HiPSS–LSTM:

A Hierarchical Probabilistic LSTM model for motion forecasting

Bartosz Zabłocki

Master Thesis
Algorithmics Group
April 7, 2019

TU Delft

HiPSS–LSTM:

A Hierarchical Probabilistic LSTM model for motion forecasting

by

Bartosz Zabłocki

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday April 7, 2020 at 10:00 AM.

Student number:	4739671	
Project duration:	February 1, 2019 – April 7, 2020	
Thesis committee:	Dr. M. T. J. Spaan,	TU Delft, supervisor
	Dr. J. C. van Gemert,	TU Delft
	Dr. B. Bakker,	Cygnify BV

Preface

It was quite an adventure, with ups and downs, and, therefore, I would like to express my gratitude to all of those who supported me during this time.

I would like to thank Dr. Bram Bakker from Cygnify, who believed in me and was patient enough to guide me through the process of developing and writing this thesis. His intuition, experience and knowledge on the topic were what I needed to develop this work. Thanks to him I was able to accomplish my goal - work on a project, which not only was ambitious but also closely related to challenges faced by Autonomous Vehicles researchers. He gave me a lot of space for development of my ideas and was also always there when I needed his guidance. Bram, I really appreciate it.

I would also like to thank Dr. Matthijs Spaan, who was my academic supervisor for this project, for making sure that the development of the thesis is on track. His feedback and ideas helped shape this research and thesis. I would like to thank Dr. Jan van Gemert for accepting my request to join the thesis committee.

A big thank you goes to my girlfriend, Malwina, who was there for me whenever I needed it. Thank you for listening and advising me when I was in doubt and for sharing the joy and excitement with me, whenever I had a breakthrough. Your companion has an indescribable value.

To my parents and sister, who always believed in me and supported in every single decision I took. I would never be here and accomplish this without you. Dziękuję!

Bartosz Zabłocki
Delft, March 2020

Contents

1 Introduction **3**
- 1.1 Importance of a path prediction for ADAS . . . 3
- 1.2 System overview . . . 4
- 1.3 Related work . . . 5
  - 1.3.1 Probabilistic approaches . . . 5
  - 1.3.2 Inverse Reinforcement Learning . . . 5
  - 1.3.3 Deep Learning approaches . . . 6
- 1.4 Research questions . . . 7
- 1.5 Contributions . . . 8

2 Background on the HiPSS architecture **11**
- 2.1 Recurrent Neural Networks . . . 11
- 2.2 Long Short–Term Memory (LSTM) networks . . . 12
  - 2.2.1 LSTM modes . . . 12
  - 2.2.2 Encoder–Decoder LSTM architecture . . . 13
  - 2.2.3 SS-LSTM architecture . . . 13

3 Problem description **17**
- 3.1 System's requirements . . . 17
  - 3.1.1 Input modalities . . . 17
  - 3.1.2 Output representation . . . 18
  - 3.1.3 Other requirements . . . 18
- 3.2 Performance Metrics . . . 18
  - 3.2.1 Standard Metrics . . . 18
  - 3.2.2 High–level network performance metrics . . . 19

4 Hierarchical Probabilistic SS-LSTM (HiPSS-LSTM) **21**
- 4.1 Reasoning and architecture overview . . . 21
- 4.2 High–level prediction – detailed information . . . 22
  - 4.2.1 Architecture . . . 22
  - 4.2.2 Training process . . . 22
- 4.3 Low–level prediction network – detailed information . . . 24
  - 4.3.1 Architecture . . . 24
  - 4.3.2 Cost function . . . 25
  - 4.3.3 Training . . . 27
- 4.4 Input representation – implementation details . . . 27
  - 4.4.1 Observed trajectory . . . 28
  - 4.4.2 Social forces representation . . . 28
  - 4.4.3 Scene representation . . . 28

5 Dataset **31**
- 5.1 Requirements . . . 31
- 5.2 Available datasets . . . 31
  - 5.2.1 Generating lane–level dataset with Argoverse . . . 32

6 Experiments and results **35**
- 6.1 Hyperparameters . . . 35
  - 6.1.1 Learning rate tuning . . . 36
  - 6.1.2 Activation functions . . . 36
  - 6.1.3 Optimizer . . . 36
  - 6.1.4 Final destination input representation (for the low–level network) . . . 36
  - 6.1.5 Scene encoder's layers configuration . . . 37

6.2 Experiments and results . . . 38
6.2.1 An ablation study . . . 40
6.3 High–level network performance . . . 40
6.4 Visualization and discussion on results . . . 42
6.4.1 Influence of social forces . . . 42

7 Conclusions, limitations and future work **47**
7.1 Conclusions . . . 47
7.2 Limitations . . . 47
7.3 Future work . . . 48

Appendices **51**

A Machine Learning - the essentials **53**
A.1 Feedforward network . . . 53
A.1.1 Activation functions . . . 53
A.2 Cost functions, stochastic gradient descent and backpropagation . . . 54
A.3 Long Short–Term Memory (LSTM) networks . . . 55

Bibliography **59**

Abstract

When driving a car, people can usually predict the intention of other road users with high confidence. They can spot small variations in a recent trajectory, take into account road infrastructure, traffic rules and other factors, which influence a future trajectory. As a result, they can interact with other drivers smoothly and safely.

Even though human drivers are generally very good at driving, they still make a lot of errors, which lead to many accidents. Therefore, future technology should support the drivers in this task and eventually replace them, to increase safety on roads. The future vehicle should be able to understand and predict the behavior of the surrounding road users and react accordingly. However, predicting trajectories of other road users is an extremely complex task for an algorithm, because of many different factors influencing road users' decisions. Existing approaches predicting future trajectories are mostly based only on the past, observed trajectories, often resulting in very unrealistic predictions. The predicted trajectories are often not in line with road infrastructure, nor do they take into account other road users surrounding them. Moreover, current methods usually predict only one most likely trajectory, disregarding all other possibilities. The main objective of this thesis is to find a Deep Learning architecture that can predict a few most likely future destinations for a target vehicle, and generate accurate trajectories leading to each of them, while taking into account infrastructure, possible trajectories there, target vehicle's recent behavior and other road users around the target vehicle.

This work uses a well–performing architecture for pedestrian trajectory prediction in crowded spaces and further develops it, by adapting it to a different domain – vehicle motion prediction in an urban environment. Moreover, this work extends the method, to predict a set of possible trajectories. This was achieved by implementing two–stage architecture. The first stage predicts several high–level destinations, which indicate an estimated location where a target vehicle will be after 3 seconds. Each high–level destination is defined with a probability indicating confidence of the prediction. The second stage uses those predictions to generate low–level trajectories, which are 30 consecutive locations, leading from the last observed position towards the high–level goal.

This two–step architecture is trained on an extensive dataset, which was specifically made for autonomous–vehicle applications. To make even better use of the dataset, the network was pretrained on lane centerlines before it was trained on real trajectories. The centerlines were supplied with the dataset. The results show high prediction accuracy, both in terms of the high–level goals, and in terms of the low–level trajectories generated for each high–level goal.

1 Introduction

This thesis aims to find a Deep Learning architecture, which will be able to reliably predict a few best vehicle's future trajectories. The motion prediction is an active research topic and Deep Learning based techniques are widely used to solve this challenge. This thesis focuses specifically on a system predicting future trajectories of road vehicles.

1.1. Importance of a path prediction for ADAS

Whether you are driving a car or biking on a bicycle, your safety and the safety of other road users should be what matters the most. There are 3,287 accidents world–wide involving road users daily [2]. A lot of them are due to human error. The trend has been changing over the years, but we are far from having safe roads. According to research, traveling by car is the most dangerous means of transport. Moreover, unless action is taken, road traffic injuries are predicted to become the fifth leading cause of death by 2030 [2]. Around 94% of accidents are caused by human error [35]. And yet, the number of cars on roads still increases, leading to even more accidents.

At the same time, car manufacturers and governments are improving the safety of drivers and other road users: from making seat belts obligatory equipment in 1983, to improvements in light technology, to Advanced Driver Assistance Systems (ADAS). Those ADAS systems include safety features such as emergency braking, blind–spot warning, traffic warnings, adaptive cruise control, etc. They all increase safety on roads. All the current safety systems are reactive – they activate only at the time or just before an event. The emergency braking will turn on, only at the latest possible moment, right before the crash. Blind–spot warning will alert the driver only when they indicate an intention to change a lane. Sometimes, these kinds of warnings are presented too late. The next generation of those systems should be predictive. They should monitor the environment and reason about it in real–time. They should be able to assess the probability of a pedestrian jaywalking or another car steering suddenly into the path of a driving vehicle. They should predict hazardous situations multiple seconds before they occur.

One way of improving those systems is to create a predictive motion forecasting system. A system that would analyze the current behavior of other road users and predict a set of plausible trajectories they would take. This is still an active research topic. Several works are treating this problem, but most of them focus on predicting a single best trajectory, instead of generating multiple feasible ones. This master thesis is focusing on creating a system that can be used as a predictive driver assistance system. The system generates a set of feasible trajectories a target vehicle could take in an urban environment. Moreover, the trajectories are reliably ranked based on a certainty of the prediction. Generation of low–level trajectories takes into account other road users and scene layout – which areas are drivable or not. This kind of system could alert a driver about a car that is on a collision course with them, or about a cyclist who suddenly appears in front of the car bonnet because they came unnoticed. These hazardous situations could be avoided if the car could inform drivers about their intention. The same system could eventually be used for fully autonomous cars, helping them to decide on how to interact with other road users.

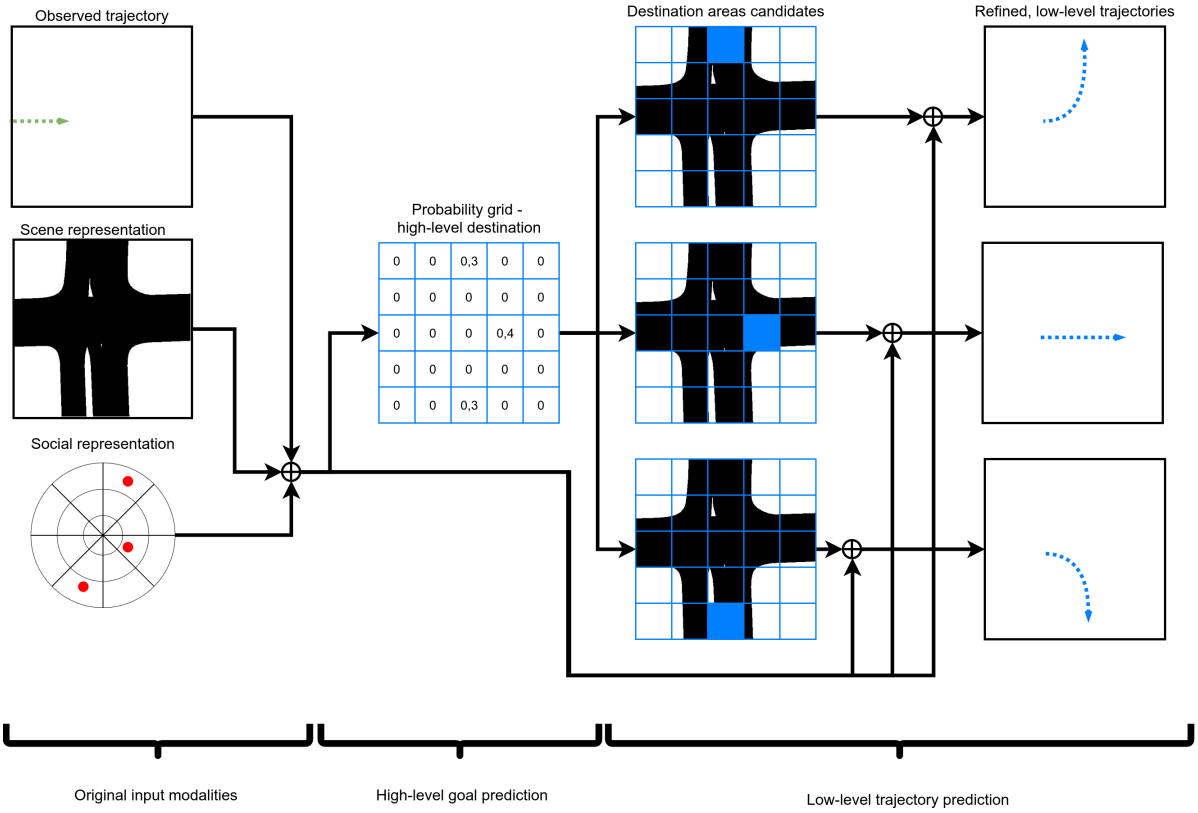


Figure 1.1: An overview of the prediction process. Firstly, all the inputs (observed trajectory, scene and social representations) are concatenated together to be fed to a network. For the first stage, the high–level network, scene has to be discretized to a grid. The grid size corresponds to a resolution of an output – the more specific the output is, the higher resolution of the grid should be (in this work tested resolutions range from 5 by 5 up to 15 by 15 cells). Then, a high–level network estimates destinations on the discretized grid, by assigning a probability to every single grid cell, indicating how likely the given grid cell is to be a destination. Next, for every distinct destination, a low–level network can generate a precise, position by position trajectory. The prediction is based on an original resolution of the scene. Depending on a task, the network would usually predict trajectories for top–K or a few destinations with a likelihood value above some threshold.

1.2. System overview

The goal of this work is to create a system based on short observation histories together with context information that can predict where a target vehicle will go in the near future. Based on current observations there are usually a few plausible scenarios for what will happen next. Humans can anticipate the near future and act upon it well, leading to reasonable decisions when driving a car. Next–generation vehicles should be able to anticipate the near future as well, to support their steering decisions or warn a human driver. The architecture designed in this work, called Hierarchical Probabilistic Social–Scene Long Short–Term Memory (HiPSS–LSTM) extends state–of–the–art techniques for a singular path prediction, to multiple paths prediction with an uncertainty measure. It is an architecture, which based on target vehicle's trajectory, drivable areas and neighboring vehicles can predict a few most likely high–level goals and generate precise, low–level trajectories leading to them, which are consistent with drivable areas.

The created network consists of two separate elements; high– and low–level predictions, as presented in the Figure 1.1. Those two elements are separate networks, which run sequentially, one after another.

The high–level prediction network is the first stage in the process. Its goal is to *roughly* predict a few target vehicle's destinations, along with an uncertainty corresponding to them. This network uses the vehicle's observed trajectory, scene and social representation as inputs. Social input is represented as a circular occupancy map. It is a data structure, which discretizes a number of road users depending on distance and direction, into a few *bins*. This simulates relations between road users, which mostly depend on distance and direction. The scene is a rasterized, bird–eye view representation of drivable areas in the target vehicle's proximity. This is a binary mask, where drivable areas are marked with „1”

(such as roads) and all the others are marked with „0". For the high–level network, the scene has been discretized into a square grid representation. The network has been trained to assign a probability to each grid cell, where each probability represents a likelihood of the target ending up in the location corresponding to the given grid cell after a fixed amount of time. The output of the high–level network serves as one of the inputs for the low–level prediction network.

The low–level prediction network takes as an input the same modalities as the high–level network (an observed trajectory, scene and social representation) and a representation of one of the destinations predicted in the previous stage. This means, that for each predicted destination, a low–level network predicts one, precise trajectory, which is leading to the grid cell provided as an input. The prediction has to be made for all desired destinations separately. The final results are a few plausible, accurate trajectories, leading towards different high–level goals, where each of the goals is described with a likelihood measure. An overview of the process is presented in Figure 1.1.

1.3. Related work

There are a few distinct approaches to a trajectory prediction problem, starting with probabilistic approaches, to modern deep–learning approaches. Some of the works focus on pedestrian trajectory prediction and some on road users' trajectory prediction. Both types of works are interesting because they usually take the same kind of inputs into account and output a representation of a trajectory. Since the problems are very similar to each other, literature studied for this work contains both types of them.

1.3.1. Probabilistic approaches

There are a few works that use a probabilistic approach to predict future trajectory. Among them are approaches that are using Dynamic Bayesian Networks (DBNs) [3, 23] or approaches that are using Kalman Filters (KFs) [32]. In both cases, the path prediction is done iteratively – prediction of a first step is based on recent observations and then the following steps are predicted using previous predictions. The advantage of using a KF is that it generates a probability distribution of the prediction. The result of this iterative process comes down to a single trajectory with a growing uncertainty with every predicted step.

Early work on path prediction used modalities like walking speed and acceleration [32]. Later, DBNs have been used for that task as well. In a more restricted case, where the task was only to predict if a pedestrian crossed a street, Kooij et al. [23] used a model with a Switching Linear Dynamical System (SLDS). This work used features extracted from onboard car videos, such as pedestrian's head orientation, distance to the car and distance between the pedestrian and a roadway. The prediction was only done for a very small time horizon – 1 second.

1.3.2. Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL), in contrast to Reinforcement Learning (RL), is a method of learning an agent's objectives, values, or rewards by observing its behavior. It is a method, which relies on Markov Decision Processes, in which a reward function has to be manually designed. IRL, on the other hand, infers the reward function in an iterative process from observations, and then the found reward function can be used to find the best policy in an RL problem [1]. A challenge comes from the choice of representation of input modalities. Generally, the more dimensions the input has, the more precise the results can be, but at the same time, the complexity of the model rises exponentially [38]. This implies two issues: a need for exponentially more data; and that learning and execution time is substantially higher.

Despite the obstacles, this method is successfully used in [42], where authors decided to combine kinematic and environmental context, to lower the number of features and model's complexity. They also collected their dataset of off–road driving, to collect enough data. The environment is represented as a top–down view picture, in a form of heat–map reflecting the height of an environment around. This representation makes the algorithm detect obstacles (which are higher than the target vehicle and therefore depicted with a different color). The output is a probability distribution of future paths, which again, looks like a heat–map depicting probabilities of future positions. There are two downsides – the predicted trajectory cannot be directly represented as consecutive positions; and very high execution time, despite the lowered model's complexity. In consequence, future path prediction takes too long to be executed in real–time.

A similar concept to this work's approach was implemented in [30]. It incorporates two–stage path generation. Firstly, a destination point (on a discretized top–down view map representation) is predicted, using Recurrent Mixture Density Networks (RMDNs), which estimate Mixture Density Function. Each component of the function represents a destination alternative. Since each function is a Probability Density Function, not only a location is estimated but also a certainty. In the second phase, the destination is reused and goal–oriented path planning algorithm is used. The problem then is defined as a Markov Decision Process, where a goal is to select an appropriate set of actions a_i for every state in S_t to reach one's goal. The training of policy, transition models and reward function from observed trajectories is done with Inverse Reinforcement Learning. This work focuses on path prediction for pedestrians. The approach is similar to the approach taken in this work – first a high–level destinations are estimated and then a low–level trajectory is generated. This work uses different components to execute this task. Early experiments showed that RMDNs are not the best fit for the task. As every function's component is described with 3 variables (mixing coefficient, mean and standard deviation) and every component describes one alternative location, the output space grows really fast and each of the parameters is very sensitive and hard to train. Additionally, the low–level path generation technique presented in the aforementioned work is too slow to be used in real–time applications.

1.3.3. Deep Learning approaches

Deep learning approaches usually incorporate one or multiple methods out of the following: Generative Adversarial Networks (GANs), Recurrent Neural Networks(RNNs) and/or Convolutional Neural Networks (CNNs).

GANs are deep generative models composed of two networks, a generator and a discriminator. The generator is a network that learns to generate some images. The second network is a discriminator. It tries to distinguish real data from candidates synthesized by the generator. As those two networks compete with each other, the generator learns to generate more and more realistic images and discriminator improves on distinguishing real and fake images. When they converge, the generator outputs so realistic images, that the discriminator cannot distinguish them from real ones.

GANs were used in [17] where the generator and discriminator were LSTM networks. The goal of the work was to predict a trajectory of multiple pedestrians, based on observed path, taking into account „social interactions". The social interactions were learned by predicting the trajectories simultaneously for multiple pedestrians. The method is able to output multiple predictions by exploring a latent space. Therefore, to output all possible trajectories, the whole latent space needs to be explored. To generate distribution map of trajectories, authors had to sample 300 points from the latent space. In practice, this means that they had to run the network 300 times, to reliably generate all possible predictions for one observed trajectory. In a real–time scenario, this might be a bottleneck.

Most of the approaches in the trajectory prediction domain use some kind of RNNs as the main element of architecture. RNNs, as described in Section 2.1, are networks created for sequence modeling. Since a trajectory can be interpreted as a sequence of positions, it seems that RNNs could be a good fit for the task. In a special case, CNNs can also be used for sequence modeling [39].

In a very constrained problem in [43], RNNs were used to predict a few possible trajectories, along with a likelihood, but only on five intersections. The architecture incorporated RNNs to model Mixture Density Function. The resulting probability distribution had to be then processed and clustered to several distinct trajectories. Authors do not compare the algorithm to any publicly available dataset, so the ability of the model to generalize to any other intersection is uncertain.

Another work [25] introduced stochastic recurrent encoder-decoder network architecture. The overall model comprises three components, which include a Conditional Variational Autoencoder (CVAE) followed by an RNN encoder-decoder to generate a set of plausible future states, followed by an inverse optimal control-based ranking and refinement module to rank the predictions and enforce global consistency. The method takes into account scene context along with a social context, by processing all the target objects at once. The method produces a fixed number of predictions (some might have a likelihood level equal to zero).

Another work [36] claims to yield better accuracy results and better transferability across different datasets than the aforementioned [25] work. It uses similar inputs, observed positions and a map representation, but the positions are represented in an image–like data structure, which size corresponds to the size of the map's representation. Each position is marked as a white „dot" on an image. Other road users are represented similarly. Those inputs can be seen as 3 different images, one for each

modality. Then, using CNNs and ConvLSTMs [33], a forecasted trajectory is created. The consecutive positions are initially encoded in the same way as input data (2D array), so it needs a postprocessing step, to represent trajectory in terms of coordinates. Authors claim that the method can generate multiple possible trajectories, but when trained on KITTI dataset, the trajectories are usually in the same high–level direction (e.g. all go along the same straight line, with only small deviations).

Instead of predicting consecutive coordinates in the Cartesian space directly, Xu et al. in [40] decided to predict a displacement between the current and next frames. Their architecture is based on LSTMs and they are also processing all the target pedestrians at once, to model the dynamic relations between them, but their social interactions modeling is more sophisticated. The main module, „crowd interaction module" combines two other modules – „motion encoder" (a few stacked LSTMs which are fed with consecutive observed positions) and a „location encoder module" which encodes jointly social interactions of all the target pedestrians. The „location encoder module" is based on the idea that social interactions should not be based solely on Euclidean distance between them. Given two pedestrians, even if their distances to the target person are the same, they might influence their choices differently (e.g. when one pedestrian is going directly towards a target pedestrian, it probably influences the target pedestrians choices more, than a pedestrian who is going in the same direction next to him). To solve this issue, authors implemented a neural network with a few layers for each pedestrian, whose results are at the end multiplied together, modeling interactions in a non–linear way.

In [14], the authors used only CNNs and Feedforward Networks. All the input modalities were represented in a top–down view, rasterized representation. Each color resembled a different kind of information, like a target vehicle, other vehicles, drivable areas, lane–markings. The only modality without a fixed color is lanes. Every lane, depending on a heading direction, is depicted with a different color in an HSV color space. Therefore, lanes that are going in the opposite direction are represented by colors diametrically opposite to each other on the HSV color cylinder. Along with the image representation, there is also a vector defining a dynamic context (like speed and heading). The rasterized representation and the vector are processed with a sequence of CNNs and Feed Forward Networks. The output is a $3n$-dimensional vector, where n is a predicted number of steps. The first two dimensions are coordinates of a target and the third dimension depicts the standard deviation of the position (the bigger, the less certain prediction). The network cannot yield top–K predictions, only the single best prediction.

A method called Social–Scene LSTM (SS–LSTM) [41] was created to predict a trajectory of pedestrians in crowded environments. The authors identified that for the task, the most important input modalities need to capture current pedestrian dynamics, social forces and scene layout. Each of the inputs is encoded separately with an LSTM and all the encoded vectors are then merged. Such a vector is then encoded with another LSTM. The whole network is trained to output a single best prediction.

This work In this work, it was decided to build an architecture on top of the SS–LSTM architecture [41]. The architecture follows a modular approach, which gives flexibility (there is a separate encoder for each modality – trajectory, scene layout and social factors). Therefore, new components can be easily added and it can be adapted to alternative outputs. The network can be used for both the high-level prediction and for the low-level prediction, with relatively few changes.

1.4. Research questions

The main research question of this work is **how to create a motion forecasting system, which outputs multiple feasible predictions and ranks them based on how likely they are to be chosen?**

There are four other subquestions, which help to find an answer to the research question:

1. What kind of input modalities must be taken into account and how should they be represented to create a reliable system for a trajectory prediction? This will help determine what are the most which are the most predictive modalities for a motion prediction. Those will serve as an input to the network.

2. What kind of deep learning architecture fits best to generate a set of feasible trajectories? It is important to find an architecture, which can output several trajectories. Additionally, all the trajectories have to be realistic and as compatible with a road layout as much as possible.

3. How to represent a certainty of predictions? It is desired to be able to assess the certainty of a prediction – both for ranking purposes (to be able to select the top N most likely predictions) and for importance assessment (e.g. to indicate to a human driver how likely they are to be on a collision course with another car).

4. What time horizon, both for observation and prediction, should be taken into account? For safety reasons, the quicker a prediction can be made, the better. The biggest factor influencing prediction time is the time needed for capturing a target vehicle's dynamics. Therefore, minimal observation time should be found, which would not worsen prediction results. Furthermore, the longer the predicted path, the better, so various prediction times should also be considered.

1.5. Contributions

There are several contributions this work introduces.

- **Multimodal prediction**: if the system is meant to be used in real–life, it has to account for a multimodal nature of future predictions. It has to be able to output a diverse set of feasible trajectories. A few recent works took it into account as well and created systems that can output multiple predictions, such as [25, 36], but it is hard to determine how diverse those predictions are. [43] on the other hand, focused on outputting a diverse set of predictions, but the task was constrained to a very limited number of types of intersections. This work aims to create a diverse set of predictions on a high–level, following with accurate predictions on a low–level, so that each trajectory is substantially different from another.

- **Pretraining technique**: the high–level and low–level prediction networks are pretrained on an artificially generated dataset first, and only then they are trained on real trajectories. This has been incorporated, because all the datasets are biased towards one kind of trajectories – the ones that are going straight ahead. Only a fraction of the trajectories are turning left or right on an intersection. Therefore, generating an artificial dataset (described below), in which a number of each type of trajectories is controlled, is a step to counteract this bias.

- **Artificial Dataset**: to be able to incorporate the aforementioned pretraining step, a fully artificial dataset has been created. It was achieved using lane–level information included in the dataset's HD maps. To generate the trajectories, each lane was analyzed and transformed into a set of points. The points along one lane have to be correctly distanced from each other – they have to resemble a real trajectory. Every lane in the dataset was processed and saved in this fashion, resulting in a big dataset of artificial trajectories. This also has other implications:

  1. since the artificial dataset is generated manually, there is the same number of trajectories going straight, to the left and to the right, to counteract the bias in the dataset with real trajectories. The dataset constitutes of 125373 different trajectories;
  2. moreover, if the network is trained on those trajectories alone, the indirect lane–level information is directly associated with a scene layout. This happens, because the network is trained to output a trajectory that is to output a trajectory that is consistent with a lane. Therefore, there is no need for an explicit lane–level input data, making it far more robust to different kinds of situations, such as incomplete lane–level information;
  3. additionally, the network learns to generate smooth and reasonable trajectories, which are more compliant with a layout of drivable areas, since all the trajectories are strictly inline with drivable areas.

  The only known work taking into account lane–level information is done in [14] and in [10] in a baseline method provided by authors of a dataset. There the lane centerlines are explicitly provided as an input. This work is the only one that uses the lane–level information in such an indirect fashion, incorporating knowledge on lanes into a network.

- **Architecture transfer and adaptation to a different domain**: early in the development a connection between pedestrians' trajectory prediction and vehicles' trajectory prediction has been

noticed. At the moment of writing this thesis, there was more work on *pedestrians' trajectory prediction in crowded spaces* done, than on a *vehicles' trajectory prediction*. The domains seemed similar enough to try to use a model from the well-researched area, on to the other. According to the expectations, the SS–LSTM model [41] performed well enough, to be further developed and adapted, which was done in this thesis.

- **Destination path planning network**: as the second part of the network had to generate a low–level trajectory given a rough destination, a network has to be created for that. What is interesting, due to the data–driven approach, the network values the generation of feasible paths more, than just the generation of paths that reach a goal. What it means, is that when an unreachable for some reason goal is fed to the network, it will generate a trajectory leading towards the goal, creating a realistic trajectory, but might not reach it. It is a very desired behavior, as the final output should always be a reasonable trajectory.

Outline of the thesis

This thesis describes the fundamental Machine Learning and Deep Learning elements used to create the HiPSS architecture in Appendix Chapter A and Chapter 2. Chapter A includes information on Feedforward networks, activation functions, cost functions, gradient descent and backpropagation. Chapter 2 includes information on Recurrent Neural Networks, Long Short–Term Memory Networks, its modes and Encoder–Decoder variation and detailed information on SS–LSTM architecture, which the HiPSS–LSTM is based on. Next, the thesis introduces and defines the problem tackled in this thesis – it describes a set of inputs and desired outputs that the system has to return. It also introduces metrics used to measure accuracy of the predictions in Chapter 3. The following chapter (Chapter 4) describes the created architecture in detail. It introduces a reasoning behind the choices in the architecture and describes the system in detail. Next, there is a description of the dataset used (Chapter 5). Then, experiments and results of the created architecture are discussed and compared to baselines (Chapter 6). The thesis ends with conclusions, limitations of the method and possible future work discussion in Chapter 7.

2

Background on the HiPSS architecture

This chapter introduces background information on the technical aspects of the work. It describes techniques, approaches and algorithms which underlie the algorithms chosen to create the system described in this work – the HiPSS–LSTM.

This work uses heavily the latest inventions in Artificial Intelligence and Deep Learning. The core elements of the created architecture are Recurrent Neural Networks (Long Short–Term Memory networks in particular). Basic information on the Machine Learning fundamentals, which this work is based on, such as Feedforward networks, activation functions, cost functions, gradient descent and backpropagation can be found in the Appendices Chapter A.

2.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [19] are a family of neural networks for processing sequential data. If RNNs were compared to feedforward networks, the main difference is that outputs of an RNN model are fed back into itself [16]. Moreover, a feedforward network has separate parameters for each input feature, whereas RNNs share the same weights across several time steps. Recurrent networks are generally described using the following formula (whereas Recurrent *Neural* Networks would add to it some extra architectural features, such as output layers):

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{2.1}$$

Equation (2.1) is recurrent, because the definition of $\boldsymbol{h}$ at time t refers back to the same definition at time $t-1$. The equation in a form of a graph is illustrated in Figure 2.1.

When a recurrent network is trained to predict a future sequence, based on some past representation, it typically uses the hidden state $\mathbf{h}^{(t)}$ to select relevant patterns from the input, up to t. It maps an arbitrary length sequence $(\mathbf{x}^t, \mathbf{x}^{t-1} \ldots \mathbf{x}^1)$ to a fixed–length vector $\mathbf{h}^t$. It could be interpreted that $\mathbf{h}^{(t)}$ stores previous information, to later use it to create an output. Moreover, the fact that the parameters are reused for every input, decreases the number of parameters to learn, therefore enables the model to be estimated with fewer examples. This also increases generalization capabilities.

Figure 2.2 illustrates the full graph of Recurrent Neural Network, with all its vital elements. It's a network in which the connections are created between hidden units (other variations include connections

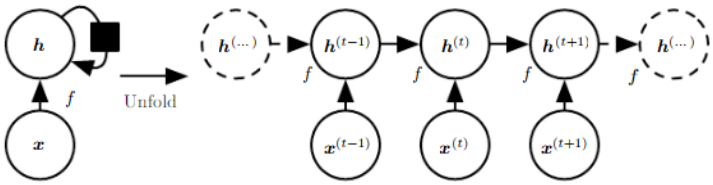


Figure 2.1: Two representations of a recurrent network with no outputs. It represents input **x** fed into state **h** that is passed forward through time. The left diagram is a folded representation, where the black square represents a single time step delay. The right diagram represents the same network, but unfolded. Its size depends on the input sequence length. Source: [16].

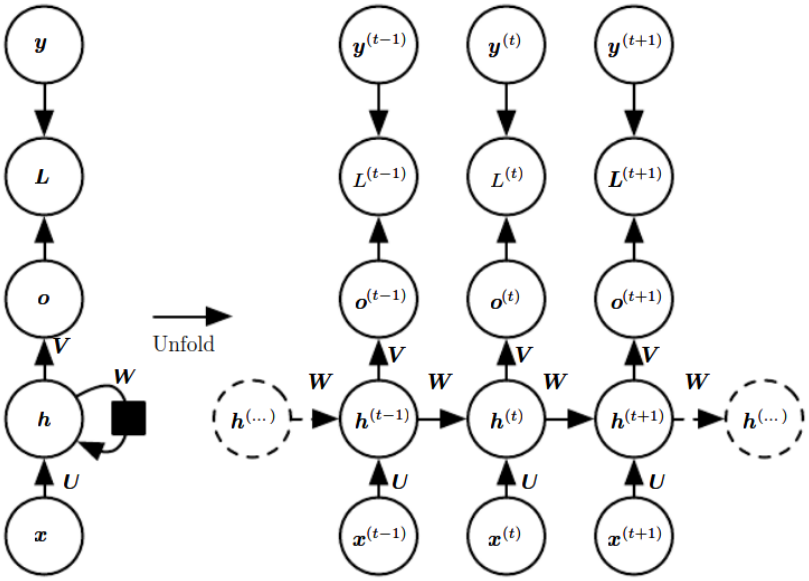


Figure 2.2: The full graph of Recurrent Neural Network containing all the vital elements. The graph represents the flow of information in a network. The network maps input sequences x to corresponding output values o. A loss L measures how much a value o differs from the corresponding training target y. The RNN has input to hidden connections parametrized by a weight matrix U, hidden-to-hidden recurrent connections parametrized by a weight matrix W, and hidden-to-output connections parametrized by a weight matrix V. Source: [16].

from output to a hidden unit or no intermediate outputs – just a single output at the end).

The problem of long–term dependencies – vanishing gradient
There is one very common problem with the RNNs. It is called a problem of long–term dependencies. This problem arises when pieces of relevant information in an input sequence are far apart. In case of sentence translation, if an RNN had a long sentence to translate, it would look like it cannot „memorize" the first words of the sentence. Some experiments [5] indicated, that the parameters $\boldsymbol{h}$ tend to settle in sub–optimal solutions, which take into account short–term dependencies and no long–term ones. What happens is that „the derivatives of the output at the time t with respect the unit activations at time 0 tend rapidly to 0 as t increases for most input values" [6]. In such a situation, simple gradient descent techniques (Section A.2) appear to be unfitting. This phenomenon is also called „vanishing gradient". This problem makes the RNNs unsuitable for a lot of tasks. A solution to the problem came with an invention of Long Short–Term Memory networks.

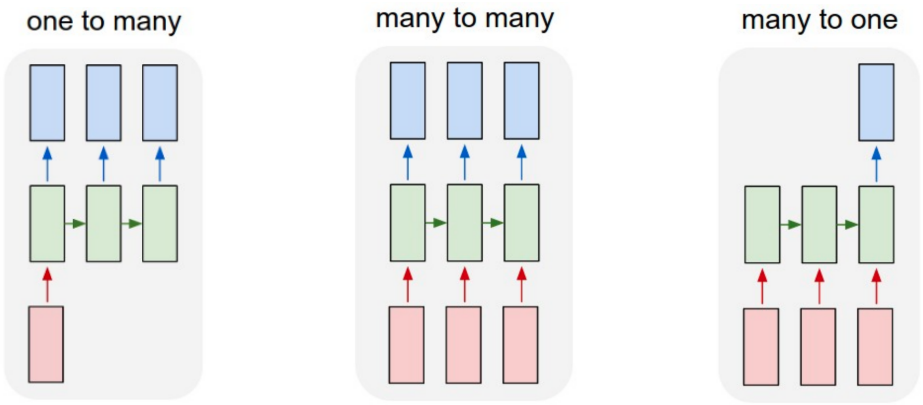
2.2. Long Short–Term Memory (LSTM) networks

LSTM networks originate from RNNs and were invented as a remedy for vanishing gradient problem. They outperform vanilla RNNs in many sequence modeling domains, such as text translation or video analysis. The details are presented in Section A.3, but as the LSTMs are used extensively in this work, the key points on LSTMs are presented here.

2.2.1. LSTM modes

There are a few variants of LSTMs to consider. The decision has to be made on „processing mode". There are three main modes to choose from:

- one to many - LSTM takes as an input one vector and output a sequence (many) vectors - e.g. image captioning takes an image and outputs a sentence describing the image (Figure 2.3a).
- many to many - LSTM takes as an input a sequence of vectors and outputs the same number of corresponding vectors - e.g. sentiment analysis where each word in a sentence is classified as expressing positive or negative sentiment (Figure 2.3b). It is a mode in which the LSTM returns a sequence, which is the same length as an input. Moreover, the first element of the sequence is



(a) LSTM „one to many" mode.

(b) LSTM „many to many" mode.

(c) LSTM „many to one" mode.

Figure 2.3: Different LSTM modes. **(a)** The LSTM takes one input (red rectangle), processes it (green rectangles) and outputs a sequence (blue rectangle). **(b)** The LSTM takes a sequence as an input and outputs a sequence with the same length as an input. Worth to note is that the first output cell is generated based only on the first sequence element. **(c)** The LSTM takes a sequence as an input and outputs only one vector, which was processed by all the LSTM's internal cells. Source: [20].

based only on the first element from the input sequence. Only the last output element is calculated based on all the previous elements. It is important to understand the implications of that. If the network was trained in a way such that the observed path is an input sequence, the resulting sequence would not resemble the predicted path. Only the last output can be assumed correct since it is generated based on all the input sequences, which is the full observed path. Even if the network is trained against a correctly predicted sequence, the first element rarely resembles the ground truth element.

- many to one - LSTM takes as an input a sequence of vectors and outputs one vector - (Figure 2.3c). In this mode, the network outputs only the last state, which is a result of operations made on a whole input sequence. In this case, the output state is influenced by all the input vectors before. To generate a full, n–seconds trajectory, the prediction would have to be made n–times in an iterative manner. The first iteration, based on observed trajectory, would predict a step in $t + 1$ and following iterations would take the previous prediction along with the parts of observed trajectory as input and predict further steps.

2.2.2. Encoder–Decoder LSTM architecture

„Many to many" mode comes with certain limitations. A length of output sequence has to be the same as the length of an input sequence. Therefore, the „Encoder–Decoder LSTM" setup has been invented [11]. The architecture was developed for natural language processing problems where it demonstrated state-of-the-art performance, specifically in the area of text translation. The key idea is that the encoder maps the input sequence to a fixed–length vector and that the decoder maps from that fixed–length vector to an output sequence, which can have a different sequence length than the input sequence. As presented in Figure 2.4, the network can be seen as a combination of two LSTM networks – „many to one", which encodes the input and produces „encoder vector", and the "one to many", which decodes the „encoder vector" and outputs the vector of a desired length. When trained jointly, the „encoder vector" can be interpreted as an encoded representation of an input vector. Such a setup is highly flexible and effective, which is why it is used extensively in this work.

2.2.3. SS-LSTM architecture

SS-LSTM architecture [41] has been a great inspiration for this work. It was created to predict pedestrians' trajectory in a dense crowd. SS-LSTM stands for Social-Scene-LSTM. It uses three different LSTMs to capture 3 different modalities – current target person's trajectory, social and scene information (Figure 2.5).

The authors identified these three modalities to be crucial in a future trajectory prediction. Along

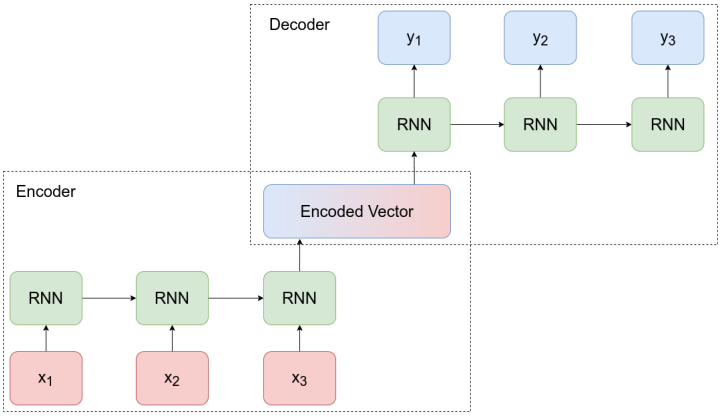


Figure 2.4: Scheme of an Encoder–Decoder network. This allows having an RNN network in a setup many–to–many, where all the inputs are taking into account when creating output. The encoder is set up in a many–to–one mode, and the decoder as a one–to–many. The output of the encoder is an input for the decoder.

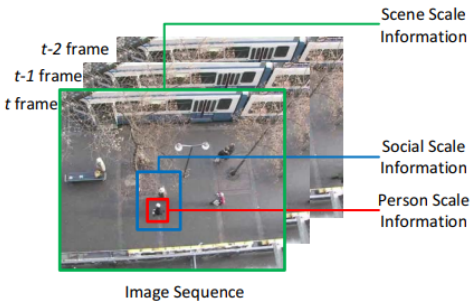


Figure 2.5: Scheme of an SS-LSTM network. The network incorporates information from 3 different scales for pedestrian trajectory prediction: person scale, which captures individual pedestrian's past trajectory information; social scale, which captures the neighborhood information of each pedestrian; and scene scale, which captures the scene layout features. Source: [41].

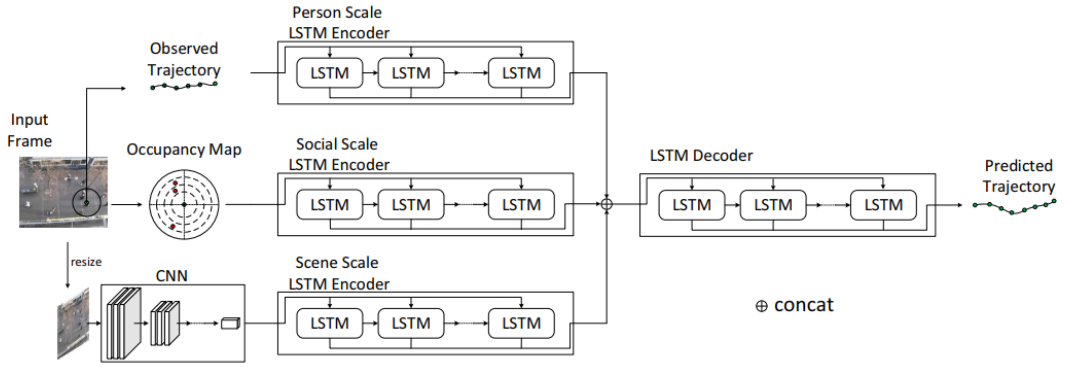


Figure 2.6: Architecture of the SS-LSTM network, depicting three separate input modalities – observed trajectory, occupancy map and scene input. Observed trajectory and occupancy map are encoded with a single LSTM each, and the scene is first processed with a CNN network, and then with an LSTM, to create an encoded vector. The three encoded modalities are then concatenated to an „encoder vector", which is then decoded to form a predicted trajectory. Source: [41].

Methods	ETHhotel		ETHuniv		UCYuniv		zara01		zara02		average	
	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE	ADE	FDE
Linear	0.137	0.261	0.143	0.298	0.099	0.197	0.141	0.264	0.144	0.268	0.133	0.257
LSTM	0.128	0.216	0.160	0.263	0.151	0.267	0.163	0.302	0.134	0.243	0.147	0.258
S-LSTM-g	0.076	0.125	0.195	0.366	0.196	0.235	0.079	0.109	0.072	0.120	0.124	0.169
S-LSTM-c	0.065	0.085	0.165	0.313	0.122	0.185	0.051	0.079	0.074	0.117	0.095	0.156
S-LSTM-l	**0.053**	**0.079**	0.149	0.295	0.099	0.159	0.052	**0.078**	0.069	0.104	0.084	0.143
SS-LSTM-g	0.081	0.129	0.170	0.291	0.108	0.157	0.057	0.089	0.072	0.109	0.097	0.155
SS-LSTM-c	0.066	0.101	0.182	0.328	0.098	0.144	0.051	0.081	0.065	0.107	0.092	0.152
SS-LSTM-l	0.070	0.123	**0.095**	**0.235**	**0.081**	**0.131**	**0.050**	0.084	**0.054**	**0.091**	**0.070**	**0.133**

Figure 2.7: SS-LSTM performance comparison to other methods – Linear model, Vanilla LSTM [18] and S-LSTM [27]. *–g. *–c and *–l describe different circular occupancy map modes – grid maps, circle maps and log maps. The ADE is an average displacement error and the FDE is a final displacement error (details in Section 3.2.1). Source: [41].

with an observed trajectory, there has to be the information of a movement of neighboring pedestrians, which influences the target pedestrian trajectory. Along with that, a scene level, which encodes scene layout adds information of „walkable" areas, obstacles or other important elements. Having these three modalities processed together, a trajectory prediction can be made.

The architecture of the work relies on a variation of an Encoder–Decoder network. First, each of the levels is processed separately: the observed trajectory is processed with an LSTM; the social scale is represented as a circular occupancy map, which is processed with another LSTM; the scene scale is processed with a Convolutional Neural Network (CNN) and an LSTM. The CNN is there to extract meaningful features and lower the image dimensionality. All three outputs are then concatenated, creating an „encoder vector". The vector is processed with a decoding LSTM which outputs an n–step predicted trajectory (Figure 2.6).

The setup which was tested by the authors involved 8 consecutive frames as an observation and 4,8,12 and 16 frames as a prediction horizon, which corresponds to 0.8, 1.6, 2.5 and 3.2 seconds respectively. The performance was measured against the previous state-of-the-art technique, Scene-LSTM network [27], and proved to be better in two of the tested datasets – UCY [26] and The Town Centre dataset [4]. The performance was worse for the ETH dataset [29]. Detailed results are presented in Figure 2.7.

The SS–LSTM architecture takes as inputs all the elements that are also crucial for the trajectory prediction task in our thesis work, as described in Section 3.1.1. Therefore, our work uses the design decisions from the SS-LSTM as underlying principles. It also uses three separate inputs, representing person, scene and social level, encoding them to one „encoder vector" and later decoding them in the decoder part to a final trajectory. The changes are described in detail in the following chapters.

3

Problem description

This chapter analyzes which modalities should be used as an input for the prediction system and how the output of both high– and low– level prediction network should be represented. Based on previous works, we make a decision on which modalities to use. Moreover, there are a few metrics that can be used to asses an accuracy of a prediction system. This chapter introduces and defines the ones that will be used for the assessment of this system.

3.1. System's requirements

In the previous chapter we already mentioned the main input modalities. Let's review now in more detail what we want from those input modalities, and what is realistic in terms of final applications and available datasets. What is more, the two–step architecture introduces an intermediate representation – the output of the high–level network, which is an input for the low–level network. This section defines it as well.

3.1.1. Input modalities

To reliably predict the path of a target road user, a system must have some amount of knowledge about the target road user and about a context in which it operates. There are some modalities which the system could take into account.

There is one thing common in all the related works' approaches – one of the input modalities is always an observed for a few seconds trajectory. This is the most obvious and crucial information that needs to be taken into account. In all the works (Section 1.3) it is represented as a sequence of absolute or relative positions. In most of the newest approaches, an observation window is set to 2 seconds [10, 25, 37], as this proved to be enough to reason about a general „intention" of a driver, and is realistic in terms of tracking – when a vehicle appears in a range of a modern vehicle's sensors, it is usually far away, so it can be tracked for 2 seconds, without introducing a direct danger. Moreover, the biggest dataset available [10] (more in Section 5.2) sets the task as 2 seconds of observation and 3 seconds of prediction, as this is approximately the longest it takes to make a full maneuver by a driver. The data is collected at a frequency of 10Hz, resulting in 20 steps of observation and 30 steps to predict. The same data collection frequency is used in the aforementioned works.

The other often repeating input modality is a piece of information on a surrounding context. Context is a piece of information about target vehicle position, surrounding, drivable areas, obstacles, lane–level information (including lane's location, direction and type), traffic signs, traffic lights and so on. These are pieces of information, which can be easily spotted, fused and interpreted together by a human driver. Advanced self–driving car companies are using HD maps, which include most of those modalities in a digital form, to get as much context information as possible. The HD maps are great, but they come with limitations. First of all, the areas covered by HD maps are very limited. The commercial maps are covering only a few cities accurately. Secondly, the environment, in the long run, is dynamic – it constantly changes, new work roads are being conducted, new roads created, so the map has to be constantly updated. This is still an active research topic, to keep them automatically up to date.

This work focuses on creating the most robust and universal system as possible. Given these constraints, this architecture cannot rely on *all* the HD map modalities during an interference. It should be able to yield valid predictions using data acquired mostly by onboard sensors and use detailed map information as little as possible. On the other hand, most related work confirms that basic geometric information describing the road infrastructure and/or drivable areas in some detail is necessary.

A trade-off solution could be to represent only drivable areas around the target vehicle, without additional modalities such as traffic signs, lanes directions, etc. The system must know which areas are allowed to be driven on and which are not – areas such as pavements, obstacles or buildings. These are underlying constraints for the algorithm. Therefore, this work considers drivable areas representation (scene), as an equally important input modality as the observed trajectory.

Moreover, this work takes into account a social context, as all the vehicles interact with each other and influence each other decisions. The social interactions representation is taken directly from the SS–LSTM architecture [41], which performed well in a crowded scene–setting. Interactions are represented in terms of distance and an absolute direction from a target vehicle to other road users. Out of three options (rectangular–map, circular–map and circular–log–map), the chosen representation has been „circular–map".

3.1.2. Output representation

The most popular and direct method of representing a trajectory is a sequence of coordinates. The consecutive coordinates can be in an absolute coordinate system (latitudes and longitudes), relative to the first position or relative to some other point (e.g. the last observed position). One requirement is that those positions should be easily transferable to meters, to make the performance understandable for humans and easily compared across different works. This work chose to represent the positions relative to the last observed points. Moreover, the positions are normalized to a scene representation, so that the most south–west corner of the scene has always coordinates of $(0, 0)$ and the north–east has coordinates $(1, 1)$. Moreover, the last observed position is always in $(0.5, 0.5)$. The size of the scene is chosen to fit the longest trajectories, so that the network always predicts positions from a range of $[0, 1]$.

The other part of the output should be some measure of the likelihood of the path being chosen. The likelihood can serve as a sorting argument, where a system would rank the generated paths.

With the 2 seconds as observation time, the prediction time window used in other works is usually between 1 and 4 seconds. Chang et al. [10] claim that every important maneuver, like a turn on an intersection, which takes the longest, takes usually no more than 5 seconds. Previous works also noticed a high accuracy loss [25], with an increase of the prediction window. Therefore, the prediction window has been decided to be 3 seconds.

3.1.3. Other requirements

The overall requirement for the network is that it has to output a reliable and realistic set of predicted trajectories and reasonably assess their probability. This will be measured according to Section 3.2. Additionally, the trained network should output predictions close to real–time on a decent, commercially available computer with a GPU.

3.2. Performance Metrics

There are a few metrics, which are considered „standard" and are used in a few modern publications, including this work. Additionally, this work introduces a specific metric to measure the high–level network accuracy alone.

3.2.1. Standard Metrics

The standard metrics applied to this problem are Average Displacement Error (ADE) and Final Displacement Error (FDE). These metrics have been used in [10, 14, 25, 36]. The ADE is defined as the average L2 (Euclidean) distance between the ground–truth and predicted trajectories, over all vehicles and all time steps. The FDE is an average L2 distance between the last step of ground truth and the last step of prediction. It is important to look at both metrics at the same time, as they complement each other.

Most of the papers focus on a single trajectory prediction, where a set of those two metrics is enough

to describe a quality of predictions. In the case of this work, where there is always a set of predictions, each with a predicted probability of occurring, those metrics cannot be applied directly. One work which also faced a similar issue [25] introduced an oracle metric, where out of the top–K of predictions, the only one which is taken into account, is the one with the lowest ADE and FDE error in comparison to the ground truth. In the work, the K has been set to 50. The underlying assumption is that it does not matter how high the predicted likelihood associated with the prediction is, as long as the correct prediction is identified.

The metrics have been refined by authors of the Argoverse dataset [10]. Along with the dataset they created a benchmark and defined a few metrics to evaluate submissions. The leaderboard is sorted based on an oracle FDE (in this case called minimum FDE (minFDE), which is more coherent) with K set to 6. This is lower than in the [25], so it encourages to have only a few good predictions, with a good likelihood estimation. Other metrics used are minimum ADE (minADE), which refers to ADE of the trajectory which has the lowest FDE (and not necessarily the lowest ADE), since the evaluation should take into account the single–best forecast.

Other metrics proposed by Chang et al. are Drivable Area Compliance (DAC) and Miss Rate (MR). DAC measures how many of the generated trajectories are exiting at any point drivable area. If m out of n trajectories exit the drivable area, the score is $(n - m)/n$. MR tells in a binary way if a destination point is missed. If there are n samples and m of them had the last coordinate of their best trajectory more than 2.0 meters away from ground truth, then the miss rate is m/n.

These metrics measure well the final performance of the network. They take into account the multi-modal nature of predictions by evaluating all of them, but considering only the best one. The likelihood estimation is only used indirectly. It is not explicitly used in a metric, but it is used for ranking the predictions. In the end, it does not matter if a system assigned a low likelihood to a trajectory, as long as it identified it as one of K (which is relatively low) most likely trajectories.

3.2.2. High–level network performance metrics

The metrics listed in Section 3.2.1 are good for measuring the overall performance. In the case of this work, it is also useful to measure the performance of the network's first stage – high–level prediction. The most important measure here is a High–Level Miss Rate (HLMR) indicating if the ground truth grid cell is within a top–K predictions. If there is m out of n samples where at least one out of K best grid cells is chosen correctly, then the score is m/n. This could be used to evaluate how many high–level goals are covered correctly. This could indicate how many best trajectories (K) should be taken into account, to minimize the minFDE error.

4

Hierarchical Probabilistic SS-LSTM (HiPSS-LSTM)

In this chapter, the description of the developed architecture is presented. We describe general reasoning and motivation behind the architectural choices followed by detailed information on design choices, architecture and training process.

4.1. Reasoning and architecture overview

Based on the Research Question set in Section 1.4, the prediction system must fulfill two main requirements: it has to predict a set of feasible and diverse trajectories and assign a likelihood to each of them.

To fulfill the first requirement, there are a few approaches present in the literature. One way is to have an architecture which can output multimodal results by design, e.g. using a set of ConvLSTMs or MDNs [37, 43] or by introducing a latent variable to account for the ambiguity of the future (as in [25]), but there the generated predictions had to be ranked and refined). Another approach is to have an architecture that splits the problem into two stages – first, which would determine a set of destinations and second, which would generate trajectories from the last observed position towards each of the destinations [30]. In this work, it was decided to focus on the latter approach. The main reason to divide the task this way is that even though the high–level and low–level prediction problems are related, they are still fundamentally different. Even though they might use similar inputs, the mapping function the network has to learn, will be completely different. Therefore, there must be two networks, trained separately.

Estimation of a goal can be formulated as assigning a probability to a corresponding regions over a scene. This can be done using MDNs [30] or, as in this work, the problem can be treated as a multi–class classification problem, where a network is trained to assign a probability to a few categories (which represent plausible destinations, in case of this work). This approach has been very successful in multi-class object detection tasks.

In the second stage, the low–level network can be trained as a goal–oriented path planning task, taking into account social interactions and road infrastructure constraints. Some approaches use IRL [30], but these are too slow to be executed in real–time. This approach modified the SS–LSTM network [41] for this task, which can model both social forces and output trajectories compliant with a scene layout.

In conclusion, the network consists of two stages:

1. a stage that estimates roughly all likely high–level destinations, along with a probability assigned to each of them;

2. a stage that generates a low–level trajectory, starting from the last observed point to each of the estimated destinations.

4.2. High–level prediction – detailed information

The task of the high–level prediction network is to generate a few final destination candidates. This suggests outputting some form of a probability distribution over a scene. There were two approaches considered and tested:

- predicting a set of Gaussian functions using Mixture Density Networks ,
- discretizing a scene to a fixed–size square grid and assigning a probability to each of the grid cells.

Predicting a Mixture of Gaussians was done by [43], where it was applied to a very constrained problem (one type of intersection only), and in [30]. Even though both authors reported satisfactory results for their dataset, the results were unsatisfactory when tested for the HiPSS–LSTM, due to the input complexity and scene variability. Moreover, authors of [30] reported a very high accuracy decrease, with bigger time–windows for prediction, making it not very robust.

The second approach, discretizing a scene to a fixed–size square grid, is a better approach for the task. The network would be trained to assign a probability to every grid cell.

Worth to note, is that the grid should be of an odd size (e.g. 5x5), because the input data is processed in a way, that the last observed point is always in the middle. If the grid height (or width) was even, this point would lay exactly on an intersection of 4 grid cells and the next predicted positions would be more likely to follow along the border of two grid cells. As the position measurements are usually noisy, the ground truth data could often switch from one grid cell to another. Making it an odd number, the training process deals with some more consistent ground–truths for the first steps.

The network's task can be compared to a multiclass classification problem, where a network assigns a probability to every single class. A class with the highest probability is the final class. In this work's case, the classes are different grid cells. A top–K most likely destinations will be taken into account and refined to a low–level trajectory in the next step. The resolution of a grid can be as small as 5 by 5, whereas upper bound is theoretically unlimited. Different grid sizes are tested in Chapter 6.

4.2.1. Architecture

The scheme of the architecture is shown in Figure 4.1. It is based on the SS-LSTM architecture (described in Section 2.2.3), which is an encoder–decoder network (Section 2.2.2). This means, that inputs are encoded by one part of the network and later decoded by another part.

The three input modalities are on top of the Figure and their format is described in Section 4.4. Social representation and observed trajectory are supplied to its encoders, which are LSTM networks. The scene representation is already concise and does not have to be further encoded. It is treated as an encoded vector.

The encoded vectors are then concatenated and fed into an LSTM decoder. The decoder is expected to return only one value, the final destination, so in theory, it could be set up in the many–to–one mode (Section 2.2.1). It turns out the performance is better when the LSTM is in many–to–many mode. This is due to a fact, that in the many–to–one mode, during backpropagation, there is only one sequence element used for gradient–descent, whereas in many–to–many, there is as many sequence elements as output steps, which effectively trains the LSTM better. Therefore, the LSTM is trained against n–steps grid cells (number of positions in a ground–truth trajectory).

As the last refinement step, the output of LSTM decoder is concatenated with a scene input and fed to a feed–forward network. This step forces the network to refine the predictions for the last time, to make them more consistent with drivable grid cells. The last layer, which is a Dense layer, has a Softmax activation function (described in Section A.1.1) which distributes a probability to each element of a flatten grid.

Only at the very end of the network, the first n-1 predictions are discarded and only the last element from the sequence is fed to the next, low–level network.

4.2.2. Training process

The training of the high–level prediction network is divided into two stages. Firstly, the network is trained on an artificially generated dataset (described in Section 5.2.1). These are trajectories, which are perfectly aligned with lanes on the road. They contain no noise, are smooth and give the network knowledge of „perfect trajectories". Secondly, the network is trained on the original dataset.

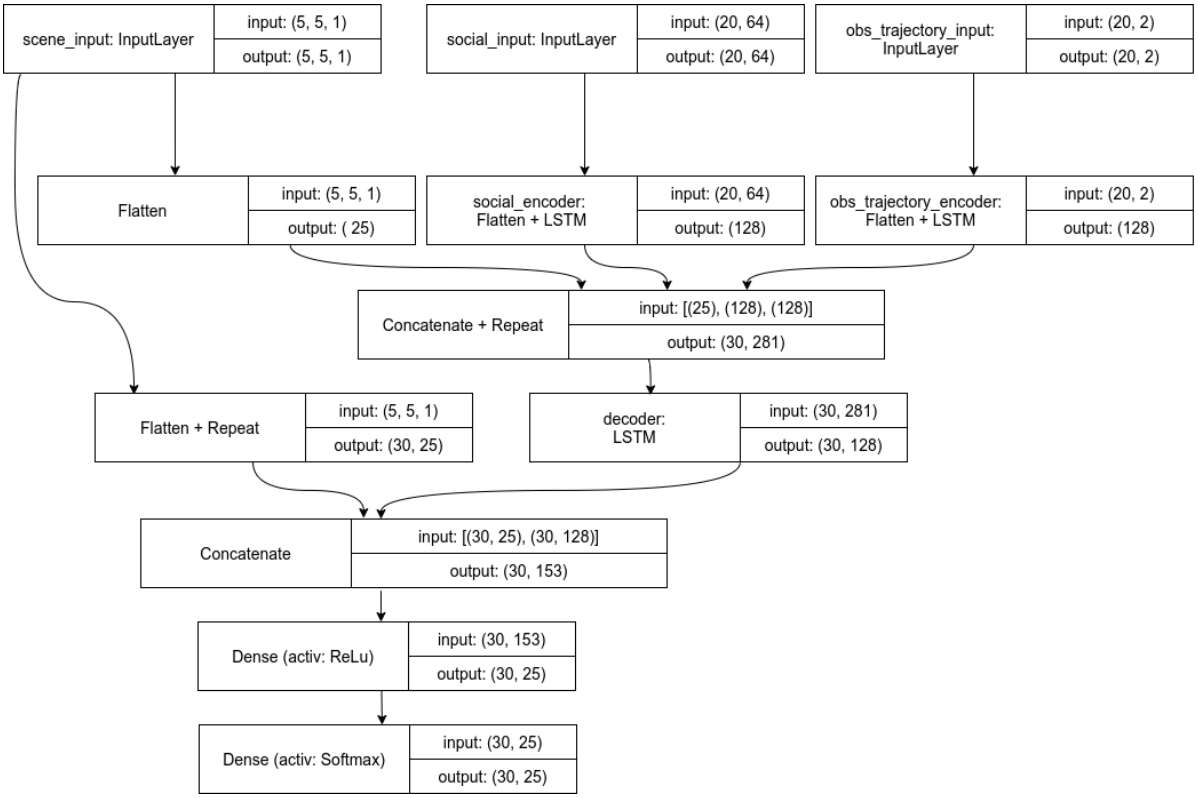


Figure 4.1: High–level prediction network architecture, with inputs on top, and outputs on the bottom. Each block symbolizes a layer or collection of similar layers. Each of them has an input and output shape indication. This graph assumes the high–level grid resolution to be 5x5. Inputs to the network are: scene representation, social representation and observed trajectory. Based on those inputs, the network outputs a heat–map, representing probabilities of where the target object will end up. The resolution of the scene input representation matches a resolution of the output heat–map. The output has shape of 30 by 25, because it outputs 30, 5x5 heatmaps, one for each predicted step.

Introducing the artificial dataset and two–staged training overcomes a problem of a biased dataset. The available dataset is biased in terms of types of trajectories – most of them, are trajectories going straight ahead. A network trained on this dataset alone, would be biased towards predicting straight trajectories more often.

Therefore, a different approach was taken. It was inspired by Curriculum Learning [7]. The approach describes a training process, where a network is trained on easier examples first, and only then on harder ones. In a typical setting, a separation between easy and hard samples would be done either by a simple classifier or by estimating how noisy a sample is. Since real trajectories are always subjected to contain some level of noise (due to the data collection process, which is sensitive to sensor imperfections, wrong distance estimation or occlusions), this subset is treated as the more difficult one. On the other, the artificial dataset is made out of lane centerlines, so it contains substantially less noisy data. It is treated as the easier dataset.

After training the network on a training subset of the artificial dataset, it performs well on the out–of–sample validation dataset, predicting all the possible and reasonable destinations well. It also performs quite well on many real trajectories by now. But it does not reflect real–world probabilities of those different directions being chosen by road users. It does not take into account small variations in the observed trajectory, to determine a probability of a certain destination. To make it more adjusted to the real–world trajectories, as they contain more subtle clues, more noise and are often slightly off the „perfect trajectory", the second training step is applied – the model is retrained on a dataset with real trajectories. Starting with the weights that were learned in the previous phase, the model now starts with a lower error rate, as it would when trained from scratch, and achieves higher generalizability.

As the network outputs a vector that corresponds to the input scene grid representation, the ground–truths supplied to the network have to be in the grid format as well. Therefore, the ground–truth trajectory is discretized into the grid representation, where each point is translated to a vector of a shape corresponding to the flatten grid shape. The vector contains only zeros, except for the element corresponding to the point – this one has a value of 1. Moreover, to reduce memory usage, only indexes of the ones are stored (sparse representation). The discretization process is presented in Figure 4.2. This process happens for each ground–truth during batch loading process. This is a preferred way of doing it over storing it as a separate dataset, because it allows for more flexibility during experimentation.

A cost function applied to this training process is the sparse–cross–entropy cost function (Section A.2). Its main advantage is that it assigns a probability to each of the classes from a dataset. It is usually used in models with multi–class classification, where the classes with a probability above some threshold are used as an output. In our case, the assigned probability can be interpreted as a likelihood for the given destination area to be an actual destination.

The network is trained for 200 epochs, which is when a value of the cost function already converged. The network is trained with gradient descent variation called Adam, with a decaying learning rate starting at 10^{-3}.

For the pretraining, there are 152373 trajectories, which are randomly divided into a train and validation dataset. The training dataset contains 100298 trajectories, whereas the validation 25075 trajectories. The training is executed on a Titan XP GPU. It is trained for 200 epochs (which is when it is converged) and it takes approximately 1 day. In the second phase, training the real–trajectories dataset is divided upfront by the dataset supplier [10] and it contains 205942 train trajectories and 39472 validation trajectories. It converges after 100 epochs and the training takes 1 day on the same GPU.

4.3. Low–level prediction network – detailed information

The low–level architecture's task is to do a destination path planning. Starting with the last observed position, it has to generate a step–by–step trajectory, which leads towards the final grid cell.

4.3.1. Architecture

This is the second part of the full architecture. The scheme of it is shown in Figure 4.3. It is also based on the SS–LSTM architecture. This network's task is to generate a low–level trajectory in a form of n–coordinates, which represent positions over time. The network takes 4 inputs: social representation, observed trajectory, scene representation and a final grid cell. The final grid cell is provided by the high–level network. Observed trajectory and social representation are exactly in the same form, as for

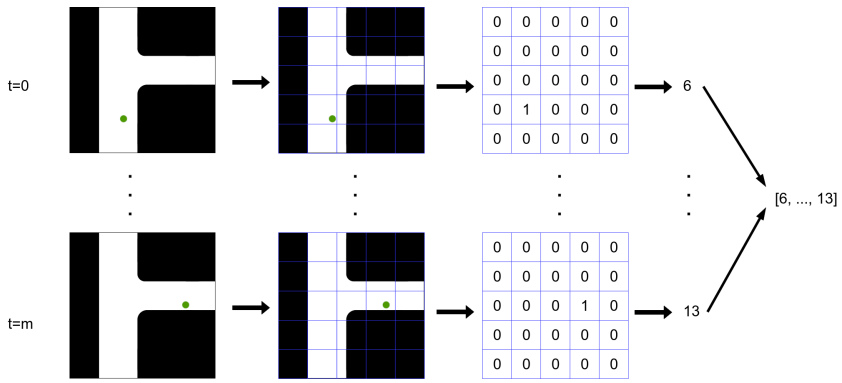


Figure 4.2: Discretization scheme for the ground–truth data. The process discretizes each trajectory point one by one. It finds to which grid cell the datapoint belongs to, and creates one–hot encoded matrix, which is flatten and transformed to a sparse encoding. Created sparse encoding are concatenated together, representing the full ground–truth trajectory on a grid. This graph assumes the high–level grid resolution to be 5x5.

the high–level network. Scene representation differs. The high–level network required a low–resolution representation of drivable areas, whereas this network requires higher–quality raster. In this work, a quality of 72 by 72 pixels was chosen and proved to yield good results. Final grid cell provides the network with an information about an estimated destination. This input is crucial, as without it, it is impossible to firmly predict one trajectory. Due to this input, the network can generate a trajectory that is coherent and guides to the appointed destination.

Each input, except for scene representation, is then processed by an encoder separately. The scene is processed by several CNNs. The CNNs extract meaningful features for the network in the training process. This part of architecture resembles the ConvNet architecture [24], which has proven to work well for the image recognition task. The idea is that it learns to extract and represent in a dimensionally lower space which areas are drivable and which are not. There are three sets of CNNs, followed by Pooling Layers and Batch Normalization Layers. The CNNs have filters of size 32, 64 and 96 with a kernel of size (3, 3). Max Pooling Layers have a pooling size of (2, 2). Following the groups, there is a Dropout layer with a ratio of 0.2, which blocks a random 20% of inputs to go through, which has been proven to decrease overfitting.

Observed trajectory and social representation, as they are sequential, are encoded by an LSTM. The final grid cell is a one–hot encoded vector, which proved to be the best representation. Another option was to represent it as one number, representing a flatten index in a fixed–size vector (dense representation), but this reported a worse performance. This input does not have to be encoded, as it is already in a concise form, usable by the network. Those encoded vectors are then concatenated together and fed into a decoder LSTM. The decoder works in the „many–to–many" mode and this time all the outputs are important, as they represent a trajectory over time. A final feedforward network shapes the output to resemble a trajectory representation – 30 consecutive coordinates, on a two–dimensional Cartesian space. The network is trained with gradient descent variation called Adam, with a decaying learning rate starting at 10^{-3}.

4.3.2. Cost function

There have been a few cost–functions tried out. A few setups have been tested:

- Mean Squared Error (MSE) alone,
- a sum of MSE and a custom function *scene loss*,
- a custom function *destination loss*

MSE

MSE is an average L2 distance between the predictions $\hat{y}$ and ground–truth y trajectories:

$$mse = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \tag{4.1}$$

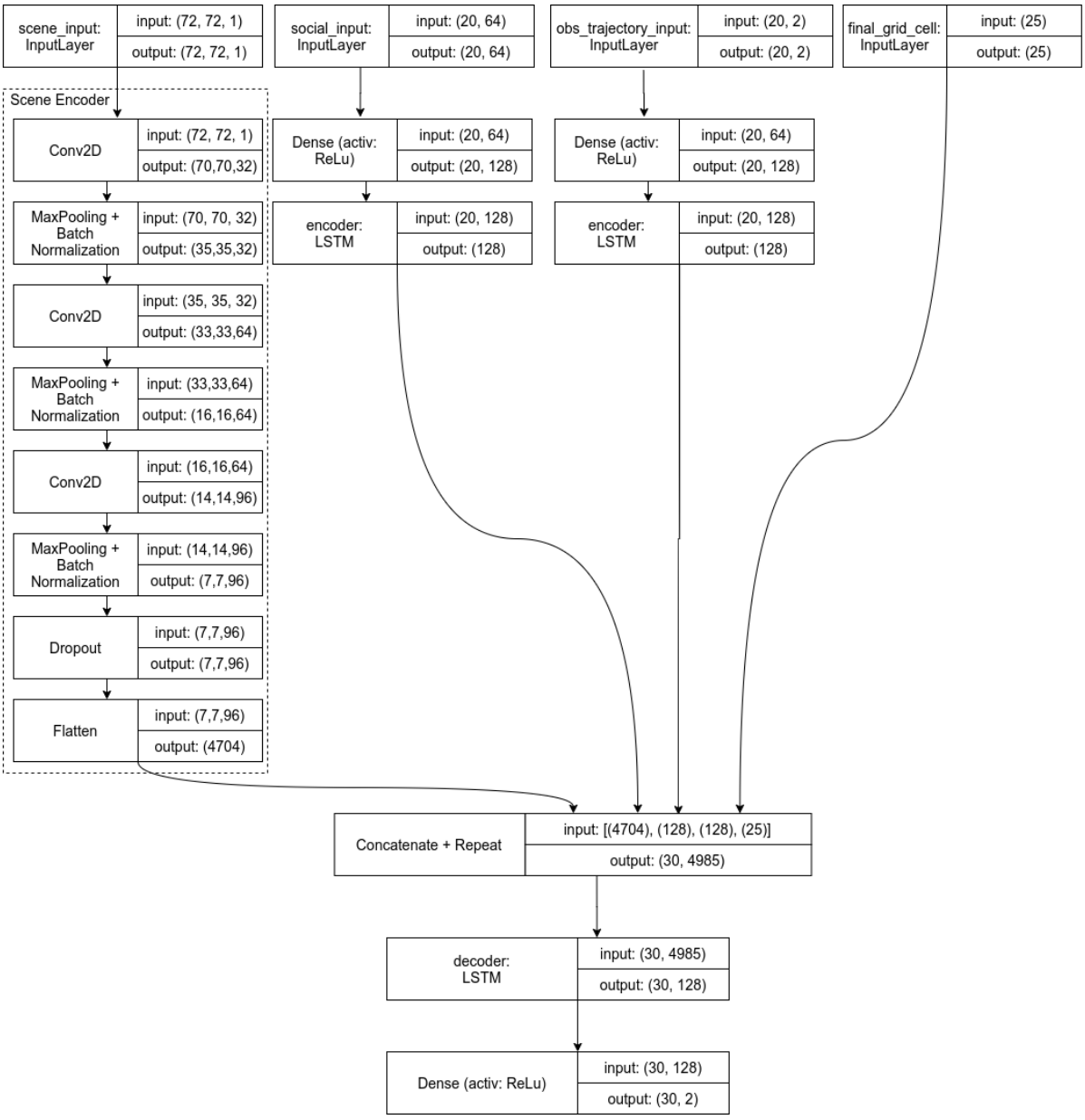


Figure 4.3: Low–level prediction network architecture, with inputs on top, and outputs on the bottom. Each block symbolizes a layer or collection of similar layers. Each of them has an input and output shape indication. This graph assumes the high–level grid resolution to be 5x5. Inputs to the network are: scene representation, social representation, observed trajectory and destination area (final grid cell). Based on those inputs, the network predicts a low–level trajectory.

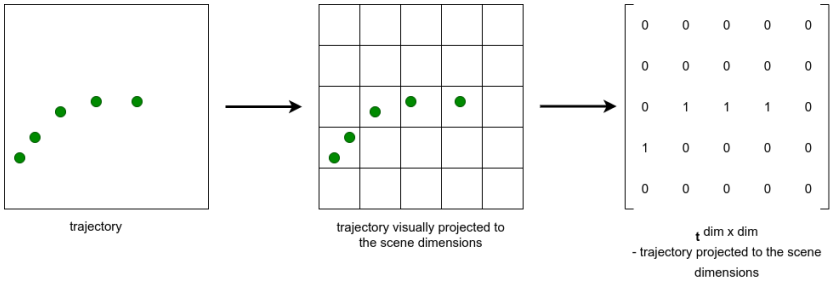


Figure 4.4: Visualization of the trajectory projection process.

Scene loss
Scene error calculates a number indicating to what extent a generated path is within boundaries of drivable areas. The function penalizes trajectories that are outside of drivable areas, encouraging the network to output trajectories complying with a scene input. Scene loss is always applied along with MSE as a sum of both.

$$scene\_error = (vec(\mathbf{s}^{dim \times dim}) \cdot vec(\mathbf{t}^{dim \times dim}))/\sum_{i=1}^{n} \mathbf{t}_i^{dim \times dim} \tag{4.2}$$

where $vec(\cdot)$ is a vectorization operation, $dim \times dim$ is an input scene resolution, **s** is a matrix representing scene input, which is normalized to have values in range $[0; 1]$, and **t** is a projected trajectory to the scene dimensions. The projection process is visualized in Figure 4.4. $\sum_{i=1}^{n} \mathbf{t}_i^{dim \times dim}$ is a normalization term.

$$scene\_loss = mse + scene\_error$$

Value of the $scene\_error$ is scaled, to match the order of magnitude of the mse. Scene loss is a sum of mse and $scene\_error$.

Destination loss
Destination loss is a function, which calculates a distance of a last predicted point to a center of a final grid cell. The function penalizes trajectories, which final point is far away from a center of a final grid cell. What is important, as the network's output is a vector, which represents n–steps in time, so the loss function must also generate a vector of the same shape. Therefore the first n-1 steps are evaluated with the MSE between predicted and ground truth coordinates, and the last predicted coordinates are compared to a center of a final grid cell.

$$destination\_loss = mse(\mathbf{y}_0, \hat{\mathbf{y}}_0) + \ldots + mse(\mathbf{y}_{n-1}, \hat{\mathbf{y}}_{n-1}) + mse(\mathbf{s}_n, \hat{\mathbf{y}}_n) \tag{4.3}$$

where $\hat{\mathbf{y}}$ are predicted coordinates, **y** is a ground truth trajectory, $\mathbf{s}_n$ is a center of a final grid cell and $\hat{\mathbf{y}}_n$ is the last predicted point.

4.3.3. Training
This network, as the high–level network, is pretrained on the artificial trajectories. The motivation behind the pretraining step is similar to the one in high–level network. Firstly, it follows the Curriculum Learning approach, which is beneficial for the training process. Secondly, the network will gain knowledge on „perfect" trajectories, how they keep the lanes, and what shape do they have. The pretraining is done on the generated dataset with artificial trajectories, for 80 epochs, after which the network converges. Here is the same setup – *the same* 100298 trajectories are used for training and 25075 for validation. Training with the real trajectories is done starting with the learned weights. This training is executed with the same dataset of real trajectories, so as in the high–level network, there are *the same* 100298 training trajectories and 25075 for validation. It is trained for 100 epochs, which takes approximately 1 day as well.

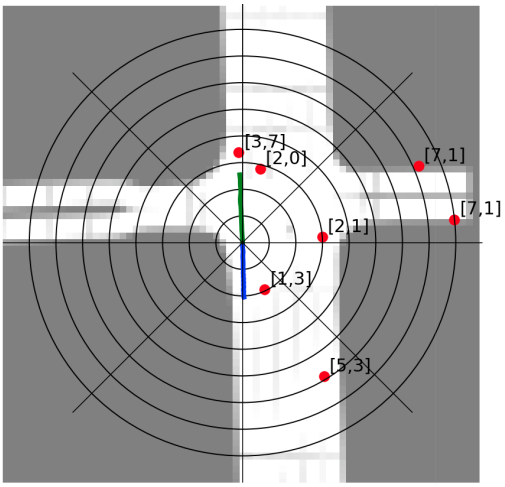
4.4. Input representation – implementation details
There are three main inputs for the network – observed trajectory, scene and social representation.

4.4.1. Observed trajectory

The observed trajectory is represented as 20 consecutive points on a top–down view map. These points represent target vehicle locations collected at 10Hz for seconds. This number of data points has been used in many works before [10, 14, 25, 36, 41] and proves to be representative to predict a trajectory in the next 3 seconds. Each observed trajectory has been normalized – every point has a value from a $< 0, 1 >$ range. Moreover, the last observed step is always in the center (coordinates of $(0.5, 0.5)$). This representation helps the network to generalize to different situations since there is less variety in the training examples. Moreover, since the scene representation is static, the observed trajectory is encoded as coordinates relative to a static scene (binary representations of drivable and non–drivable areas).

4.4.2. Social forces representation

Social forces are represented in the same way as in [41]. A method based on a circular occupancy map has been implemented. An occupancy map is built by partitioning the pedestrian's neighborhood into non-overlapping cells, on a circular grid. The circular–shaped grid, compared to the traditional rectangular shape neighborhood setting, is more appropriate because the significance of social influence is mainly governed by the distance between the i^{th} pedestrian and the neighboring pedestrians. The circular–map discretizes linearly each distance to one of 8 values and direction to one of 8 values, creating a circular grid of size 8 by 8. Subsequently, the number of vehicles in each cell is counted and saved. This is then represented as an array with 64 elements, where each element represents the number of vehicles in a given cell. Such an array is generated for every observed step, to capture the dynamics of other road users. An example of such social input is presented in Figure 4.5, where a circular map is presented visually. A corresponding social input matrix is presented in eq. (4.4).



$$\begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 2. & 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix} \quad (4.4)$$

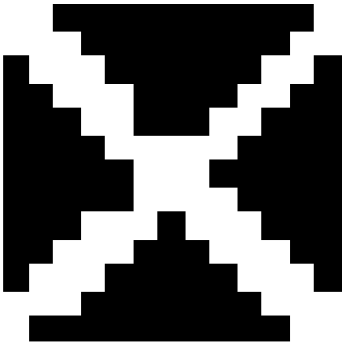
Figure 4.5: An example of a social input representation. The green line is an observed trajectory, blue is ground–truth trajectory, red dots are other vehicles in the last observed timestamp (in the twentieth step). Circles and cells created with straight lines represent cells in a corresponding scene matrix. The numbers next to the red circles are indexes in the matrix. The scene matrix corresponding to this timestamp is presented in eq. (4.4).

4.4.3. Scene representation

As described before, the scene is represented differently for the high– and low–level networks. In this section we elaborate and add more details on this modality.

For the low–level network, the scene representation is a square cut–out of a full neighborhood map (Figure 4.6a). It is a binary mask of a resolution 72 by 72 pixels, which proved to be a good trade–off between the size and quality. A bigger resolution would dramatically increase the network size (more neurons needed to process the mask) and lower resolution would decrease the map quality. The parameter which had to be chosen was a scale of the map. It has been chosen in a way, that the longest trajectory from a training set still fits in the mask. An example of an observed trajectory combined with scene representation is presented in Figure 4.7. As shown there, the input can sometimes be noisy, which reflects real–life situations.

(a) Input scene representation for the low–level network.



(b) Input scene representation for the high–level network.

Figure 4.6: The same scene input example, in a format for (a) low–level network and (b) high–level network

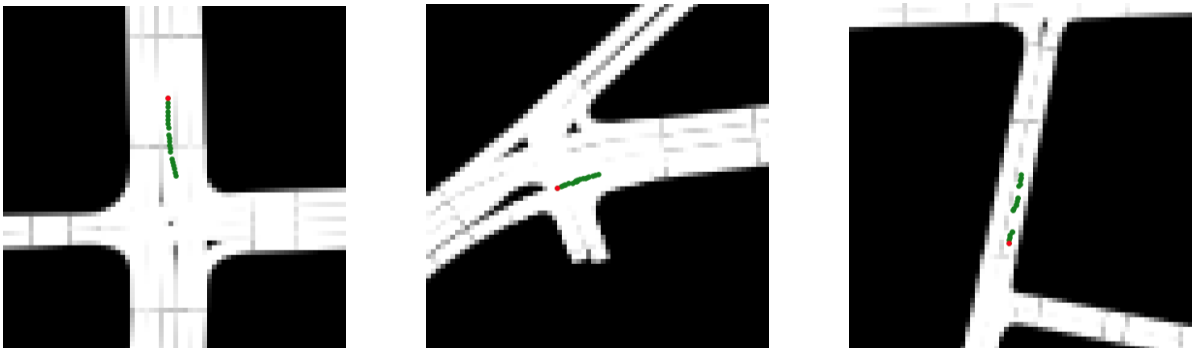


Figure 4.7: Examples of two input modalities – scene and observed trajectory, merged into one visualization. The red dot represents the first data point of the observed path. (a) and (b) are examples of typical input data, and (c) is a noisy input, which reflects real–life measurements, which are often noisy.

For the high–level network, the scene is discretized in a preprocessing phase to a square grid (Figure 4.6b). It was discretized in a way, that each cell has a value of ”0” or ”1”. It takes ”0” when a ratio of the drivable area compared to a non–drivable area under the given grid cell is lower than a certain threshold (0.3 in this work), ”1” is assigned otherwise. Moreover, the grid size is the same, as the output destination grid size.

Such an input and output combination simplifies the task for the network, explicitly supplying it with information on plausible grid cells to choose, but at the same time, the network can still assign a probability to a grid cell which was not marked as a „drivable area”, which makes it more robust to unexpected behavior.

Worth mentioning is that the observed trajectory and social representation are inputs that have a representation for every observed timestamp. The scene representation is static, which means there is one scene representation for all the observed steps.

5 Dataset

Every supervised deep learning algorithm needs to be trained on pairs of inputs and outputs. The more parameters an architecture has, the more data it needs to train. Moreover, the quality of data is also very important - the less noise the data has, the better. There are a few datasets available that can be used for the task of trajectory prediction. In this chapter, we describe them and choose the most suitable one.

5.1. Requirements

The search for the dataset begun with creating a list of requirements. The ideal dataset has to resemble the data that can be collected by a modern car with some additional advanced safety features.

Therefore, the dataset should contain (or it should allow extracting) the following modalities:

1. observed and ground–truth trajectories to predict of at least 4 seconds long, recorded at a frequency of at least 10Hz;
2. positions of other road users for every observed frame;
3. HD map of a neighborhood, which especially contains drivable areas and lane–level data.

5.2. Available datasets

Several datasets for self–driving cars have been studied in depth

The KITTI Vision Suite [15] The KITTI Vision Benchmark Suite is a collection of datasets and ground–truths for many car–related vision tasks and benchmarks. The suite includes tasks such as optical flow evaluation, depth prediction, SLAM evaluation, 2D and 3D object recognition and tracking and road segmentation. The dataset does not directly introduce a trajectory forecasting task. It would have to be reasoned from other tasks, such as tracking. Even then, it turns out that the dataset is very small in terms of scenes covered. The tracking task consists of 21 training sequences, which is too little for trajectory prediction.

Cityscapes [13] This dataset contains dash camera videos, along with annotations (for object detection and classification tasks), vehicle odometry and GPS positions. It has not been designed for trajectory prediction, position estimations of other vehicles are hard to estimate so it is pointless to use it as such. Also, there is a lot of videos from the camera available on request, but they are missing all the annotations.

EuroCity Persons [8] The Eurocity Persons dataset is the largest benchmark dataset for person detection. It has been collected in Europe, in 12 countries, across 4 seasons with different weather and lighting conditions. It consists of 47300 images and has 238200 persons annotated. It is a great dataset for VRUs detection, but it is not fitting for the trajectory forecasting task. It lacks tracking data

(a) Vector Map: Lane–Level Geometry (b) Rasterized Map: Drivable Area

Figure 5.1: Argoverse HD maps are very extensive. They include lane–level geometry encoded as vectors (a) and drivable areas masks, to determine which part of a scene can be used by a vehicle (b). Source: [10].

and the images are taken with a very low (0.25Hz) frequency. Lack of HD maps (thus no available representation of a surrounding) is also an issue.

NuScenes [9] A very fresh, large–scale, high–quality dataset created by Aptiv, a company developing self–driving cars. The dataset includes modalities from a full sensor suite of the car – 1 lidar, 5 radars, 6 cameras, an Inertial Measurement Unit (IMU) and GPS. Additionally, there are HD maps available to download. It consists of 1000 scenes of 20 seconds each. It comes with 1.4M 3D bounding boxes. Unfortunately, there are 2 limitations. The bounding boxes are annotated at 2Hz (even though the lidar works at 20Hz and cameras at 12Hz) and the objects from frame to frame, are not tracked. These two drawbacks make this dataset hard to use for trajectory forecasting. Objects have to be tracked over the frames. The tracking algorithm could be included in the scope of this work, but with only 2Hz frequency, this is a challenging task. For this reason, this work is not using this dataset.

Argoverse dataset [10] It was released around the same time as the NuScenes dataset. It is also very similar – the data originates from a full sensor suite, which in this case is: 2 lidars (10Hz), 7 cameras (30Hz) and 2 stereo cameras pointing forward (10Hz), IMU and GPS. It lacks radars, but effectively the data quality is no worse than the NuScenes'.

This dataset comes in two formats, adjusted for two different tasks: 3D tracking and motion forecasting. The more suitable for the work is the motion forecasting dataset. It comes with a big number of csv files, which describe tracked trajectories. There is one csv file per each target vehicle, resulting in 324557 files and around 320h of driving data. Each csv file contains 50 consecutive positions of the target vehicle, collected at a frequency of 10HZ. 20 steps (2 seconds) will be used as an observed trajectory and 30 (3 seconds) as a target trajectory. Additionally, each file contains positions of neighboring vehicles at each time. The dataset comes with extensive HD–maps, which contain lane–level information and drivable areas (Figure 5.1). The dataset meets all the requirements set for a dataset, therefore it will be used as the main dataset.

5.2.1. Generating lane–level dataset with Argoverse

The dataset used in this work, the Argoverse[10], is relatively big, containing a lot of good quality samples. The only problem is a variety of data. To train a network that predicts a high–level trajectory a lot of very distinct samples are needed. Turns out, most of the trajectories in the Argoverse dataset, are trajectories that are going straight ahead, following the main road. A network trained solely on such a dataset would be very biased towards destinations that are in front of a vehicle and would fully ignore any other directions.

To minimize bias, this work uses a different solution. High–definition maps, which come along with the dataset, contain information on lanes for the whole area. It is typically used to position an autonomous car on a correct lane, but also can be used to improve trajectory predictions of other vehicles. Authors of the dataset, when setting a baseline for the trajectory prediction, used the information on lanes explicitly, by describing a car position relative to the closest centerline. In one of the variations of the baseline, they also used information on where the lane leads as an input, to further improve the predictions.

This work aims to build a prediction system, which is not dependent on map information to that extent (the only information extracted from an HD–map is scene layout). However, this information

can, and is, used during a training phase. To be able to use it during the training, the data containing lanes information fed into a network has to be in the same format as the data which will be used later, for prediction. Therefore, the lanes need to resemble real trajectories. In this work, a whole new dataset, containing 125373 different artificial trajectories has been generated in the following manner: all the centerlines, one by one, where taken and cut into pieces, which resemble 5 second trajectory. To maintain reality, the mean distance between consecutive steps is 0.81m, with a standard deviation of 0.25m, whereas for the real dataset, these parameters are 0.88m and 0.42m accordingly, making these trajectories less diverse, but very alike. Out of all the generated samples, an equal number of trajectories turn left, right and going straight was taken. Such a dataset is balanced in terms of high–level destinations.

6

Experiments and results

The results have been tested against the Argoverse dataset v1.1 [10], motion forecasting subset. The generated trajectories have been evaluated against a test dataset, whose ground–truth trajectories are not public. The evaluation takes place on the eval.ai platform[1], where several metrics are computed. Each metric is computed for $K = 1, 3, 6$, where K is the number of best trajectories taken into account. Those metrics are:

- minADE – minimal Average Displacement Error – the average of lowest L2 distances between predicted and ground–truth trajectories out of K best predictions,
- minFDE – minimal Final Displacement Error – the average of lowest L2 distance between the last predicted and ground–truth points out of K best predictions,
- DAC – Drivable Area Compliance – a percentage of trajectories which do not exit drivable areas at any point and
- MR – Miss Rate – a percentage of trajectories whose last predicted point is further away than a given threshold – 2 meters.
- HLMR – High–Level Miss Rate – a percentage of trajectories high–level goal is not in the set of top–K predictions.

A detailed description of the metrics can be found in Section 3.2.1.

6.1. Hyperparameters

Hyperparameters are external parameters of a model whose value cannot be estimated from data. They are often used in processes to help estimate model parameters (i.e. coefficients in a regression problem).

Final hyperparameters of the model were chosen during a manual tuning process, which involved testing multiple setups of the network. Hyperparameters that were tuned are:

- learning rate,
- last layer's activation function,
- training optimizer,
- final destination input representation (for the low–level network),
- scene encoder's layers configuration

[1] https://evalai.cloudcv.org/web/challenges/challenge-page/454/overview

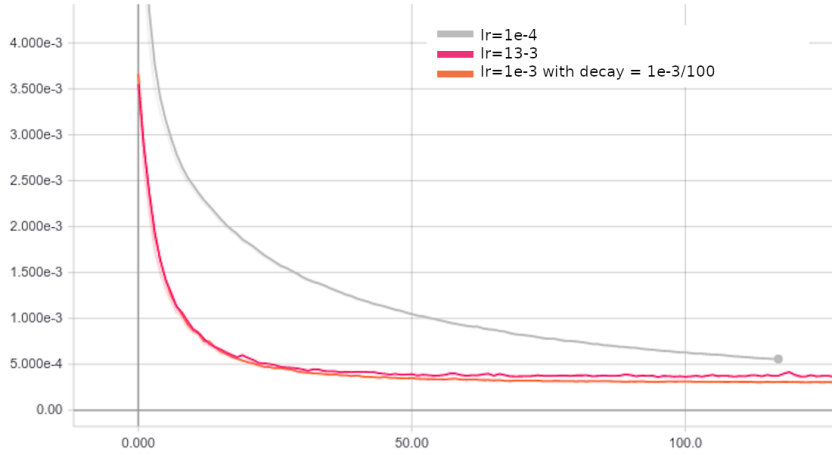


Figure 6.1: The influence of a learning rate on convergence speed. The model with $lr = 1 \times 10^{-4}$ converges the slowest (gray line), whereas models with $lr = 1 \times 10^{-3}$, both without and with a decay, converge the fastest (magenta and orange lines). The error rate presented here is calculated on a validation dataset. The experiment has not been made on the same architecture as the one described in the previous sections, but very similar. It has been done to show an effect of the learning rate on the convergence speed.

6.1.1. Learning rate tuning

The learning rate controls how quickly the model finds a minimum for the objective function. The learning rate search was performed on the low–level network. The value which was found was applied to the high–level network.

Three different learning rates were tested: 1×10^{-4}, 1×10^{-3} and 1×10^{-3} with a decay equal to $1 \times 10^{-3}/100$. The learning rate decay is a mechanism which adjusts the learning rate after every batch update, according to the formula:

$$lr = init\_lr * \frac{1.0}{1.0 + decay * iterations} \tag{6.1}$$

The rule of thumb says the decay value should be the initial learning rate divided by the total number of epochs we are training the network for. The network was trained for around 100 epochs, therefore the decay was set to $1 \times 10^{-3}/100$.

The results show that a network with a learning rate set to 1×10^{-4} was converging the slowest, whereas networks with learning rate equal to 1×10^{-3}, both with and without the decay, converged in almost the same time. The results are presented in Figure 6.1.

6.1.2. Activation functions

Two activation functions for the last layer in the low–level network were tested: linear and Rectified Linear Units (ReLU). Both of them converged and achieved similar results in terms of MSE on the validation dataset, as presented in Figure 6.2. ReLU converged slightly faster, after 15 epochs, whereas the model with a linear activation converged after 18 epochs. The difference is of negligible importance.

6.1.3. Optimizer

Only one optimizer was tested, which yielded satisfactory results from the beginning. The optimizer was Adam [22] with a learning rate set to 1×10^{-3}. The algorithm calculates an exponential moving average of a gradient and a squared gradient. Parameters beta1 and beta2 control the decay rates of these moving averages, and are set to 0.9 and 0.999 respectively.

6.1.4. Final destination input representation (for the low–level network)

One of the inputs for the low–level network is a final destination grid cell. This modality informs the model where a destination for the generated trajectory should be. Two different representations were tested: one–hot encoded vector and a single number. The single number is from a $< 1, dim^2 >$ range, where dim^2 is a number of all grid cells, informing which grid cell is the destination. The other option is a one–hot encoded vector. The one–hot encoded vector is a vector containing only zeros, except for

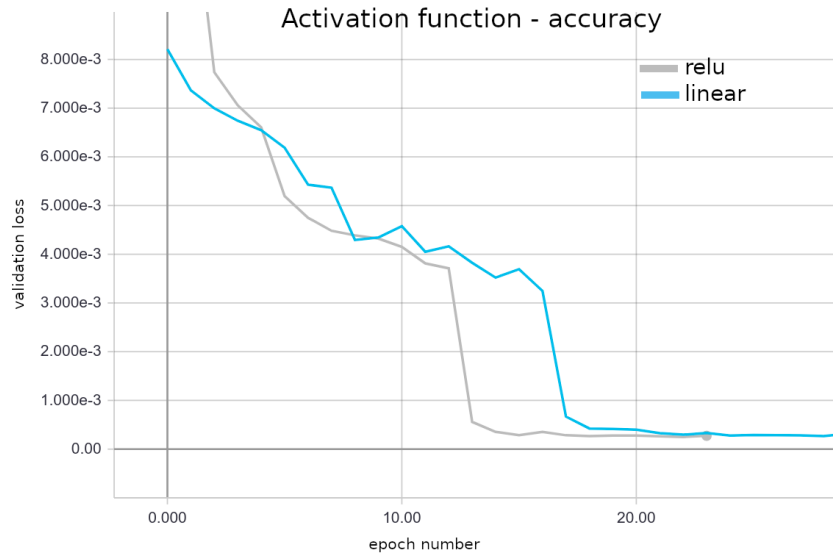
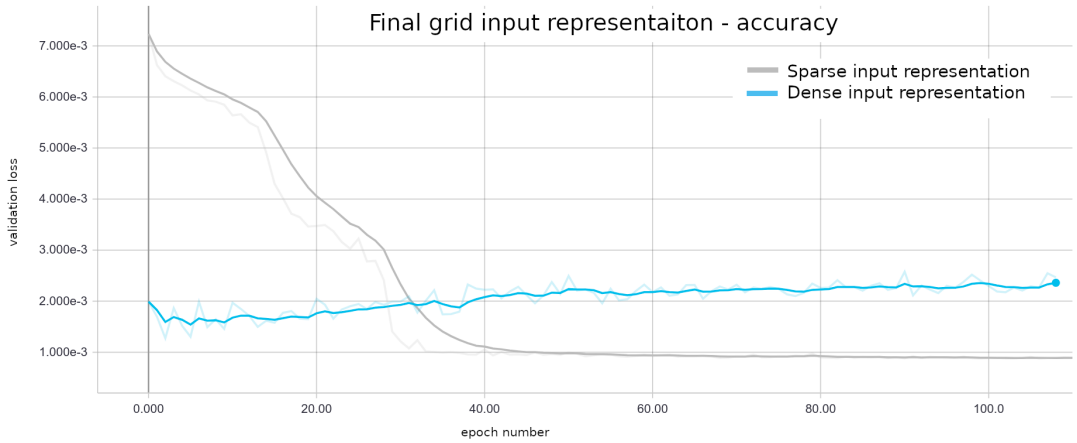


Figure 6.2: The influence of an activation function in the last layer on accuracy. Models with both activations – Linear and ReLU achieved similar results in terms of accuracy, but ReLU converged after 15 epochs, which is a little faster than the linear – 19 epochs.



one cell, at index equal to the destination grid cell number, which contains a value of one. This points to the destination grid cell. This representation has been chosen over the dense one, as the dense one performed much worse – it was overfitting on the training dataset. This can be noticed by observing the validation loss, which is presented in Section 6.1.4

6.1.5. Scene encoder's layers configuration

The scene encoder for the low– level networks is a network consisting of convolutional and max–pooling layers. Order of those layers, number of parameters in each layer, size of kernels, stride can be tuned and adjusted to the problem. In this work three different setups were tested. These setups are popular architectures for the MNIST challenge, which is a classification problem of hand–written digits. The digits, similar to scene representations, are low–quality, black and white images. Therefore, similar CNN architecture might be suitable for analyzing these scene representations. Different configurations were trained on the low–level network and tested against validation dataset. The system was evaluated end–to–end, so for each observed trajectory there were 6 ranked predictions. They were ranked according to the DAC metric.

The first setup is similar to the VGG architecture [34] with the trend Conv-Conv-Pool-Conv-Conv-Pool. The details of the setup are presented in Table 6.1. The second follows trend Conv-Pool-Conv-Pool – Table 6.2. The last tested setup follows Conv-Pool-Conv-Pool as well as, but the second to last

Layer	Kernel size	Stride	Number of filters	Padding	Output shape
Input Image	-	-	-	-	(72,72,1)
Conv2D	3	1	16	same	(72,72,16)
Conv2D	3	1	16	same	(72,72,16)
MaxPool + BatchNormalization	2	2	-	valid	(36,36,16)
Conv2D	3	1	32	valid	(34,34,32)
Conv2D	3	1	32	valid	(32,32,32)
MaxPool + BatchNormalization	2	2	-	valid	(16,16,32)
Flatten	-	-	-	-	(8196)
Dense	-	-	-	-	(512)

Table 6.1: The scene encoder's layers configuration, based on VGG architecture.

Layer	Kernel size	Stride	Number of filters	Padding	Output shape
Input Image	-	-	-	-	(72,72,1)
Conv2D	3	1	32	valid	(70,70,32)
MaxPool + BatchNormalization	2	2	-	valid	(35,35,32)
Conv2D	3	1	64	valid	(33,33,64)
MaxPool + BatchNormalization	2	2	-	valid	(16,16,64)
Conv2D	3	1	128	valid	(14,14,128)
MaxPool + BatchNormalization	2	2	-	valid	(7,7,128)
Flatten	-	-	-	-	(6272)
Dense	-	-	-	-	(64)

Table 6.2: The scene encoder's layers configuration, based on Conv-Pool-Conv-Pool trend.

layer is a Dropout layer and the last layer is not dense, but just a flattening transformation, as shown in Table 6.3.

Turns out, the last setup scored best on the validation dataset, with 94.23% DAC, the second model scored 92.61% DAC and the VGG–like model scored 92.91% DAC. Therefore, the last setup is used for the final evaluation.

6.2. Experiments and results

Once the right setup of layers and their sizes was found, several network configurations were trained and evaluated.

The most influential parameter to tune for this architecture is the grid dimensionality (the scene input resolution and output grid resolution of the high–level network). This directly changes the accuracy of the final predictions, since the output of the high–level network can be seen as a definition of a goal, towards which the low–level network has to generate a trajectory. The lower the resolution, the less accurate the goal definition. Therefore, the first experiments were designed to find the best grid resolution. Once the best grid cell resolution was found, the model was trained with a few variations:

Layer	Kernel size	Stride	Number of filters	Padding	Output shape
Input Image	-	-	-	-	(72,72,1)
Conv2D	3	1	32	valid	(70,70,32)
MaxPool + BatchNormalization	2	2	-	valid	(35,35,32)
Conv2D	3	1	64	valid	(33,33,64)
MaxPool + BatchNormalization	2	2	-	valid	(16,16,64)
Conv2D	3	1	96	valid	(14,14,96)
MaxPool + BatchNormalization	2	2	-	valid	(7,7,96)
Dropout (0.2)					(7,7,96)
Flatten	-	-	-	-	(4704)

Table 6.3: The scene encoder's layers configuration, based on Conv-Pool-Conv-Pool trend with a dropout layer.

	K = 1				K = 3				K = 6			
	minADE	minFDE	DAC	MR	minADE	minFDE	DAC	MR	minADE	minFDE	DAC	MR
Constant Velocity	3.53	7.89	0.88	0.83	-	-	-	-	-	-	-	-
Argoverse Baseline (NN+map(prune))	3.38	7.62	**0.99**	0.86	2.11	4.36	**0.97**	0.68	1.68	3.19	**0.94**	0.52
Vanilla SS–LSTM	2.79	6.30	0.88	0.84	2.79	6.30	0.88	0.84	2.79	6.30	0.88	0.84
HiPSS–LSTM 5x5	2.68	**5.93**	0.91	0.81	1.95	3.97	0.86	0.73	1.80	3.53	0.74	0.71
HiPSS–LSTM 7x7	2.72	6.02	0.92	0.82	1.90	3.87	0.89	0.72	1.71	3.30	0.80	0.70
HiPSS–LSTM 11x11	**2.67**	6.02	0.93	0.81	**1.74**	**3.58**	0.91	0.65	1.47	2.83	0.88	0.59
HiPSS–LSTM 13x13	2.71	6.03	0.94	**0.80**	1.78	3.59	0.92	0.61	1.47	2.74	0.90	0.53
HiPSS–LSTM 13x13 w/ pretraining	2.68	6.00	0.93	**0.80**	1.76	3.59	0.91	0.61	**1.45**	**2.71**	0.89	0.52
HiPSS–LSTM 13x13 w/ dest_loss	2.69	6.02	0.93	**0.80**	1.76	3.60	0.92	0.62	1.46	2.77	0.89	0.54
HiPSS–LSTM 13x13 w/ scene_loss	2.74	6.02	0.93	**0.80**	1.81	3.61	0.92	0.62	1.52	2.77	0.89	0.54
HiPSS–LSTM 15x15	2.75	6.21	0.93	0.81	1.82	3.73	0.92	**0.59**	1.48	2.78	0.90	**0.49**
HiPSS–LSTM 13x13 no scene	2.74	6.28	0.87	0.83	1.87	4.01	0.86	0.67	1.56	3.16	0.83	0.61
HiPSS–LSTM 13x13 no social	2.82	6.24	0.92	0.81	1.83	3.69	0.91	0.63	1.50	2.80	0.89	0.55

Table 6.4: Forecasting errors for baselines and different networks' configurations, for a 3 seconds prediction horizon and top 1, 3 and 6 predictions. minADE is minumum Average Displacement Error (lower is better), minFDE is minimum Final Destination Error (lower is better), DAC is Drivable Area Compliance (higher is better) and MR is Miss Rate (lower is better). For detailed definitions refer to Section 3.2.1. „5x5" and similar represent high–level grid resolution, which the models were trained on. The model with „pretraining" is a model with the low–level network pretrained on the artificial dataset (Section 5.2.1), „dest_loss" and „scene_loss" are models, where the low–level network is trained with Destination Loss and Scene Loss respectively (Section 4.3.2). The entries marked as „no social" and „no scene" are models trained without those inputs. Results of Vanilla SS–LSTM were pasted to columns with K=3 and K=6 for comparison purposes, even though the method always outputs one trajectory (K=1).

with low–level network pretraining and with different, custom losses. As an ablation study, models without scene and social elements were tested as well.

The tested setups are:

- *HiPSS–LSTM* with high–level prediction grid sizes of *5x5*, *7x7*, *11x11*, *13x13* and *15x15* – these are setups with different grid sizes, trained with basic loss functions – the high–level network with „cross–entropy" and the low–level network with MSE. The low–level network is trained only on real trajectories.
- *13x13 HiPSS–LSTM with low–level scene loss* and *13x13 HiPSS–LSTM with low–level destination loss* – these are models trained for the best performing grid size, where the low–level network was trained with an additional loss function – *scene loss* or *destination loss*.
- *13x13 HiPSS–LSTM with low–level pretraining* – this model is trained for the best performing grid size, but the low–level network was trained with the artificial dataset first, and only then with the dataset containing real trajectories.
- *13x13 HiPSS–LSTM without the scene input representation* and *13x13 HiPSS–LSTM without the social input representation* – this ablation study aims to show an influence of scene and social representation on the predicted trajectories.

The performance is also compared to a baseline, which is a linear interpolation of recent trajectory, with a constant velocity. Moreover, it is compared to a reported by Argoverse Baseline using Neural Networks with scene representation [10], which was the best performing method created by them. Since the HiPSS–LSTM is modeled on SS–LSTM [41], it was also compared to that. The SS–LSTM (noted as Vanilla SS–LSTM in Table 6.4) is a model created as described in the original paper, trained on the Argoverse dataset. Since it can output only unimodal prediction, the reported scores are for K=1.

To make sure that the results are meaningful and comparable across each other, all the networks were trained for long enough, to converge during training (based on validation score). The high–level network was trained on the artificial dataset for 150 epochs and then on real trajectories for an additional 80 epochs. Low–level network training process required the same number of epochs. The learning rate used is $1e-3$ for all the experiments. All the networks are trained on the same train, validation and test dataset. Moreover, the order of samples fed into the network is also the same. Batch size is always set to 32. The only thing that changes from experiment to experiment is weights initialization, which is always random.

The results are presented in Table 6.4. The best performing method in terms of minFDE is the one trained to output high–level probabilities on a grid of size 13x13, with a combination of a low–level network which was pretrained on artificial trajectories.

All the HiPSS–LSTM configurations with resolution above 11x11 scored better in terms of minFDE than all the baseline methods. They also score equally well or better in terms of the MR score.

There is a noticeable influence of high–level output grid dimensionality on the performance of the network, as presented in Figure 6.3a. A model with a grid resolution of 13x13 scored best in terms of teh minFDE, but both lower and higher grid resolutions reported a worse performance. Lower grid resolutions, such as 5x5, scored worse because the low–level network was supplied with a very rough destination. It was too vague to correctly lead a path to it. On the other hand, a high–resolution grid, such as 15x15 starts to be too fine–grained. With such a resolution, the network no longer predicts all the reasonable high–level destinations, within the top 6 results. This is presented in Figure 6.4, where in Figure 6.4c all the selected high–level points (indicated with a red dot) are present close to each other. This makes the prediction error higher, for higher grid resolutions. Too small grid resolution leads to a different problem, mentioned already, that the destination area is too vague, as presented in Figure 6.4a. The best trade–off found is a 13x13 grid (example presented in Figure 6.4b), where the destinations are quite well-defined, but they are rarely concentrated around one high–level destination area.

On the other hand, with the increased high–level grid resolution, the Miss Rate (MR) error decreases, as presented in Figure 6.3b. This once again proves that the better the destination area is defined, the better the low–level trajectory and the last predicted point can be. If the destination area, represented by a grid cell is smaller, a trajectory generated by the low–level network can be more precise. Following this reasoning, a method with the destination loss was implemented and evaluated. Interestingly, the method's MR is higher than an MR of a corresponding method trained with a simple MSE. This is due to a fact, that the destination loss drags trajectories towards the center of the destination area, and not only towards boundaries of it, as it is with the MSE (MSE is calculated against the ground–truth trajectories, whose last point is always within boundaries of the destination area). The center of the destination area is not always within 2m threshold, which is required by the MR metric.

One thing that the network struggles with is the DAC (Drivable Area Compliance), which in every configuration of HiPSS–LSTM model is lower than Argoverse Baseline (NN) and often lower even when compared to Constant Velocity model. This is because, this metric takes into account all the trajectories generated (top 1, 3 or 6) and not only the best one (as it does with the minADE, minFDE and MR). Therefore, when the high–level prediction model outputs a destination that is outside of a drivable area, and the low–level network creates a trajectory leading to this destination, part of the trajectory could be outside of drivable areas, which increases the DAC error. On the other hand, a trajectory prediction system should take into account, that a future trajectory can lead outside of drivable areas, as the HiPSS–LSTM does, because some situations require this. Unfortunately, training the network with a manually designed loss, which takes into account drivable area compliance, does not improve the DAC measure.

6.2.1. An ablation study

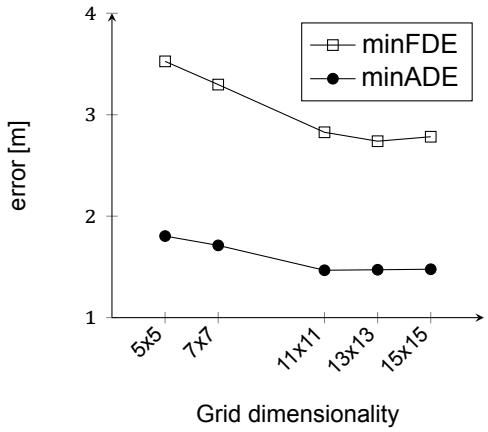
As an ablation study, two models with an incomplete set of inputs were trained – a model without a scene representation as an input and a model without a social representation as an input.

The model without scene feature (HiPSS–LSTM 13x13 no scene), achieved the lowest DAC score, according to expectations. It also scored very low on minADE and minFDE. This proves, that scene representation is a strongly predictive feature.

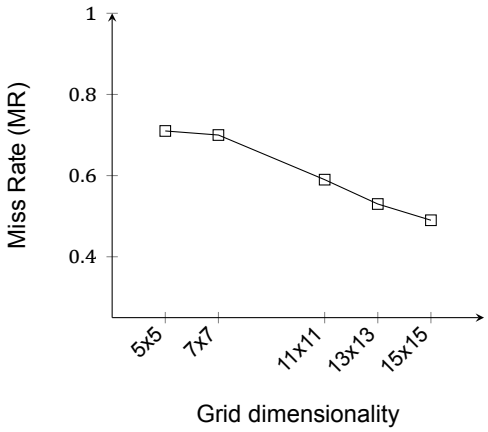
The lack of social input feature (HiPSS–LSTM 13x13 no social) did not have such a big impact as lack of scene input, but also worsened the results, comparing to the corresponding model with this input feature present (HiPSS–LSTM 13x13). It scored slightly lower on minADE and minFDE, which suggests that this feature is desired and has an impact on prediction results.

6.3. High–level network performance

To fully assess the performance of the networks, apart from the results of the overall performance (high– and low–level networks together in Section 6.2), the high–level network was analyzed separately. Since this network cannot be assessed on the test dataset supplied by the Argoverse (since there is no ground–truths available online), this network is evaluated on a validation dataset, which was not directly used in the training process (it was only used to monitor convergence of the model). The metric used is High–Level Miss Rate (HLMR) (described in detail in Section 3.2.2).

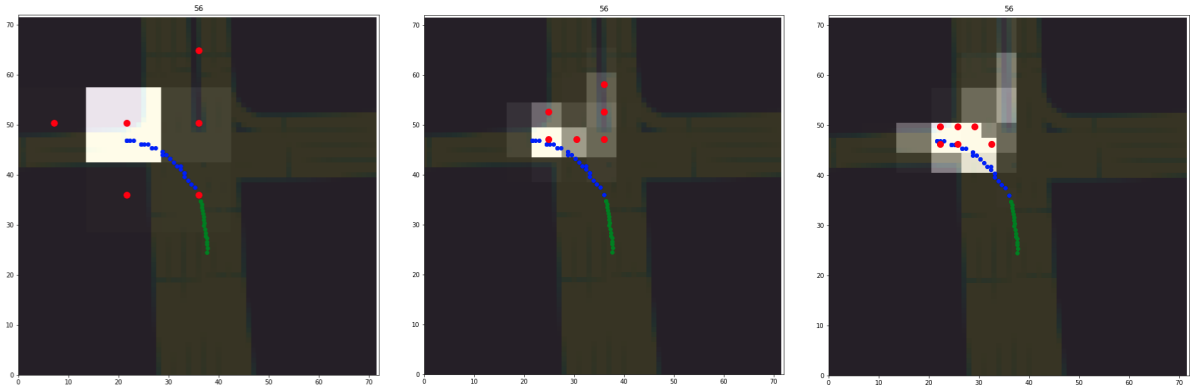


(a) The relation between grid dimensionality and minADE and minFDE errors.



(b) The relation between grid dimensionality and MR error.

Figure 6.3: The graphs present a correlation between grid dimensionality and error measures – minFDE and minADE on (a) and Miss Rate (MR) on (b) for *HiPSS–LSTM* with high–level prediction grid sizes of *5x5*, *7x7*, *11x11*, *13x13* and *15x15* for K=6. Lower error is better.



(a) Visualization for a 5x5 prediction grid.

(b) Visualization for an 11x11 prediction grid.

(c) Visualization for a 21x21 prediction grid.

Figure 6.4: Example of a too high grid resolution problem. Green points are representing an observed trajectory, blue low–level ground truth trajectory, squares are predicted high–level destinations, where a brighter color means higher probability, red dots, which are marked on top of 6 squares, mean that these are top–6 high–level goals. When the resolution of the high–level grid is too high (c) the top 6 destinations are often around one, same area, meaning that generated low–level trajectories will not cover all reasonable high–level goals. On the other hand, a too low resolution (a) yields too vague destination points for the low–level network to be able to generate good low–level trajectories. Therefore it is better to have a bit lower resolution (b), but a diverse set of high–level predictions.

	K=30	K=20	K=10	K=6	K=3	K=1
5x5	0.0%	0.0%	0.01%	0.17%	1.94%	23.82%
7x7	0.0%	0.01%	0.32%	1.10%	4.56%	31.41%
11x11	0.13%	0.38%	1.67%	4.24%	12.73%	47.07%
13x13	0.44%	0.89%	2.82%	6.41%	16.82%	51.05%
15x15	0.91%	1.61%	4.40%	9.14%	21.64%	55.88%

Table 6.5: HLMR error for various combinations of high–level grid resolutions and top–K predictions. The error grows with the resolution increase and the decrease of K parameter. Theoretically, it would be best to choose the biggest possible K and the lower grid resolution. But that would generate a lot of trajectories leading towards a loosely defined goal. For the low–level network it is best to pick the highest resolution possible. This table shows, that for K=6, grid resolution of 13x13, 6.41% of times would not select the correct high–level destination among the top 6 predictions. For grid size 15x15 it rises to 9.14%. On the other hand, the bigger the K, the error is lower again.

The precision of low–level trajectories is highly dependent on a performance of the high–level network. The better the high–level destination is defined, the more precise low–level trajectories are. As presented in Table 6.5, the HLMR worsens with every increase of the resolution. However, the more generated trajectories are taken into account, the HLMR error is lower. As an example, for the top 20 predictions for grid resolution 15x15, the HLMR stays below 1%. Trajectories generated towards those destinations would score better on MR metrics.

6.4. Visualization and discussion on results

The system in most cases predicts a diverse set of trajectories, where usually one of the top 6 predictions (based on predicted probability) is compliant with the ground–truth trajectory, as presented in Figure 6.5.

There are some failed predictions when the system misses the correct high–level destination, as presented in Figure 6.6. Since the low–level network generates trajectories from the last observed point to the high–level destination area, none of the low–level trajectories is correct when the high–level goal is missed.

Other types of failed cases include predictions which were correct on the high–level, but due to the big imprecision of high–level goal definition, they were not correct on a low–level. In this case, the predicted trajectories are sometimes shifted to a side or miss the last ground–truth point substantially, as shown in Figure 6.7a and Figure 6.7b. Sometimes the predicted trajectories are unrealistic when compared to an associated scene layout as presented in Figure 6.7c. This suggests that the scene encoder might have not encoded the scene well enough to be able to output predictions inline with the road layout.

6.4.1. Influence of social forces

It is difficult to visually identify if the network takes into account social factors and to what extent on any of the given samples. All the samples that I examined, where one could argue that the predictions try to avoid driving into other vehicles, are ambiguous. Therefore, to address this issue, the test of the influence of social factors was performed on partially manipulated data. One sample was selected, with a fairly simple situation – observed trajectory is a path going straight ahead, along a road around which is also straight, with one very common, perpendicular intersection, placed at the end of the ground–truth trajectory. Next, all the surrounding vehicles were erased, and only one was placed in a small distance from the end of the observed trajectory. It is expected, that the predicted trajectories would go around the vehicle which is standing still in front of the ego vehicle. In another case, all vehicles were erased and none was added. In this case, it is expected that the predicted trajectories would go directly straight ahead. Unfortunately, the predicted trajectories for both cases are the same. This proves the fact, that social element needs further investigation and improvements.

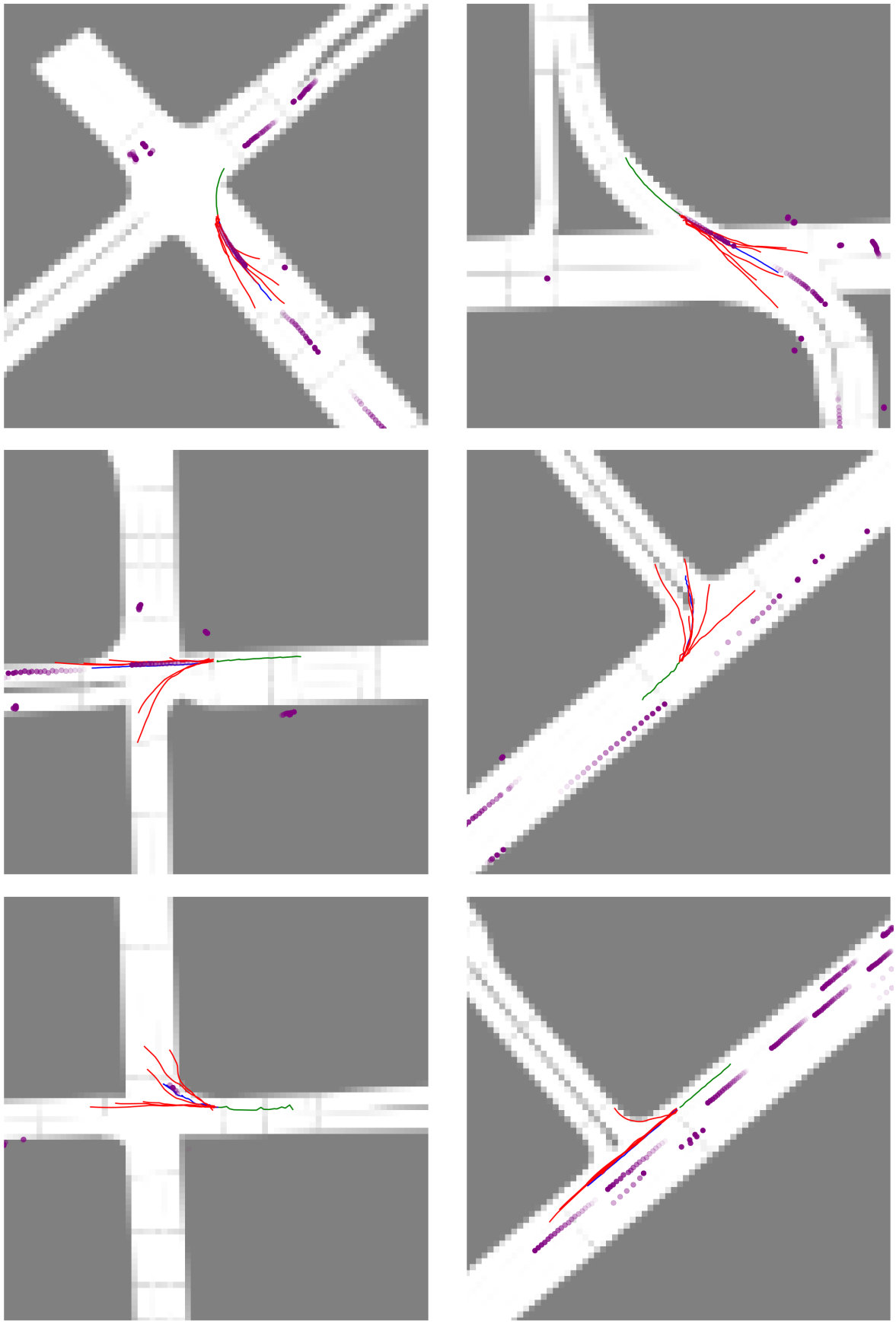
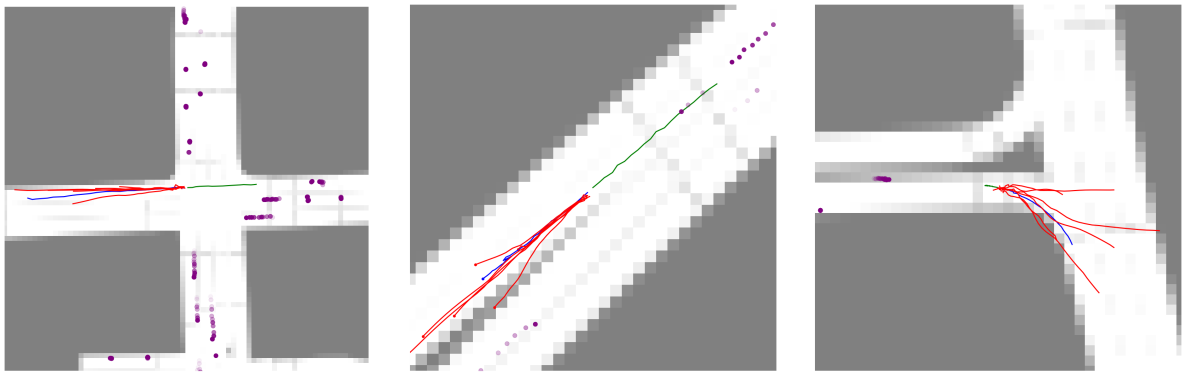


Figure 6.5: Examples of successful predictions. The predictions are accurate and diverse. The green line is an observed trajectory, blue is ground–truth trajectory, reds are the best 6 predictions. The purple dots are other vehicles, where the lightest purple are vehicles in the first timestamp and the darkest in the last observed timestamp.

Figure 6.6: Examples of failed predictions, when the high–level goal is not predicted correctly. None of the predicted trajectories is correct, because the high–level network failed to predict the correct destination grid cell. The green line is an observed trajectory, blue is ground–truth trajectory, reds are the best 6 predictions. The purple dots are other vehicles, where the lightest purple are vehicles in the first timestamp and the darkest in the last observed timestamp.

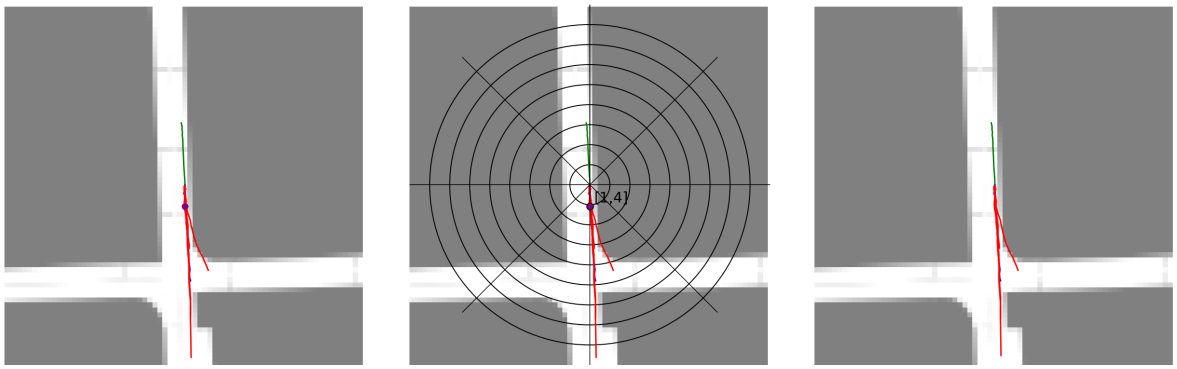


(a) All the predictions are in a correct direction, but are shifted north or south comparing to the ground–truth.

(b) High–level direction is correct, but the prediction does not identify motion model correctly and predicts trajectories which are either too long or too short. Visualization is zoomed–in for clarity.

(c) An example of predictions not inline with an underlying scene, resulting in unrealistic predictions. Visualization is zoomed–in for clarity.

Figure 6.7: Examples of failed predictions, when the high–level goal was identified correctly, but the low–level network failed to correctly estimate motion model, leading to too long or too short predicted trajectories or when the underlying road layout was not identified correctly. The green line is an observed trajectory, blue is ground–truth trajectory, reds are the best 6 predictions. The purple dots are other vehicles, where the lightest purple are vehicles in the first timestamp and the darkest in the last observed timestamp.



(a) Situation with one vehicle standing still in front of the ego vehicle. It is expected, that the predicted trajectories would go around the standing vehicle. Unfortunately, the model seems to ignore the vehicle and plans the trajectories directly through the standing vehicle.

(b) Visualization of social forces in the last observed timestamp. The standing vehicle is in range of the ego vehicle, so the prediction model takes it into account.

(c) The same situation, without the vehicle standing still. The predicted trajectories are straight, as expected.

Figure 6.8: Test on the influence of social forces. The same situation with and without neighboring vehicles is resolved by the prediction model. Unfortunately, the model seems to ignore the vehicle standing still in front of the ego vehicle and plans trajectories directly through the standing vehicle.

7

Conclusions, limitations and future work

The main goal of this thesis was to create a system, which can reliably predict multiple possible future trajectories. This was achieved by designing a Deep Learning model, which was fine–tuned and trained for this task. During the design and experimentation process, some new ideas emerged on how to tackle this problem differently, how to further improve the model and what limitations the current solution has.

7.1. Conclusions

An important aspect of this work is the network's ability to output truly multimodal predictions. This should be a requirement for every system dealing with a similar problem. Human drivers sometimes make a last–second decision. These are the most dangerous situations and the system should be able to anticipate them, even if they are not the most likely to happen. This work creates such a system. The top–K predictions are diverse, every single one is on a high–level different from another. The system can predict high–level destinations outside of drivable areas, therefore, even if a human driver behaves in a non–standard way and goes outside of drivable areas, there is a high chance that the system predicts it.

Achieving high performance by the high–level network was possible only when the pretraining step was applied. The network was pretrained on an artificial, balanced dataset – it has the same number of trajectories that are going straight ahead, turning left and turning right. If the network was trained only on trajectories supplied with a dataset, it would be biased towards straight trajectories. The network benefits now from both data sources – from the lane–level information and the real trajectories.

The high–level network would not solve the trajectory prediction task alone. Equally import stage, which is a trajectory planning, is done by the second stage of the architecture – low–level trajectory planning. The low–level trajectory planning depends on the previous stage, which delivers rough destinations. It was also pretrained on the artificial dataset. Since the destinations are not just two coordinates, but rather an area that should be reached, the low–level network has to be able to lead a trajectory from a starting point towards the destination area. It has to take into account that the high–level prediction might not be reachable, but it still has to output a valid trajectory leading towards it. And since it was trained on real trajectories, that is exactly what it does – creates realistic trajectories leading towards the goal.

Overall, it seems that as long as the destination point is well–defined and the network knows the current dynamics and context of the target vehicle, generating the low–level trajectory is feasible. Another difficulty is present in defining the destination point as precisely as possible. This depends on the correct choice of the high–level grid dimensionality. The experiments showed, that the best trade–off between accuracy and diversity is when the grid has 13 by 13 cells resolution.

7.2. Limitations

This work uses a new, vast and comprehensive dataset, which contains a lot of various trajectories. It performs well on this dataset, but it has not been tested against any other dataset. Therefore, transfer and generalization ability of the model to other datasets is unknown at this point.

Moreover, this dataset contains only trajectories from two American cities. It can be argued, that American cities are very specific when it comes to a road arrangement. For instance, they seem to contain less arched roads and small intersections comparing to European cities. The roads in the USA are usually made of multiple lanes, which might not be a rule in other cities. In consequence, an assumption that it works on all possible kinds of roads cannot be made.

Additionally, as the dataset does not contain illegal turns, all the predictions seem to be very reasonable. The method might also not predict correctly rare maneuvers such as U–turns, as there are none or very little such turns in the dataset. As the method is data–driven, it needs to be trained against those behaviors specifically.

7.3. Future work

Trajectory prediction is an open research topic and there is still a lot to discover. This work attempted to find the best possible trajectory prediction model, but given the results, there are still some improvements that can be made.

The network proved to be highly dependent on the first prediction stage – the high–level network and the prediction depicted by it. It was reported, that the grid dimensionality of 13x13 performed the best, but the performance dropped again with an increase of the grid resolution. This is due to the fact, that with such a high granularity, a set of top–K destinations are concentrated around one point, sometimes missing out different high–level destination. To counteract this, a refinement step could be incorporated at the end of high–level network, which would cluster neighboring high probability destinations together, separating a few different on a high–level destination from each other. The low–level network would be fed with a very limited number (1 or 2) of small destinations within the clustered area and would lead a trajectory to it.

As pointed out in the Limitations section, the model is trained on one dataset which was collected on American streets. Future work would involve validating it against other datasets, to see how well the model generalizes to other cases. If not, it should be redesigned and/or trained on other datasets to generalize better. It also should be tested against more unpredictable and rare behavior such as U–turns.

An element that could use some improvement is the social element. In this implementation the social forces and not reliably taken into account, as proven in section 6.4.1. It might be due to a lack of suitable data to train the model to take other vehicles into account – there might be too little samples with interactions. In this case, the social model should be trained separately or some heuristic should be designed which would take other vehicles reliably into account. The other reason might be that the circular occupancy grid processed with an LSTM is not encoding the interactions well enough. In this case, this element needs to be redesigned.

Judging by the results, the model could use some improvements in compliance with an underlying road layout, which is reflected in the DAC metric. This method yields approximately 90% DAC in the best case. To improve this, the scene encoding module should be further analyzed and improved.

An important modality that should be taken into account is a direction of a lane. A predicted trajectory should not be on a lane that is of an opposite direction. This kind of information could be encoded in the scene input. Different lane directions would be depicted with different colors, as in [14]. The convolutional neural network processing the scene input would be trained to differentiate between these colors and pass on the relevant information further to the network. As this work focuses on predicting a trajectory of vehicles, some additional cues could be taken into account. For instance, the network could take into account right of way rules or a turn indicator which could be a strong visual cue. As human drivers rely on it a lot, the same could apply to an automated system. As this network was designed to be modular, adding such an input is straight forward. It can be added as an input, next to the existing ones – the observed trajectory, social and scene representations. On the other hand, these modalities carry a danger – for instance, a fact of detecting an active indicator doesn't directly translate to a maneuver in that direction. It could even mislead a system, resulting in an incorrect prediction. Therefore, if these modalities were taken into account, they should only reinforce the already identified future trajectory. Future work could look into those subtle, but meaningful cues, and take them into account when modeling future behavior.

This work has provided important insights into the domain of future trajectory prediction. Hopefully, one day technology like this will be incorporated into all vehicles, increasing safety on roads and

reducing car accidents, making car transportation much safer than it is today.

Appendices

A

Machine Learning - the essentials

In this chapter, we introduce fundamental, but useful knowledge to understand the HiPSS architecture.

A.1. Feedforward network

The goal of a Feedforward Neural Network, also called multilayer perceptrons (MLPs), Fully Connected Neural Network, or ”Dense” in Keras framework, is to approximate some function f^*. „A feedforward network defines a mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation”[16]. They are called feedforward, because the information only travels forward in the neural network, from the input **x** then through hidden (single or many) layers, used to define f and finally to the output **y**. A scheme of the network is presented in Figure A.1. There are no feedback connections in which outputs of the model are fed back into itself (these kinds of networks are called *Recurrent Neural Networks* and are described in Section 2.1).

Feedforward networks form an underlying basis for many of many modern, widely used deep learning architectures and are of extreme importance to machine learning practitioners. They are a groundwork for many important commercial applications. The usual task for a network is to learn to classify an input **x** to a category **y**, in case of classifiers or to learn a function, which maps an input to a value y, in case of regression. One of the most popular architecture is Convolutional Neural Network, which is a specialized type of feedforward network. They are used for object recognition from photos – one of the most popular applications of deep learning.

As Goodfellow et al. explained in their work, feedforward neural networks are called networks because „they are typically represented by composing together many di□erent functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, we might have three functions $f^{(1)}, f^{(2)}$, and $f^{(3)}$ connected in a chain, to form $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$”. The number of functions contributing to the chain is described as a „depth” of a network. The term „deep learning” arose from this terminology. During training, $f(\mathbf{x})$ is derived during neural network training to match $f^*(\mathbf{x})$, using labels $y^*(\mathbf{x})$ which are provided for each example **x**. The algorithm used to derive the function is called **stochastic gradient descent**.

A.1.1. Activation functions

Activation Function is a function that introduces non–linearity to a network. The function is applied to every hidden layer’s output. The most relevant for this work are ReLU and Softmax functions.

ReLU

ReLU (Rectified Linear Unit) is the most commonly used activation in this work. The formula for it is simple:

$$R(z) = max(0, z)$$

It proved to be an effective choice in many different works, as well as this one.

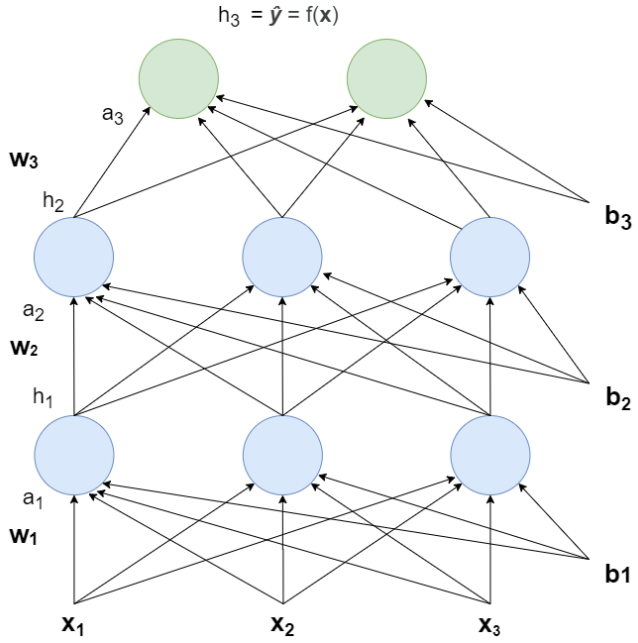


Figure A.1: Scheme of a feedforward network, where $\mathbf{x}$ is an input, $\boldsymbol{\theta} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \boldsymbol{w_3}\}$ and $\boldsymbol{w_k}, k \in \{1, 2, 3\}$ describe transformation matrix between input and output (w.r.t. to $\boldsymbol{w}$) layers. This network has 3 hidden layers, each containing 3 units.

Softmax

Softmax is a function which is widely used for classification problems. It is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

A.2. Cost functions, stochastic gradient descent and backpropagation

Cost function

It is a function that measures the performance of a Machine Learning model for given data. Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number. Depending on the problem Cost Function can be formed in many different ways. The purpose of the Cost Function is to be either:

- Minimized – then returned value is usually called cost, loss or error. The goal is to find the values of model parameters for which Cost Function return as a small number as possible.
- Maximized – then the value it yields is named a reward. The goal is to find the values of model parameters for which the returned number is as large as possible.

The two most important cost functions for this work are Mean Squared Error (MSE) and Cross–Entropy.

- MSE is a mean of the squared differences between predicted y and actual y.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Cross–Entropy measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from

the actual label.

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x)$$

- Sparse–cross–entropy is the same as the cross-entropy, but different on the implementation level. The sparse variation takes as an input an index of a class, to which a ground–truth belongs, and transforms it into a one–hot encoded vector. Then, the cross-entropy is calculated.

Stochastic gradient descent and backpropagation

Stochastic gradient descent [21] is a first–order iterative optimization algorithm for finding a minimum of a function. The algorithm learns parameters $\boldsymbol{\theta}$ by iteratively optimizing for a cost function. The formula summarizing the algorithm is the following:

$$\boldsymbol{\theta_{i+1}} = \boldsymbol{\theta_i} - \alpha \nabla J(\boldsymbol{\theta}) \tag{A.1}$$

where $\boldsymbol{\theta}$ are the parameters $\boldsymbol{\theta}$ to learn, α is a learning rate, $\nabla J(\boldsymbol{\theta})$ is a gradient (derivative) of a cost function J, with respect to the parameters. The algorithm is simple and efficient, if there is a way to promptly calculate the derivative. To calculate it, an algorithm called **backpropagation** is used. The backpropagation [31] allows the information to flow backward from the cost function through the network in order to compute the gradient using previously computed gradients and a chain rule[16].

A.3. Long Short–Term Memory (LSTM) networks

The LSTM network has been invented by Hochreiter and Schmidhuber as a remedy for the vanishing gradient problem (Section 2.1). There were a few changes introduced to an RNN network, which made it possible to be used with gradient–descent based algorithms. „It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities”[18]. The architecture and a gradient–based algorithm, makes the error flow–through internal states constant (thus never exploding nor vanishing). This was a key for a very successful algorithm creation, which is now widely used in a variety of tasks.

To explain how LSTMs work, let's refer to the schematics of an RNN network in a different form, as presented in Figure A.2a. As discussed previously, recurrent networks have a form of a chain, where an output of one module is used as an input to the next one. In standard RNNs, every output is merged with a sequence element using a single $tanh$ activation layer.

In comparison to RNNs, as visualized in Figure A.2, an LSTM has a different structure. Instead of having a single neural network layer ($tanh$), there are four, each of them has a different role. Moreover, the LSTM outputs another vector, cell state $\boldsymbol{C}$ along with a hidden state $\boldsymbol{h}$. The cell state is subjected to only to some small, linear interactions throughout the whole process.

Sets of activation layers with a merging operation are called „gates”. Thus, there are three gates in an LSTM network: “forget gate”, “input gate” and „output gate” (Figure A.3).

Forget gate – role of this gate is to keep or discard some information from a cell state. It takes $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ as arguments, and outputs a number between 0 and 1 for each number in the cell state $\boldsymbol{C}_{t-1}$ (eq. (A.2)).

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W_f} \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b_f}) \tag{A.2}$$

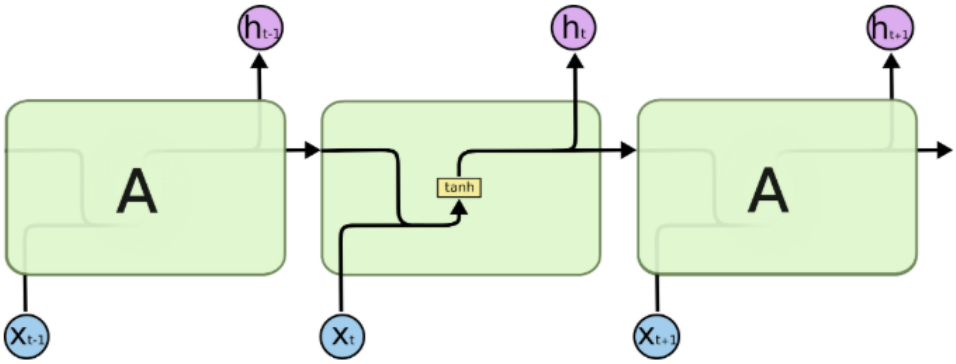
Input gate – it calculates what to store in the cell state. This consists of two steps. Firstly, a sigmoid function selects values to update (eq. (A.3)) and in the same time, $tanh$ function creates a vector of new candidate values $\widetilde{\boldsymbol{C}}_{\boldsymbol{t}}$, that could be added to the state (eq. (A.4)).

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W_i} \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b_i}) \tag{A.3}$$

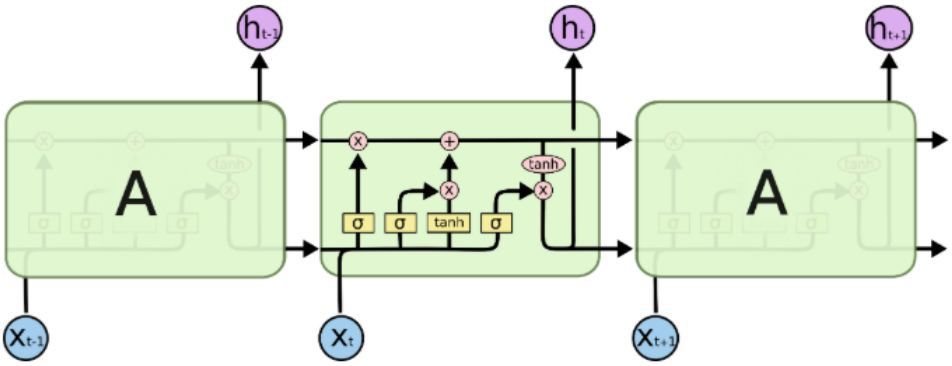
$$\widetilde{\boldsymbol{C}}_t = tanh(\boldsymbol{W_C} \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b_C}) \tag{A.4}$$

To create the new cell state $\boldsymbol{C}_t$, the old state $\boldsymbol{C}_{t-1}$ must be updated with values computed in the gates. The old state has to be multiplied by $\boldsymbol{f}_t$, to „forget” selected values. Then, the new candidate values, scaled with $\boldsymbol{i}_t$ to the level indicating how important the new values are, are added (eq. (A.5)).

$$\boldsymbol{C}_t = \boldsymbol{f}_t \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \widetilde{\boldsymbol{C}}_t \tag{A.5}$$



(a) Schematics of an RNN, picturing one neural activation layer – $tanh$



(b) Schematics of an LSTM network, picturing four neural activation layers – three σ and one $tanh$.

Figure A.2: Simplified schematics of an RNN and LSTM architectures (Source: [12]).

Output gate – the output hidden vector $\boldsymbol{h}_t$ is a filtered version of a state cell $\boldsymbol{C}_t$. The calculation of the vector can be broken down to three steps. Firstly, a squashed cell state vector with $tanh$ layer is created. The resulting vector has values in range from -1 to 1. Then, using values of $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ and a sigmoid layer, a filter is created. The filter is multiplied with the squashed cell state vector, creating a new hidden state $\boldsymbol{h}_t$.

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W_o} \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b_o}) \tag{A.6}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t tanh(\boldsymbol{C}_t) \tag{A.7}$$

Despite the complicated nature of LSTMs, they proved to be successful in many applications concerning sequence modeling. They used to achieve state–of–the art results in the text translation, video classification and other tasks involving pattern recognition in sequences. This work is also using LSTMs as a vital element.

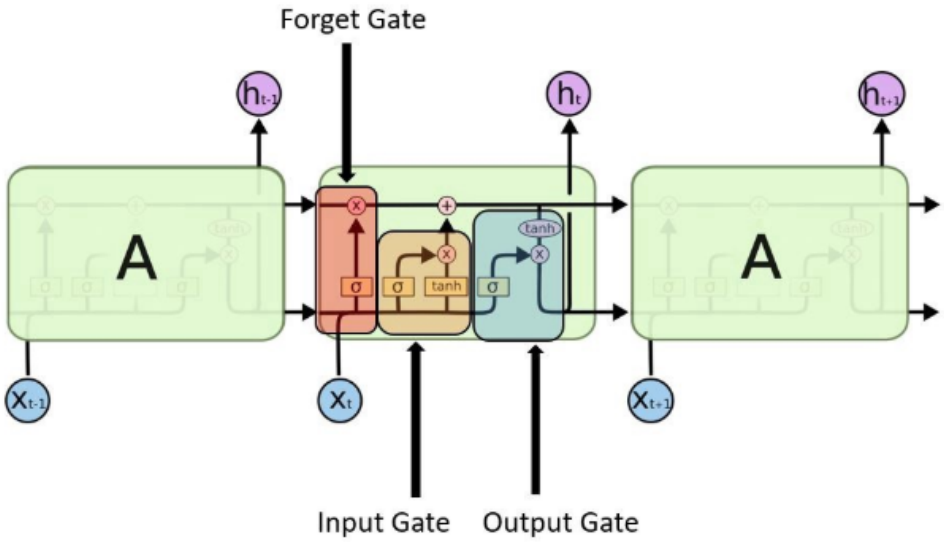


Figure A.3: Schematics of an LSTM network, depicting three gates – forget, input and output (Source: [28])

Bibliography

[1] What is inverse reinforcement learning – goodaudience.com. https://blog.goodaudience.com/what-is-inverse-reinforcement-learning-e333228af146.

[2] Road safety facts – asirt.org. https://www.asirt.org/safe-travel/road-safety-facts/.

[3] Lamberto Ballan, Francesco Castaldo, Alexandre Alahi, Francesco Palmieri, and Silvio Savarese. Knowledge transfer for scene-specific motion prediction. *CoRR*, abs/1603.06987, 2016. URL http://arxiv.org/abs/1603.06987.

[4] B. Benfold and I. Reid. Stable multi-target tracking in real-time surveillance video. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3457–3464, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-0394-2. doi: 10.1109/CVPR.2011.5995667. URL https://doi.org/10.1109/CVPR.2011.5995667.

[5] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe. Global optimization of a neural network-hidden markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259, March 1992. ISSN 1941-0093. doi: 10.1109/72.125866.

[6] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE International Conference on Neural Networks*, pages 1183–1188 vol.3, March 1993. doi: 10.1109/ICNN.1993.298725.

[7] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

[8] Markus Braun, Sebastian Krebs, Fabian B. Flohr, and Dariu M. Gavrila. Eurocity persons: A novel benchmark for person detection in traffic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019. ISSN 0162-8828. doi: 10.1109/TPAMI.2019.2897684.

[9] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[10] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[11] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

[12] colah.github.io. Understanding lstms – colah.github.io blog post. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[13] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *CoRR*, abs/1808.05819, 2018. URL http://arxiv.org/abs/1808.05819.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: socially acceptable trajectories with generative adversarial networks. *CoRR*, abs/1803.10892, 2018. URL http://arxiv.org/abs/1803.10892.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[19] Michael I. Jordan. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pages 112–127. IEEE Press, Piscataway, NJ, USA, 1990. ISBN 0-8186-2015-3. URL http://dl.acm.org/citation.cfm?id=104134.104148.

[20] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[21] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952. doi: 10.1214/aoms/1177729392. URL https://doi.org/10.1214/aoms/1177729392.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[23] Julian Francisco Pieter Kooij, Nicolas Schneider, Fabian Flohr, and Dariu M. Gavrila. Context-based pedestrian path prediction. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 618–633, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10599-4.

[24] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540667229.

[25] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher Bongsoo Choy, Philip H. S. Torr, and Manmohan Krishna Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. *CoRR*, abs/1704.04394, 2017. URL http://arxiv.org/abs/1704.04394.

[26] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. *Computer Graphics Forum*, 26(3):655–664, 2007. doi: 10.1111/j.1467-8659.2007.01089.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01089.x.

[27] Huynh Manh and Gita Alaghband. Scene-lstm: A model for human trajectory prediction. *CoRR*, abs/1808.04018, 2018. URL http://arxiv.org/abs/1808.04018.

[28] Aditi Mittal. Understanding lstms – towardsdatascience.com blog post. https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[29] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *ICCV*, pages 261–268. IEEE Computer Society, 2009. ISBN 978-1-4244-4419-9. URL http://dblp.uni-trier.de/db/conf/iccv/iccv2009.html#PellegriniESG09.

[30] Eike Rehder, Florian Wirth, Martin Lauer, and Christoph Stiller. Pedestrian prediction by planning using deep neural networks. *CoRR*, abs/1706.05904, 2017. URL http://arxiv.org/abs/1706.05904.

[31] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

[32] Nicolas Schneider and Dariu Gavrila. Pedestrian path prediction with recursive bayesian filters: A comparative study. pages 174 – 183, 09 2013. doi: 10.1007/978-3-642-40602-7_18.

[33] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 802–810, Cambridge, MA, USA, 2015. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969239.2969329.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[35] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. *Washington, D.C., U.S. Department of Transportation DOT, National Highway Traffic Safety Administration NHTSA, 2018, 3 p.; NHTSA Traffic Safety Facts Crash Stats; A Brief Statistical Summary ; March 2018 / DOT HS 812 506*, 2018.

[36] Shashank Srikanth, Junaid Ahmed Ansari, Karnik Ram R., Sarthak Sharma, J. Krishna Murthy, and K. Madhava Krishna. INFER: intermediate representations for future prediction. *CoRR*, abs/1903.10641, 2019. URL http://arxiv.org/abs/1903.10641.

[37] Shashank Srikanth, Junaid Ahmed Ansari, Karnik Ram R., Sarthak Sharma, J. Krishna Murthy, and K. Madhava Krishna. INFER: intermediate representations for future prediction. *CoRR*, abs/1903.10641, 2019. URL http://arxiv.org/abs/1903.10641.

[38] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[39] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

[40] Y. Xu, Z. Piao, and S. Gao. Encoding crowd interaction with deep neural network for pedestrian trajectory prediction. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5275–5284, June 2018. doi: 10.1109/CVPR.2018.00553.

[41] H. Xue, D. Q. Huynh, and M. Reynolds. Ss-lstm: A hierarchical lstm model for pedestrian trajectory prediction. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1186–1194, March 2018. doi: 10.1109/WACV.2018.00135.

[42] Yanfu Zhang, Wenshan Wang, Rogerio Bonatti, Daniel Maturana, and Sebastian A. Scherer. Integrating kinematics and environment context into deep inverse reinforcement learning for predicting off-road vehicle trajectories. *CoRR*, abs/1810.07225, 2018. URL http://arxiv.org/abs/1810.07225.

[43] Alex Zyner, Stewart Worrall, and Eduardo M. Nebot. Naturalistic driver intention and path prediction using recurrent neural networks. *CoRR*, abs/1807.09995, 2018. URL http://arxiv.org/abs/1807.09995.