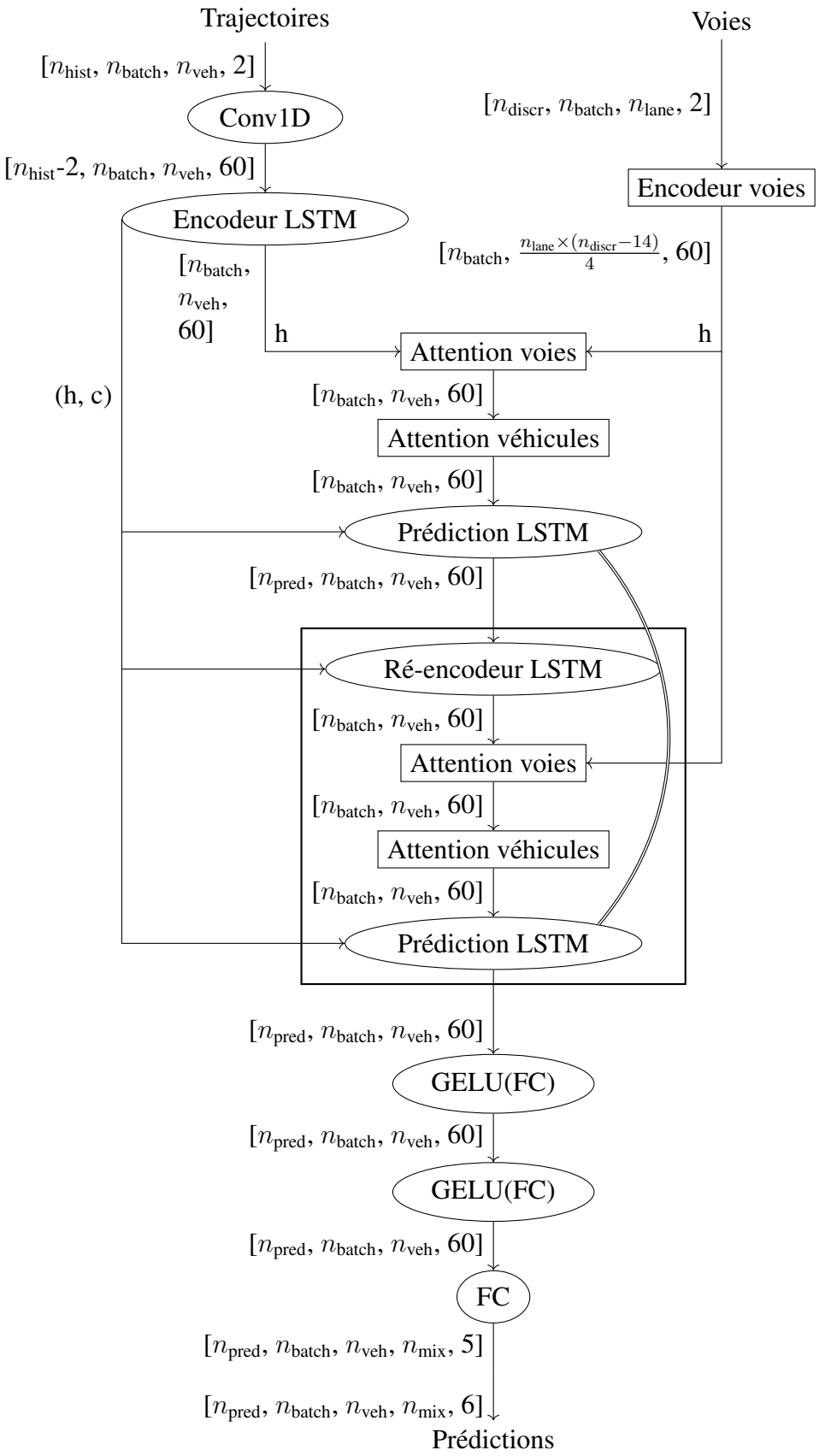


Figure 11.13: Représentation graphique de l'architecture employée. La couche "prédiction LSTM" est utilisée deux fois avec des poids partagés. Les dimensions des tenseurs sont écrites entre crochets. Les couches totalement connectées sont notée FC (Fully Connected).

11.8.1.1 Données d'entrée

Nous souhaitons minimiser les transformations préalable des données si celles-ci peuvent être directement incluses et apprises par notre modèle. Ainsi, nous jugeons qu'il est préférable d'éviter la représentation sous forme d'image ou la création de données dérivées additionnelles. Nos entrées sont simplement les positions successives des véhicules et des voies environnants dans un repère local. La première couche de convolution 1D est capable de calculer les grandeurs cinématiques ou géométriques dérivées des séquences de positions.

11.8.1.2 Encodeur des voies

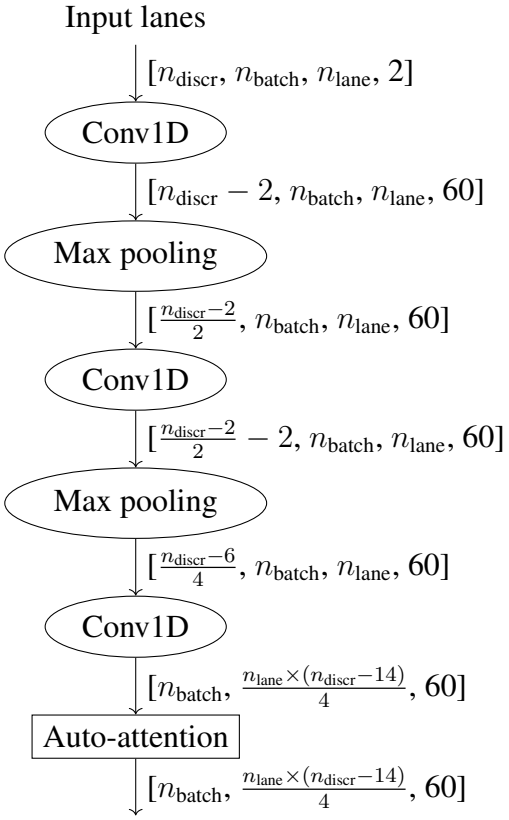


Figure 11.14: Représentation graphique de l'encodeur des voies. Les convolutions utilisent un noyau de taille 3.

L'encodeur de voies est composé de trois couches de convolutions 1D avec des noyaux de taille 3 sans padding. Les deux premières couches sont suivies de deux couches de max-pooling avec une taille de noyau 2. Les morceaux de centre de voies donnés en entrée sont en relation les uns aux autres. Dans la mesure où certaines voies s'intersectent ou sont la continuation l'une de l'autre et d'autres sont parallèles, il semble cohérent de permettre au modèle de représenter ces relations entre voies. Ainsi, une couche d'auto-attention entre les voies encodées est utilisée.

Les encodeurs devraient être en mesure de représenter les données d'entrée dans un espace pour lequel une simple combinaison convexe de certains paramètres représentant les différents objets permet la représentation d'interactions complexes. Ces encodeurs perdent la notion du temps pour les trajectoires (et d'espace pour les voies) en encodant chaque séquence en un seul vecteur. Cela permet une interaction simple entre des vecteurs plutôt qu'une interaction spatio-temporelle.

11.8.1.3 Bloc d'Attention

Le bloc d'attention effectue l'opération d'auto-attention à têtes multiples qui est centrale dans notre modèle. Il est illustré par la figure 11.15. Chaque tête d'attention pondère les éléments relatifs à tous les véhicules. Autrement, dans le cas de l'attention entre véhicules et voies, chaque tête d'attention pondère les éléments relatifs aux voies relativement à chaque véhicule. Les calculs effectués par ces têtes d'attention sont décrits dans la section suivante.

Chaque tête d'attention calcule des mélanges différents des éléments des tenseurs encodés relatifs à chaque entrée. La dimension des tenseurs de sortie de chaque tête est la dimension d'entrée divisée par le nombre de tête. De cette manière, la quantité de calculs est indépendante du nombre de têtes d'attention. Une couche totalement connectée combine la concaténation des tenseurs calculés par les différentes têtes d'attention. Les tenseurs d'entrée encodant les trajectoires de chaque véhicule sont ajoutés aux résultats comme dans les réseaux résiduels [He+16]. Optionnellement, une couche de normalisation [BKH16] peut être ajoutée après cette somme.

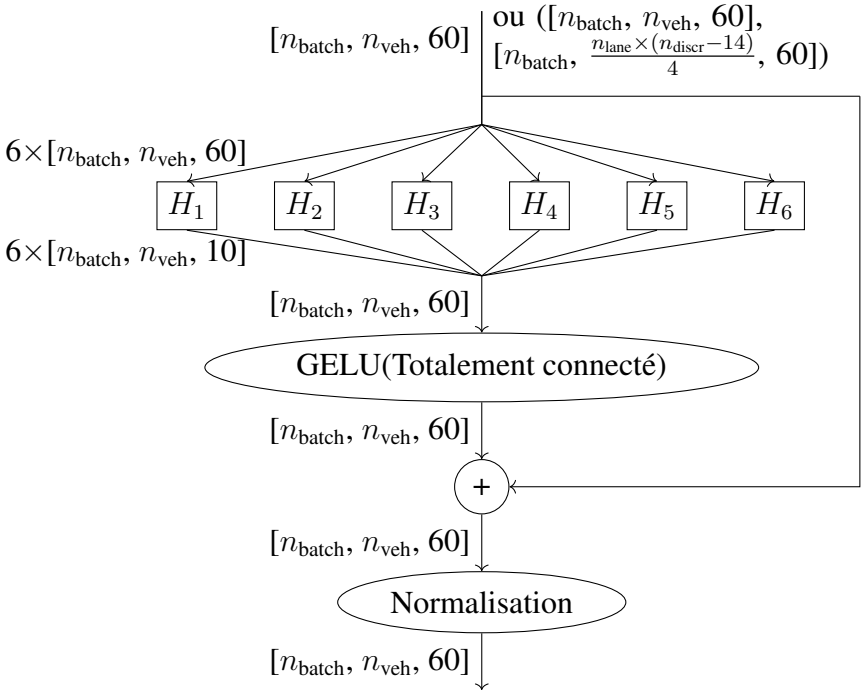


Figure 11.15: Bloc d'attention à têtes multiples. Les blocs H_1 à H_6 sont des têtes d'attention.

11.8.1.4 Têtes d'Attention

Les têtes d'attention sont représentées par les figures 11.16 et 11.17. Elles intègrent un encodage de relation qui est une modification de l'attention de [Vas+17] proposée dans [Sch+19]. Les blocs L sont totalement connectés sans activation et comptent $\frac{60}{n_{\text{heads}}}$ unités. Il y a n_{heads} têtes d'attention, chacune utilisant ses propres paramètres définissant les matrices L. Les multiplications matricielles sont implicites tandis que le symbole $\odot$ décrit la multiplication terme à terme.

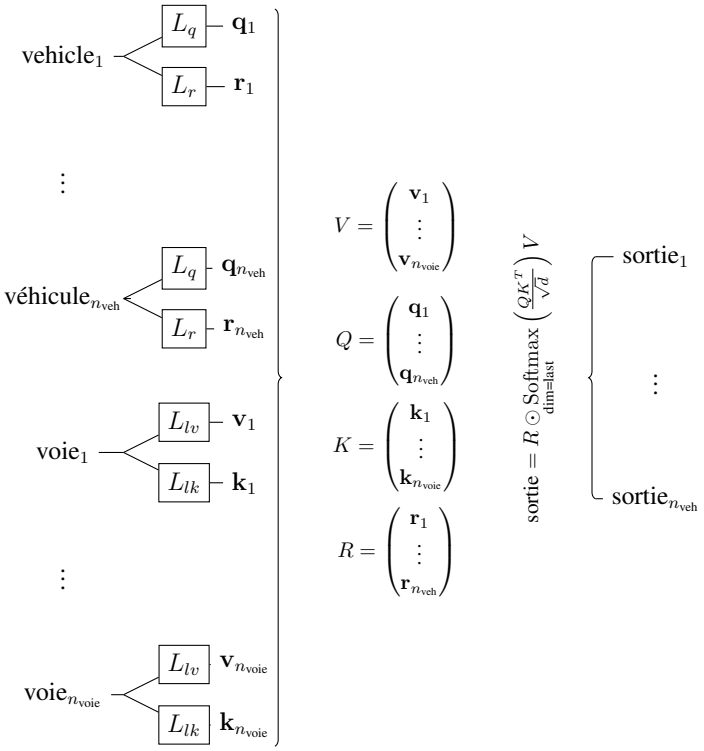


Figure 11.16: Représentation de l'attention de voie pour une tête. Les blocs L_q, L_r, L_{lv}, L_{lk} sont des couches totalement connectées avec biais.

On note d la dimension du tenseur de clés. Dans notre cas avec 6 têtes et une dimension d'entrée de 60, $d = 10$. Le calcul d'auto-attention effectué par chaque tête est donné par :

$$\text{sortie} = R \odot \underbrace{\underset{\text{dim=-1}}{\text{Softmax}}\left(\frac{QK^T}{\sqrt{d}}\right)}_{\text{matrice d'attention}} V \tag{11.17}$$

Nous utilisons deux types d'attention. L'attention aux voies qui met en relation les véhicules et les voies et l'auto-attention qui met en relation les véhicules entre eux. Ces deux types ont la même architecture mais l'attention aux voies calcule le tenseur de valeur v et les clés associées k à partir des tenseurs encodant les voies et le tenseur de requête q et le tenseur de relation r à partir du tenseur encodant les trajectoires des véhicules. L'auto-attention entre véhicules calcule ces quatre tenseurs à partir du même tenseur encodant les trajectoires des véhicules.

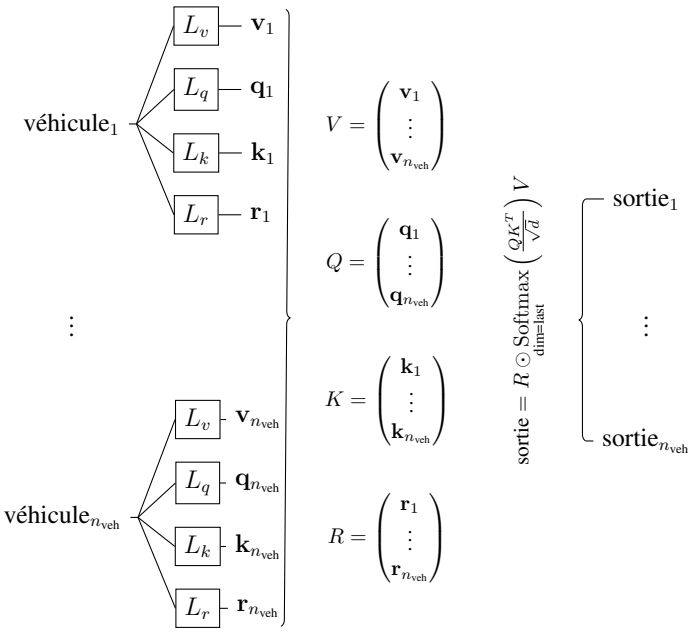


Figure 11.17: Représentation de l'auto-attention d'une tête. Les blocs L_q, L_v, L_k, L_r sont totalement connectés.

Les couches d'attention ont les propriétés mathématiques désirées : elles sont équivariantes et définies pour n'importe quel nombre d'entrées. Cela nous permet de les entrainer avec un nombre dynamique d'objets dans les scènes routières. L'attention permet l'apprentissage de corrélations entre les entrées et donc d'incorporer les interactions. De plus ces interactions peuvent aussi se faire avec le tracé des voies ou d'autres types d'objets ce qui permet de rendre le modèle flexible et adaptable à de nouvelles observations tout en ne dépendant pas totalement de la présence de ces entrées.

Après l'attention, le tenseur est répliqué n_{pred} fois de manière à former une séquence temporelle qui peut être traitée par la couche LSTM notée «Prédiction LSTM» dans la figure 11.13. L'état caché initial de cette couche est l'état caché final (h, c) de la couche «Encodeur LSTM». Cette étape est appelée prédiction parce qu'elle fait apparaitre la dimension temporelle dans le futur mais elle laisse les tenseurs dans un état encodé. C'est-à-dire qu'à chaque pas de temps, ce tenseur contient toute l'information utilisée par le modèle pour faire une prédiction à ce pas de temps mais qu'il n'est pas directement interprétable. Nous utilisons un tenseur répliqué plutôt que d'utiliser la récurrence dans une boucle pour bénéficier d'une optimisation de performance.

11.8.1.5 Bloc de Ré-Encodage

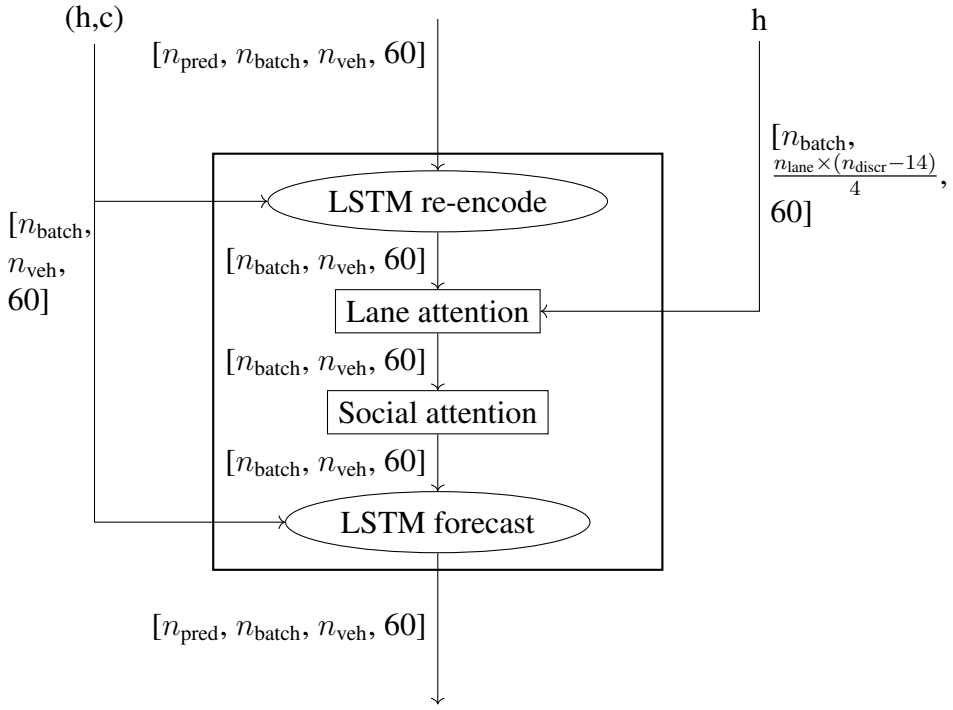


Figure 11.18: Illustration de bloc de ré-encodage.

Le bloc de ré-encodage représenté sur la figure 11.18 est similaire au premier encodage : la séquence d'entrée est encodée avec un LSTM, une couche d'attention aux voies et une d'attention de véhicule à véhicule sont utilisées avant de réutiliser la même couche «prédiction LSTM» pour former de nouveau une séquence temporelle.

L'interaction calculée par le précédent bloc d'attention est faite pour l'encodage des séquences passées. Nous voulons pousser le modèle à considérer des interactions dans le futur. Pour assurer que la notion de futur est la même avant et après le réencodage, la même couche «prédiction LSTM» est employée. Ainsi, le bloc réencodage/attention/prédiction peut être exécuté plusieurs fois en boucle. Nous expérimentons avec une et deux itérations.

11.8.1.6 Décodage

Le décodage est fait avec 3 couches totalement connectées. Les deux premières utilisent des fonctions d'activation GELU [HG16] tandis que la dernière couche utilise l'activation spécifique pour l'expression d'un mélange Gaussien.

11.8.1.7 Fonction de coût

Le modèle est entrainé par rétropropagation du gradient calculé pour la fonction de coût NLL. Deux coûts subsidiaires sont ajoutés : un coût d'erreur et une pénalisation des poids.

Le coût NLL - Nous calculons la moyenne temporelle sur l'horizon de prédiction de la NLL des mélanges Gaussien définis précédemment pour chaque instant de prédiction.

Coût d'erreur à 2 mètres - Nous utilisons un coût L_1 saturé pour la trajectoires dont la position finale est la plus proche de la position finale observée. C'est à dire $||\tilde{\mathbf{y}} - \hat{\mathbf{y}}_{m^*}||_2$, avec $\mathbf{y}$ désignant une trajectoire complète, n'affectant la prédiction que si l'erreur commise est entre 1 et 3 mètres. La figure 11.19 représente le graphe de cette fonction. Ce coût évite que les prédictions soient proches mais supérieures à 2 mètres de la position finale observée ce qui devrait diminuer le taux d'erreur à plus de 2 mètres. De plus, il évite dans une certaine mesure les prédictions moyennes irréalistes entre deux modes possibles.

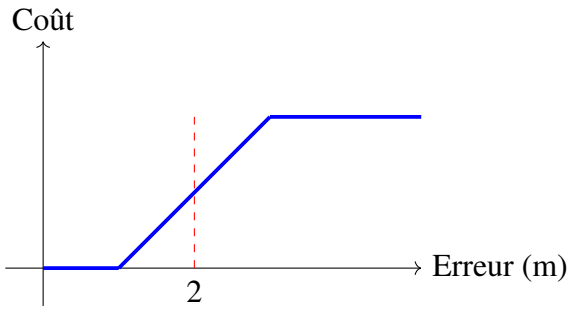


Figure 11.19: Graphe du coût d'erreur à 2 mètres.

Pénalisation des poids - Nous ajoutons un coût L_2 sur les poids du réseaux pondéré arbitrairement par 10^{-5} pour que son effet reste faible.
La fonction de coût est donc donnée par :

$$\begin{aligned}
&\text{Coût}(\tilde{\mathbf{y}}, \{\hat{\mathbf{y}}, \Sigma, p\}_{m\in[\![1,n_{\text{mix}}]\!]}, \Theta) = \\
&\quad - \ln\left(\sum_{m=1}^{n_{mix}} \exp\left(-\text{NLL}(\tilde{\mathbf{y}} - \hat{\mathbf{y}}_m, \Sigma_m)\right) p_m\right) \\
&\quad + \text{coût_erreur}(\tilde{\mathbf{y}} - \hat{\mathbf{y}_{m^*}}) \\
&\quad + 10^{-5}||\Theta||^2
\end{aligned} \tag{11.18}$$

11.8.1.8 Hyperparamètres

Nous avons construit notre modèle avec de nombreux hyperparamètres. Les dimensions des tenseurs cachés, le nombre de têtes d'attention, le nombre de couches, les dimensions de noyaux de convolution, le type d'optimisation, le taux d'apprentissage, la taille de batch, la normalisation ou non des données d'entrée. Pour beaucoup de paramètres, différentes valeurs donnent des résultats très similaires et une valeur arbitraire est fixée. Pour les paramètres jugés les plus importants, nous faisons une étude ablative.

11.8.2 Base de données

La base de données NGSIM que nous avons utilisée jusqu'ici nous a permis des comparaisons avec de nombreux autres travaux. Cependant, les situations sur autoroute qu'elle contient ne présentent pas de cas très interactifs et ne contient qu'un type de réseau routier sans intersection. Nous utilisons donc la base de données Argoverse qui contient des situations en ville principalement à des intersections. De plus c'était au moment de nos travaux la plus grande base de données publique pour la prédiction de trajectoire. La documentation concernant cette base est disponible sur GitHub[2], et dans le papier de référence [Cha+19b]. Elle contient 323557 scènes de 5 secondes. Les deux premières secondes sont utilisées pour prédire les trois suivantes. Deux compétitions de prédiction de trajectoire ont été organisées avec cette base de donnée. L'évaluation est faite avec 78143 scènes pour lesquelles seules les deux premières secondes sont diffusées. Les organisateurs comparent les prédictions émises par les participants aux trois secondes observées suivantes qu'ils ont tenu secrètes et ne diffusent que les résultats moyens en ligne[3].

11.8.3 Base de comparaison

Nous établissons les résultats du modèle de prédiction à vitesse constante décrit dans la première section avec cette nouvelle base de données dans le tableau 11.6.

Table 11.6: Évaluation du modèle de prédiction unimodal à vitesse constante sur la base de données Argoverse.

Time horizon	1s	2s	3s
RMSE (m)	1.90	4.41	7.72
FDE (m)	1.38	3.20	5.65
NLL	3.58	5.12	6.20
MR	0.23	0.55	0.72

De même, nous établissons les résultats du modèle multi-modal à vitesse constante dans le tableau 11.7. La distribution de variation de vitesse initiale calculée sur cette base de données est représentée sur la figure 11.20

[2] https://github.com/argoai/argoverse-api

[3] https://evalai.cloudcv.org/web/challenges/challenge-page/454/leaderboard/1279

Table 11.7: $\sigma_\theta = 0°$, $\sigma_{\alpha_v} = 20\%$

Time horizon	1s	2s	3s
minRMSE (m)	1.30	3.10	5.48
minFDE (m)	0.83	1.92	3.35
NLL	3.85	6.25	7.98
MR	0.10	0.24	0.44
Sim	0.42	0.06	0.01

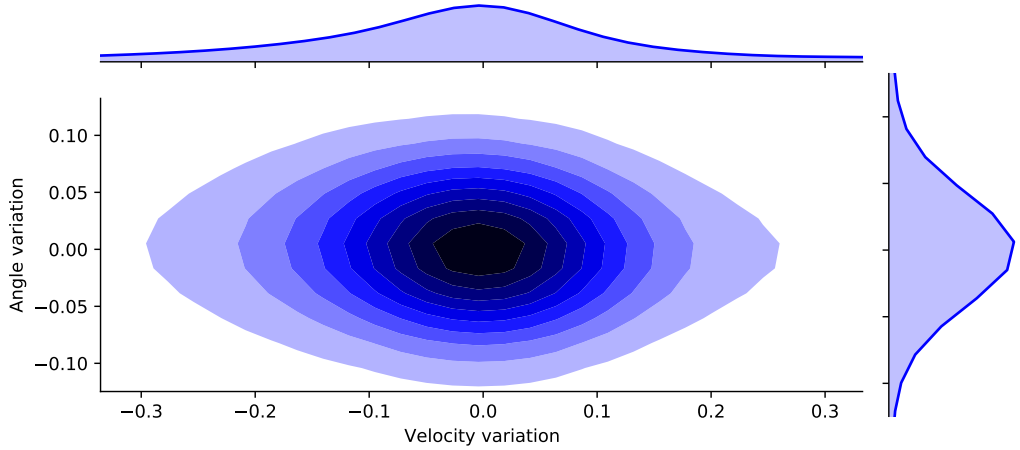


Figure 11.20: Distribution de variation de module de vitesse et d'angle initial (en radians) pour la base de données Argoverse.

Pour dépasser les performances de cette base de comparaison, nous souhaitons produire des modèles qui produisent moins de 44% d'erreur de plus de 2 mètres à 3 secondes pour 6 propositions de trajectoires. Et une distance finale d'erreur minimale inférieure à 3,35 mètres.

11.8.4 Évaluation comparée du modèle

Dans un premier temps, nous évaluons et comparons notre modèle avec la base de données NGSIM. Le tableau 11.8 présente les résultats obtenus. Tous les résultats sauf GRIP [LYC19] sont entrainés et calculés avec le même jeu de données et les mêmes fonctions d'évaluation. Nous ne comparons les résultats que pour la prédiction du véhicule ego dans chaque scène. Les modèles comparés sont :

- le modèle unimodal à vitesse constante
- le modèle multi-modal à vitesse constante
- le modèle CSP(M) de [DT18] dont nous reproduisons les résultats
- le modèle GRIP [LYC19] en recopiant leurs résultats publiés
- notre modèle décrit précédemment (à partir de 11.8)

Table 11.8: Comparaison des résultats NLL, RMSE, FDE et MR avec différentes méthodes. Les résultats *CSP(M) sont reproduits avec des corrections mineures.

Time horizon		1s	2s	3s	4s	5s
NLL	CV	0.82	2.32	3.23	3.91	4.46
	MM CV	2.45	3.35	4.12	4.81	5.39
	CSP(M) [DT18]*	-0.41	1.07	1.93	2.55	3.08
	SAMMP	-0.36	0.70	1.51	2.13	2.64
RMSE (m)	CV	0.76	1.82	3.17	4.80	6.70
	MM CV	0.88	2.04	3.49	5.21	7.15
	CSP(M) [DT18]*	0.59	1.27	2.13	3.22	4.64
	GRIP [LYC19]	0.37	0.86	1.45	2.21	3.16
	SAMMP	0.51	1.13	1.88	2.81	3.98
FDE (m)	CV	0.46	1.24	2.27	3.53	4.99
	MM CV	0.52	1.32	2.39	3.67	5.14
	CSP(M) [DT18]*	0.39	0.91	1.55	2.36	3.39
	SAMMP	0.31	0.78	1.35	2.04	2.90
MR	CV	0.02	0.20	0.44	0.61	0.71
	MM CV	0.01	0.03	0.10	0.20	0.30
	CSP(M) [DT18]*	0.004	0.03	0.12	0.28	0.44
	SAMMP	0.002	0.02	0.08	0.15	0.23

De même, nous établissons une comparaison de résultats avec les données de la base Argoverse. Pour cela, nous reportons les résultats du dernier concours de prédiction de trajectoire dans le tableau 11.9.

Table 11.9: Résultats à 3 secondes sur la base de test d'Argoverse. Ces résultats ont été recopiés le 21 Août 2020 depuis EvalAI.

	ADE	minADE	FDE	minFDE	MR
Notre modèle	**1.68**	0.97	**3.73**	1.42	**0.13**
Waymo Poly [Cha+19a]	1.71	0.89	3.85	1.50	**0.13**
Waymo TNT [Zha+20]	1.78	0.94	3.91	1.54	**0.13**
Alibaba	1.97	0.92	4.35	1.48	0.16
Uber ATG-LaneGCN [Lia+20]	1.71	**0.87**	3.78	**1.36**	0.16
Argo CMU Wimp [Kha+20]	1.82	0.90	4.03	1.42	0.17

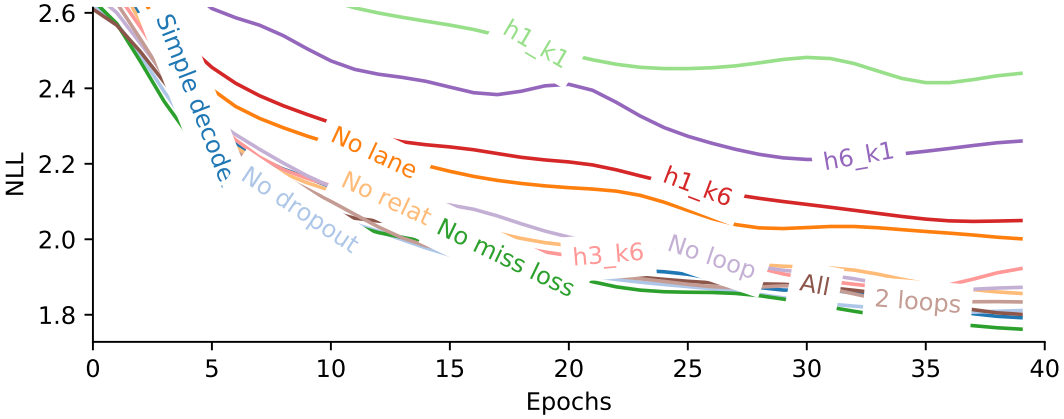
Tous ces résultats sont très similaires et sont obtenus par des méthodes proches. En particulier l'auto-attention est très utilisée. Les principales différences sont :

- L'utilisation de trajectoires de références dans TNT [Zha+20].
- La représentation des voies sous forme de graphe dans TNT [Zha+20] et Uber [Lia+20].
- Une représentation sous forme d'image de la scène dans Poly [Cha+19a] et Uber [Lia+20].
- Des hypothèses explicites de choix de voies qui conditionnent la prédiction dans Wimp [Kha+20].

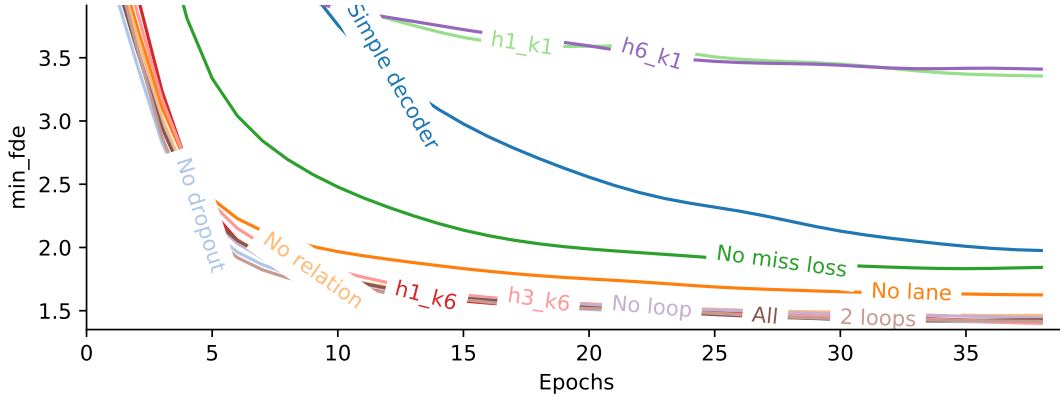
11.8.4.1 Étude ablative

Un entrainement complet de notre modèle demande environ 300 «epochs» (c'est à dire que chaque séquence est utilisée 300 fois dans une étape de descente du gradient) ce qui est effectué en 20 heures avec une carte graphique Nvidia Tesla v100. Les ressources de calcul nous ont été offertes par l'IDRIS sous l'allocation 2019-39282 de GENCI. Pour l'étude ablative, nous entrainons nos modèles pour 40 epochs seulement pour accélérer l'étude. Les différentes configurations étudiées sont listées ci-dessous :

- Simple decoder: Une seule couche totalement connectée est utilisée après «prédiction LSTM».
- No dropout: Un dropout de 20% des séquences d'entrée et des voies n'est pas appliqué.
- No lane: Les centres de voies ne sont pas fournis au modèle.
- No relation: L'encodage de relation [Sch+19] utilisé dans l'attention n'est pas utilisé.
- No miss loss: Le coût d'erreur n'est pas ajouté au coût à minimiser.
- h1_k1: Une seule tête d'attention est utilisée et une seule prédiction est émise.
- h6_k1: Six têtes d'attentions sont utilisées et une seule prédiction est émise.
- h1_k6: Une seule tête d'attention est utilisée et six prédictions sont émises.
- h3_k6: Trois têtes d'attention sont utilisées et six prédictions sont émises.
- No loop: La boucle de réencodage n'est pas utilisée.
- 2 loops: La boucle de réencodage est utilisée deux fois de suite.
- All: Tous les paramètres par défaut (6 têtes d'attention, 6 prédictions, 3 couches de décodeur, 20% de dropout, utilisations des voies, relation, coût d'erreur, une boucle de réencodage).



(a) NLL



(b) minFDE

Figure 11.21: Courbes de convergence de validation de NLL pour les modèles ablatés.

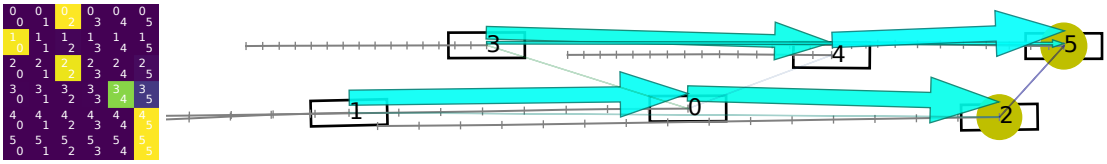
Les courbes de la figure 11.21a caractérisent la qualité des distributions prédites. Elles sont d'autant plus basses qu'elles représentent de grandes vraisemblances de données de validation pour les distributions prédites. Les courbes de la figure 11.21b caractérisent la distance finale moyenne sur l'ensemble de validation entre la meilleure des trajectoires proposées et l'unique trajectoire réelle observée. Sur ces deux critères, on observe un faisceau de résultats similaires montrant relativement peu d'influence de certains paramètres. Cependant les résultats obtenus pour certains choix de paramètres sont clairement moins bons. En particulier, les prédictions unimodales («k1») donnent de moins bons résultats en terme de distance mais aussi de vraisemblance. L'addition de têtes d'attention (passer de «h1» à «h3» ou «h6») améliore la vraisemblance mais il faut aussi ajouter des modes pour obtenir de meilleurs résultats. Il est intéressant de remarquer que la simplification du décodeur en passant de 3 couches à une dégrade la minFDE mais pas la vraisemblance.

Retirer les voies des données d'entrée dégrade aussi les résultats, tant en termes de distance que de vraisemblance. Enfin, le dernier critère significatif est le coût d'erreur (miss loss). L'ajout de ce coût dégrade légèrement la NLL mais améliore nettement la minFDE.

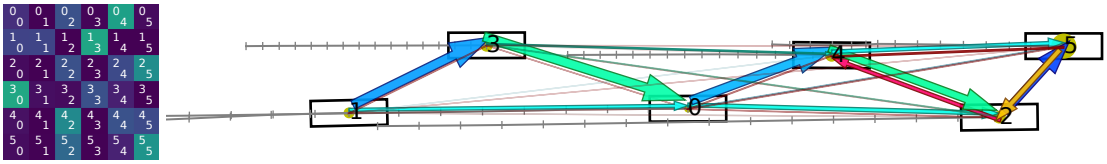
11.8.5 Interprétation des résultats

11.8.5.1 Graphes d'Attention

Données autoroute NGSIM - La figure 11.22 représente la matrice d'attention et le graphe équivalent pour la première couche d'auto-attention. Les flèches entre les véhicules représentent l'attention et sont d'autant plus épaisses que la valeur est grande. La couleur des flèches est liée à leur orientation. Un cercle représente l'attention d'un véhicule à lui-même. Les coefficients d'attention sont également représentés par les couleurs dans la matrice à gauche allant du violet foncé pour une attention très faible au jaune pour une attention élevée. Sans être supervisé pour cette tâche, le modèle apprend à reconnaître certains types de relations entre les véhicules de la scène. C'est le cas de l'attention au véhicule de devant représenté dans la figure 11.22a, ou de l'attention au véhicule le plus proche de la figure 11.22b. D'autres têtes d'attentions forment des graphes qui ne sont pas interprétables si facilement.



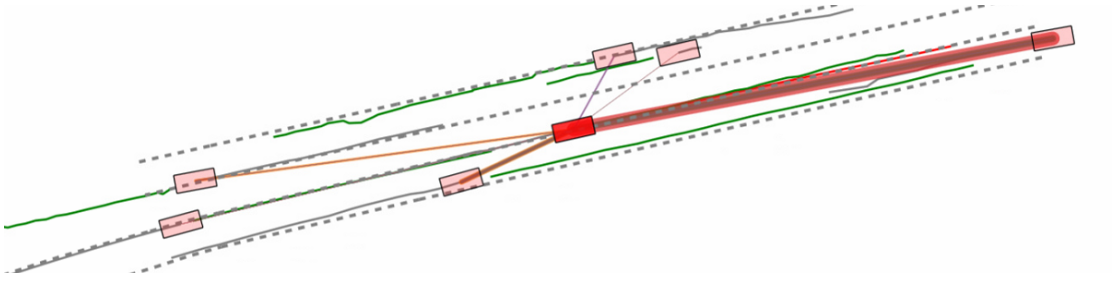
(a) Tête spécialisée dans l'attention du véhicule de devant.



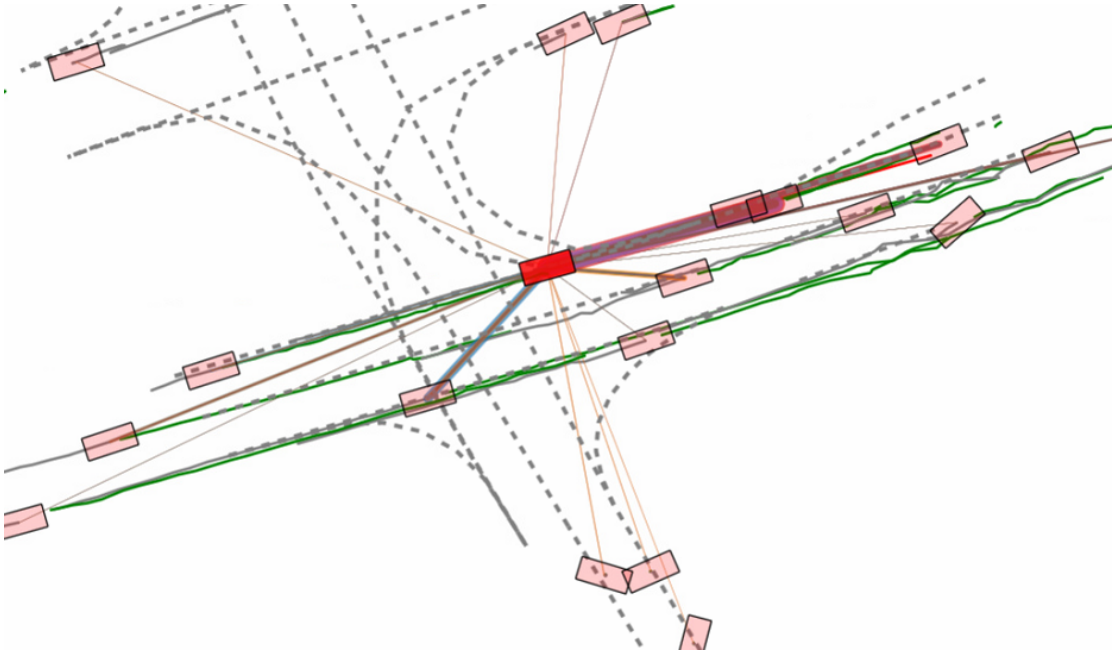
(b) Tête spécialisée dans l'attention du véhicule le plus proche.

Figure 11.22: Une scène de la base NGSIM avec tous les véhicules observés, leurs positions passées en gris. Les matrices d'attention de deux têtes ainsi que les graphes orientés correspondant sont représentés.

Données urbaines Argoverse - Avec les données Argoverse, le réseau routier est bien plus complexe, comme on le voit sur la figure 11.23b. Les types d'interaction sont plus complexes mais on distingue tout de même des régularités sur les représentations 11.23. Le véhicule de devant reçoit le principal de l'attention et on distingue une sensibilité à la distance.



(a) Une scène simple.

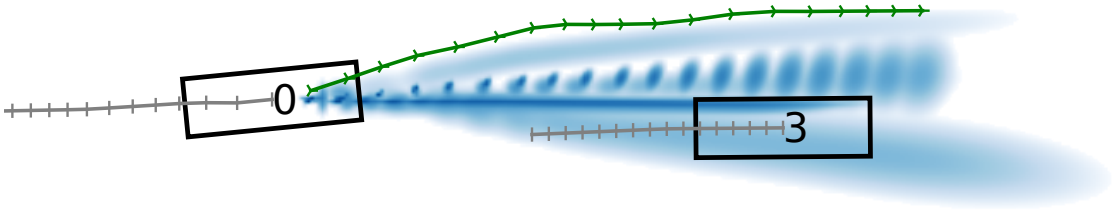


(b) Une scène complexe.

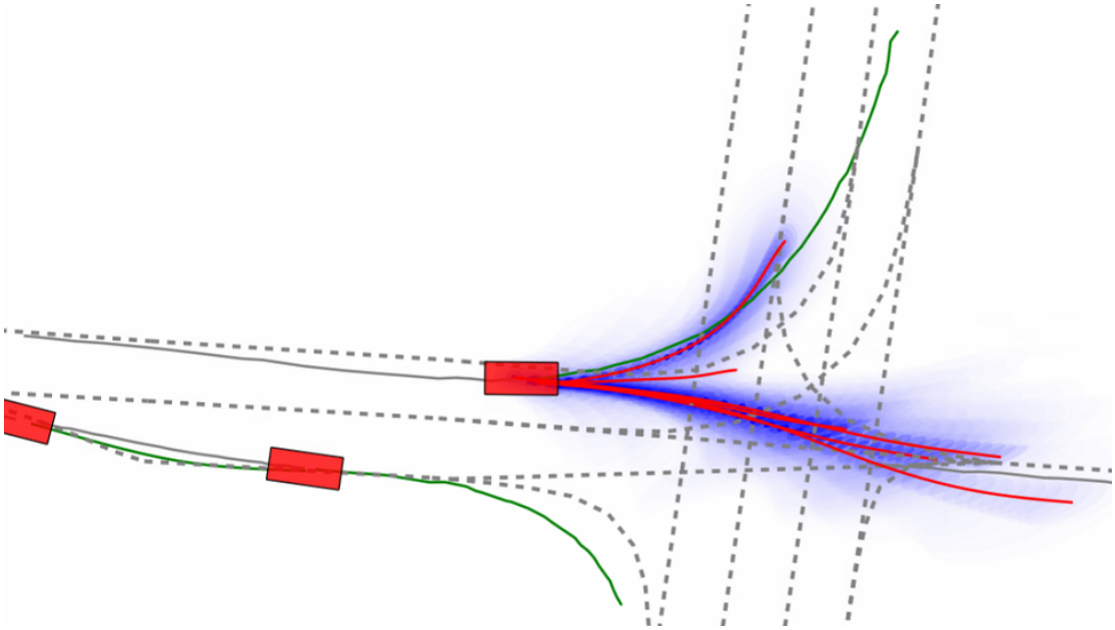
Figure 11.23: Deux scènes tirées de la base Argoverse. Les voies sont représentées en pointillés gris. Les trajectoires passées sont en gris, le futur réel en vert et la prédiction du véhicule ego est en rouge. L'attention que le véhicule ego porte aux véhicules alentours est représentée par des arêtes, colorées selon la tête d'attention, et d'épaisseur représentant le taux d'attention.

11.8.6 Multimodalité

Comme vu dans la distribution de variation de vitesse initial du modèle multimodal à vitesse constante, les principaux modes sont déterminés par des variations du module de la vitesse du véhicule et non de direction. Pour faciliter la visualisation des modes, la figure 11.24 représente des cas pour lesquels les modes sont latéraux, c'est à dire avec une grande variation d'orientation. On observe une diversité de prédictions émises par notre modèle.



(a) NGSIM



(b) Argoverse

Figure 11.24: Scènes routières avec les prédictions représentées par des ellipses transparentes bleues. Le futur observé réel est représenté en vert.

11.9 Conclusion

Dans cette thèse, nous avons étudié la prédiction de trajectoire de véhicules depuis les simples modèles cinématiques jusqu'aux réseaux neuronaux. Nous avons établi des résultats à vitesse constante unimodaux et multimodaux. Des critères d'évaluation précisément établis nous permettent la comparaison de différentes approches et l'analyse de différents aspects des modèles prédictifs. Suite à notre analyse de l'état de l'art, nous avons opté pour l'étude de modèles de prédiction avec des réseaux neuronaux Nous avons adapté la représentation de la scène routière aux besoins des réseaux neuronaux mais les résultats obtenus n'étaient pas convainquants. Nous avons donc utilisé une architecture particulière qui s'adapte aux données représentant la scène routière. Cette approche donne de bien meilleurs résultats. Une dernière amélioration nécessaire est la prédiction multimodale. Nous avons obtenu des résultats multimodaux simplement en entrainant un modèle à prédire les paramètres d'un mélange Gaussien et en appliquant un coût NLL saturé et coût d'erreur. Le modèle obtenu a donné les meilleurs résultats deux années consécutives (2019 et 2020) à la compétition de prédiction de trajectoires Argoverse et permet dans une certaine mesure une interprétation des interactions entre véhicules. Depuis, grâce à une communauté de recherche très active, de nombreuses améliorations des modèles de prédiction de trajectoire ont vu le jour et permettent des résultats encore meilleurs.

Bibliography

[ADY13] Ossama Abdel-Hamid, Li Deng, and Dong Yu. "Exploring convolutional neural network structures and optimization techniques for speech recognition." In: *Interspeech*. Vol. 11. 2013, pp. 73–5.

[Agu+17] Eirikur Agustsson et al. "Soft-to-hard vector quantization for end-to-end learning compressible representations". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1141–1151.

[AJ08] Dan Ariely and Simon Jones. *Predictably irrational*. Harper Audio New York, NY, 2008.

[AK07] Daniel Aarno and Danica Kragic. "Evaluation of Layered HMM for Motion Intention Recognition". In: *IEEE International Conference on Advanced Robotics* (2007).

[AL17] Florent Altché and Arnaud de La Fortelle. "An LSTM network for highway trajectory prediction". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 353–359.

[Ala+16] Alexandre Alahi et al. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971.

[AMP10] Stefan Atev, Grant Miller, and Nikolaos P Papanikolopoulos. "Clustering of vehicle trajectories". In: *IEEE transactions on intelligent transportation systems* 11.3 (2010), pp. 647–657.

[Bec+18] Stefan Becker et al. "Red: A simple but effective baseline predictor for the trajnet benchmark". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[Bha+18] Raunak P Bhattacharyya et al. "Multi-Agent Imitation Learning for Driving Simulation". In: *arXiv* (2018). arXiv: `arXiv:1803.01044v1`.

[Bha+19] Apratim Bhattacharyya et al. "Conditional flow variational autoencoders for structured sequence prediction". In: *arXiv preprint arXiv:1908.09008* (2019).

[BIS09] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*. Vol. 56. springer, 2009.

[Bis94] Christopher M Bishop. "Mixture density networks". In: (1994).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. “Estimating or propagating gradients through stochastic neurons for conditional computation”. In: *arXiv preprint arXiv:1308.3432* (2013).

[Bru+13] Joan Bruna et al. “Spectral networks and locally connected networks on graphs”. In: *arXiv preprint arXiv:1312.6203* (2013).

[CAN20a] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. “Learning Neural Event Functions for Ordinary Differential Equations”. In: *arXiv preprint arXiv:2011.03902* (2020).

[CAN20b] Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. “Neural Spatio-Temporal Point Processes”. In: *arXiv preprint arXiv:2011.04583* (2020).

[CD19] Ricky TQ Chen and David K Duvenaud. “Neural networks with cheap differential operators”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 9961–9971.

[CH06] Patrick Pakyan Choi and Martial Hebert. “Learning and Predicting Moving Object Trajectory: a piecewise trajectory segment approach”. In: (2006). URL: `http://repository.cmu.edu/cgi/viewcontent.cgi?article=1336%7B%5C&%7Dcontext=robotics`.

[CH07] James Colyar and John Halkias. *Next Generation Simulation dataset*. 2007.

[Cha+19a] Yuning Chai et al. “Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction”. In: *arXiv preprint arXiv:1910.05449* (2019).

[Cha+19b] Ming-Fang Chang et al. “Argoverse: 3d tracking and forecasting with rich maps”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757.

[Che+18] Ricky T. Q. Chen et al. “Neural ordinary differential equations”. In: *arXiv* (2018).

[CR18] Joseph Curro and John Raquet. “Deriving confidence from artificial neural networks for navigation”. In: *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. IEEE. 2018, pp. 1351–1361.

[Cui+19] Henggang Cui et al. “Multimodal trajectory predictions for autonomous driving using deep convolutional networks”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2090–2096.

[CW03] Miguel Á Carreira-Perpiñán and Christopher KI Williams. “On the number of modes of a Gaussian mixture”. In: *International Conference on Scale-Space Theories in Computer Vision*. Springer. 2003, pp. 625–640.

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems*. 2016, pp. 3844–3852.

[Din+18] Guohui Ding et al. “Game-theoretic cooperative lane changing using data-driven models”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3640–3647.

[Dju+18] Nemanja Djuric et al. “Motion prediction of traffic actors for autonomous driving using deep convolutional networks”. In: *arXiv preprint arXiv:1808.05819* 2 (2018).

[DT18] Nachiket Deo and Mohan M Trivedi. “Convolutional Social Pooling for Vehicle Trajectory Prediction”. In: (2018). arXiv: `arXiv:1805.06771v1`.

[FHL19] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective”. In: *arXiv preprint arXiv:1912.02757* (2019).

[Fis+19] Jaime F Fisac et al. “Hierarchical game-theoretic planning for autonomous vehicles”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9590–9596.

[Gao+20] Jiyang Gao 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. 11525–11533.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[Gér19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O’Reilly Media, 2019.

[Gon+17] David Sierra González et al. “Interaction-aware driver maneuver inference in highways using realistic driver models”. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 1–8.

[Gra13] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013).

[Gua+20] Jiaqi Guan et al. “Generative hybrid representations for activity forecasting with no-regret learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 173–182.

[Gup+18] Agrim Gupta et al. “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.

[He+16] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[HG16] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[HM95] Dirk Helbing and Peter Molnar. “Social force model for pedestrian dynamics”. In: *Physical review E* 51.5 (1995), p. 4282.

[HN99] Lars U Hjorth and Ian T Nabney. “Regularisation of mixture density networks”. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99.(Conf. Publ. No. 470)*. Vol. 2. IET. 1999, pp. 521–526.

[Hou14] Adam Houenou. “Calcul de trajectoires pour la préconisation de manoeuvres automobiles sur la base d’une perception multi-capteur : application à l’évitement de collision”. PhD thesis. 2014. URL: https://tel.archives-ouvertes.fr/tel-00977389.

[HR18] Richard Harang and Ethan M Rudd. “Towards principled uncertainty estimation for deep neural networks”. In: *arXiv preprint arXiv:1810.12278* (2018).

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[HS86] G. E. Hinton and T. J. Sejnowski. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. Ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning and Relearning in Boltzmann Machines, pp. 282–317. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104291.

[HS97] Sepp Hochreiter and Jürgen Schmidhubre. “Long short-term memory”. In: 9.8 (1997), pp. 1–32.

[HZT18] Yeping Hu, Wei Zhan, and Masayoshi Tomizuka. “Probabilistic prediction of vehicle semantic intention and motion”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 307–313.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[Ju+19] Ce Ju et al. “Interaction-aware kalman neural networks for trajectory prediction”. In: *arXiv preprint arXiv:1902.10928* (2019).

[Kah11] Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

[Kal60] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960).

[Kha+20] Siddhesh Khandelwal et al. “What-If Motion Prediction for Autonomous Driving”. In: *arXiv preprint arXiv:2008.10587* (2020).

[KL51] S. Kullback and R. A. Leibler. “On information and sufficiency”. In: (1951).

[Kra+18] Robert Krajewski et al. “The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems”. In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2018-Novem (2018), pp. 2118–2125. DOI: 10.1109/ITSC.2018.8569552.

[KS20] A. Kawasaki and A. Seki. “Multimodal Trajectory Predictions for Urban Environments Using Geometric Relationships between a Vehicle and Lanes”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[KTH07] Arne Kesting, Martin Treiber, and Dirk Helbing. “General Lane-Changing Model MOBIL for Car-Following Models”. In: *Transportation Research Record* 1999 (Jan. 2007), pp. 86–94. DOI: 10.3141/1999-10.

[KW14] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: (2014).

[Lia+20] Ming Liang et al. “Learning lane graph representations for motion forecasting”. In: *arXiv preprint arXiv:2007.13732* (2020).

[Llo82] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[LVL14] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. “A survey on motion prediction and risk assessment for intelligent vehicles”. In: *ROBOMECH Journal* (2014). ISSN: 2197-4225. DOI: 10.1186/s40648-014-0001-z. URL: http://www.robomechjournal.com/content/1/1/1.

[LYC19] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. “Grip: Graph-based interaction-aware trajectory prediction”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 3960–3966.

[LYC20] Xin Li, Xiaowen Ying, and Mooi Choo Chuah. “GRIP++: Enhanced Graph-based Interaction-aware Trajectory Prediction for Autonomous Driving”. In: (2020).

[Mak+19] Osama Makansi et al. “Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7144–7153.

[Mer+19] Jean Mercat et al. “Kinematic Single Vehicle Trajectory Prediction Baselines and Applications with the NGSIM Dataset”. In: *arXiv* (2019), arXiv–1908.

[Mer+20] Jean Mercat et al. “Multi-Head Attention for Joint Multi-Modal Vehicle Motion Forecasting”. In: *IEEE International Conference on Robotics and Automation*. 2020.

[Mes+19] Kaouther Messaoud et al. “Non-local Social Pooling for Vehicle Trajectory Prediction To cite this version : HAL Id : hal-02160409 Non-local Social Pooling for Vehicle Trajectory Prediction”. In: (2019).

[Met+16] Luke Metz et al. “Unrolled generative adversarial networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[Min09] Barbara Minto. *The pyramid principle: logic in writing and thinking*. Pearson Education, 2009.

[OP00] Nuria Oliver and Alex P. Pentland. “Graphical Models for Driver Behavior Recognition in a SmartCar”. In: *IEEE Intelligent Vehicles Symposium* (2000).

[Ort+11] Michaël Garcia Ortiz et al. “Behavior prediction at multiple time-scales in inner-city scenarios”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2011, pp. 1068–1073.

[PB09] Vincenzo Punzo and Maria Teresa Borzacchiello. “Estimation of vehicle trajectories from observed discrete positions and Next - Generation Simulation Program (NGSIM) data”. In: (2009).

[Qi+17] Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[Rag+17] Maithra Raghu et al. “On the expressive power of deep neural networks”. In: *international conference on machine learning*. 2017, pp. 2847–2854.

[Ram+20] Hubert Ramsauer et al. “Hopfield networks is all you need”. In: *arXiv preprint arXiv:2008.02217* (2020).

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. ISBN: 026268053X.

[Rup+17] Christian Rupprecht et al. “Learning in an uncertain world: Representing ambiguity through multiple hypotheses”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3591–3600.

[Sad+18] Amir Sadeghian et al. “SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints”. In: (2018). arXiv: arXiv:1806.01482v2.

[Sal+20] Tim Salzmann et al. “Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control”. In: *arXiv preprint arXiv:2001.03093* (2020).

[Sch+19] Imanol Schlag et al. “Enhancing the transformer with explicit relational encoding for math problem solving”. In: *arXiv preprint arXiv:1910.06611* (2019).

[Sch+20] Christoph Schöller et al. “What the constant velocity model can teach us about pedestrian motion prediction”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1696–1703.

[SLY15] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. “Learning structured output representation using deep conditional generative models”. In: *Advances in neural information processing systems*. 2015, pp. 3483–3491.

[Sri+17] Akash Srivastava et al. “VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 3308–3318. URL: http://papers.nips.cc/paper/6923-veegan-reducing-mode-collapse-in-gans-using-implicit-variational-learning.pdf.

[SSS17] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “On a formal model of safe and scalable self-driving cars”. In: *arXiv preprint arXiv:1708.06374* (2017).

[SWA14] Matthias Schreier, Volker Willert, and Jürgen Adamy. “Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems”. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2014, pp. 334–341.

[THH00] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested Traffic States in Empirical Observations and Microscopic Simulations". In: *Physical Review E* 62 (Feb. 2000), pp. 1805–1824. DOI: 10.1103/PhysRevE.62.1805.

[TPB00] Naftali Tishby, Fernando C Pereira, and William Bialek. "The information bottleneck method". In: *arXiv preprint physics/0004057* (2000).

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[Vig19] Jesse Vig. "Visualizing attention in transformer-based language representation models". In: *arXiv preprint arXiv:1904.02679* (2019).

[VV+17] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6306–6315.

[Wan+18] Xiao Wang et al. "Human-Like Maneuver Decision Using LSTM-CRF Model for On-Road Self-Driving". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* 2018-Novem (2018), pp. 210–216. DOI: 10.1109/ITSC.2018.8569524.

[Woo+17] Hanwool Woo et al. "Lane-Change Detection Based on Vehicle-Trajectory Prediction". In: 2.2 (2017), pp. 1109–1116. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2660543.

[Xu+19] Yifei Xu et al. *Energy-Based Continuous Inverse Optimal Control*. 2019. arXiv: 1904.05453 [cs.LG].

[Xu+20] Yifei Xu et al. "Energy-Based Continuous Inverse Optimal Control". In: *arXiv* (2020). URL: https://arxiv.org/abs/1904.05453.

[Yan+18] Xinchen Yan et al. "MT-VAE: Learning motion transformations to generate multimodal human dynamics". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 265–281.

[YK19] Ye Yuan and Kris Kitani. "Diverse trajectory forecasting with determinantal point processes". In: *arXiv preprint arXiv:1907.04967* (2019).

[YT13] Peter K. Willett Yaakov Bar-Shalom and Xin Tian. *Tracking and data fusion: a handbook of algorithms, by*. Vol. 4. Mar. 2013, pp. 102–104.

[Zha+19] Pu Zhang et al. "Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12085–12094.

[Zha+20] Hang Zhao et al. "TNT: Target-driveN Trajectory Prediction". In: *arXiv preprint arXiv:2008.08294* (2020).

[Zhu+19] Yanliang Zhu et al. "Starnet: Pedestrian trajectory prediction using deep neural network in star topology". In: *arXiv preprint arXiv:1906.01797* (2019).

Appendices

Appendix A

Kalman Filter

A.1 Constant Velocity Forecast

Proposition 1. *Considering the state* $X(t) = (x(t), v_x(t))^T$ *and the vehicle kinematics given by the equalities* (A.1)*,*

$$\begin{cases} \dot{x}(t) = v_x(t) \\ \dot{v_x}(t) = a_x(t) \end{cases} \tag{A.1}$$

With $A_x = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}$*, the forecast of the state vector is written:*

$$X_{k+1} = A_x X_k + V_{k+1}$$

Proof. The differential equation(A.2) is obtained using the vehicle kinematics.

$$\dot{X}(t) = \underbrace{\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}}_{\Lambda} X(t) + \underbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}_{B} a_x(t) \tag{A.2}$$

The solution of this differential equation is to be estimated every dt seconds. The notation X_k is used to refer to $X(kdt)$. The exact solution of the equation (A.2) is given by the following equation:

$$X(t) = e^{(t-kdt)\Lambda} X_k + \int_{kdt}^{t} e^{(t-\tau)\Lambda} B a_x(\tau) d\tau \tag{A.3}$$

The exponential of the matrix Λ multiplied by a real coefficient $(t - kdt)$ is by definition:

$$\begin{aligned} e^{(t-kdt)\Lambda} &= \sum_{n \in \mathbb{N}} \frac{(t-kdt)^n}{n!} \Lambda^n \\ &= \underset{n=0}{Id} + \underset{n=1}{(t-kdt)\Lambda} + \underset{n>1}{0} + \dots \end{aligned}$$

Thus the equation (A.3) becomes:

$$\begin{aligned} X(t) &= \begin{pmatrix} 1 & t-kdt \\ 0 & 1 \end{pmatrix} X_k + \int_{kdt}^{t} \begin{pmatrix} 1 & t-\tau \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} a_x(\tau) d\tau \\ &= \begin{pmatrix} 1 & t-kdt \\ 0 & 1 \end{pmatrix} X_k + \underbrace{\int_0^{t-kdt} \begin{pmatrix} \sigma \\ 1 \end{pmatrix} a_x(t-\sigma) d\sigma}_{V(t)} \end{aligned} \quad \begin{matrix} \searrow \ \sigma = t-\tau \\ \searrow \ t = (k+1)dt \end{matrix}$$

$$X_{k+1} = \underbrace{\begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix}}_{A_x} X_k + V_{k+1}$$

This is the evolution equation from one time step to the next used in chapter 1. □

Proposition 2. *We consider an acceleration $a_x(t)$ modeled as a discrete-time centered white noise. This is expressed by the following equalities:*

$$\begin{aligned} \mathbb{E}[a_x(t)] &= 0 \\ \mathbb{E}[a_x(t)a_x(t+\sigma)] &= \begin{cases} q_{a_x} & \text{if } \sigma \in [0, dt[\\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{A.4}$$

where dt is the discrete time step. Then, the two first moments of V_k are:

$$\begin{aligned} \mathbb{E}[V_k] &= 0 \\ \mathbb{E}[V_k V_k^T] &= q_{a_x} \begin{pmatrix} \frac{dt^4}{4} & \frac{dt^3}{2} \\ \frac{dt^3}{2} & dt^2 \end{pmatrix} \end{aligned}$$

Proof. The first moment is given by:

$$\begin{aligned} \mathbb{E}[V_k] &= \mathbb{E}\left[\int_0^{dt} a_x(kdt-\sigma) \begin{pmatrix} \sigma \\ 1 \end{pmatrix} d\sigma \right] \\ &= \int_0^{dt} \underbrace{\mathbb{E}[a_x(kdt-\sigma)]}_{=0} \begin{pmatrix} \sigma \\ 1 \end{pmatrix} d\sigma \\ &= 0 \end{aligned}$$

The second moment is given by:

$$\begin{aligned} \mathbb{E}[V_k V_k^T] &= \int_{\sigma=0}^{dt} \int_{\tau=0}^{dt} \mathbb{E}[a_x(kdt-\sigma)a_x(kdt-\tau)] \begin{pmatrix} \sigma \\ 1 \end{pmatrix} \begin{pmatrix} \tau & 1 \end{pmatrix} d\tau d\sigma \\ &= q_{a_x} \int_{\sigma=0}^{dt} \int_{\tau=0}^{dt} \begin{pmatrix} \sigma\tau & \sigma \\ \tau & 1 \end{pmatrix} d\tau d\sigma \\ &= q_{a_x} \int_{\sigma=0}^{dt} \begin{pmatrix} \sigma\frac{dt^2}{2} & \sigma dt \\ \frac{dt^2}{2} & dt \end{pmatrix} d\sigma \\ &= q_{a_x} \begin{pmatrix} \frac{dt^4}{4} & \frac{dt^3}{2} \\ \frac{dt^3}{2} & dt^2 \end{pmatrix} \end{aligned}$$

□

In contrast with the hypothesis of a discrete-time centered white noise acceleration used in the chapter 1, we compute the moments of V for a continuous-time white noise acceleration. We show that we do not obtain the same variance (the mean value remains the same).

Proposition 3. *We consider an acceleration $a_x(t)$ modeled as a continuous-time centered white noise. This is expressed by the following equalities:*

$$\begin{aligned} \mathbb{E}[a_x(t)] &= 0 \\ \mathbb{E}[a_x(t)a_x(\tau)] &= q_{a_x}\delta(t-\tau) \end{aligned} \tag{A.5}$$

where δ is the Dirac function. Then, the second moment of V_k is:

$$\mathbb{E}[V_k V_k^T] = q_{a_x}\begin{pmatrix} \frac{dt^3}{3} & \frac{dt^2}{2} \\ \frac{dt^2}{2} & dt \end{pmatrix}$$

Proof. The second moment is given by:

$$\begin{aligned} \mathbb{E}[V_k V_k^T] &= \int_{\sigma=0}^{dt}\int_{\tau=0}^{dt} \mathbb{E}[a_x(kdt-\sigma)a_x(kdt-\tau)] \begin{pmatrix}\sigma \\ 1\end{pmatrix}\begin{pmatrix}\tau & 1\end{pmatrix} d\tau d\sigma \\ &= \int_{\sigma=0}^{dt}\int_{\tau=0}^{dt} q_{a_x}\delta(\tau-\sigma)\begin{pmatrix}\sigma\tau & \sigma \\ \tau & 1\end{pmatrix} d\tau d\sigma \\ &= \int_{\sigma=0}^{dt} q_{a_x}\begin{pmatrix}\sigma^2 & \sigma \\ \sigma & 1\end{pmatrix} d\sigma \\ &= q_{a_x}\begin{pmatrix} \frac{dt^3}{3} & \frac{dt^2}{2} \\ \frac{dt^2}{2} & dt \end{pmatrix} \end{aligned}$$

□

A.2 Innovation and Update

The Kalman filter is a recursive method to compute $\hat{x}_{k|k}$, $P_{k|k}$, $\hat{x}_{k+1|k}$, and $P_{k+1|k}$. At time step k, we note the observation $z_k = Hx_k + v_k$. With v_k a sample from $\mathcal{N}(0, R)$. Thus the observation at time step k knowing the previous observation is:

$$\begin{aligned} z_k|z_{k-1} &= Hx_k|z_{k-1} + v_k|z_{k-1} \\ &= Hx_k|z_{k-1} + v_k \end{aligned} \quad \big) \text{ The noise is independent.} \tag{A.6}$$

We note the condition "$|Z_{k-1} = z_{k-1}$" using "$|z_{k-1}$."

Proposition 4. *$Z_k|z_{k-1}$ and $X_k|z_{k-1}$ are jointly Gaussian. This means that the concatenated random vector is a Gaussian:*

$$\begin{pmatrix} X_k|z_{k-1} \\ Z_k|z_{k-1} \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma) \tag{A.7}$$

with $\mu = \begin{pmatrix}\mu_x \\ \mu_z\end{pmatrix} = \begin{pmatrix}\hat{x}_{k|k-1} \\ H\hat{x}_{k|k-1}\end{pmatrix}$ *and* $\Sigma = \begin{pmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{22} & \Sigma_{21}\end{pmatrix} = \begin{pmatrix} P_{k|k-1} & P_{k|k-1}H^T \\ HP_{k|k-1} & HP_{k|k-1}H^T + R\end{pmatrix}$

Proof. From equation (A.6), we know that $\begin{pmatrix} 0 \\ Z_k - HX_k \end{pmatrix} | z_{k-1} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & R \end{pmatrix}\right)$.

From the forecast model assumption, we know that $X_k \sim \mathcal{N}(\hat{x}_{k|k-1}, P_{k|k-1})$.
This implies $\begin{pmatrix} X_k \\ HX_k \end{pmatrix} | z_{k-1} \sim \mathcal{N}\left(\begin{pmatrix} \hat{x}_{k|k-1} \\ H\hat{x}_{k|k-1} \end{pmatrix}, \begin{pmatrix} P_{k|k-1} & P_{k|k-1}H^T \\ HP_{k|k-1} & HP_{k|k-1}H^T \end{pmatrix}\right)$

The sum of these two Gaussians is the joint Gaussian of state and observation written in (A.7). □

Proposition 5. *If* $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma)$ *is partitioned with* $\mu = \begin{pmatrix} \mu_x \\ \mu_z \end{pmatrix}$, $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{22} & \Sigma_{21} \end{pmatrix}$, *then* $X_1|(X_2 = x_2)$ *is Gaussian thus* $X_1|(X_2 = x_2) \sim \mathcal{N}(\mu_{1|2}, \Sigma_{11|2})$. *With* $\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$ *and* $\Sigma_{11|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$.

Two remarks: Being a Gaussian partition is a strong condition: the conditioned part is affected both on its covariance and mean. The conditioned covariance is the Schur complement of Σ.

Corollary 1. *The proposition 5 applies to the Kalman recursive step using equation* (A.7) *to update* $X_{k|k-1}$ *with the observation* z_k*:* $X_{k|k-1}|z_k \sim \mathcal{N}(\hat{x}_{k|k}, P_{k|k})$ *with*

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + \underbrace{P_{k|k-1}H^T \overbrace{(HP_{k|k-1}H^T + R)}^{S_k}{}^{-1}}_{K_k} \underbrace{(z_k - H\hat{x}_{k|k-1})}_{e_k} \\ P_{k|k} &= P_{k|k-1} - \overbrace{P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1}}^{K_k} HP_{k|k-1} \end{aligned} \tag{A.8}$$

The equation (A.8) is a combined equation of innovation and update steps of the Kalman filter described in section 1.2.1. This shows that under Gaussian assumptions, a step of the Kalman filter is a Bayesian update of the state estimation with a new observation.

Appendix B

Euler Spirals

In chapter 7, Euler spirals sometimes named Clothoids are used as local lane coordinate systems. The local coordinates express longitudinal and lateral vehicle positions in their lanes.

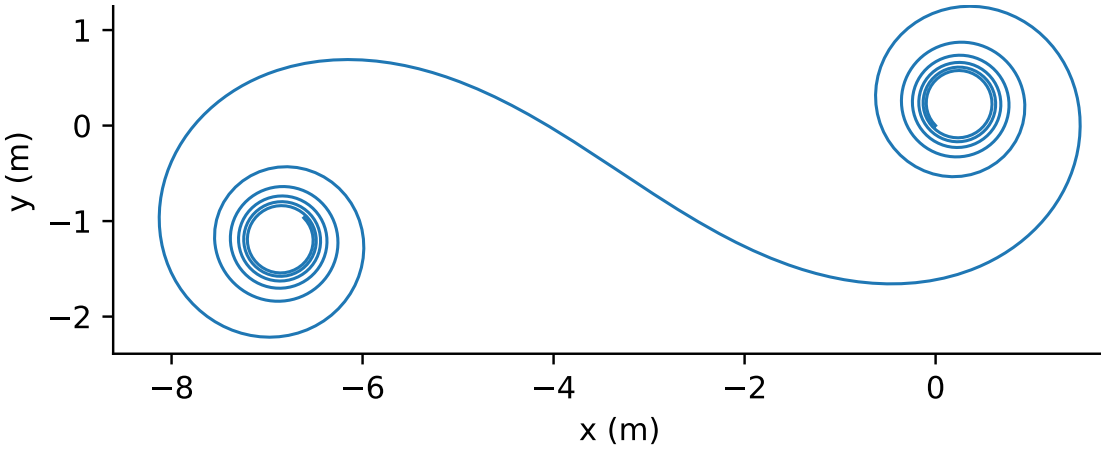


Figure B.1: Representation of an Euler spiral with parameters $(M_0 = (0,0), \Phi_0 = \frac{3\pi}{4}, c_0 = -3, c_1 = \frac{3}{25}, L = 50)$

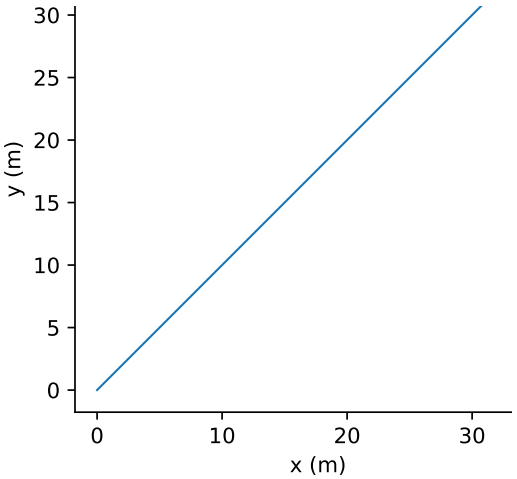
B.1 Definition and Properties

Definition 1. *For an orthonormal basis (Ω, x, y), a limited length Euler spiral is defined as $C = (M_0, \Phi_0, c_0, c_1, L)$. A representation is given in figure B.1. Its parameters are defined as follows:*

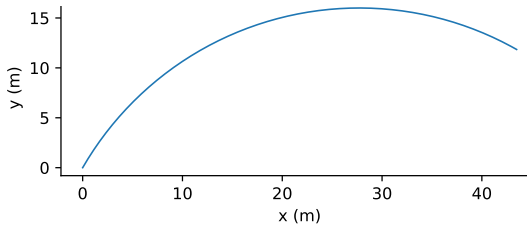
- M_0 *is the origin* (x_0, y_0)
- Φ_0 *is the angle at origin*
- c_0 *is the curvature at origin*
- c_1 *is the curvature variation rate*
- L *is the arc length*

Property 1. *For all $l \in [0, L]$ the distance along the curve from the origin, the curve has the following properties:*

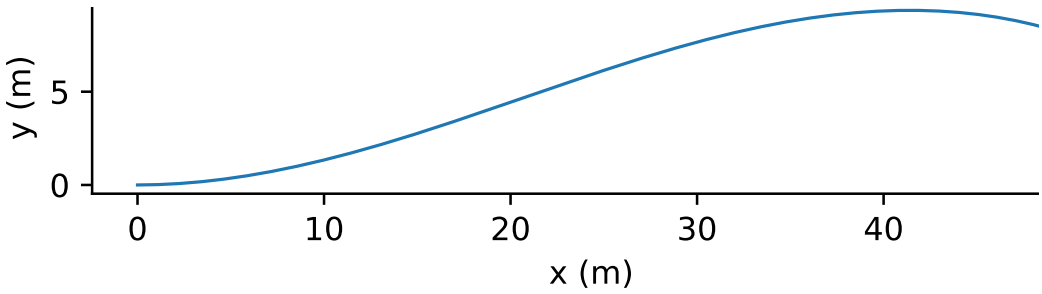
- $c(l) = c_0 + c_1 l$ *is the curvature*
- $\begin{array}{rcl} \Phi(l) & = & \Phi_0 + \int_0^l c(u)du \\ & = & \Phi_0 + c_0 l + c_1 \frac{l^2}{2} \end{array}$ *is the tangent angle*
- $M(l) = \big(x(l), y(l)\big) : \begin{cases} x(l) = & x_0 + \int_0^l \cos\left(\Phi(u)\right) du \\ y(l) = & y_0 + \int_0^l \sin\left(\Phi(u)\right) du \end{cases}$ *is the position*



(a) Linear case with parameters $(M_0 = (0,0), \Phi_0 = \frac{\pi}{4}, c_0 = 0, c_1 = 0)$



(b) Circular case with parameters $(M_0 = (0,0), \Phi_0 = \frac{\pi}{3}, c_0 = \frac{-1}{32}, c_1 = 0)$



(c) Particular case with parameters $(M_0 = (0,0), \Phi_0 = 0, c_0 = \frac{1}{32}, c_1 = \frac{-3}{2048})$

Figure B.2: Examples of Euler spirals with lengths $L = 50$.

Property 2. *The Euler spiral C is a rotated and translated version of $C' = (\Omega, 0, c_0, c_1, L)$ with* $\begin{pmatrix} x(l) \\ y(l) \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} \cos(\Phi_0) & -\sin(\Phi_0) \\ \sin(\Phi_0) & \cos(\Phi_0) \end{pmatrix} \begin{pmatrix} x'(l) \\ y'(l) \end{pmatrix}$
and $\begin{pmatrix} x'(l) \\ y'(l) \end{pmatrix} = \begin{pmatrix} \int_0^l \cos\left(c_0 u + \frac{c_1}{2}u^2\right) du \\ \int_0^l \sin\left(c_0 u + \frac{c_1}{2}u^2\right) du \end{pmatrix}$

Proposition 6. *The coordinates of the Euler spiral points can be computed using the Fresnel integrals:*

$$\mathcal{C}(t) = \int_0^t \cos(\frac{\pi}{2}u^2)du$$
$$\mathcal{S}(t) = \int_0^t \sin(\frac{\pi}{2}u^2)du$$

Proof. Let us define $\omega(u) = \sqrt{\frac{|c_1|}{\pi}}(u + \frac{c_0}{c_1})$ and use it in the position equation:

$$x'(l) = \sqrt{\frac{|c_1|}{\pi}} \int_{\omega(0)}^{\omega(l)} \cos\left(\text{sign}(c_1)\frac{u^2\pi}{2} - \frac{c_0^2}{2c_1}\right) du$$
$$y'(l) = \sqrt{\frac{|c_1|}{\pi}} \int_{\omega(0)}^{\omega(l)} \sin\left(\text{sign}(c_1)\frac{u^2\pi}{2} - \frac{c_0^2}{2c_1}\right) du$$

Let us define $(x''(l), y''(l))$ as Fresnel integrals:

$$x''(l) = \mathcal{C}\left(\sqrt{\frac{|c_1|}{\pi}}(l + \frac{c_0}{c_1})\right)$$
$$y''(l) = \text{sign}(c_1)\mathcal{S}\left(\sqrt{\frac{|c_1|}{\pi}}(l + \frac{c_0}{c_1})\right)$$

Then $(x'(l), y'(l))$ can be expressed as:

$$\begin{pmatrix} x'(l) \\ y'(l) \end{pmatrix} = \sqrt{\frac{\pi}{|c_1|}} \begin{pmatrix} \cos\left(-\frac{c_0^2}{2c_1}\right) & -\sin\left(\frac{c_0^2}{2c_1}\right) \\ \sin\left(\frac{c_0^2}{2c_1}\right) & \cos\left(-\frac{c_0^2}{2c_1}\right) \end{pmatrix} \begin{pmatrix} x''(l) - x''(0) \\ y''(l) - y''(0) \end{pmatrix}$$

□

Property 3. *The Fresnel integrals can be approximated using some algebraic fractions noted A and B :*

$$\mathcal{C}(t) \approx \frac{1}{2} - R(t)\sin\left((\frac{\pi}{2}(A(t) - t_2)\right)$$
$$\mathcal{S}(t) \approx \frac{1}{2} - R(t)\cos\left((\frac{\pi}{2}(A(t) - t_2)\right)$$

However, this approximation is not very good for small c_0 and c_1 which is the case in our applications where the roads are often straight lines and the ego vehicle is aligned with most lanes it observes. In that case, we use the local quadratic approximation (Simpson 3 points integrals) of the curve to approximate the integral for the position given in the property 1.

Property 4. *For f a function defined and integrable on $[a,b]$, we note $h = \frac{b-a}{n}$. Then the Simpson approximation of the integral of f on $[a,b]$ using $n+1 \geq 3$ equally spaced function estimations is written:*

$$\int_a^b f(u)du \approx S(a,b,h) = \frac{h}{3}\left(f(a) + f(b) + 2\sum_{j=1}^{\frac{n}{2}-1} f(u_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(u_{2j-1})\right)$$

We have defined the properties of the Euler spirals and how to compute the coordinates of a point from its distance along the curve. We also want to compute the Euler spirals parameters to fit a given curve and to compute the projections of given coordinates on the curves.

B.2 From coordinates to Euler Spirals

The current road lines perception system used in Renault cars is a smart camera from Mobileye that gives a polynomial fit of the perceive road markings. The polylines of an HD-map such as the one given with the Argoverse dataset can be represented with local polynomial fit. With the highD dataset application from chapter 7, we used the lane assignments of the tracked vehicle positions. All the coordinates for each assignment gives a 2D point cloud. We can make a piecewise polynomial fit of these points (in highD the observed road segments can be fitted as is without considering smaller pieces). In all cases, the local lanes are known with a polynomial fit. To be expressed as a polynomial, the local road segment must have a simple shape such that for one inverse image there is only one image. The road segment might need to be rotated by an offset Φ_{offset}.

Property 5. *An Euler spiral matching a polynomial p can be estimated from its origin position and its three first derivatives estimated at the origin:*

- $M_0 = (x_0, y_0)$ *in the absolute coordinates system*
- $\Phi_0 = \arctan\left(\frac{dp}{dx}(x_0)\right) + \Phi_{offset}$
- $c_0 = \frac{\frac{d^2p}{dx^2}(x_0)}{\left(1+\left(\frac{dp}{dx}(x_0)\right)^2\right)^{\frac{3}{2}}}$
- $c_1 = \frac{\frac{d^3p}{dx^3}(x_0)\left(1+\left(\frac{dp}{dx}(x_0)\right)^2\right) - 3\frac{dp}{dx}(x_0)\left(\frac{d^2p}{dx^2}(x_0)\right)^2}{\left(1+\left(\frac{dp}{dx}(x_0)\right)^2\right)^3}$
- L *is a set value*

Using the polynomial fit of the lanes and this property, we define local Euler spirals. We want to locate the surrounding vehicle positions relatively to the Euler spirals. The coordinates are supposed to be known in the absolute referential and should be expressed as longitudinal and lateral offset from the centerline of their lane represented with an Euler spiral. We want to project the car coordinates on the Euler spiral.

Definition 2. *For a point $P = (x,y)$ and an Euler spiral $C = (M_0, \Phi_0, c_0, c_1, L)$, we define the dot product $f(l) = \left\langle P - M(l), \frac{dM}{dl}(l)\right\rangle$. The projection of P on C is $M(l^*)$ with $l^* \in [0,L]$ such that $f(l^*) = 0$.*

Remark 1. *This projection may not exist and may not be unique.*

Property 6. *If it exists, the value l^* can be estimated using the Newton method. It is an iterative procedure that finds the zero value of the tangent at each step:*

$$\begin{aligned} f(l_n) + (l_{n+1} - l_n)\frac{df}{dl}(l_n) =& 0 \\ \textit{thus } l_{n+1} =& l_n - \frac{f(l_n)}{\frac{df}{dl}(l_n)} \end{aligned}$$

The stop criteria for the procedure are:

$$\begin{aligned} \left|\frac{df}{dl}(l_n)\right| <& \epsilon_1 \quad \textit{Local minimum almost reached by } l_n \\ n >& \textit{maximum number of iterations} \quad \textit{Avoid infinite loops} \\ \left|\frac{f(l_n)}{\frac{df}{dl}(l_n)}\right| <& \epsilon_2 \quad \textit{Zero value almost reached by } l_n \end{aligned}$$

With ϵ_1 and ϵ_2 small constants.

For $M(l) = \begin{pmatrix} x_0 + \int_0^l \cos(\Phi(u))\,du \\ y_0 + \int_0^l \sin(\Phi(u))\,du \end{pmatrix}$, with $\Phi(l) = \Phi_0 + c_0 l + \frac{c_1}{2}l^2$, we compute $\frac{df}{dl}(l)$ as follows:

$$\begin{aligned} f(l) &= \left\langle P - M(l), \frac{dM}{dl}(l) \right\rangle \\ \frac{df}{dl}(l) &= \left\langle P - M(l), \frac{d^2M}{dl^2}(l) \right\rangle - ||\frac{dM}{dl}(l)||^2 \\ \frac{dM}{dl}(l) &= \begin{pmatrix} \cos(\Phi(l)) \\ \sin(\Phi(l)) \end{pmatrix} \\ ||\frac{dM}{dl}(l)||^2 &= 1 \\ \frac{d^2M}{dl^2}(l) &= (c_0 + c_1 l)\begin{pmatrix} -\sin(\Phi(l)) \\ \cos(\Phi(l)) \end{pmatrix} \\ \frac{df}{dl}(l) &= (c_0 + c_1 l)\left\langle P - M(l), \begin{pmatrix} -\sin(\Phi(l)) \\ \cos(\Phi(l)) \end{pmatrix} \right\rangle - 1 \end{aligned}$$

We have defined Euler spirals and how to compute the coordinates of the points along the curve, how to compute their parameters from polynomial fits of the lanes, and how to project a coordinate on a given Euler spiral. Using these tools, we can transform a dataset given in absolute coordinates into one in relative coordinates along local Euler spiral fits of the lanes. This offers an abstraction from the road shape and was used in chapter 7. The road shapes are usually made of Euler spiral splines, this make Euler spirals a good candidate for lane fitting. Moreover, contrarily to polynomials, Euler spirals have the advantage of begin easy to rotate.

Appendix C

KL divergence of Gaussians

The Kullback-Leibler divergence is given by: $D(p||q) = \mathbb{E}[\ln(p) - \ln(q)] = \mathbb{E}[\ln(p)] - \mathbb{E}[\ln(q)]$. In the following, we compute the two terms separately with the hypotheses $p \sim \mathcal{N}(\mu, \Sigma)$, and $q \sim \mathcal{N}(0, I)$.

C.1 Computing $\mathbb{E}_p[\ln(p)]$

For $p \sim \mathcal{N}(\mu, \Sigma)$, we want to compute $\mathbb{E}_p[\ln(p)]$. We write D the dimension. The Probability Density Function (PDF) $p(\mathbf{x})$ is expressed as:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)} \tag{C.1}$$

First we expand the logarithm of the Gaussians PDF:

$$\begin{aligned}\mathbb{E}_p[\ln(p)] &= \frac{1}{2}\mathbb{E}_p\left[-\ln(|\Sigma|) - D\ln(2\pi) - (\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right] \\ &= -\frac{1}{2}\ln(|\Sigma|) - \frac{D}{2}\ln(2\pi) - \frac{1}{2}\mathbb{E}_p\left[(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right]\end{aligned} \tag{C.2}$$

The expected quadratic form is equal to D. We show it below by expanding the expression and using the Green identity of the integration by parts.

$$\begin{aligned}&\mathbb{E}_p[(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)] = \\ &\int_{\mathbb{R}^D} (\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}\mathbf{dx}\end{aligned} \tag{C.3}$$

We want to express this formula using the functions u and v defined below:

$$\begin{aligned}v(\mathbf{x}) &= (\mathbf{x}-\mu)^T \\ u(\mathbf{x}) &= e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)} \\ \mathrm{div}(v)(\mathbf{x}) &= D \\ \mathrm{grad}(u)(\mathbf{x}) &= -\Sigma^{-1}(\mathbf{x}-\mu)e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}\end{aligned} \tag{C.4}$$

We obtain the following expression where the Green's identity appears:

$$\mathbb{E}_p[(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)] = -\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\int_{\mathbb{R}^D} v\,\mathrm{grad}(u) \tag{C.5}$$

Let us recall the Green's identity for integration by parts that we want to use:

$$\int_\Omega v(\omega)\,\mathrm{grad}(u)(\omega)d\omega = \int_\Gamma u(\gamma)v(\gamma)\cdot n d\gamma - \int_\Omega u(\omega)\,\mathrm{div}(v)(\omega)d\omega \tag{C.6}$$

The integral over the frontier is null because for all $i \in [\![1, D]\!]$, $u(x_i)v(x_i) \xrightarrow[x_i\to\pm\infty]{} 0$.
The other term reduces to the integral of the Gaussian PDF:

$$\begin{aligned}\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\int_{\mathbb{R}^D} u(\mathbf{x})div(v)(\mathbf{x})\mathbf{dx} &= \frac{D}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\int_{\mathbb{R}^D} u(\mathbf{x})\mathbf{dx}\\ &= \frac{D}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\int_{\mathbb{R}^D} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)}\mathbf{dx}\\ &= D\end{aligned} \tag{C.7}$$

Replacing the expected quadratic form in equation (C.2) with 1, we obtain the final result:

$$\mathbb{E}_p[\ln(p)] = -\frac{D}{2}\ln(2\pi) - \frac{1}{2}(D + \ln(|\Sigma|)) \tag{C.8}$$

C.2 Computing $\mathbb{E}_p[\ln(q)]$

For $p \sim \mathcal{N}(\mu, \Sigma)$, and $q \sim \mathcal{N}(0, I)$, we want to compute $\mathbb{E}_p[\ln(q)]$. With $\Sigma = \mathrm{diag}(\sigma)$. We write D the dimension.
The Probability Density Function (PDF) $p(\mathbf{x})$ is expressed as:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)} \tag{C.9}$$

The PDF $q(\mathbf{x})$ is expressed as:

$$q(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}}}e^{-\frac{1}{2}\mathbf{x}^T\mathbf{x}} \tag{C.10}$$

We expand the logarithm of $q(\mathbf{x})$ and recall that on each independant dimension i, $var(x_i) = \mathbb{E}[x_i^2] - \mathbb{E}[x_i]^2$ and thus $\mathbb{E}[x_i^2] = \sigma_i^2 + \mu_i^2$:

$$\begin{aligned}\mathbb{E}_p[\ln(q)] &= \frac{1}{2}\mathbb{E}_p\left[-D\ln(2\pi) - \mathbf{x}^T\mathbf{x}\right]\\ &= -\frac{D}{2}\ln(2\pi) - \frac{1}{2}\mathbb{E}_p[\mathbf{x}^T\mathbf{x}]\\ &= -\frac{D}{2}\ln(2\pi) - \frac{1}{2}\sum_{i=1}^{D}\sigma_i^2 + \mu_i^2\end{aligned} \tag{C.11}$$

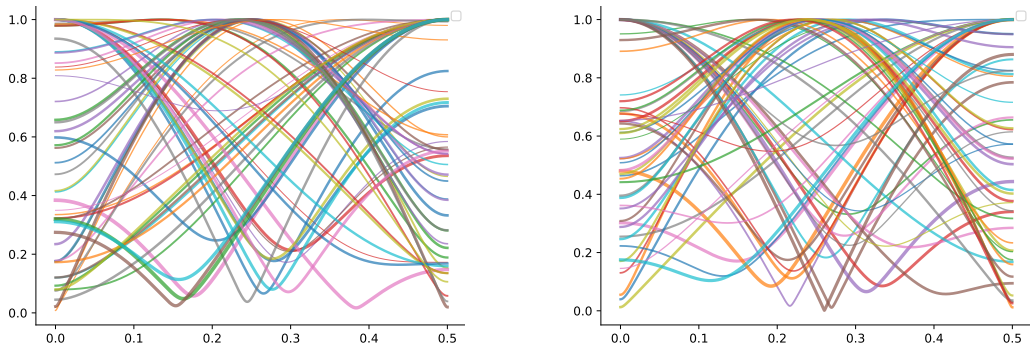
Appendix D

Fourier Interpretation of the Convolution Kernels

This appendix is a supplementary analysis of the convolution kernel in our final model defined in chapter 9. Figure D.1 shows the normalized frequency response moduli of the convolution kernels at initialization and figure D.2 after training. The convolution kernels are composed of 3×2 float values. For this analysis, we separate the x and y axis. Thus we consider 3 values noted k_{-1}, k_0, k_1. The Fourier transforms are computed as:

$$\mathrm{FT}(\nu) = k_{-1}e^{2i\pi\nu} + k_0 + k_1 e^{-2i\pi\nu} \tag{D.1}$$

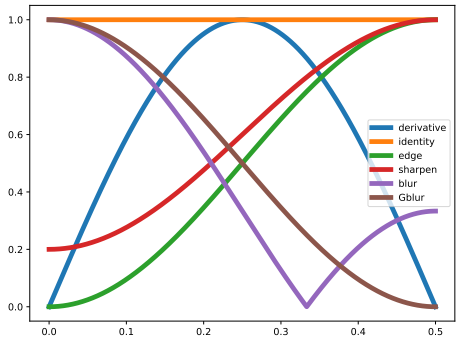
Figure D.2 represents modulus plots $|\,\mathrm{FT}\,|(\nu)$ of the filters. Figure D.3 represents phase plots $\mathrm{angle}(\mathrm{FT}(\nu))$ of the filters.



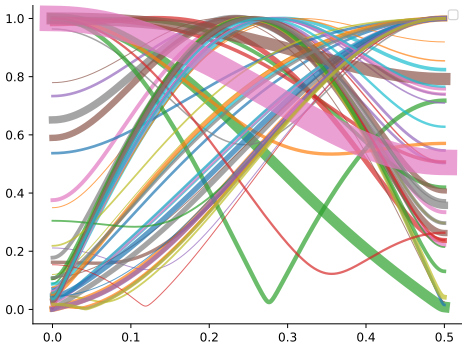
(a) Moduli of the filters along the x axis at initialization.

(b) Moduli of the filters along the y axis at initialization.

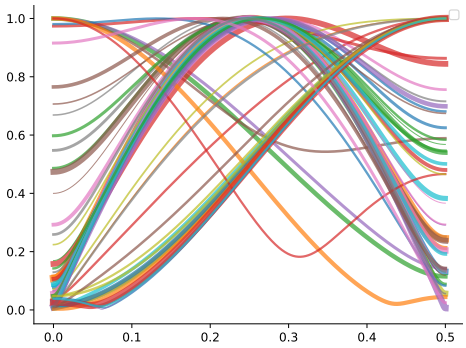
Figure D.1: Moduli of the normalized kernels Fourier transform. The plots are normalized with the maximum moduli value and their line width are proportional to this value.



(a) Reference kernel frequency response moduli: derivative [-1 0 1], identity [0 1 0], edge [-1 2 -1], sharpen [-1 3 -1], blur [1 1 1], Gblur [2 4 2].

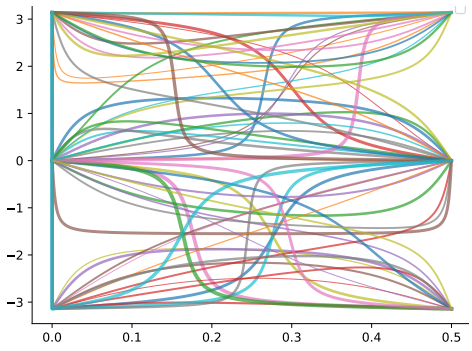


(b) Moduli of the filters along the x axis after training.

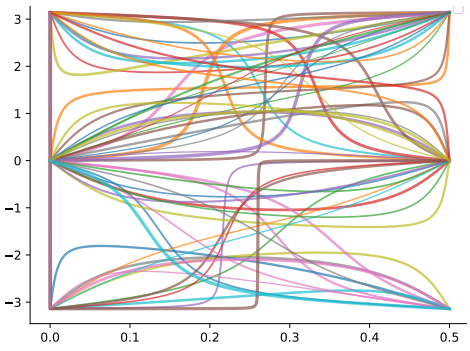


(c) Moduli of the filters along the y axis after training.

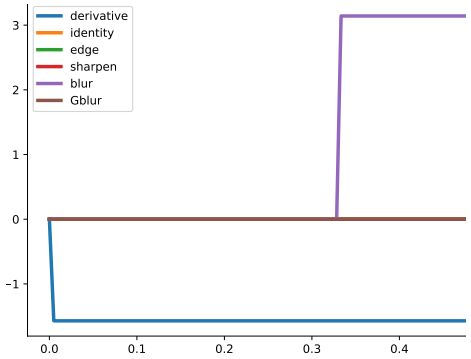
Figure D.2: Moduli of the normalized kernels Fourier transform. The plots are normalized with the maximum moduli value and their line width are proportional to this value.



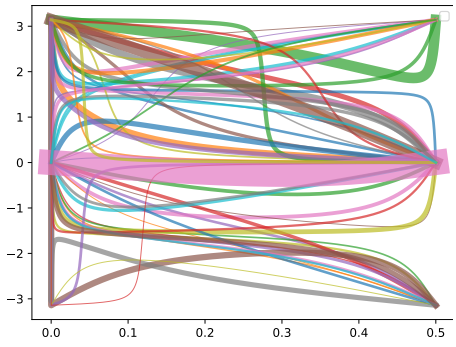
(a) Kernel frequency response phase in radian for the filters along the x axis at initialization.



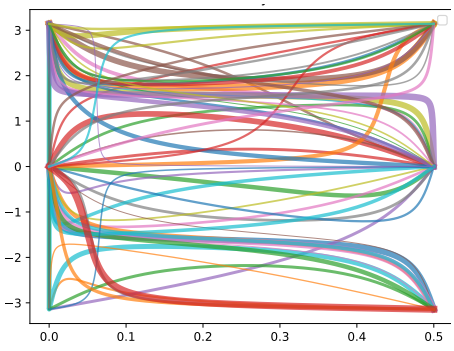
(b) Kernel frequency response phase in radian for the filters along the y axis at initialization.



(c) Reference kernel frequency response phases: derivative [-1 0 1], identity [0 1 0], edge [-1 2 -1], sharpen [-1 3 -1], blur [1 1 1], Gblur [2 4 2].



(d) Kernel frequency response phase in radian for the filters along the x axis after training.



(e) Kernel frequency response phase in radian for the filters along the y axis after training.

Figure D.3: Phase of the kernels Fourier transform. The line width are proportional to the maximum module value.