CIMAT

Centro de Investigación en Matemáticas, A.C.

VAE MODELS IN TRAJECTORY PREDICTION

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Orientación en

Ciencias de la Computación y Matemáticas Industriales

Presenta

Jose Antonio Gallardo Monroy

Director de Tesis:

Dr. Jean Bernard Hayet

Guanajuato, Gto., 4 de Noviembre de 2020

CIMAT

Centro de Investigación en Matemáticas, A.C.

VAE MODELS IN TRAJECTORY PREDICTION

T E S I S

Que para obtener el grado de

Maestro en Ciencias

con Orientación en

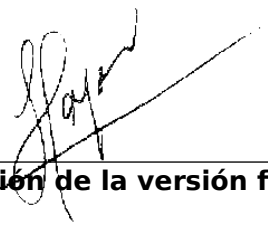
Computación y Matemáticas Industriales

Presenta

Jose Antonio Gallardo Monroy

Director de Tesis:

Dr. Jean Bernard Hayet



Autorización de la versión final

Guanajuato, Gto., 4 de Noviembre de 2020

Dedico este trabajo de tesis a mi hermano Luis Gonzalo y a papá Gonzalo.

Agradecimientos

A Jean Bernard, por asesorarme en la tesis y ser tan bueno conmigo; por tenerme paciencia y apoyarme. Gracias por alentarme a ser mejor y aportar cosas buenas en mi vida.

A Johan Van Horebeek, por ser mi sinodal y revisar la tesis. Aún llevo muchas de sus enseñanzas conmigo, principalmente en redacción. Gracias por estar presente los últimos años, ha hecho una diferencia y parte de lo que soy es gracias a usted.

A Pastor López, por ser mi sinodal y darme comentarios tan valiosos. Le agradezco mucho la ambilidad con la que me trató durante la revisión y tiempo atrás cuando fui su alumno.

I would like to thank Julien Pettre for having me in France. Even when nothing went as expected, I learned a lot of things, and it was an unique and good experience.

I would like to express my gratitude to Janet Izzo for being so kind with me. Talking to you was the push I needed to finish this work.

A Mayté González, por apoyarme en lo que ciertamente ha sido mi etapa más difícil y por ayudarme a ver las cosas de una manera diferente.

A Claudia Esteves, por alegrarme tantos días y estar al pendiente de mi.

A Carlos Segura, por haberme enseñado tantas cosas desde el momento en que llegué a CIMAT. Usted es una de las personas que más admiro.

A Mariano Rivera, por transmitir a sus alumnos la pasión que tiene en aprendizaje máquina. Aprendí y disfruté mucho haciendo sus tareas.

A Israel Becerra, por alentarme a hacer el proyecto del cuál estoy más orgulloso. Gracias por la sencillez y humildad con que se maneja.

I would like to extend my gratitude to Sigfrido for his teachings. Sometimes I was stressed or worried because of other courses, but he always made me feel better with his activities and his good mood.

A Hector Becerra y a Noemí Rentería, por organizar actividades de atletismo. Correr es una de las cosas por las cuáles recordaré esta etapa de mi vida con mucho cariño. Haber compartido algunos de esos momentos con ustedes y otras personas es algo por lo que siempre estaré agradecido.

A mi hermano Luis Gonzalo, por apoyarme en todo lo que hago. Gracias, porque aún cuando eres menor que yo, me has dado el ejemplo en muchas cosas. Estoy muy orgulloso de ti, eres un chico noble, inteligente y bondadoso. Ser tu hermano es sin duda uno de los mejores regalos que me ha dado la vida, te amo.

A Hanna Ehrlich, por su amor y compañía. Gracias por compartir tantas cosas lindas conmigo; por llenar mis días de alegría y adornarlos con chispas de creatividad repentinas. Muchas gracias por todo el cariño y apoyo que me has dado, no imagino esta etapa de mi vida sin ti.

A Julián Candela, por tantos momentos agradables y también incómodos; gracias por cuidar de mi. Doy gracias a Dios porque hayas venido a México y pudiéramos coincidir.

A Rogelio Cruz, por su amistad y paciencia. Gracias por compartir conmigo momentos deportivos e impulsarme a dar siempre un poco más. Espero que la vida nos de la oportunidad de participar juntos en un triatlón.

I would like to thank Alberto Ávila for helping me with English. It was nice to make a friend in México when I was in another country.

A Puri Méndez, por ser mi amiga durante tantos años y recordarme constamente quién soy. Simplemente gracias.

A mi mamá, por no rendirse y buscar siempre una manera de seguir. Gracias, porque cada vez entiendo mejor lo mucho que te has esforzado para que yo esté bien. Gracias por tener tanta fé en mi, te amo.

A papá Gonzalo, porque su cariño y apoyo es uno de mis más grandes pilares. Gracias, porque aún cuando hemos estado lejos, sigo aprendiendo de ti a través de recuerdos.

A mi tío Gon, porque durante esta etapa ha sido mi soporte racional más importante. Gracias, porque además de eso, tu amor me ha fortalecido.

A mi tía Blanca, por su complicidad y apoyo. Hay cosas que solo tú entiendes y me alegra poder recurrir a ti en todo momento.

Al Centro de Investigación en Matemáticas A.C. (CIMAT), por haberme dado la oportunidad de estudiar el grado de maestría en esta institución. Aprendí muchísimo este tiempo.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por haberme otorgado una beca que me permitió vivir en Guanajuato durante mis estudios de maestría.

A Intel, por haberme apoyado económicamente los últimos meses que estuve en Guanajuato, lo cual me permitió concluir con este trabajo. También gracias por el apoyo que me fue otorgado a principios de año para realizar la estancia académica en Francia.

A EDUCAFIN, por el apoyo económico brindado para comprar los boletos de avión a Francia.

Abstract

This work is focused on pedestrian trajectory prediction, an important problem in different contexts, such as robot navigation and autonomous cars. Trajectory prediction has been tackled with different kind of models, but Variational Autoencoders (VAEs) are among the less explored; this is mainly because other authors have focused on the most popular generative models: Generative Adversarial Networks (GANs). Even when GANs perform better than VAEs in other tasks, VAEs are end-to-end models and therefore easier to train. In this thesis, the aim is to explore the potential that VAE architectures have in trajectory prediction from a practical perspective.

We propose two VAE architectures. Both models use LSTMs to encode and decode. The difference is the nature of the latent space, which can be continuous or discrete. In the VAE with continuous latent space, as usual, it is assumed that the latent variables follow a standard normal distribution. The VAE with discrete latent space is a simplification of a model presented by Salzmann et al. [22], where the authors assume a discrete categorical latent variable z (just one variable $z \in \{1, \ldots, k\}$).

We work with two datasets. The first one is a popular trajectory prediction benchmark known as ETH-UCY. In this benchmark, both our continuous and discrete latent variable VAE models can compete with state of the art methods. Ablation studies are realized to have a better insight of how each component affects the models. The second dataset is an artificial crossroad dataset that we made in Unity (a very popular video game engine) to explore the incorporation of scene maps to the models. The results on Crossroad are not compared with other methods results, but we perform a local comparison and ablation studies as well.

Contents

List of Figures

List of Tables

Acronym List

ADE Average Displacement Error. 16, 17, 43

AE Autoencoder. 10, 11, 27–29, 72, 78–82

APG Angular Pedestrian Grid. 10, 12, 59, 60, 74, 85

convLSTM Convolutional Long Short-Term Memory. 43, 44

FDE Final Displacement Error. 17, 43

FPS Frames Per Second. 37

GAN Generative Adversarial Network. 18, 47, 63

GMM Gaussian Mixture Model. 49, 51, 52

IPA Invalid Positions Amount. 73–75, 80, 82, 83

KL Kullback–Leibler. 10, 11, 30, 48, 49, 54–58, 79, 80

LSTM Long Short-Term Memory. 10, 26, 27, 38–40, 43–46, 50–53, 64, 72

mADE Minimum Average Displacement Error. 10, 17, 53, 56–59, 62–64, 71, 73–75, 82, 83

mFDE Minimum Final Displacement Error. 17, 56, 59, 62–64, 73–75, 82, 83

NLP Natural Language Processing. 18, 51, 64

NN Neural Network. 16, 19, 21–24, 26, 27, 29, 31, 32, 41, 43, 48, 49, 52, 59, 67, 69

ReLU Rectified Linear Unit. 22, 23

RNN Recurrent Neural Network. 18, 26, 29

SDVAE Simplified Discrete latent variable Variational Autoencoder. 10–12, 49, 50, 53, 59–65, 69, 71, 72, 74–77, 79, 80, 82–86

SS Semantic Segmentation. 10–12, 15, 19, 25, 38, 42–45, 71–75, 77, 78, 82, 83

VAE Variational Autoencoder. 10–12, 18, 19, 29–31, 44–50, 53, 55–58, 61–64, 67, 71, 72, 82–86

VVAE Vanilla Variational Autoencoder. 12, 61, 63

Chapter 1

Introduction

There are many science fiction works where the authors explore ideas for the future. Some of them involve robots immersed in the society or autonomous cars on the streets, but it is hard to design and imagine systems that could make inference as human beings do. For all kind of inference in autonomous systems, understanding the development of a dynamic scene in the short and in the long term is essential to plan. Depending on the goal, there could be several aspects to be considered. For example, take the case of moving an agent from one point to another. If the agent is a robot in a crowded environment, with the aim of avoiding collisions, it is essential to know where the robot is allowed to be and how other agents such as pedestrians are moving. If the agent is an autonomous car in the street, it has to stay out of the sidewalks and give preference to the pedestrians. It also has to deal with more cars on the street and different kind of vehicles, such as bicycles.

The examples I comment above are the main inspiration and motivation for this thesis, which is focused on pedestrian trajectory prediction. The problem consists in trying to predict the next positions of a pedestrian given the last ones, that have been observed. In some cases, there is more available information to do the prediction, such as the Semantic Segmentation (SS) over an image of the surroundings, but in general, there could be nothing else but the observed positions.

Although there is no complete understanding of how the data is internally used by human beings, they can naturally collect it and make decisions. The agents (robots, autonomous cars) need to collect data to understand the scene as well. It is collected

by sensors, such as cameras, but even when these are maybe the most popular devices, extracting data from images involves some processing. This leads to a bunch of vision problems, such as target tracking, object detection, image classification, segmentation, depth estimation and object reconstruction. It is believed that all this kind of information is used unconsciously by people to trace a trajectory, and it is the reason why much research aims to effectively incorporate additional features to trajectory prediction models.

Trajectory prediction has been studied from many different approaches and covering all of them is beyond the scope of this modest academic work. In this work, we use Neural Networks (NNs). These parametric models achieve the state of the art results in the problem of interest. NNs do not only lead to the best results in this problem, but also in many others. In contrast with other models, NNs allow an easy integration of features of different nature. Mainly, the problem is to design the structure of the network.

1.1 Problem Statement

The problem we are interested in is very easy to state and, actually, it was already done above. This section is presented for describing it in more mathematical terms and for explaining how the models are evaluated.

Problem statement: given the last observed positions $x = [x_1 \ \dots \ x_l]^T \in \mathbb{R}^{l \times 2}$ of an agent, the problem is to predict its next positions $y = [y_1 \ \dots \ y_{l'}]^T \in \mathbb{R}^{l' \times 2}$. All the agent positions x_i and y_j are in $\mathbb{R}^2$. In total, there are $l + l'$ time steps $[x_1 \ \dots \ x_l \ y_1 \ \dots \ y_{l'}]^T \in \mathbb{R}^{(l+l') \times 2}$ and consecutive observations are supposed equidistant in time. In the Figure 1.1, an example of the observed and predicted trajectories is shown.

To evaluate the quality of a prediction $\hat{y} = [\hat{y}_1 \ \dots \ \hat{y}_{l'}]^T \in \mathbb{R}^{l' \times 2}$, the Euclidean distances $d_i = d(y_i, \hat{y}_i)$ are computed for each timestep i, as depicted in Figure 1.1. Two metrics are defined based on these distances. The first one is the Average Displacement Error (ADE):

$$ADE(y, \hat{y}) = \frac{1}{l'} \sum_{i=1}^{l'} \|y_i - \hat{y}_i\|, \tag{1.1}$$

and the second one is the Final Displacement Error (FDE):

$$FDE\left(y, \hat{y}\right) = \left\| y_{l'} - \hat{y}_{l'} \right\|. \tag{1.2}$$

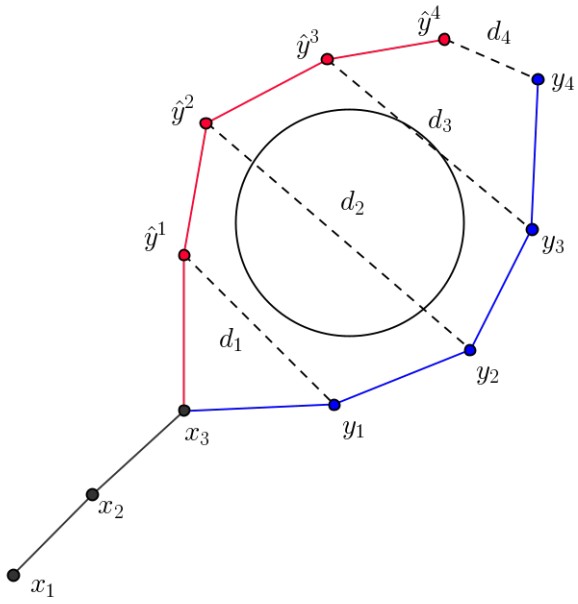


Figure 1.1: Example of an observed trajectory $x = [x_1\ x_2\ x_3]^T$ with the (true) future positions $y = [y_1\ \dots\ y_4]^T$ and a possible prediction $\hat{y} = [\hat{y}_1\ \dots\ \hat{y}_4]^T$. The Euclidean distances $d_i = d\left(y_i, \hat{y}_i\right)$, for $i = 1, \dots, 4$, are computed to evaluate the prediction $\hat{y}$.

Models at the state of the art usually output more than one prediction. To evaluate a model that gives k predictions as an output, $\hat{Y} = \left\{\hat{y}^1, \dots, \hat{y}^k\right\}$, where each prediction $\hat{y}^i \in \mathbb{R}^{l' \times 2}$, the metrics ADE and FDE are extended as follows. Among the k predictions, only the best is considered. The Minimum Average Displacement Error (mADE) is defined as

$$mADE\left(y, \left\{\hat{y}^1, \dots, \hat{y}^k\right\}\right) = \min_{1 \le i \le k} \left\{ADE\left(y, \hat{y}^i\right)\right\}, \tag{1.3}$$

and the Minimum Final Displacement Error (mFDE) is defined as

$$mFDE\left(y, \left\{\hat{y}^1, \dots, \hat{y}^k\right\}\right) = \min_{1 \le i \le k} \left\{FDE\left(y, \hat{y}^i\right)\right\}. \tag{1.4}$$

This problem is going to be tackled with an encoding-decoding scheme (see Section 2.5), where both encoder and decoder are Recurrent Neural Networks (see Section 2.4). This kind of networks were introduced by Cho et al. [3] in the area of Natural Language Processing (NLP) to cope with sequence-to-sequence problems, such as machine translation. Since trajectory prediction is also a sequence (observed trajectory) to sequence (predicted trajectory) problem, the community has applied many ideas from NLP to trajectory prediction.

It is important to notice that, given an observed trajectory, the prediction is not always clear and may have multiple modes. For example, in Figure 1.1, the prediction $\hat{y}$ is totally understandable. This is why state of the art models give multiple outputs instead of a single prediction. In Figure 1.1, we could say that there are at least two modes: one where the agent goes to the left, and another where it goes to the right. In this work, we use Variational Autoencoders (see Section 2.6), which follow an encoding-decoding scheme, and allow us to generate multiple predictions.

There are different flavors in state of the art methods. As our main research direction, we have neural generative models: Generative Adversarial Networks (GANs) and VAEs. Mainly, the efforts have been directed toward using GANs to improve the quality of the predictions [9, 12, 21], but some works have shown that VAEs have great potential as well [11, 15, 22]. On the other hand, there are neural models which handle a grid of the scene, and a distribution over the grid is learned. That strategy has been used to give a fixed number of predictions [18, 20], or an infinite amount of predictions following sampling approaches [7, 17].

We use a VAE model because we consider it as a natural extension of a single output encoding-decoding model, and because VAEs are easier to train than GANs. We work with two datasets. The first one is a popular trajectory prediction benchmark known as ETH-UCY. In this benchmark, our models can compete with state of the art methods, and ablation studies are realized to have a better insight of how each component affects the models. The second dataset is an artificial crossroad dataset that we made in Unity (a very popular video game engine) to explore the incorporation of scene maps to the models. The results on Crossroad are not compared with other methods results, but we perform a local comparison and ablation studies as well.

1.2 Aim and Objectives

Trajectory prediction has been tackled with different kind of models, but Variational Autoencoders (VAEs) are among the less explored. Even though there are some very pertinent related works to ours, we notice that too often the authors do not share the code to replicate the results. In this thesis, the aim is to explore the potential that VAEs have in this problem and to provide the community with a clean and well organized code to reproduce the results. Punctually, the objectives are:

- Doing a review of the state of the art in pedestrian trajectory prediction.
- Designing and implementing a couple of VAE architectures for this problem.
- Getting and processing a recognized benchmark to have a point of comparison with other methods in the literature.
- Designing and creating an artificial dataset to explore the incorporation of Semantic Segmentation (SS) to improve the quality of the predictions.
- Testing the proposed architectures with the obtained and generated data, and realizing ablation studies to have a better insight of how each component affects the overall performance.

1.3 Thesis Structure

The thesis is organized as follows:

- Since the proposed models are Neural Networks (NNs), a brief summary of NNs is given in the Chapter 2. More than an extensive and detailed text, it is rather a gentle introduction. References for a further understanding are indicated in that chapter.
- The datasets in which we have worked are described in Chapter 3. In the Section 3.1, the ETH-UCY benchmark is explained. This dataset is used to compare the proposed VAE models with existing methods in the literature. In the Section 3.2, the artificial dataset that we have designed to work with SS

is detailed. In this dataset, there is no comparison with other methods, but a local comparison among our proposed models is performed.

- The proposed models are explained in Chapter 4. Also, the state of the art is summarized through this Chapter. Before explaining our main two proposals, an introductory model is explained in the first Section; then, each proposed model has its own Section. After explaining a model, its relation with the state of the art is explained to put it in perspective.

- The results are presented in the Chapter 5. In the Section 5.1, the ETH-UCY dataset results are presented; in the Section 5.2, the results obtained with our artificial dataset are presented. Our code is available on GitHub [1].

- Finally, the conclusions and future work are stated in the Chapter 6.

[1] https://github.com/jagmonroy/cvae_tp

Chapter 2

About Neural Networks

In this chapter, a brief Neural Networks (NNs) summary is given to understand this work. I encourage the interested reader in peeping into specialized works [8] for much more details than what we recall here.

2.1 Chapter Introduction

NNs are parametric models that have been widely used in the last years. It is important to stress that, in spite of the recent "hype", they are not a new approach. Their origin can be traced back to 1958, when Frank Rosenblatt introduced the perceptron [29]. Backpropagation, the most popular algorithm to compute the gradient in NN training, was introduced by Paul Werbos in 1974 [29]. There are two good reasons for revisiting neural networks decades after their introduction: faster computers and much more data available.

In video games, it is required to realize expensive matrix operations and many people have been developing new hardware technologies to do it faster. As a consequence, GPU's were created. These devices can perform a lot of simple operations (in the order of hundreds or even thousands), such as sums and multiplications, in parallel. At the same time, the most basic NN models depend heavily on matrix multiplications, vector additions and applications of functions to all the entries of tensors (multi-dimensional arrays); even more complicated models are based on those basic operations. This is why GPU's have been intensively used to train and test

neural networks.

In some cases, simpler parametric models perform similar to NNs. When the model needs to capture a linear behavior, to give an example, classic models may be enough. When a nonlinear behavior is introduced, kernel methods [23] have been the most important tool before NNs (re-)emerged. The idea of these methods is to transform the data into another space and then use the classic and well known methods in that space. The problem is that it is not always clear which transformation to use to transform the data to find a linear pattern. In some way, during the NN optimization process, some similar process occurs, as NN weights are adjusted with the aim of transforming the data and enhancing linear patterns in the new space. The advantage of this, is that it is not necessary to think much about the explicit transformation. The disadvantage is that some interpretability is lost. In this sense, NNs have been criticized, but the fact is that they achieve the state of the art in several areas.

Most of the current NNs applications are limited to real numbers: vectors, matrices and tensors. All the models in this work cope with real numbers.

2.2 Dense Layers

Nowadays, there are plenty of ideas involved in NNs and, regardless their origin, it is common to start talking about layers. Specifically, *dense layers*. They can be seen as functions that map a real vector to another real vector. These layers consist of combining the application of an affine transformation and then a nonlinear transformation:

$$\begin{aligned} f :& \mathbb{R}^n \to \mathbb{R}^m \\ & x \mapsto f(x) = g\left(Ax + b\right), \end{aligned} \tag{2.1}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $g : \mathbb{R} \to \mathbb{R}$ is applied to each entry of $Ax + b$. The function $g\left(\cdot\right)$ is called *activation function* and there are a few popular choices. For example, the sigmoid function, the hyperbolic tangent function, the Rectified Linear Unit (ReLU) function and even the identity function are some of them.

Another important concept is the one of *depth*, which is, roughly, the number of layers the model has. It is possible to think of the function $f(\cdot)$ of Eq. 2.1 as a NN with depth one. A two layers-deep NN with dense layers has the form

$$NN(x) = f^2\left(f^1\left(x\right)\right),$$

where $f^1\left(\cdot\right)$ and $f^2\left(\cdot\right)$ are consecutive dense layers. An l layers-deep NN with only dense layers is the composition of l dense layers. It should be noticed that the codomain of $f^i\left(\cdot\right)$ and the domain of $f^{i+1}\left(\cdot\right)$ must be the same for $i \in \{1, 2, \ldots, l-1\}$ to ensure consistency.

In this kind of networks, the selection of the element-wise activation function $g^i\left(\cdot\right)$ depends on whether $i \neq l$ or $i = l$ (last layer). For $i \neq l$, we usually use the same function, which could be the sigmoid, $\tanh\left(\cdot\right)$ or ReLU. The last activation function $g^l\left(\cdot\right)$ is chosen depending on the problem: in a regression problem, the identity function could be used; for a binary classification problem, the sigmoid function; in a multi-class classification problem, a softmax function. Nothing of this is a rule. As far as I know, in general, there is no mathematical way of knowing when a structure is better than another, that is why testing different configurations is important. Usually, a novel work in NNs brings a new architecture which performs better in a specific problem, according to some criteria. These novel works also use to be more experimental than theoretical, but there are many flavors.

There is an important result that applies to NNs with only dense layers, known as *the universal approximation theorem*. It says that any continuous function can be approximated as close as you want with this kind of NNs [16]. It is just an existence theorem and it does not say how to find the parameters, however it is useful to have this result in mind.

2.3 Useful Operators

Dense layers can be seen as the base element of NNs, but they have some issues, such as the required memory and the cost of computing the output when the dimension of the space where the input belongs is large. In the face of these adversities, researchers have proposed different kind of alternative layers. Actually, talking about what a

layer is becomes subjective, and depends on the author. This section briefly explains three operations that are common with multi-dimensional arrays and useful in order to construct NN layers. These operations are reshaping, concatenating and slicing.

Reshaping consists in modifying the tensor shape. In this work, reshaping is used to pass from a matrix to a vector and vice versa. For example, the matrix

$$M = \left[\left[1, 2, 3\right], \left[4, 5, 6\right]\right]$$

has shape $(2, 3)$, and reshaping to (6) leads to the vector

$$v = \left[1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6\right].$$

Similarly, applying a reshaping operation to the vector v into dimensions $(2, 3)$, makes it go back to the matrix M. This operation can be useful when images are involved.

Concatenating "combines" vectors along a dimension. For example, the tensor $[[1, 2]]$ has shape $(1, 2)$. If it is concatenated with itself along the first dimension, the result would have shape $(2, 2)$:

$$\left[\left[1, 2\right], \left[1, 2\right]\right].$$

If it is concatenated with itself along the second dimension, the result would have shape $(1, 4)$:

$$\left[\left[1, 2, 1, 2\right]\right].$$

This operation is used in different situations, but a simple example is the one of combining features in the same array: if there is an array with 6 features and another one with 8 features, concatenating them allows to have finally one array with the 14 features.

Slicing takes a slice of an array. For example, taking a row of the matrix M is easy because of the way it is stored in memory, but extracting a column is not straightforward. A C programmer would just use a for loop to get the data. In Python, using loops for big arrays is a bad idea, and it is faster to use slicing operations. The command to extract the first column from the matrix M, would be

$M[:, 0]$. The two points ":" in the first index ask to iterate over all the elements along the first dimension (rows), and the second index 0 is for conserving the first position of all the rows.

We are going to see in this thesis that a way to incorporate the information coming from Semantic Segmentations (SSs), is by coping with n image sequences stored in a 4-dimensional array IS. If the sequence length is t and the images have size $w \times h$, IS has shape (n, t, w, h). To get a planar representation of each image, a function $f(\cdot)$ can be applied, but some functions only take as input 3-dimensional objects. Slicing can be used in that case to get $f(IS[:, i])$ for $i \in \{1, 2, \ldots, t\}$, and then concatenating the t outputs from the consecutive application of $f(\cdot)$ allows to create a sequence. Each slice $IS[:, i]$ has shape (n, w, h).

2.4 Convolutional and Recurrent Layers

Depending on the nature of the data, different kind of layers can be used. In the case of images, convolutional layers are the most popular choice. When sequences are involved, recurrent layers are perhaps the first option.

A simple approach to process images is to flatten them to use dense layers: if the image has shape (n, m, c), it is reshaped to $(n \cdot m \cdot c)$. However, by doing this, the Markovian structure of the images (where close pixels values are correlated) is more difficult to be learned by the model. On the other hand, processing big images with dense layers could be very expensive. A convolutional layer uses convolutions (strictly speaking, it uses correlations), a well known image processing tool [5]. The idea is to find automatically some "good" filters to extract convenient features and solve a higher level problem (classification, regression...) with these extracted features. The values of the filters, also called kernels, are adjusted during the training phase. These layers became popular since the publication of AlexNet [13], in 2012. Krizhevsky et al. [13] used convolutional layers and they achieved impressive results (at that time) in a natural image classification problem. Some of the filters in the first convolutional layer of AlexNet [13] resemble Gabor filters. Compared with dense layers, convolutional layers reduce the amount of parameters and they take advantage of the two dimensional structure.

In the same way as I commented for images, a sequence of n vectors $[x_1 \ \dots \ x_n]^T \in \mathbb{R}^{n \times d}$ can be concatenated to get a vector in $\mathbb{R}^{n \cdot d}$ and then we could use a dense layer to process it. This has similar issues as using flattened images: $n \cdot d$ can be big and the natural temporal structure of the data (a Markov model, for example) is lost. There is a kind of NNs designed to deal with sequences: Recurrent Neural Networks (RNNs).

RNNs use recurrent layers. There are different layers of this kind, but the most basic of them is known as recurrent layer. It manages an internal state $h_t \in \mathbb{R}^{d'}$ and gives an output $out_t \in \mathbb{R}^{d'}$; at each time step t of the sequence, the state h_t and the new input x_{t+1} are used to compute the next state h_{t+1}. The dimension d' is a hyperparameter. The vectors h_t and out_t can theoretically have a different dimension, but, in practice, they are usually the same. If there is nothing else before the RNN, the initial state h_0 can be the null vector. The recurrence is given by:

$$\begin{aligned} h_t\left(h_{t-1}, x_t\right) &= g^1\left(A_h h_{t-1} + A_x x_t + b_1\right), \\ out_t\left(h_t\right) &= g^2\left(A h_t + b\right), \end{aligned}$$

where $g^1(\cdot)$ and $g^2(\cdot)$ are activation functions and the entries of $A_h \in \mathbb{R}^{d' \times d'}$, $A_x \in \mathbb{R}^{d' \times d}$, $A \in \mathbb{R}^{d' \times d'}$, $b_1 \in \mathbb{R}^{d'}$ and $b \in \mathbb{R}^{d'}$ are network parameters to adjust during the training phase.

The idea is to try to encode the first t sequence terms $x_1, \dots, x_t$ through the internal state h_t and then use it to give an output out_t. Usually, the activation functions $g^1(\cdot)$ and $g^2(\cdot)$ are $\tanh(\cdot)$. However, since the $\tanh(\cdot)$ derivative codomain is $(0, 1]$, training models with these layers can be hard because of the vanishing gradient problem (using the chain rule, it is possible to see that, in the end, there would be n multiplied factors, all between 0 and 1, in some gradient terms).

There is another kind of recurrent layers called Long Short-Term Memory (LSTM). It has more elements and handles two internal states $c_t \in \mathbb{R}^{d'}$ and $h_t \in \mathbb{R}^{d'}$, instead of one. The dimension d' is, again, a hyperparameter. There are three gates to weight the states:

$$f_t\left(h_{t-1}, x_t\right) = \sigma\left(A_f h_{t-1} + B_f x_t + b_f\right),$$
$$i_t\left(h_{t-1}, x_t\right) = \sigma\left(A_i h_{t-1} + B_i x_t + b_i\right),$$
$$o_t\left(h_{t-1}, x_t\right) = \sigma\left(A_o h_{t-1} + B_o x_t + b_o\right).$$

The first two components f_t and i_t are used to weight two terms and compute the state c_t:

$$c_t = f_t \circ c_{t-1} + i_t \circ g^1\left(A_c h_{t-1} + B_c x_t + b_c\right),$$

where $\circ(\cdot,\cdot)$ is the Hadamard product (element-wise product). Since the sigmoid function $\sigma(\cdot)$ has codomain $(0,1)$, f_t determines "how much" the history c_{t-1} is taken into account in c_t. The term $g^1\left(A_c h_{t-1} + B_c x_t + b_c\right)$ tries to capture how the state h_t evolves with the new observation x_t and i_t determines "how much" it has to be considered. The state h_t is computed with c_t and o_t:

$$h_t = o_t \circ g^2\left(c_t\right).$$

The interpretation of o_t is analogous. An output out_t is computed by passing h_t through a dense layer as in recurrent layers. The parameters to adjust during training are the entries of A_f, B_f, A_i, B_i, A_o, B_o, A_c, B_c, b_f, b_i, b_o and b_c. Their dimensions depend on d and d', and it is not complicated to find them. The activation functions $g^1(\cdot)$ and $g^2(\cdot)$ use to be $\tanh(\cdot)$. The vanishing gradient problem is less common in LSTMs, since all the states h_t are better related because of f_t, i_t and o_t.

2.5 The Encoding-Decoding Scheme

A kind of NNs named Autoencoders (AEs) follow the encoding-decoding scheme. As we make heavy use of this concept in this thesis, first, I am going to explain AEs and then I am going to generalize it to the encoding-decoding scheme.

The AE objective is to reduce the data dimensionality. It uses two functions: an encoder $enc(\cdot)$ and a decoder $dec(\cdot)$. Both are implemented as NNs. If the data live in a space X, the encoder tries to extract d-dimensional features from the data:

$$enc : X \to \mathbb{R}^d$$

$$x \mapsto enc(x).$$

To have a dimensionality reduction effect, d has to be lower than the dimension of X. The decoder goes back to the original space X:

$$dec : \mathbb{R}^d \to X$$

$$x \mapsto dec(x).$$

If the encoder does a good job in extracting features, then it would be possible reconstruct the input x from $enc(x)$ with the decoder. The reconstruction is

$$\hat{x} = dec\left(enc\left(x\right)\right).$$

To adjust the parameters of the encoder and the decoder during training, a loss function $L\left(x, \hat{x}\right)$ is defined. For example, if $X = \mathbb{R}^{d'}$, the encoder function $enc\left(\cdot\right)$ and the decoder function $dec\left(\cdot\right)$ could be dense layers. In this case, the loss function could be the squared reconstruction error:

$$L\left(x, \hat{x}\right) = \left\|x - \hat{x}\right\|^2.$$

For several instances, the average over the instances is taken.

In a more general context, the encoding-decoding scheme extracts features from the input data with an encoder function:

$$enc : A \to \mathbb{R}^d$$

$$x \mapsto enc(x).$$

Then, the decoder uses those features to get something of interest, not necessarily the reconstructed data as in the case of AE:

$$dec : \mathbb{R}^d \to B$$

$$x \mapsto dec(x).$$

An AE is a particular case such that $A = B$. There are several examples of the encoding-decoding scheme. An interesting example are NNs used to describe images by text. In this case, the model receives an image and it extracts features with the encoder; the decoder uses those features to get a text description. The encoder could use convolutional layers and the decoder could be a RNN.

2.6 Variational Autoencoders

In this section, Variational Autoencoders (VAEs) are introduced from a geometric perspective, which is more intuitive than the statistical formulation, in my opinion. For a first formal review, I consider Doersch [4] to be a very good tutorial.

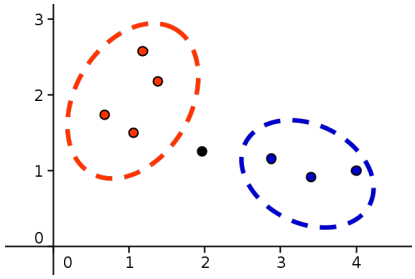


Figure 2.1: Fictional AE embedding of two classes.

AEs have been thought to do dimensionality reduction, as explained above. In general, they are not useful to generate new data. Decoding random points from the domain decoder could lead to "weird" things. Let us consider a simple and fictional example: an AE trained to map several images of the digits 1 and 8 to $\mathbb{R}^2$. As depicted in the Figure 2.1, the digits 1 could be mapped inside the red curve and the digits 8 inside the blue one (or the opposite). If a point inside a curve is decoded, the result would probably be much alike a digit 1 or a digit 8. The decoder should have learned to decode the points that the encoder gave during the training phase, but there is no guarantee for points outside the curves.

A VAE takes care of the latent space to generate new data. It assumes a latent variable $z \in \mathbb{R}^d$ that follows a given distribution $p(z)$. In this way, the encoder is

asked to map the input x to a conditional distribution $q(z|x)$, from which samples can be drawn easily (e.g., a Gaussian). In order to keep $q(z|x)$ close to $p(z)$, a penalization term is included in the loss function:

$$D\left(q(z|x)\|p(z)\right),$$

where $D\left(\cdot,\cdot\right)$ is a divergence. It is not a distance, but it tries to quantify how far is the distribution $q(z|x)$ from the distribution $p(z)$. Usually, the Kullback–Leibler (KL) divergence $D_{KL}\left(\cdot,\cdot\right)$ is taken.

To decode, during training, a sample from $q(z|x)$ is taken and the decoder tries to reconstruct the input from this sample:

$$L\left(x,\hat{x}\left(z_0\right)\right),$$

where $z_0 \sim q(z|x)$. Putting the two terms together, the loss function is:

$$L\left(x,\hat{x}\left(z_0\right)\right) + D\left(q(z|x)\|p(z)\right),$$

where the first term is the reconstruction loss and the second one is used for keeping $q(z|x)$ close to $p(z)$, as mentioned above.

In practice, it is assumed that z follows a standard normal distribution $p(z)$; the approximated posterior distribution $q(z|x)$ is assumed normal as well and the encoder learns its statistics $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$: $\mathcal{N}\left(\mu,\sigma\cdot I\right)$. There is a closed formula to compute

$$D_{KL}\left(\mathcal{N}\left(\mu,\sigma\cdot I\right)\|\mathcal{N}\left(0,I\right)\right),$$

which simplifies the expression of the loss.

Considering again the same simple and fictional example that we described before (the one of the digits): in the best case, in the VAE latent space, the data clouds are located around the origin and close enough of each other, as depicted in Figure 2.2. If this is achieved, a random point in this latent space would be in a zone that the decoder knows and the decoding (mapping from the latent space from the original space) would make sense. It is important to notice that the region space corresponding to a category does not have to be a Gaussian. What it is assumed

Gaussian is the latent space distribution and $q(z|x)$.

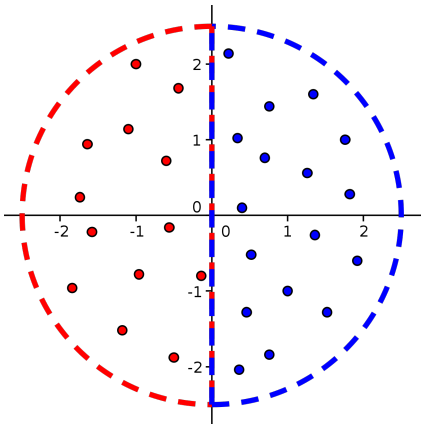


Figure 2.2: Fictional and ideal VAE embedding of two classes.

2.7 General Comments

The potential of NNs has been experimentally explored, but in most cases, the theory is still a gray zone. There are models that are better understood than others, but in practice they are often not as good as others, such as VAEs. Recently, Vahdat and Kautz [27] proposed a nouveau (new) VAE and, as far as I know, they are the first known authors who successfully trained a VAE to generate 256×256 images of people. They do not study the statistical properties of their model, and the proposal is mostly based on experience and experimental observations.

Optimization has its own problems. Since the NNs parameters are adjusted using gradient-based methods, all the limitations that optimization has are inherited in NNs. For example, some deeper models get lower performance, even when some parameter configurations can get exactly the same results than less deep models. He et al. [10] noticed this phenomenon and they proposed an architecture called ResNet for deeper models. To show that their proposal facilitates deeper models training, they successfully trained a 101-depth NN to classify natural images into 1000 different categories.

Libraries such as Tensorflow and PyTorch have made NNs research much easier. The advantage of these libraries is that, as long as you can build the network using

their functions, they can automatically compute the gradient during training with backpropagation. Nevertheless, even when a lot of people can code NNs, their most amazing applications require resources that few people have.

Chapter 3

Datasets

In this Chapter, the datasets used in the experiments are presented. Each dataset has its own Section.

3.1 ETH-UCY

In pedestrian trajectory prediction, maybe the most popular benchmark is ETH-UCY. It consists of five different scenes with pedestrians: ETH, HOTEL, UNIV, ZARA1 and ZARA2. The first two scenes (ETH and HOTEL) belong to ETH Zurich and can be found on their Computer Vision Group page [1].

Figure 3.1: Frames from the ETH and HOTEL sequences, respectively.

[1] https://icu.ee.ethz.ch/research/datsets.html (2020-07-10).

The other three scenes (UNIV, ZARA1 and ZARA2) come from the UCY Computer Graphics Lab [2]. The ZARA1 and ZARA2 sequences are filmed at the same place at different times.

Figure 3.2: Frames from UNIV, ZARA1, and ZARA2 videos, respectively.

The authors behind ETH-UCY have applied a homography (i.e., a 2D projective mapping) to have an approximation of the pedestrians world coordinates in meters, given their position in the image. In the trajectory prediction community, these trajectories are taken to train and test the model. It is hard to know who initially proposed this benchmark, and the oldest paper I could track is Social-LSTM [2]. Yamaguchi et al. [30] worked with it as well, but with a slightly different protocol.

In this benchmark, the prediction model is tested on each scene, which means that there are typically five different reported tests. To test the model on a scene, all the trajectories of that scene are reserved for testing; the trajectories from the other four scenes are used to train and validate the model. Social-GAN [9], a work that can be seen as a continuation of Social-LSTM [2], provides the code for splitting the data into training, validating and testing [3]. The training and validation data contain trajectories of the four scenes.

To tell the truth, I consider ETH-UCY problematic. Some authors try to compare their results with those of others but they do not always follow the same protocol. Some of them work with their own partitions. Strictly speaking, if the partition is changed, then the experiment is not the same. On the other hand, the ETH scene contains longer trajectories than the other scenes, because it has been acquired with

[2] https://graphics.cs.ucy.ac.cy/research/downloads/crowd-data (2020-07-10).
[3] https://www.dropbox.com/s/8n02xqv3l9q18r1/datasets.zip?dl=0 (2020-07-10).

a different framerate. This last detail was indicated by [7, 31] and commented in the Social-GAN [9] repository [4]. Some authors "correct" the length of ETH trajectories and they achieve "better" results, but as I said, it is no longer the same experiment and it is not comparable. In order to avoid problems, it is better to work with the split data as it is given, for a fair comparison among methods.

The intention of Social-GAN [9] was to predict 12 positions in the next 4.8 seconds, given 8 positions in the last 3.2 seconds. As far as I know, the only scene which does not satisfy this property is ETH. According to [31], the time duration between 6 frames corresponds to 0.4 seconds. Social-GAN [9] took the observations every 10 frames and this means that, in this scene, the problem is to predict 12 positions in the next 7.92 seconds, given 8 positions in the last 5.28 seconds, which is a different problem per se (and, a priori, a more difficult one). In all the cases, the goal is to predict the next 12 positions given the last 8 positions. The same time duration is kept between each pair of observations. This time duration is 0.66 seconds in ETH and 0.4 seconds in the others. In Table 3.1, the number of trajectories with length 20 ($= 8 + 12$) is reported for each scene.

Scene	Amount of trajectories with length 20
ETH	364
$HOTEL$	1197
$UNIV$	24334
$ZARA1$	2356
$ZARA2$	5910

Table 3.1: Amount of trajectories in ETH-UCY.

Even though ETH has fewer trajectories than other scenes, its impact in the results is important because of the challenge that longer trajectories implies, as I have mentioned it.

Information about pedestrians density is another important parameter, and it can be found in the TrajNet challenge [5]. This is another benchmark, but it is useful to get an idea.

[4] https://github.com/agrimgupta92/sgan/issues/84 (2020-07-10).
[5] http://trajnet.stanford.edu/data.php?n=1 (2020-07-12)

3.2 Crossroad

This second dataset comes from an artificial crossroad that I made in Unity3D [26], with free assets provided by Adobe Mixamo [1]. There are only pedestrians and they do not get out of the paths. Only one scene was created. A camera is collocated above the scene and the trajectories are saved in pixel coordinates. This allows to interpret the segmentation as a map of the scene. The trajectories are generated with four different camera configurations. The intention is to use three configurations to train and the last one to test. In the Figure 3.3, images of the four configurations and their respective segmentations are shown.

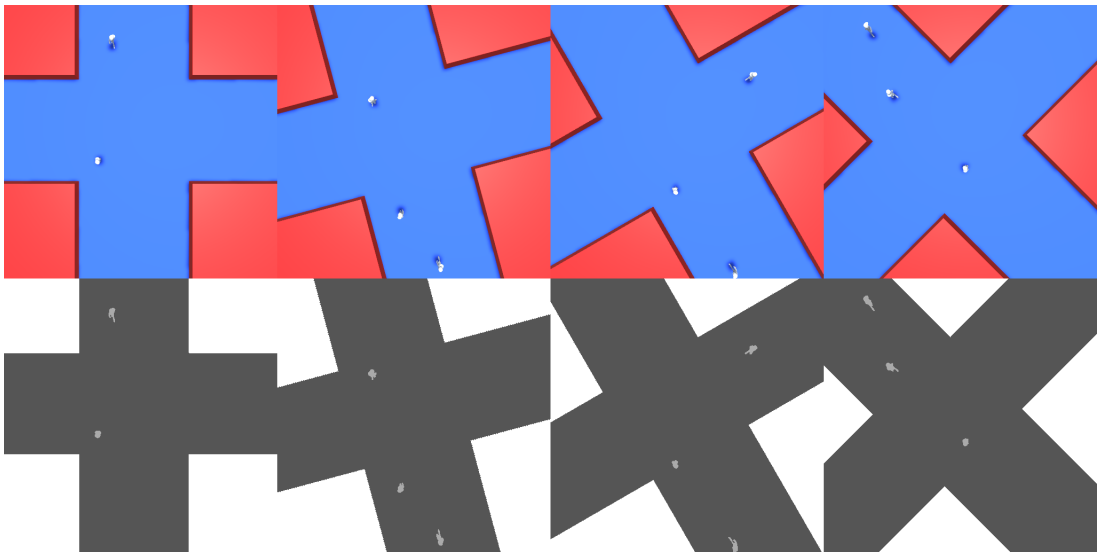


Figure 3.3: Synthetic crossroad data examples. The default setting is the one on the left. The second configuration from left to right results from rotating the camera of the first configuration by 15°. The third configuration comes from rotating the camera of the first configuration by 30° and the last one by a rotation of 45°.

To generate trajectories, the *NavMesh* functionality from Unity3D is used. There are four control points, each one at an end of a path. The starting and ending points are randomly selected from the control points. Pedestrians interaction is clumsy, as sometimes they collide or form a non natural move to avoid collisions. In this context, this is not considered an issue, as the intention is primarily to see how the prediction model is affected by incorporating the segmentation of the image.

Qualitatively, it is a low density dataset, so there are few weird trajectories distorted because of pedestrian interactions.

Since the trajectories are generated and saved at the same time, the frame rate is not constant. In average, it is 43 FPS. This depends on how many calculations *NavMesh* has to do to get the simulation done. All the pedestrians are walking at the same speed and their locations are saved every 15 frames. The goal is to predict 12 positions in the next 4.2 seconds, given 8 positions in the last 2.8 seconds, hence the conditions are rather similar to the context of ETH/UCY. A video is available on github [6] to get an idea of how fast the pedestrians go trough the scene.

Although the same scene is used for different camera configurations, each configuration will be referenced as a different scene and they will be numbered (1,2,3,4) from left to right according to the Figure 3.3. After processing, 16582 trajectories of length 20 were obtained. In Table 3.2, the amount of trajectories with length $20 = (8 + 12)$ for each scene is reported.

Scene	Amount of trajectories with length 20
1	3482
2	3205
3	4174
4	5721

Table 3.2: Amount of trajectories in our synthetic crossroad dataset.

[6] https://github.com/jagmonroy/cvae_tp/tree/master/crossroad/datasets

Chapter 4

Models

In this chapter, the models that we propose are described. In the first section, I explain an introductory model. This model has things in common with the other models and it is worth understanding it well to go smoother in the rest of the sections. Since hyperparameters depend on the dataset and the model, in order to avoid confusions, this Chapter is only focused on describing and explaining the models. The choice of hyperparameters for ETH-UCY will be detailed in Subsection 5.1.1 and for Crossroad will be detailed in Subsection 5.2.1.

4.1 An Introductory Model

This Section is intended to explain some important aspects, such as the encoding, the decoding, the loss function, and a way to incorporate Semantic Segmentation.

4.1.1 Description

We recall that the prediction problem consists in predicting the next positions of an agent given its past positions. The first model I describe manages displacements as inputs (i.e. the vectors defined by two consecutive positions), instead of absolute coordinates. An encoding-decoding scheme (see Section 2.5) with LSTMs is followed, as in other works.

The encoding process is depicted in Figure 4.1. The displacements $d_1, \dots, d_l$ (between consecutive positions of the observed trajectory) are first mapped to a

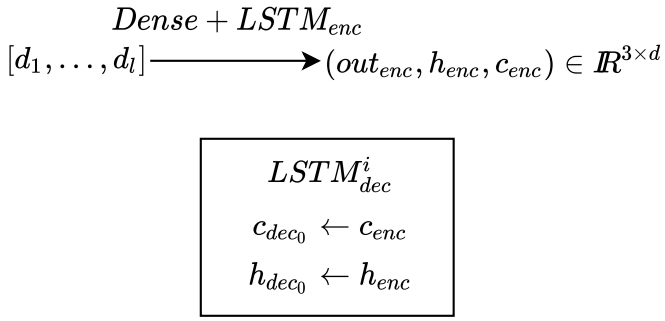


Figure 4.1: Displacements encoding and decoders states initialization. Top: the displacements $d_1, \ldots, d_l$ are first transformed with a dense layer and then encoded with a $LSTM_{enc}$. Bottom: a series of k decoders $LSTM^i_{dec}$ for $1 \leq i \leq k$ are initialized with the end states of $LSTM_{enc}$ before generating the output trajectories.

new space with a dense layer $f_e(\cdot)$ and then codified with an encoder referred to as $LSTM_{enc}$. As explained in Section 2.4, at each time step t, an LSTM layer manages two internal states h_t and c_t to give an output out_t. In this case, only the last states and the last output are going to be used in further processing:

$$[d_1, \ldots, d_l] \mapsto LSTM_{enc}([f_e(d_1), \ldots, f_e(d_l)]) = (out_{enc}, h_{enc}, c_{enc}). \tag{4.1}$$

To get a fixed number k of trajectory predictions as an output, k decoders $\{LSTM^i_{dec}\}_{1\leq i\leq k}$ are initialized with the same final states (h_{enc}, c_{enc}) of the encoder $LSTM_{enc}$. It can be seen as an ensemble of k networks [6]. Each decoder has its own weights, which allows to get multiple outputs even when all of them receive the same inputs. This is going to be better explained in the Subsection 4.1.2.

For a decoder, the objective is to get l' future displacements of the pedestrian, for time steps $l+1, l+2, \ldots, l+l'$ (the ones before, at $1, 2, \ldots, l$ have been observed). There are two options to perform this decoding. The simpler option (first row of the Figure 4.2) uses repeatedly the encoder output out_{enc} as decoder input, l' times. This is equivalent to process an l'-long sequence $[out_{enc}, \ldots, out_{enc}]$ with $LSTM^i_{dec}$. In this case, all $LSTM^i_{dec}$ outputs are used:

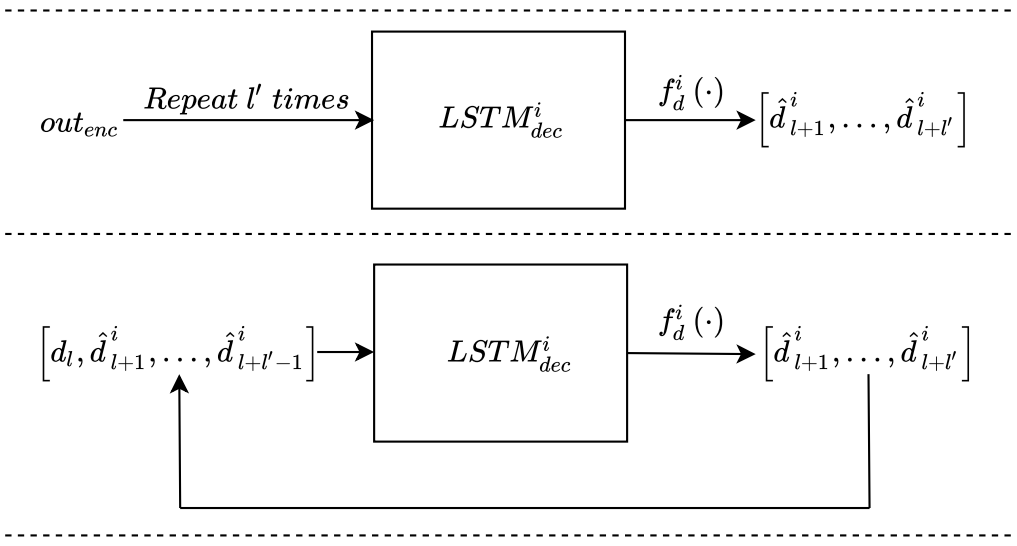


Figure 4.2: Two different decoding processes with LSTMs.

$$LSTM^i_{dec}\left([out_{enc}, \ldots, out_{enc}], h_{enc}, c_{enc}\right) = \left[out_{dec_1}, \ldots, out_{dec_{l'}}\right]. \tag{4.2}$$

To get the final prediction $\hat{y}^i$, the $LSTM^i_{dec}$ outputs are mapped to $\mathbb{R}^2$ with a dense layer $f^i_d(\cdot)$:

$$\hat{y}^i = \left[f^i_d\left(out_{dec_1}\right), \ldots, f^i_d\left(out_{dec_{l'}}\right)\right]. \tag{4.3}$$

Note that this dense layer is specific to the output decoder i.

The other option to decode, instead of repeating out_{enc}, is to use the displacements directly (second row of the Figure 4.2). The last observed displacement d_l is passed through $LSTM^i_{dec}$ to get $out^i_{dec_1}$:

$$LSTM^i_{dec}\left(d_l, h_{enc}, c_{enc}\right) = \left(out^i_{dec_1}, h^i_{dec_1}, c^i_{dec_1}\right). \tag{4.4}$$

This output $out^i_{dec_1}$ is mapped with a dense layer $f^i_d(\cdot)$ to generate the first prediction $\hat{d}^i_{l+1}$. To produce the next prediction, $\hat{d}^i_{l+1}$ goes through the decoder as if it were a new observation, to get $out^i_{dec_2}$:

$$LSTM^i_{dec}\left(\hat{d}^i_{l+1}, h^i_{dec_1}, c^i_{dec_1}\right) = \left(out^i_{dec_2}, h^i_{dec_2}, c^i_{dec_2}\right). \tag{4.5}$$

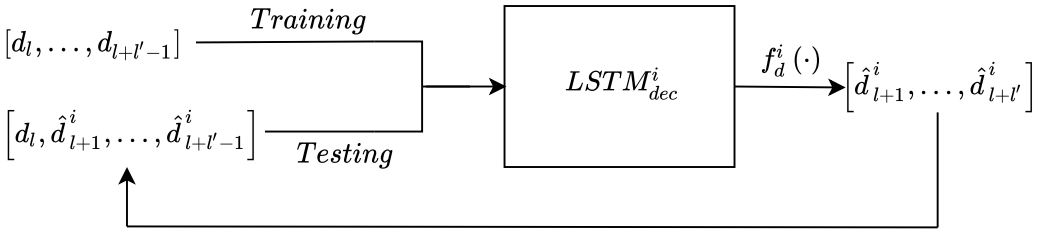


Figure 4.3: Teacher forcing.

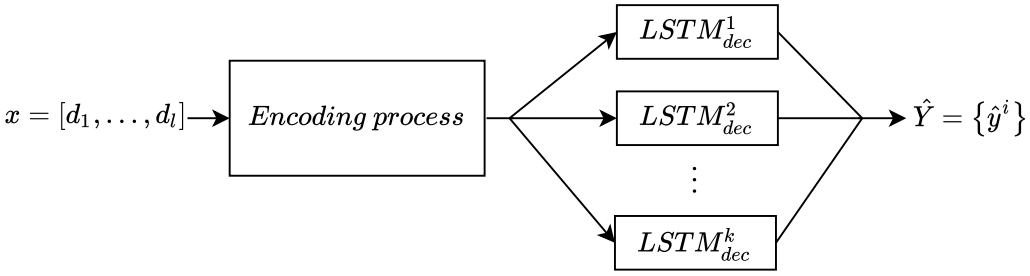


Figure 4.4: Introductory model: A general perspective.

Then, the output $out^i_{dec_2}$ is mapped with $f^i_d(\cdot)$ to the following predicted displacement $\hat{d}^i_{l+2}$. And so on.

The second decoding approach allows to use *teacher forcing*, depicted in Figure 4.3. Teacher forcing is a common technique to train recurrent NNs in sequence-to-sequence problems. In the training phase, instead of using the last prediction model $\hat{d}^i_j$ as an input to predict $\hat{d}^i_{j+1}$, the ground truth d_j is used. For testing, this trick is not possible and the predictions $\hat{d}^i_j$ have to be used, otherwise, it would be cheating.

Putting everything together, the introductory model looks like the architecture depicted in Figure 4.4.

4.1.2 Loss Function

For one instance $(x, y) \in \mathbb{R}^{l\times 2} \times \mathbb{R}^{l'\times 2}$, where x are the observed displacements and y the ground truth displacements, the loss function is defined as the minimal average

Euclidean distance between the predicted displacements $\hat{Y}(x) = \{\hat{y}^i\}_{i=1,\dots,k}$ and the ground truth displacements y. The minimum is taken over the different outputs of the k decoders:

$$L\left(\hat{Y}(x), y\right) = \min_{i=1,\dots,k} \left\{ \frac{1}{l'} \sum_{j=1}^{l'} \left\| \hat{y}_j^i - y_j \right\|^2 \right\}, \tag{4.6}$$

For several instances, as usual, the average is taken.

To start training, each decoder weights are initialized *randomly* and independently, so in the beginning the k predictions can be different. Since the loss function (Eq. 4.6) only takes into account the best prediction (according to that criteria), only the weights of the corresponding decoder will be adjusted. In the best scenario, each decoder covers a different case and the predictions are diverse. We should notice that even when this is the aim, there is nothing in the model to enhance this behavior. Actually, there is not even communication among decoders.

4.1.3 Incorporation of Semantic Segmentation

In this section, I am going to explain a possible way to incorporate Semantic Segmentation (SS) in our models.

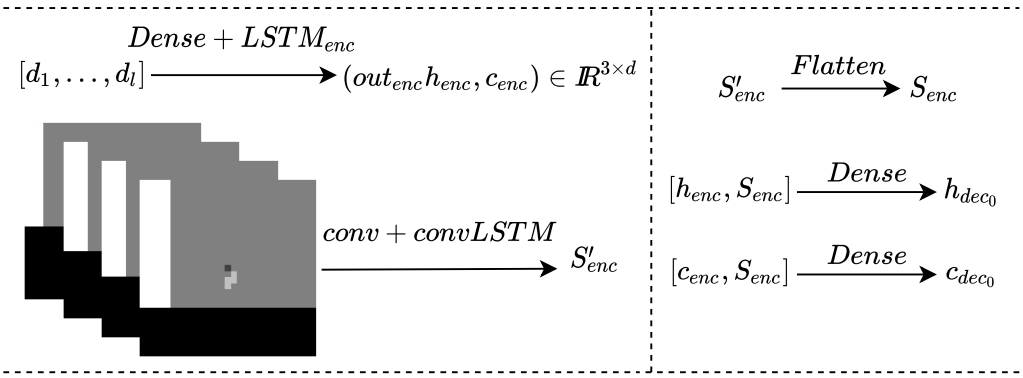


Figure 4.5: Encoding with SS.

The only difference with what I have already explained in Subsection 4.1.1, is the encoding process. There are now two encoders (Figure 4.5 on the left): One to

encode the spatial displacements (as in the previous model) and another to encode the semantics. Since the inputs are displacements, patches of the semantics centered on the pedestrian position are extracted at each time step. The semantics encoder is a convLSTM [24], which is basically a LSTM, but that uses convolutional layers instead of dense layers. Since convolutional layers preserve the two dimensional structure, the convLSTM output S'_{enc} is two dimensional. Its dimensions depend on the parameters of the convolution. To merge the information of both spatial and semantic encodings (Figure 4.5, on the right), the output from the convolutional encoder S'_{enc} is flattened and concatenated with both final states h_{enc} and c_{enc} of the spatial encoder $LSTM_{enc}$. Those concatenations are transformed with dense layers and the results are taken to initialize the states of the decoders. The rest of the prediction process is the same as in the model without SS: there are, again, two decoding options (Figure 4.2) and multiple decoders are used to get several outcomes (Figure 4.4).

I tested this model with and without SS in the Crossroad dataset (described in Section 3.2), by using $k = 3$ decoders. The SS improved the metrics ADE and FDE, but it was not impressive. The main limitation I see in this model is the tuning of a hyperparameter of the convLSTM encoder: the dilatation rate of the convolution. The goal of this parameter is to extract features of the image at certain scale. However, my experience is that, for different scenes, this parameter can be tricky to adjust.

4.1.4 Relation with State of the Art

In terms of NN, the model presented in Subsection 4.1.1 can be considered as a simpler version of a sub-network of the model presented by Liang et al. [18]. It is simpler because it does not include an attention subsystem, but both models predict multiple trajectories with multiple decoders trained with teacher forcing. Liang et al. [18] incorporate extra features to their model, such as SS, pose, and distances with other objects/pedestrians. Only a sub-branch of the network copes with displacements. Besides the prediction, the model makes a classification on an image grid about where the pedestrian is going to be and what he is going to do. In their model, the SSs of the whole frames are incorporated to the model with a size

of 32×64. This makes feasible to flatten the images and then process them with an LSTM instead of a convLSTM.

Since we are working with displacements and since there is no global reference of the scene, it does not make sense to use the SS of the whole frame. The approach followed in Subsection 4.1.3 is similar in that sense to the one followed in [19]. Pfeiffer et al. [19] also take patches centered on the pedestrian, but they apply a rotation to align the image with the direction the pedestrian is going to. The problem is that in some cases it is not clear how to rotate the images. For example, when the pedestrian stands still, it is difficult to define an orientation. In other cases, even when there is a clear going direction, it can be suddenly changed.

Liang et al. [17] present a model with a convLSTM to process the segmentation images and the trajectories. Its filters have a size of $(3, 3)$, but their SSs maps are of 36×64 as well. In the same way as in [18], their model is a combination of classification and regression. For each pixel, their model predicts an offset of where the pedestrian is going to be, starting from the center of the pixel. In the training phase, a distribution over the scene is learned. In the testing phase, they use a strategy to generate multiple, qualitatively distinct trajectories.

Ridel et al. [20] use a convLSTMs to do the encoding too. The trajectory is managed as a sequence of hot encoding grids: the position where the pedestrian is, is set to one and the rest to zeroes. The segmentation images and the hot encoding grids are transformed with two different sub-networks and concatenated. The concatenation is processed with a convLSTM to learn a distribution over the scene. This distribution is used by the model to make a fixed number of predictions. The convLSTM kernels have a size of $(11, 11)$. As far as I know, this can be considered as a big kernel, but their segmented images have a size of 128×128. I think this is to allow the network to extract further features.

4.2 Conditional VAE

In this section, a conditional VAE (see Section 2.6 for our reminders about Variational Autoencoder (VAE)) is adapted for trajectory prediction. In this context, all the models are conditioned to the observed trajectory, so the "conditional" prefix is going

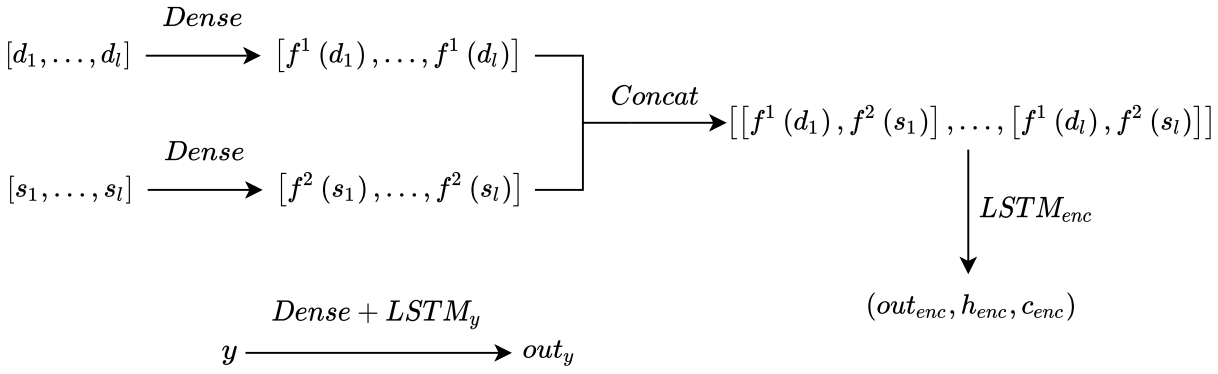


Figure 4.6: Conditional VAE encoding.

to be omitted from the names. The novelty is that I train a VAE with teacher forcing to get very competitive results in the ETH-UCY benchmark (see Section 3.1).

4.2.1 Description

The model, again, predicts displacements instead of absolute coordinates. In case of incorporating Semantic Segmentations (SSs), it is done in a different way than in Section 4.1. The difference between incorporating SS or not is subtle.

The encoding process is illustrated in the Figure 4.6. In case of incorporating semantic maps $[s_i]$, these maps are processed with a function $f^2(\cdot)$: it applies some traditional blocks of convolutional and max pooling layers and, in the end, the output is flattened. The vectorial representation $f^2(s_i)$ is concatenated with the transformed displacement $f^1(d_i)$, and the sequence of concatenated vectors

$$\left[\left[f^1(d_1), f^2(s_1)\right], \ldots \left[f^1(d_l), f^2(s_l)\right]\right]$$

is passed through the recurrent network $LSTM_{enc}$ to get $(out_{enc}, h_{enc}, c_{enc})$. In case of not incorporating semantics, only the sequence

$$\left[f^1(d_1), f^1(d_2), \ldots f^1(d_l)\right]$$

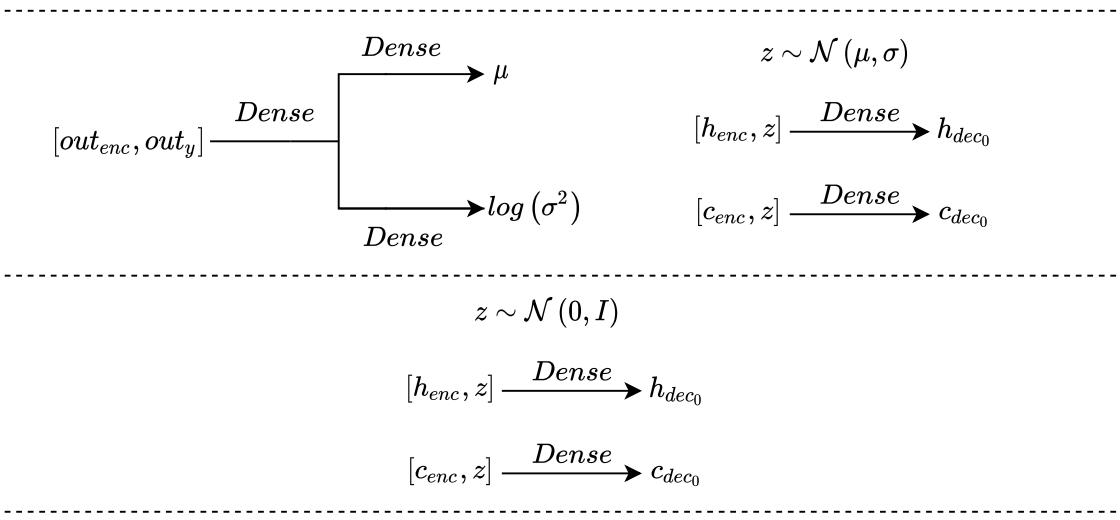


Figure 4.7: Conditional VAE: sampling at training/testing times and decoding.

is passed through $LSTM_{enc}$. Also, the ground truth y can be codified with another sub-network $LSTM_y$, as shown in the bottom of the Figure 4.6. This is done (only during training) with the intention of helping the model to do a better sampling during training.

The sampling process during training and testing is different. First, sampling in training is explained. This corresponds to the first row of Figure 4.7. There are two sub-networks forming a "recognition module" (Figure 4.7, first row on the left). Given the input, it infers the first and second moment statistics μ and σ on the latent variable. Both sub-networks share the first dense layer and each one has a last independent dense layer. In case of using the y (ground truth) encoding for sampling, the input of these sub-networks is the concatenation of out_{enc} and out_y; if not, the input is only out_{enc}. After getting the statistics μ, σ (Figure 4.7, first row on the right), a sample z of the latent variable is taken from a normal distribution $\mathcal{N}(\mu, \sigma)$; this latent variable z is then concatenated with h_{enc} and with c_{enc}. The concatenations are transformed with two different dense layers to get two vectors h_{dec_0} and c_{dec_0}. These vectors (h_{dec_0}, c_{dec_0}) are used to initialize the states of the decoder $LSTM_{dec}$ and, finally, to get a prediction. The loss function is the same used in the model previously explained (Subsection 4.1.2) plus a divergence

penalization, which can be weighted as follows:

$$L\left(\hat{y}(x), y\right) + \beta \cdot D_{KL}\left(\mathcal{N}\left(\mu, \sigma\right) \| \mathcal{N}\left(0, I\right)\right). \tag{4.7}$$

An important implementation detail is that all the dense layers have the $\tanh(\cdot)$ function as activation function. Usually, the layers used to learn the statistics μ and σ during the training phase, have the identity function as activation function. I think of the use of $\tanh(\cdot)$ in these layers as a form of regularization. Its codomain is $(-1, 1)$, so it is not possible for the entries of the statistics to have a large absolute value. I have also tried with the identity function but I observed that it worked better with the $\tanh(\cdot)$ function.

Now, sampling during testing is going to be explained. This process is depicted in the second row of Figure 4.7. The states (h_{enc}, c_{enc}) can be obtained in the same way as in training. Then, we concatenate them with a sample z taken from a standard normal distribution to produce the initial states of the decoder (h_{dec_0}, c_{dec_0}). In contrast with the model presented in Section 4.1, this model has a unique decoder, but it is possible to get as many predictions as desired, by taking different realizations of the latent variables z.

4.2.2 Relation with State of the Art

Many efforts have been directed toward using GANs to improve the quality of the predictions. As far as I know, Gupta et al. [9] took the initiative in this direction. The idea is to use the discriminator to polish the predictions from a social perspective. After Social-GAN [9], different GAN models have arisen. Some of them use information about the scene too, such as SoPhie [21] and Social-BiGAT [12]. Both of them use a pre-trained well known network [25] to extract features of the image.

Pedestrian trajectory prediction has been little explored with VAEs. As far as I know, the work of Lee et al. [15] is the most important antecedent in this context. They have designed a VAE to tackle this problem and scene features are incorporated. First, an encoding-decoding scheme is followed to do a prediction. This prediction is ranked and refined with a second decoder, which is rewarded depending on the scene. One interesting point, is that the model learns the rewards by itself, using an

inverse optimal control network.

4.3 Variant of VAE with Discrete Latent Variable

The model presented in this section is a proposal based on the work of Tim Salzmann et al [22]. The proposal is simpler and, as we will see, gets better results than the VAE of the last section. It does not achieve better results than [22], but it gets very close, which is very interesting given its simplicity.

4.3.1 Description

In the theoretical formulation of VAEs [4], the aim is to maximize

$$\underset{z\sim Q}{E}\left[\log P\left(Y|X,z\right)\right] - D_{KL}\left[Q(z|X,Y)\|P(z|X)\right], \tag{4.8}$$

In practice, $P\left(z|X\right)$ is taken as a normal standard distribution, while $P(Y|X,z)$ and $Q(z|X,Y)$ are "learned" with two distinct NNs $p_\psi(Y|X,z)$ and $q_\phi(z|X,Y)$, parameterized by ψ and ϕ, respectively. Hence, the function to maximize becomes

$$\underset{z\sim q_\phi(\cdot|X,Y)}{E}\left[\log p_\psi\left(Y|X,z\right)\right] - D_{KL}\left[q_\phi(z|X,Y)\|\mathcal{N}\left(0,I\right)\right], \tag{4.9}$$

where $q_\phi(z|X,Y)$ is assumed normal as well and such that the network learns its first two moments μ and σ, conditionally to X and Y. The distribution $p_\psi(Y|X,z)$ corresponds to the generator as we saw it above. Furthermore, the expected value is approximated by Monte-Carlo with only one sample, leaving only:

$$\log p_\psi\left(Y|X,z\right) - D_{KL}\left[q_\phi(z|X,Y)\|\mathcal{N}\left(0,I\right)\right]. \tag{4.10}$$

All of this is done in the VAE presented in the Section 4.2. When the ground truth is not used to learn μ and σ, the model tries to learn $q_\phi(z|X)$ instead of $q_\phi(z|X,Y)$.

If the latent space is *discrete*, it means that there is only a fixed number k of possible values for the latent variable $z \in \{1,2,\ldots,k\}$. Then, the expected value in Eq. 4.9 can be computed. On the other hand, instead of assuming that $P\left(z|X\right)$ is

normal, as we saw above, it can be also "learned" with a NN $p_\theta(z|X)$, parameterized by θ:

$$\underset{z\sim q_\phi(\cdot|X,Y)}{E}\left[\log p_\psi\left(Y|X,z\right)\right] - D_{KL}\left[q_\phi(z|X,Y)\|p_\theta(z|X)\right]. \tag{4.11}$$

I find this very interesting, because it is possible to iterate over the whole latent space, i.e., over the k possible values for the latent variable z. Also, the values of $q_\phi(z|X,Y)$ and $p_\theta(z|X)$ allow to *rank* the predictions in training and testing, respectively. The detail is that the number of outputs is limited to the number of possible values of the latent variable. In [22], the authors use GMM on top of that to produce as many predictions as they want.

Now the proposal is going to be explained. Asking for the value of $q_\phi(Y|X,z)$ seems to complicate the training. It makes sense, because the model is not only asked to do predictions, but also to sort them. In this work, having a ranking of the predictions is not required, so it is not necessary to learn $q_\phi(z|X,Y)$ and $p_\theta(z|X)$. In that way, the divergence term in (4.11) can be removed. If the NNs in charge of those distributions are gone, the expected value can not be computed. To overcome this, the same loss function we saw in the introductory model (Eq. 4.6) is taken:

$$\max_{z\in\{1,\dots,k\}}\left\{\log p_\psi(Y|X,z)\right\}. \tag{4.12}$$

This model is going to be called Simplified Discrete latent variable Variational Autoencoder (SDVAE). We should notice that, strictly speaking, since there is no distribution $q\left(\cdot|\cdots\right)$ learned, this model is not variational. But it is going to be considered as a VAE because it was derived from one.

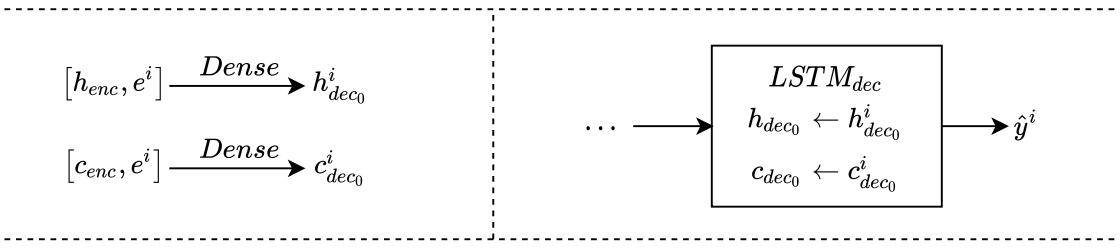


Figure 4.8: SDVAE: sampling.

The encoding process is the same as before. The latent variable z follows a categorical distribution with k possible categories. The canonical vector e_i (where at position i, the value is one and the rest zero) represents the category i. Comparing with the VAE, instead of taking a sample from a normal distribution, a canonical vector e_i is taken. To decode, as depicted on the left of the Figure 4.8, a latent variable e_i is concatenated with the final states of $LSTM_{enc}$ and the concatenations are mapped with dense layers to $\left(h^i_{dec_0}, c^i_{dec_0}\right)$. As shown on the right of the Figure 4.8, the pair $\left(h^i_{dec_0}, c^i_{dec_0}\right)$ is used to initialize the states of $LSTM_{dec}$ and to make a prediction $\hat{y}^i$. In training, for each instance (x, y), k predictions are produced instead of one and the best of them according to Subsection 4.1.2 is taken to compute the loss.

As mentioned before, this model is way simpler. For me, it loses all the beauty of the VAE with discrete latent variables, but it is still interesting from another perspective. Thinking of the work of Junwei Liang et al [18], this can be a potential direction to improve their model. An inconvenient I see on their work, is that each decoder is independent. All of them receive the same information to do the prediction, but, as far as I understand, there is no a coordination between them. Using one decoder and conditioning it to a latent code allows "synchronized" predictions. But I see a problem here as well, since maybe just one decoder is not enough to capture multi-modality and [18] has more chances in this aspect.

4.3.2 Relation with State of the Art

Salzmann et al. [22] present a VAE with discrete latent variable and other compo-

nents, such as social module interaction, attention, bidirectional LSTMs for ground truth encoding, and GMM to have as many predictions as they want. As far as I know, they achieve the best known results in ETH-UCY benchmark. Also, they got the third place in a recent trajectory prediction contest [1].

Giuliari et al. [7] have an important place in the state of the art as well. They propose a model based in Transformers [28], maybe the most popular tool in NLP at present. It does not require more data than the trajectories and it gets very good results in the ETH-UCY benchmark. They are even better than other methods that use extra features, such as Social-GAN [9], SoPhie [21] and Social-BiGAT [12].

[1] https://www.nuscenes.org/ (2020-07-18).

Chapter 5

Experiments

In this Chapter, I present some of the results I obtained with the models described in the previous chapter. Ablation studies are realized to have a better insight on the role of each component of the proposed models. The first section is about experiments performed on the ETH-UCY dataset (described in Section 3.1) and the second one is about experiments performed on our crossroad dataset (described in Section 3.2).

5.1 ETH-UCY

From my perspective, the most relevant works according the results they show, are [22] and [7]. The work of Tim Salzmann et al [22], as far as I know, achieves the best known results on this dataset up to now. They incorporate many elements to perform prediction, such as social interaction, bidirectional LSTMs and GMMs. Francesco Giuliari et al, in [7], use the popular Transformers [28], which are NNs based only in attention to work with sequences. They beat a lot (maybe all, except [22]) of the methods that incorporate extra features (semantics, appearance), by taking into account just the trajectories.

In this Section, first, ablation studies are presented and, then, some of the results obtained with our models are compared with baseline methods from the state of the art. I present two models that can compete with [7].

The procedure for testing has been explained in Section 3.1. In the training phase, the model is evaluated with the validation trajectories and the best weights

according to mADE (see Eq. 1.3) are saved. In the end, the best saved weights are loaded and the model is evaluated on the test trajectories.

After each training epoch, the model is evaluated on the test trajectories as well, to keep a register. Those results are not taken into account for the final evaluation.

One of the things I disagree with in this benchmark, is that the authors report the results of one run (whole training and testing process) and no more, while their system is stochastic (final results depend on the weight initialization). This does not say anything about the stability of the model. Maybe the bad or good results were outliers. It is important to do more than one run, so in the results presented below, I run each model five times instead of just once.

In general, when teacher forcing is not used to train, the decoding is done by repeating l' times out_{enc}, as shown in the first row of Figure 4.2. When teacher forcing is used, it is done as depicted in Figure 4.3.

5.1.1 Hyperparameters

In this Section, all the models are trained with stochastic gradient descent with a learning rate 0.005, momentum 0.9 and batch size 128. When the tested scene is one of ETH, HOTEL, ZARA1 or ZARA2, the SDVAE models are trained through 50 epochs and VAE models through 100 epochs. When UNIV is the tested scene, the number of epochs is doubled because there are less training samples than in other scenes (see Table 3.1). With VAE models, the divergence penalization weight β is a hyperparameter (Eq. 4.7) and comments on its value will be made in the next Subsection.

All the LSTM layers have the same hyperparameters (encoder and decoder in SDVAE and VAE): the activation function is the $\tanh(\cdot)$ function; the hidden state dimension and the output dimension are both 256. Before encoding the displacements, they are mapped from $\mathbb{R}^2$ to $\mathbb{R}^d$ with a dense layer. With SDVAE models, and with VAE models, $d = 64$ and $d = 128$, respectively. When the models are trained with teacher forcing, there is another dense layer to map the displacements to decode, and the same values of d are used as in the initial mapping.

These parameters were empirically adjusted. In the case of the hidden state dimension, I tried with values less than 100 but the performance was worse. For

values greater than 256, the performance does not improve too much and, eventually, it starts getting worse. The value d actually does not seem to affect the performance, but I put that dense layer to increase the dimension from 2 to 256 in two steps instead of doing it directly.

5.1.2 VAE: Ablation Study

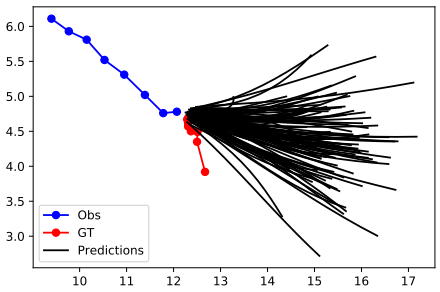
The first ablation we study is to evaluate how teacher forcing and the inclusion of ground truth data (to try to improve the sampling in training) affect the model performance. The model used here is the one described in Section 4.2 and the results are presented in the Table 5.1. As mentioned above, each number is the average of five runs (whole training and testing process). The columns labeled as "100" and "20" correspond to $k = 100$ and $k = 20$ samples generated from the VAE. In the columns with header "20S", the process is different. We initially generate 100 predictions $S_i = \{\hat{y}^i\}$ and then we select 20 of them, following a greedy approach implemented to cover the support of the predictive distribution: first, the pair of trajectories with the furthest end-points according to the euclidean distance, are taken into a set S. The next trajectory $\hat{y}^i$ to include in S is the one that satisfies

$$\arg \max_{\hat{y}^i \in S_i - S} \left\{ \min_{\hat{y}^j \in S} \left\| \hat{y}^i_{l'} - \hat{y}^j_{l'} \right\| \right\},$$

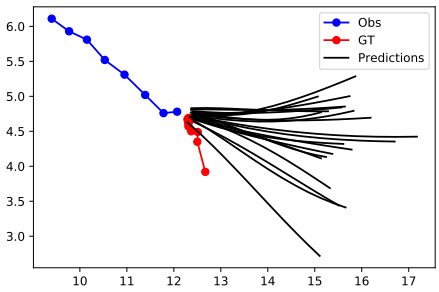
i.e., the furthest trajectory from the current S, and so on.

From Table 5.1, we can say that, independently from whether the ground truth is used or not, the model improves with teacher forcing. If teacher forcing is not used, then using ground truth helps to improve the results. Regarding to the KL divergence regularization (see Eq. 4.7): in the beginning, this loss is in the order of $1e-4$. If no regularization is used, it goes up to the order of $1e-2$, and, at some point, the model starts failing in validation and test predictions. Therefore, its weight was empirically adjusted to $\beta = 0.25$. In this way, the loss stays in the order of $1e-4$ and there is no overfitting.

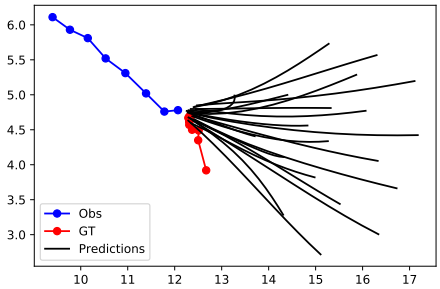
When more samples are taken, the latent space is better covered and it is possible to find a higher diversity of predictions. It is totally understandable that taking 100 samples leads to better results than taking just 20. The challenge is to filter those



(a) 100 predictions.



(b) Random selection of 20 among the 100.



(c) Greedy selection of 20 among the 100.

Figure 5.1: VAE: greedy selection. In the Subfigure 5.1a, 100 predictions are shown. In the Subfigure 5.1b, 20 of the 100 predictions are randomly selected. In the Subfigure 5.1c, 20 of the 100 predictions are selected following the greedy strategy.

100 predictions and keep the performance as good as possible. The followed strategy is simple, but effective. In the Figure 5.1, we can see the benefit of taking more trajectories (Subfigure 5.1a). If just 20 samples are taken (Subfigure 5.1b), then less area is covered. Following the greedy approach avoids concentrating the predictions in some region (Subfigure 5.1c).

The second ablation is to study the effect of the KL divergence regularization term of Eq. 4.7. The numerical results are similar to the ones presented in Table 5.1 and they are not going to be shown. As mentioned before, there is a detail, eventually, the latent space of the model trained with ground truth and without teacher forcing, becomes useless to validation and test trajectories. This can be seen in the Figure 5.2. Nevertheless, validation trajectories capture the moment when the model

	TeacherF(1). Y(1).			TeacherF(0). Y(1).		
	100	20	20S	100	20	20S
ETH	0.50/0.88	0.61/1.20	0.53/0.95	0.61/1.20	0.68/1.30	0.62/1.20
HOTEL	0.14/0.26	0.21/0.41	0.19/0.37	0.28/0.60	0.32/0.67	0.30/0.65
UNIV	0.23/0.38	0.31/0.58	0.27/0.47	0.39/0.83	0.43/0.92	0.40/0.85
ZARA1	0.17/0.27	0.24/0.45	0.22/0.38	0.27/0.54	0.33/0.68	0.28/0.57
ZARA2	0.16/0.26	0.22/0.42	0.21/0.38	0.23/0.44	0.28/0.55	0.25/0.49
avg	0.24/0.41	0.32/0.60	0.28/0.51	0.35/0.71	0.41/0.83	0.37/0.74
	TeacherF(1). Y(0).			TeacherF(0). Y(0).		
	100	20	20S	100	20	20S
ETH	0.51/0.91	0.62/1.20	0.54/0.98	0.75/1.60	0.80/1.70	0.75/1.60
HOTEL	0.14/0.26	0.21/0.42	0.19/0.38	0.30/0.65	0.33/0.71	0.31/0.67
UNIV	0.23/0.39	0.31/0.59	0.27/0.48	0.41/0.90	0.45/0.97	0.42/0.91
ZARA1	0.17/0.27	0.24/0.45	0.22/0.38	0.33/0.74	0.36/0.82	0.33/0.75
ZARA2	0.16/0.26	0.22/0.41	0.21/0.38	0.26/0.57	0.29/0.62	0.27/0.58
avg	0.24/0.42	0.32/0.61	0.29/0.52	0.41/0.90	0.44/0.96	0.42/0.91

Table 5.1: VAE: ablation study on the use of teacher forcing and ground truth encoding (see Section 4.2 for details on these two features). In all numeric entries, the average mADE/mFDE of five runs is shown, in meters. The blue columns show the metrics obtained by taking 100 samples from the VAE; the white columns show the metrics obtained with 20 samples; in the gray column, 100 predictions are done with 100 samples and then 20 of them are selected following a greedy approach to have diverse samples.

starts failing and good weights are loaded in the end. The model with the ground truth (red color in Figure 5.2) behaves worse than the clean model (green color in Figure 5.2), but then starts failing (error going up). When the KL divergence regularization is used, this behaviour is mitigated and the improvement is gradual. This can be seen in the Figure 5.3. Also, it is possible to see that the error on the ETH scene is going down (red color), but it is converging slower compared with the models trained with teacher forcing. As mentioned in Section 3.1, ETH trajectories are longer in time than others, even when all trajectories have the same number of observations. In this scene, in average, the improvement is 0.03/0.10 ($mADE/mFDE$) when the KL divergence regularization is used.

In Figures 5.2 and 5.3, it can be seen that there is no big difference between the models with teacher forcing (blue and orange colors).

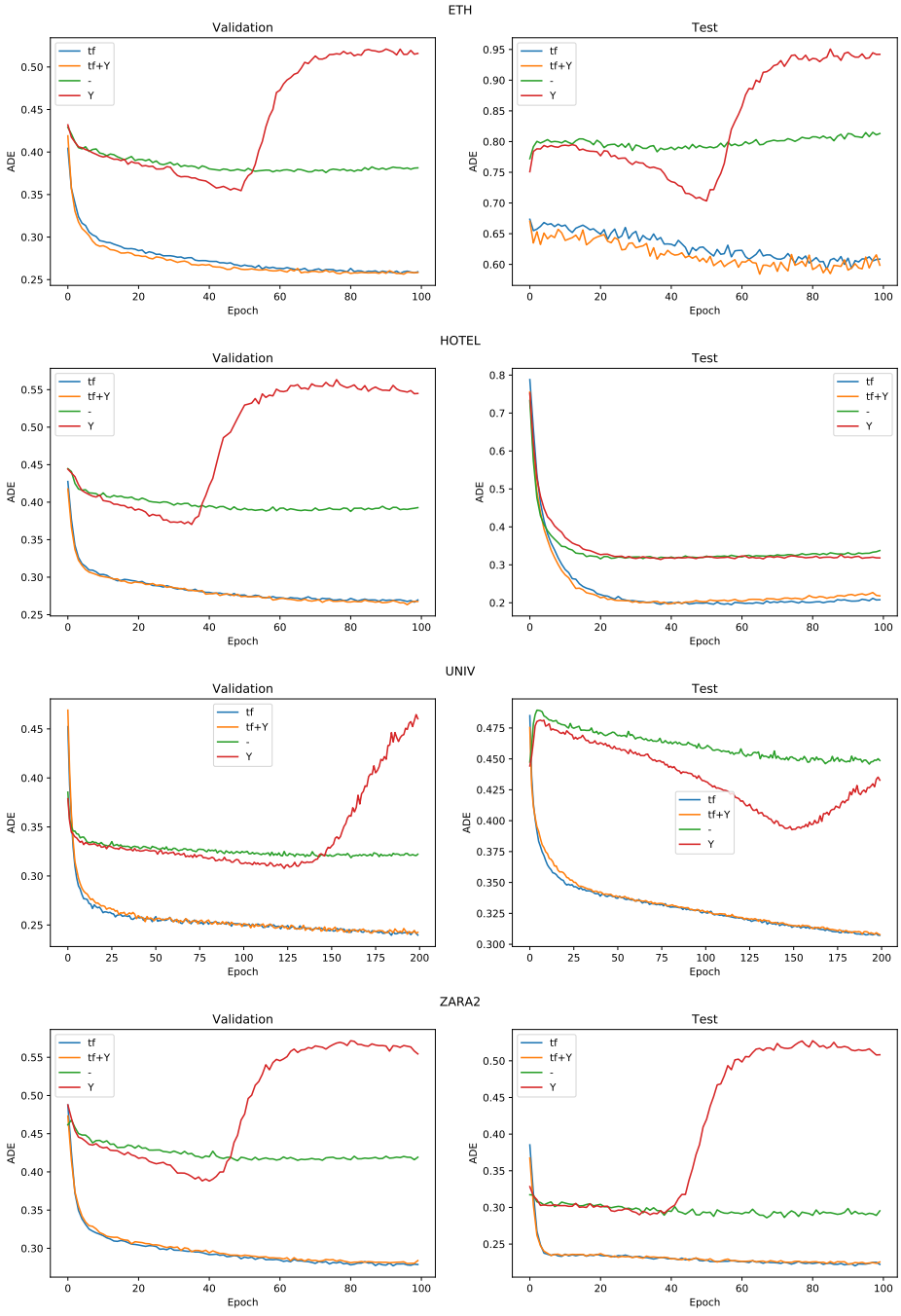


Figure 5.2: VAE: ablation study. mADE progression along training, on four ETH-UCY scenes, without KL divergence regularization. Validation on the left and test on the right.

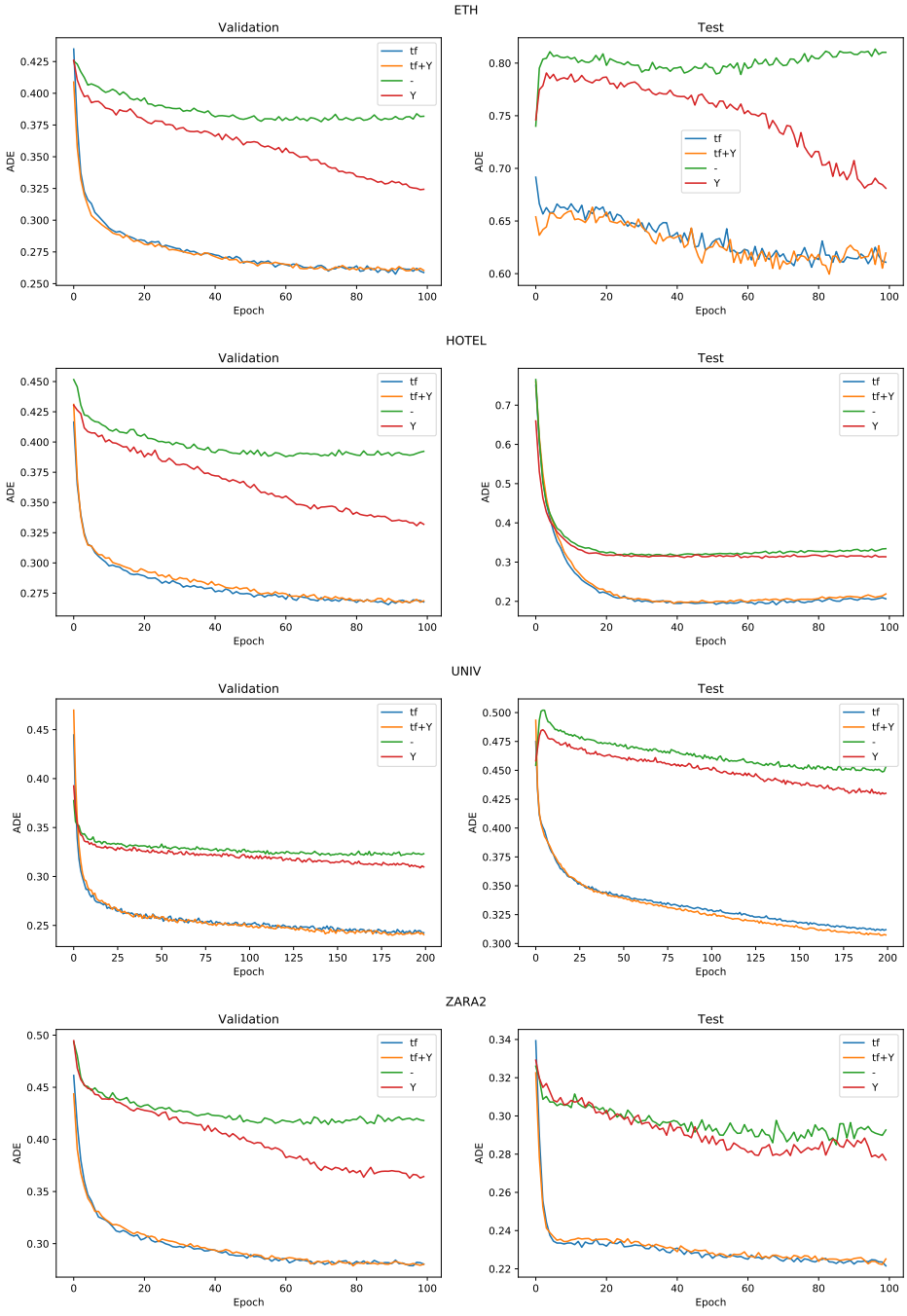


Figure 5.3: VAE: ablation study. mADE progression along training, on four ETH-UCY scenes, with KL divergence regularization. Validation on the left and test on the right.

As shown in the Table 5.1, numerically, the results of the models using teacher forcing are very similar as well. It is just a matter of hundredths of meter in the prediction.

5.1.3 SDVAE: Ablation Study

The effect of teacher forcing in the SDVAE model of Section 4.8 is experimentally studied too. The results appear in Table 5.2. Each number is the average of five runs. In this model, the results with and without teacher forcing are similar. The biggest gap is obtained on the ETH scene. This could be because of the difference between test and train trajectories. It is already known (see Section 3.1) that ETH trajectories are longer in time than others, even when all trajectories have the same number of observations (because of the framerate). Lamb et al. [14] introduce professor forcing and they explain a potential issue in teacher forcing: "when running a recurrent NN in sampling mode, the region occupied by the hidden states of the network diverges from the region occupied when doing teacher forcing". This is a possible explanation for the results, but I can not tell for sure.

	TeacherF(0). APG(0)	TeacherF(1). APG(0).
ETH	0.40/0.60	0.43/0.71
HOTEL	0.18/0.35	0.17/0.31
UNIV	0.26/0.43	0.26/0.46
ZARA1	0.23/0.41	0.22/0.38
ZARA2	0.17/0.29	0.17/0.30
avg	0.25/0.41	0.25/0.43
	TeacherF(0). APG(1).	TeacherF(1). APG(1).
ETH	0.40/0.60	0.43/0.72
HOTEL	0.19/0.38	0.19/0.35
UNIV	0.28/0.48	0.29/0.51
ZARA1	0.23/0.41	0.22/0.39
ZARA2	0.20/0.36	0.20/0.35
avg	0.26/0.45	0.27/0.46

Table 5.2: SDVAE: ablation study. Teacher forcing and APG. In all numeric entries, the average mADE/mFDE of five runs is shown in meters.

Many previous works have included features involving the neighboring pedestri-

ans in the scene, trying to improve the predictions with this contextual information. Commonly, they do not realize ablation studies, they just incorporate the features and look for a couple of cases to show how their model seems to learn social interactions. In Social-GAN [9], the authors find out that their model behaves worse in some scenes when social features are incorporated. This is briefly explored here in the case of SDVAE. The way of incorporating social features is similar to Pfeiffer et al. [19]. They introduce a so-called Angular Pedestrian Grid (APG). The idea is to include in the model the distances to the other pedestrians in different directions. There is an example of this concept in Figure 5.4. The pedestrian is located at the center of the black circle. Blue points are the positions of the other pedestrians. If there is a pedestrian in some direction inside the circle, the distance with the closest pedestrian in that direction is saved in the representation. The advantage of this is that an APG can be saved as an array and it can be easily incorporated in the model, as depicted in Figure 4.6.

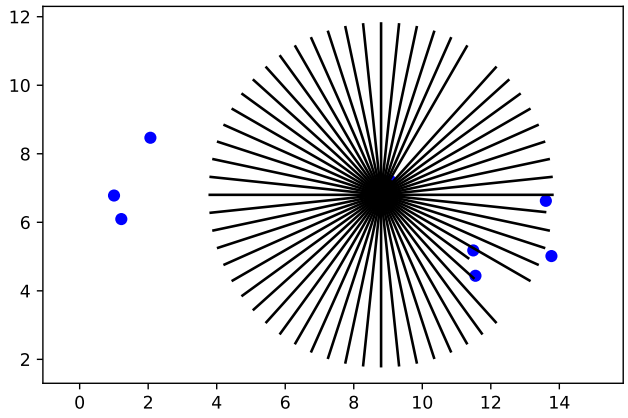


Figure 5.4: APG example.

I tried to improve the prediction results by incorporating these features but I did not see real improvements. The result are presented in the Table 5.2. It does not mean that these features are useless, maybe in another way it would work.

5.1.4 Analysis of Variance Results

In the next subsection, three of our methods are compared with the state of the art: a Vanilla Variational Autoencoder trained with the ground truth and without teacher forcing (coined as VVAE); a VAE trained with ground truth and teacher forcing (coined as 20SVAE); and SDVAE trained without teacher forcing. The S in 20SVAE means that the greedy strategy to select 20 of 100 trajectories (as explained above) is followed. In this subsection, the results of all five runs of those models are shown. The results of VVAE, which is the most unstable model among them, are presented in the Table 5.3.

In the Table 5.4, all 20SVAE results are presented. Even when the results depend on a selection after sampling 100 trajectories, it is stable (low variances). In the Table 5.5, all SDVAE results are presented. The results are stable too, but not as much as 20SVAE.

		$\mu(A)$	$\sigma(A)$	$min(A)$	$max(A)$	A
ETH	mADE	0.68	0.0102	0.67	0.69	0.67, 0.69, 0.67, 0.69, 0.68
	mFDE	1.31	0.0362	1.27	1.36	1.27, 1.35, 1.28, 1.36, 1.30
HOTEL	mADE	0.31	0.0027	0.31	0.32	0.31, 0.32, 0.31, 0.31, 0.31
	mFDE	0.66	0.0044	0.66	0.67	0.66, 0.67, 0.66, 0.66, 0.66
UNIV	mADE	0.43	0.0090	0.41	0.44	0.43, 0.43, 0.43, 0.44, 0.41
	mFDE	0.91	0.0285	0.86	0.94	0.92, 0.93, 0.92, 0.94, 0.86
ZARA1	mADE	0.32	0.0128	0.31	0.34	0.34, 0.31, 0.31, 0.32, 0.33
	mFDE	0.68	0.0391	0.63	0.73	0.73, 0.63, 0.64, 0.67, 0.71
ZARA2	mADE	0.27	0.0057	0.26	0.28	0.28, 0.27, 0.26, 0.28, 0.27
	mFDE	0.55	0.0147	0.52	0.56	0.56, 0.54, 0.52, 0.56, 0.55

Table 5.3: Results of all runs of the 20VAE model on the ETH-UCY benchmark. The position i of A is the result of the run i. For example, the results of the first run from testing on ZARA2 are 0.28/0.56.

		$\mu(A)$	$\sigma(A)$	$min(A)$	$max(A)$	A
ETH	mADE	0.52	0.0037	0.51	0.52	0.52, 0.52, 0.52, 0.51, 0.52
	mFDE	0.95	0.0094	0.94	0.97	0.95, 0.94, 0.97, 0.94, 0.95
HOTEL	mADE	0.19	0.0026	0.18	0.19	0.18, 0.18, 0.18, 0.19, 0.19
	mFDE	0.37	0.0089	0.36	0.38	0.36, 0.36, 0.36, 0.38, 0.38
UNIV	mADE	0.26	0.0016	0.26	0.27	0.26, 0.26, 0.26, 0.27, 0.26
	mFDE	0.47	0.0046	0.46	0.48	0.46, 0.46, 0.47, 0.48, 0.46
ZARA1	mADE	0.21	0.0006	0.21	0.21	0.21, 0.21, 0.21, 0.21, 0.21
	mFDE	0.37	0.0036	0.37	0.38	0.38, 0.37, 0.37, 0.37, 0.37
ZARA2	mADE	0.21	0.0008	0.21	0.21	0.21, 0.21, 0.21, 0.21, 0.21
	mFDE	0.38	0.0017	0.37	0.38	0.38, 0.38, 0.38, 0.37, 0.37

Table 5.4: Results of all runs of the 20SVAE model on the ETH-UCY benchmark. The position i of A is the result of the run i.

		$\mu(A)$	$\sigma(A)$	$min(A)$	$max(A)$	A
ETH	mADE	0.39	0.0027	0.39	0.4	0.39, 0.40, 0.40, 0.39, 0.39
	mFDE	0.60	0.0081	0.59	0.61	0.60, 0.61, 0.59, 0.59, 0.60
HOTEL	mADE	0.17	0.0061	0.16	0.18	0.18, 0.17, 0.16, 0.18, 0.17
	mFDE	0.34	0.0142	0.32	0.36	0.36, 0.33, 0.32, 0.36, 0.34
UNIV	mADE	0.25	0.0021	0.25	0.26	0.25, 0.25, 0.25, 0.26, 0.25
	mFDE	0.43	0.0058	0.42	0.44	0.42, 0.43, 0.43, 0.44, 0.43
ZARA1	mADE	0.22	0.0056	0.22	0.23	0.23, 0.22, 0.22, 0.22, 0.22
	mFDE	0.40	0.0125	0.39	0.42	0.42, 0.40, 0.39, 0.39, 0.41
ZARA2	mADE	0.16	0.0018	0.16	0.16	0.16, 0.16, 0.16, 0.16, 0.16
	mFDE	0.28	0.0061	0.27	0.29	0.27, 0.28, 0.29, 0.28, 0.28

Table 5.5: Results of all runs of the SDVAE model on the ETH-UCY benchmark. The position i of A is the result of the run i.

5.1.5 Comparison with Other Methods

In this section, in the same way as before, the entries mADE/mFDEs of a table associated to a VAE or SDVAE are the average of five runs. The results for each run have already been presented in the last subsection. The entries corresponding to other authors are the results of the only run they report in their respective papers.

In the Table 5.6, VVAE is compared with other popular methods. All of those methods use GANs as the core of their generative model and they use extra features, such as semantic maps and information about the pedestrians on the scene. Our base model VVAE (last column) can compete with them (with first position in almost all the scenes), even without using extra features, just the observed trajectories. I believe that this is because GANs are hard to train. Maybe the usual alternating training between discriminator and generator in GANs complicates things. Since VAEs are end-to-end models, I consider them easier to train.

	S-GAN [9]	SoPhie [21]	Soc-BIGAT [12]	VVAE
ETH	0.87/1.62	0.70/1.43	0.69/1.29	0.68/1.30
HOTEL	0.67/1.37	0.76/1.67	0.49/1.01	0.32/0.67
UNIV	0.76/1.52	0.54/1.24	0.55/1.32	0.43/0.92
ZARA1	0.35/0.68	0.30/0.63	0.30/0.62	0.33/0.68
ZARA2	0.42/0.84	0.38/0.78	0.36/0.75	0.28/0.55
Avg	0.61/1.21	0.54/1.15	0.48/1.00	0.41/0.83

Table 5.6: Comparison of VVAE with other methods (best of 20: mADE/mFDE in meters). The best three methods are ranked with gold, plate and bronze.

	NextP [18]	TF_q [7]	20SVAE	SDVAE
ETH	0.73/1.65	0.61/1.12	0.53/0.95	0.40/0.60
HOTEL	0.30/0.59	0.18/0.30	0.19/0.37	0.18/0.35
UNIV	0.60/1.27	0.35/0.65	0.27/0.47	0.26/0.43
ZARA1	0.38/0.81	0.22/0.38	0.22/0.38	0.23/0.41
ZARA2	0.31/0.68	0.17/0.32	0.21/0.38	0.17/0.29
Avg	0.46/1.00	0.31 / 0.55	0.28/0.51	0.25/0.41

Table 5.7: Comparison of our best VAE and SDVAE with Transformers [7] and NextP [18] (best of 20: mADE/mFDE in meters). The three methods on the right are ranked with gold, plate and bronze.

In the Table 5.7, SDVAE and 20SVAE are compared with Transformers [7] and NextP [18], two of the most efficient methods in the literature. As I mentioned in Section 4.3, I have a particular interest in exploring the potential of SDVAE compared with NextP [18]. This is a benchmark where SDVAE works better. On the

other hand, in NLP, Transformers-based methods achieve the state of the art. Their success makes it easy to think that the gap between TF_q, the main Transformers-based work in this field, and other methods (Table 5.6) is because of the Transformers capacity to process sequences. In this benchmark, our methods 20SVAE and SDVAE are models based on LSTMs able to compete with TF_q [7].

In the Table 5.8, SDVAE is compared with Trajectron and Trajectron++. Trajectron++ wins in all the cases. Even when the authors made a big jump from Trajectron to Trajectron++ in terms of the quantitative results, they do not say anything about what explained those improvements. It would be nice to know what elements are essential in that improvement and to try to get a better version of SDVAE. I believe that it is possible to achieve as good results as Trajectron++ with a simpler model. SDVAE and TF_q [7] are a good beginning. Just the time will tell.

	Trajectron [11]	SDVAE	Trajectron++ [22]
ETH	0.59/1.14	0.40/0.60	0.39/0.83
HOTEL	0.35/0.66	0.18/0.35	0.12/0.21
UNIV	0.54/1.13	0.26/0.43	0.20/0.44
ZARA1	0.43/0.83	0.23/0.41	0.15/0.33
ZARA2	0.43/0.85	0.17/0.29	0.11/0.25
Avg	0.56/1.14	0.25/0.41	0.18/0.40

Table 5.8: Comparison of SDVAE with Trajectron and Trajectron++ (best of 20: mADE/mFDE in meters). The three methods on the right are ranked with gold, plate and bronze.

5.1.6 Qualitative Results

In this section, we qualitatively analyze the outputs from our model.

Paying attention to the best predictions according to mADE of some runs, all of them are situations where the pedestrian is not moving at all. In that case, the model predicts that the pedestrian stays in the same place or that it goes in any direction. This is illustrated in the Figure 5.5. On the other hand, in the Figure 5.6, a couple of easy cases to predict are also shown.

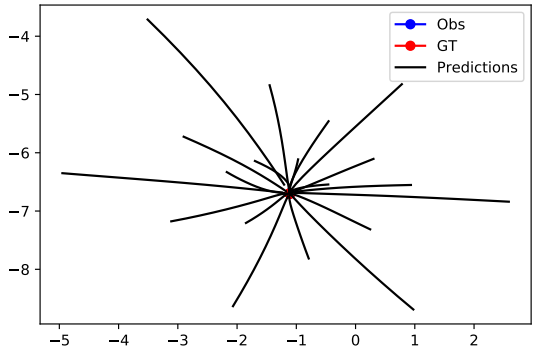
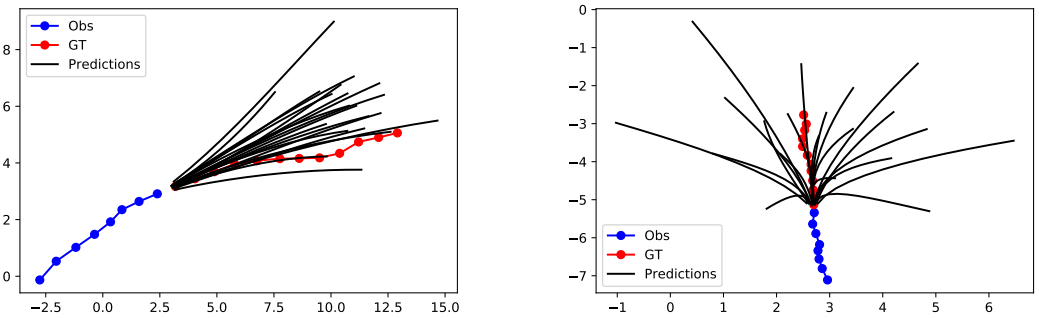


Figure 5.5: Example of SDVAE predictions when the pedestrian stands still, on the ETH-UCY dataset. One of the predicted outcomes also makes the pedestrian stay still.

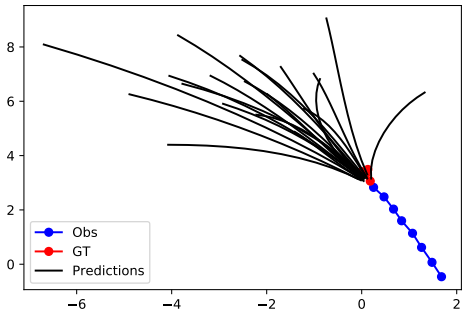


(a) An ETH case.

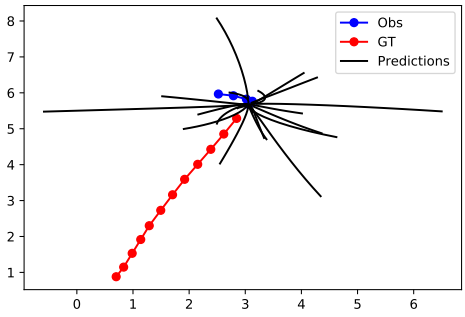
(b) A HOTEL case.

Figure 5.6: Examples of easy cases on the ETH-UCY dataset.

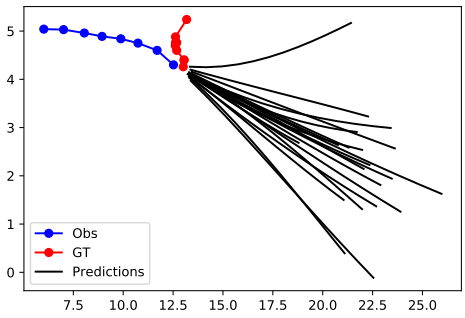
In a run, the worst case in the HOTEL dataset occurs in a situation where the pedestrian goes walking and then stops abruptly (Figure 5.7a). In the ZARA1 dataset, a pedestrian goes slow and then starts walking faster in another direction (Figure 5.7b). In the ETH, UNIV and ZARA2 datasets, there are similar cases where the pedestrian changes abruptly his direction, making the prediction generally wrong.



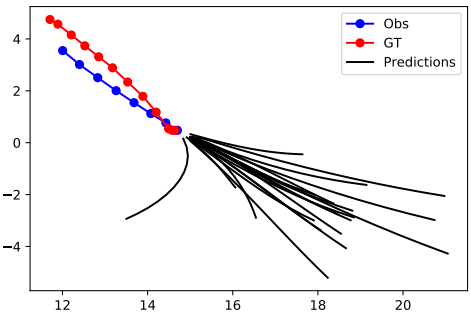
(a) Bad case in the HOTEL dataset.



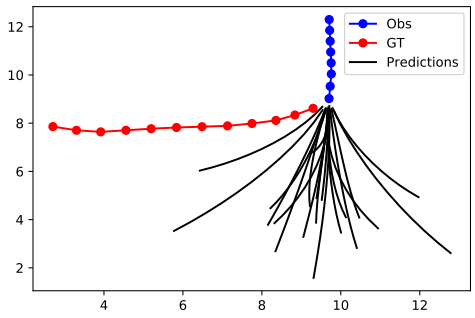
(b) Bad case in the ZARA1 dataset.



(c) Bad case in the ETH dataset.



(d) Bad case in the UNIV dataset.



(e) Bad case in the ZARA2 dataset.

Figure 5.7: Examples of bad cases on ETH-UCY.

5.1.7 Considerations on the Latent Space

From my point of view, the latent variables from the VAE latent space do not have a clear interpretation. In the experiments above, the dimension space is 128. I tried to interpret the latent variables in a 3 dimensional latent space VAE, but I was not able to find something clear either, so the comments hereafter are focused on the 128 dimensional latent space VAE.

Trying to interpret how a latent variable z_i affects a prediction, the rest of the latent variables z_j, with $j \neq i$, are set to 0 and we gradually increment the value of z_i from -2 to 2. In some trajectories, the changes in two different latent variables leads to similar predictions and, because of that, I believe that there are redundant variables. On the other hand, I found a case where a latent variable seems to affect the length of the prediction, but this same latent variable has a different effect in another case. Hence, I think it is better not to empirically conclude something about the latent variables.

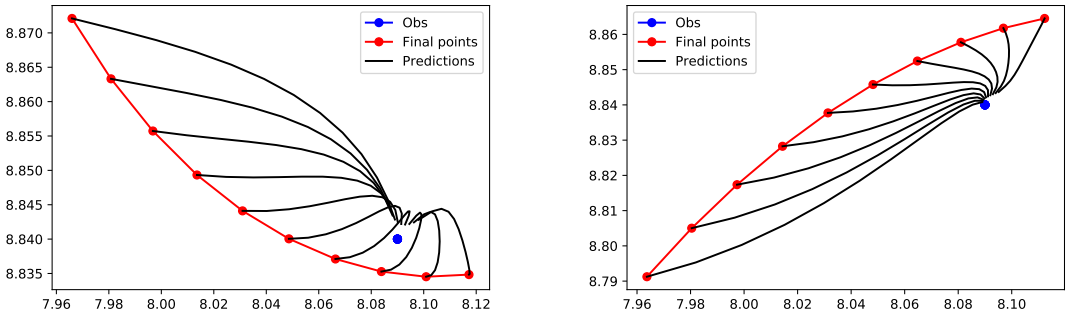
In the Figure 5.8, three cases are shown. In the Subfigure 5.8a, a case where the pedestrian stands still is shown. This is a case of large uncertainty and where the predictions are diverse. The Subfigure 5.8b shows a case with a clear tendency about the direction, and in both images the predictions are very similar. In the Subfigure 5.8c, a case where the pedestrian is moving but there is no clear tendency about the direction is shown. Varying the value of the first latent variable leads to scattered predictions; changing the value of the third latent variable leads to predictions in a more restricted area.

The model can be seen as a function:

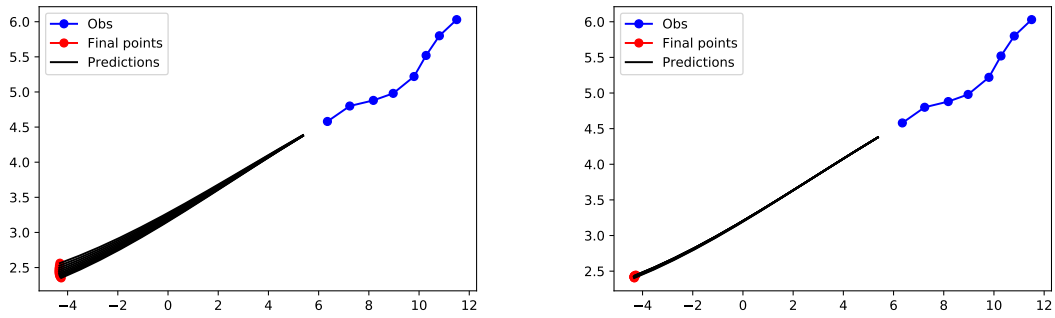
$$NN : I\!R^{l\times 2} \times I\!R^{n} \times \{1, 2, \ldots, l'\} \rightarrow I\!R^{2}$$

$$\left([d_1, \ldots, d_l], z, t\right) \mapsto NN\left([d_1, \ldots, d_l], z, t\right) = \hat{d}_t^z,$$

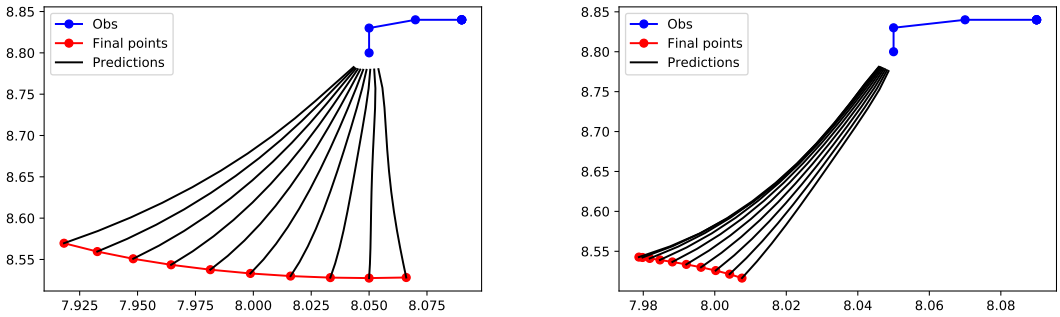
where $[d_1, \ldots, d_l]$ are the observed displacements, n is the latent space dimension, z is a sample from the latent space, l' is the number of future steps to predict and t is a particular time in the prediction horizon. Given the latent variables z and the observed displacements $[d_1, \ldots, d_l]$, $\hat{d}_t^z$ is the prediction at time t.



(a) The pedestrian stands still. On the left, the value of the first latent variable is varied; on the right, the second one is varied.



(b) Case with a clear tendency about the direction. On the left, the value of the first latent variable is varied; on the right, the second one is varied.



(c) Case with an unclear tendency about the direction. On the left, the value of the first latent variable is varied; on the right, the third one is varied.

Figure 5.8: Examples illustrating how the predictions change when a latent variable is gradually incremented from -2 to 2.

Since the network is a composition of continuous functions, given a fixed $t \in \{1, 2, \ldots, l'\}$, $NN([d_1, \ldots, d_l], \cdot, t)$ is a continuous function. In the Figure 5.8, the continuity of $NN([d_1, \ldots, d_l], \cdot, l')$ can be appreciated (red color).

With the SDVAE model, the latent variables have not an obvious role either. In the Figure 5.9, three cases are shown for illustration purposes. A color map is used, and trajectories with the same color correspond to the same latent variable (i.e. same discrete value). Even when there is not a definitive order, in many cases (qualitatively) the predictions are orientated more or less in the same direction. In the three cases, if the yellow prediction is considered in the middle, the green and the cyan ones are in one side and the purple and the blue ones are in the other side. In my opinion, in the images of the top, the yellow predictions seem to be the most reasonable predictions, but in the last one it is not. In the last image, for me, the most reasonable prediction would be the cyan.

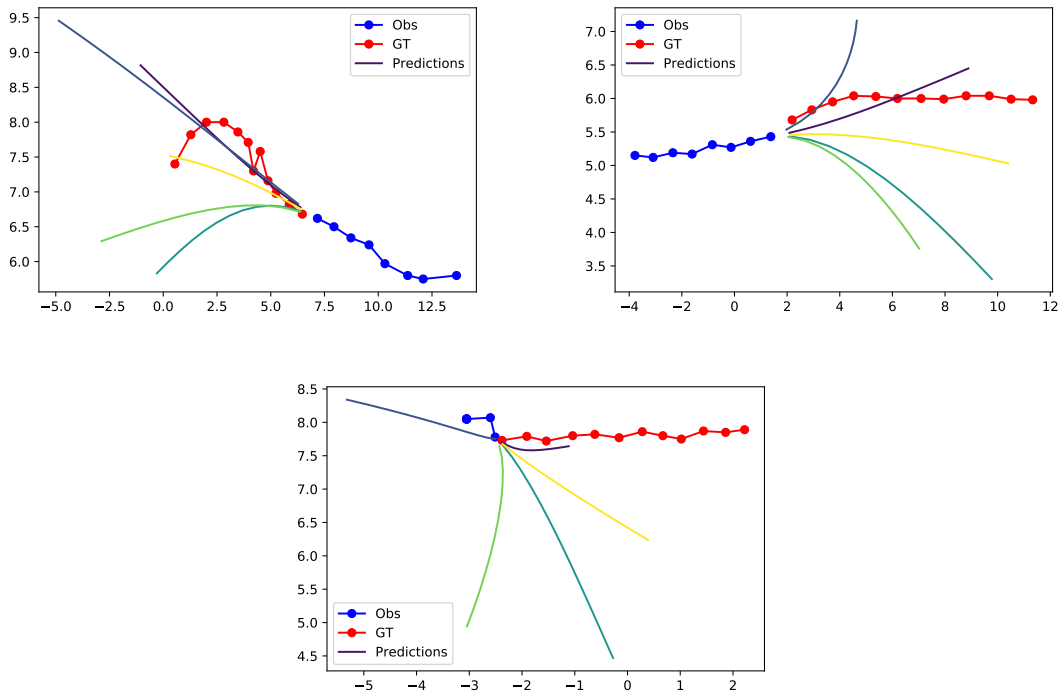


Figure 5.9: Comparison among SDVAE predictions, in three ETH cases. Predictions with the same color correspond to the same value of the discrete latent variable.

As I said before, it is hard to conclude something about the latent variables in qualitative terms. This section is rather illustrative and exploratory.

5.2 Crossroad

This dataset has been presented in Subsection 3.2. In the experiments described in this Section, three of the four scenes (1, 2 and 4) are used to train the model, while the last one (3) is reserved for testing. There is no particular reason for this arrangement. After each epoch, the model is tested and the best weights according to mADE are saved. Those weights are used in the end to evaluate the model in different aspects. It is important to notice that, in contrast with the ETH-UCY protocol followed above, this is not a rigorous machine learning process. This experiment has the intention of exploring the potential of including Semantic Segmentation (SS) and, for this purpose, keeping the evolution of the metrics in the same trajectories across the epochs is enough.

First, the results with SDVAE are shown and then the ones with VAE. The SDVAE experiments are focused on incorporating SS, and the VAE experiments on verifying that training with teacher forcing is helpful.

5.2.1 Hyperparameters

When Semantic Segmentation (SS) is used, the patches are centered in the pedestrian and they have a size of $(300, 300, 2)$. They are extracted from an image of the scene with a size of $(512, 512, 2)$. The last dimension is a hot encoding category representation: obstacle or not. To train the model, the patches are resized to $(64, 64, 2)$. In the next subsections, the trajectories are plotted in an image of the whole scene. When SS is incorporated to do the prediction, a square is drawn, and this is the last patch observed by the model.

To extract features from the patches, a six-block sub-network $f(\cdot)$ is designed as explained in Subsection 4.2: each block is composed of a convolutional layer and a maximum pooling layer. All the convolutional layers have 32 filters of size $(3, 3)$ and the $\tanh(\cdot)$ function as activation function. If a block receives an input with shape (a, a, x), it gives an output with shape $(0.5a, 0.5a, 32)$: the convolutional layer gives an output with shape $(a, a, 32)$; the maximum pooling layer takes this output and gives the final output with shape $(0.5a, 0.5a, 32)$. In the first layer, we have $x = 2$, and in the last layers, we have $x = 32$. The code is designed to receive a scalar a as

a power of two. Since there are six blocks, the encoding has dimensions

$$\left(\frac{64}{2^6}, \frac{64}{2^6}, 32\right) = (1, 1, 32)\,.$$

This output is flattened, mapped to $\mathbb{R}^{128}$ with a dense layer and concatenated with the transformed displacements.

In the Subsection 5.2.2, the weights of the sub-network $f(\cdot)$ are randomly initialized and adjusted during training. In Subsection 5.2.3, the sub-network $f(\cdot)$ is pre-trained using an Autoencoder (AE). After training the AE, the features for each patch are pre-computed and incorporated as planar vectors to the model. This is equivalent to initialize the weights of $f(\cdot)$ with the encoder weights from the pre-trained AE and freeze them during the training of the SDVAE. The AE for patches is trained with Adam through 5 epochs, with a learning rate of 0.001 and a batch size of 256 (32 sequences of patches with length 8, without considering the temporal structure).

In this Section, all the models are trained with the Adam optimizer. VAE models are trained with a learning rate of 0.001 and a batch size of 128. The learning rate and the batch size in the training of SDVAE models depend on the use of semantics: if SS is incorporated and the weights of $f(\cdot)$ are adjustable, the learning rate is 0.005 and the batch size is 128; otherwise, when no semantics is used or if a pre-encoding is done, the learning rate is 0.0005 and the batch size is 32. The SDVAE models are trained through 50 epochs and the VAE models through 100 epochs. With VAE models, the divergence penalization weight β (Eq. 4.7) is set to 0.005.

All the LSTM layers have the same hyperparamters (encoder and decoder in SDVAE and VAE): the activation function is the $\tanh(\cdot)$ function; the hidden state dimension and the output dimension are both 256. Before encoding the displacements, in both models, these displacements are mapped from $\mathbb{R}^2$ to $\mathbb{R}^{128}$ with a dense layer. When the models are trained with teacher forcing, to perform decoding, the displacements are mapped to $\mathbb{R}^{128}$ with another dense layer.

5.2.2 SDVAE: Ablation Study

To conduct the ablation studies in this Section, the average of the standard deviation of the distances between the final points and their mean is reported. Hereafter is explained how these deviations are computed and why they are used here.

The mean of the final points (over the k generated samples) is given by

$$\mu_{l'}(x) = \frac{1}{k}\sum_{i=1}^{k} \hat{y}_{l'}^{i}(x),$$

where k is the number of predictions done per data sample (x, y) and l' is the number of predicted positions. The standard deviation of the distances of the final points to their mean is evaluated, again, over the k samples:

$$\sigma_{l'}(x) = \sigma\left(\left\{\left\|\hat{y}_{l'}^{1}(x) - \mu_{l'}(x)\right\|, \ldots, \left\|\hat{y}_{l'}^{k}(x) - \mu_{l'}(x)\right\|\right\}\right).$$

If there are n tested pairs (x, y), then the average of all the values $\sigma_{l'}$ is reported:

$$\frac{1}{n}\sum_{x} \sigma_{l'}(x).$$

Ideally, when some additional feature is incorporated in our model, we would hope that in some cases the uncertainty on the final position is reduced. Hence, I propose $\sigma_{l'}$ as a proxy to quantify the improvement coming from the additional feature. By itself, this measure is not enough, because it does not involve the ground truth, but mADE/mFDE do, so $\sigma_{l'}$ can be a good complement for them.

In the case of the Crossroad dataset, the SS (see Subsection 3.2) can be used as a binary map to know when a prediction is a priori not possible because of the presence of obstacles. Hence, the average of Invalid Positions Amount (IPA) per set of predictions is reported. Given a sample (x, y), the IPA is defined as

$$IPA(x) = \sum_{i=1}^{k}\sum_{j=1}^{l'} ss_{inv}\left(int\left(\hat{y}_{j}^{i}\right)\right),$$

where we sum over the generated samples (index i) and over predicted timesteps (index j). The function $int(\cdot)$ casts all the entries of the argument to integers. The

function $ss_{inv}(\cdot)$ returns 1 if the argument corresponds to an invalid position (within the obstacles) and 0 if it corresponds to a valid position (outside of the obstacles). A prediction can be at an invalid position if it goes through a wall or out of the image. If there are n tested pairs (x, y), the average Invalid Positions Amount is reported:

$$\frac{1}{n}\sum_{x} IPA(x).$$

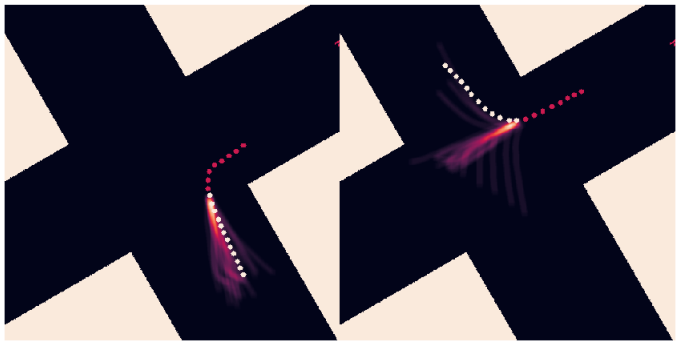
In the same way as in Subsection 5.1.3, the effect of teacher forcing and the effect of adding extra features are experimentally studied. Instead of incorporating APGs (see Subsection 5.1.3), SSs are used as depicted in Figure 4.6. The results are presented in Table 5.9. Each metric is the average of five runs. If we look only at mADE, mFDE and IPA, teacher forcing seems to help in both cases: with and without semantics. Incorporating the semantics also seems to be beneficial in both cases: with and without teacher forcing.

	TeacherF(1). SS(1).				TeacherF(1). SS(0).			
	μ	σ	min	max	μ	σ	min	max
mADE	3.718	0.10	3.57	3.86	4.321	0.10	4.26	4.54
mFDE	5.781	0.20	5.44	6.08	7.172	0.21	6.91	7.42
$\sigma_{l'}$	7.996	0.85	6.66	9.11	30.73	0.87	29.9	32.1
IPA	0.326	0.04	0.28	0.38	3.072	0.35	2.60	3.63
	TeacherF(0). SS(1).				TeacherF(0). SS(0).			
	μ	σ	min	max	μ	σ	min	max
mADE	3.275	0.09	3.14	3.37	4.472	0.04	4.42	4.52
mFDE	4.865	0.12	4.68	4.99	7.425	0.03	7.38	7.46
$\sigma_{l'}$	9.657	0.64	8.86	10.4	29.45	0.74	28.8	30.9
IPA	0.367	0.07	0.28	0.48	3.615	0.41	3.14	4.34

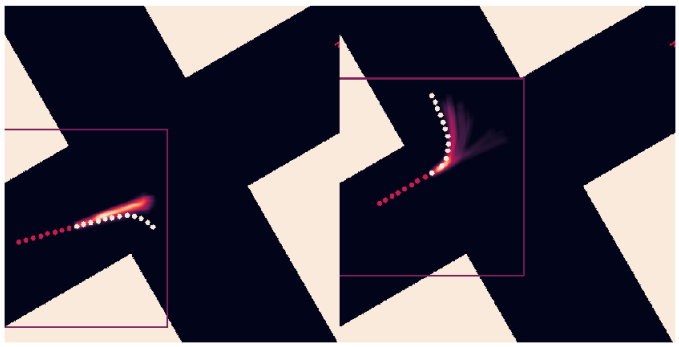
Table 5.9: SDVAE: ablation study on teacher forcing and SS on the crossroad dataset. Each configuration is run five times and statistics of four metrics are reported along the table rows.

When no semantics are used, teacher forcing increases the value of $\sigma_{l'}$. In general, this could be either good or bad, because if $\sigma_{l'}$ goes up, that means that in average,

the predictions are more diverse. Multi-modal predictions are good in some cases, but if there are very different predictions, they could be useless to make decisions based on these predictions. Incorporating semantics helps to reduce the deviation $\sigma_{l'}$ and, at the same time, it improves the other metrics. This means that the model is more confident in some cases (which does not mean that it is right) and, in general, it corresponds to a better prediction.



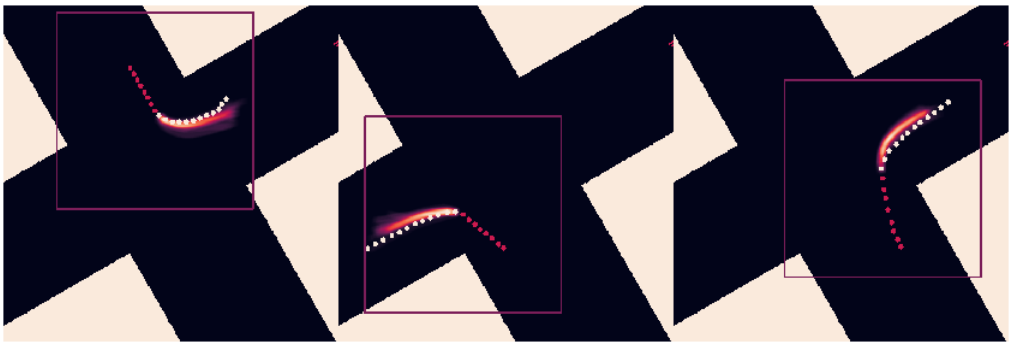
(a) Cases with minimum and maximum $\sigma_{l'}$ in SDVAE without features.



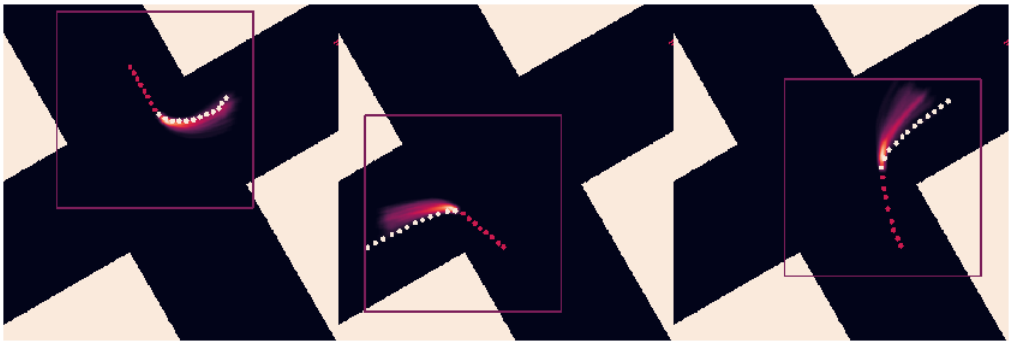
(b) Cases with minimum and maximum $\sigma_{l'}$ in SDVAE with features.

Figure 5.10: Extreme cases for $\sigma_{l'}$ for the SDVAE model trained with teacher forcing.

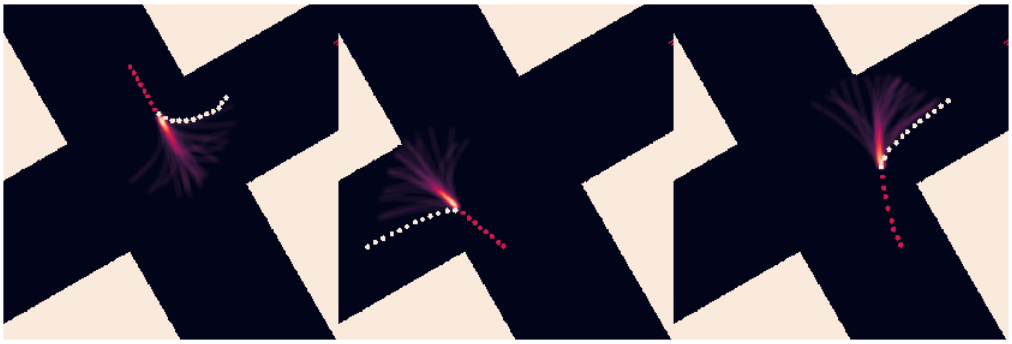
Among both models using SS, it is not possible to tell that one configuration is better than the other. If only the mADE and mFDE metrics are considered, the model trained without teacher forcing is the best. Nevertheless, the deviation $\sigma_{l'}$ is greater with this model. This means that, in average, the predictions are a bit more disperse. In this case, I consider it good and as an indication that the model has more potential in behaving better with multi-modality involved in the predictive distribution. In average, the IPA metric is very similar in both cases.



(a) SDVAE trained with teacher forcing and semantics.



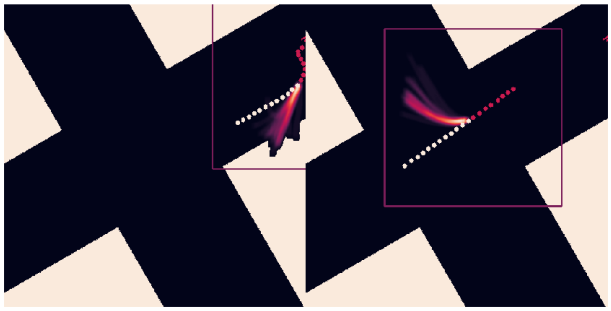
(b) SDVAE trained without teacher forcing and with semantics.



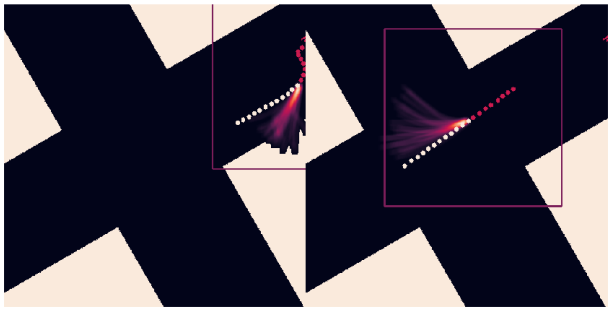
(c) SDVAE without teacher forcing and without semantics.

Figure 5.11: Good turn cases with the SDVAE model.

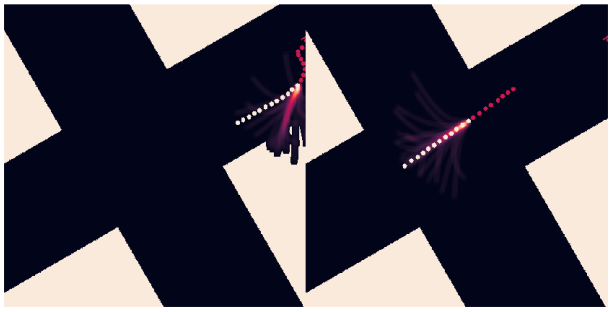
In the Figure 5.10, a few extreme $\sigma_{l'}$ cases of some runs are shown. Each row corresponds to a different run. On the top row at the right, it can be seen that the model without feature predicts that the pedestrian can turn around at any moment. On the bottom, it can be seen that the model using SS makes "cleaner" predictions; on the right, it seems to capture the multi-modality of the predictive distribution very well.



(a) SDVAE trained with teacher forcing and SS.



(b) SDVAE trained without teacher forcing and with SS.



(c) SDVAE trained without teacher forcing and without SS.

Figure 5.12: Bad cases for the SDVAE model trained with teacher forcing and SS.

In the Figure 5.11, some turn cases are shown, where the pedestrian turns and

the model seems to improve with semantics. In the last two cases from left to right, even when the models with SS predict that the pedestrian will turn, they can not predict well the length. Also, paying attention to the predictions from the model trained without teacher forcing (Figure 5.11b), it is possible to see that they are more disperse than the predictions from the model trained with teacher forcing (Figure 5.11a).

Even when it does it well in some cases, there are other cases where the predictions are pretty bad. In the Figure 5.12, we can see different such situations: on the left, the pedestrian does a weird movement and the models do not "understand" that the pedestrian can not continue straight because of the wall; on the right, the first model predicts that the pedestrian will turn, but it does not. The ideal would be a multi-modal output, something similar to the second model. The disadvantage is that the second model does not define well both modes and the outputs are disperse.

5.2.3 Extracted Features from the Map Patches

The models using Semantic Segmentation (SS) apply a function composed by blocks of convolutional and maximum pooling layers to extract features from the patches (Subsection 4.2). It is reasonable to think that it is possible to reconstruct the patches from these features. This idea is explored in this Subsection with Autoencoders (AEs) (see Section 2.5).

In order to verify that the model is extracting representative features from the patches, an encoding-decoding scheme is followed (Section 2.5). Particularly, an AE is used. The encoder is the six-block network described in Subsection 5.2.1. The decoder uses transpose convolutional layers. In this case, a transpose convolutional layer is used to map an input with a shape of $(a, a, 32)$ to a grid with a shape of $(2a, 2a, x)$. There are six such layers: the first layers have 32 filters and the last one has 2 filters. The decoder output has a shape of $(64, 64, 2)$.

With the encoder and the decoder defined, it is possible to create an AE. To perform training, we use the cross-entropy loss. The same partition as in the last Subsection is used: only patches from scenes 1, 2 and 4 are taken to train; the AE is tested on patches coming from the scene 3. There is no validation set, and the last weights are kept. Two AEs are trained. In the first one, the encoder uses the

weights of a trained SDVAE and those weights are frozen; the decoder weights are randomly initialized and trainable.

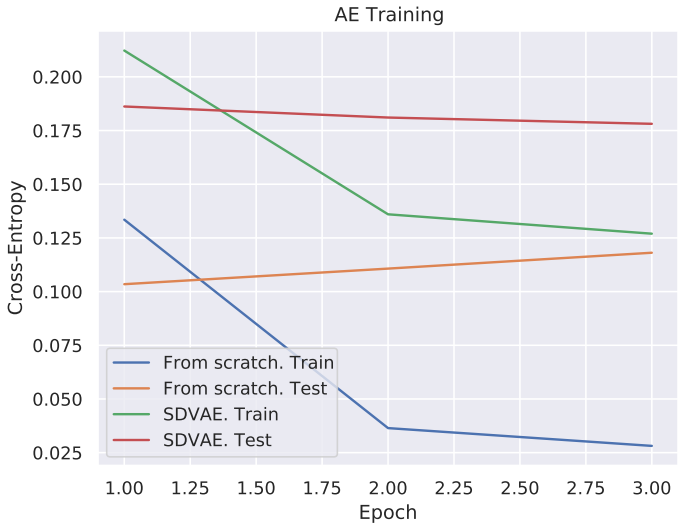


Figure 5.13: Cross-entropy in two AEs for patches.

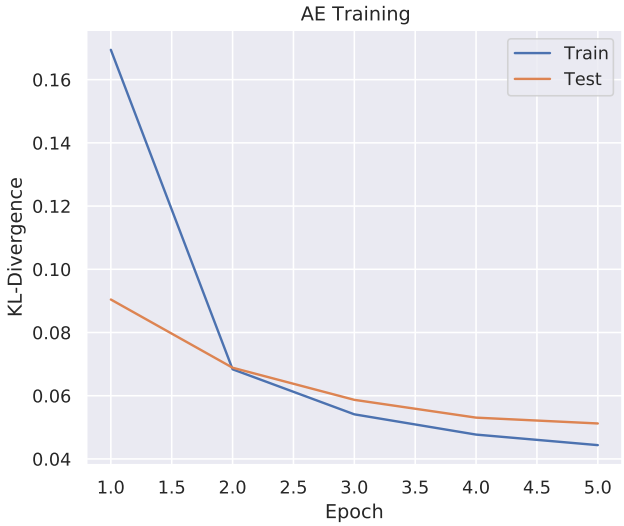


Figure 5.14: KL divergence in AE for patches trained with data augmentation.

The second one is an AE made from scratch: the encoder and decoder weights

are randomly initialized and all the parameters are trainable. In the Figure 5.13, the evolution of the loss function through epochs is shown. In both cases, the loss function looks good in training, but the test loss does not improve as well in training. When comparing the losses, the AE trained from scratch goes better. This is no surprise, as SDVAE was not trained to reconstruct the patches.

Qualitatively, the global structure of the patches can be roughly reconstructed from SDVAE features, but the details are not always very good. Some of them have holes and pieces of path are lost or invented. In the Figure 5.15, a few original patches and their reconstruction from the two AEs are shown.

	TeacherF(1). Pre-Encoding(0).				TeacherF(1). Pre-Encoding(1).			
	μ	σ	min	max	μ	σ	min	max
mADE	3.718	0.10	3.57	3.86	3.553	0.19	3.43	3.74
mFDE	5.781	0.20	5.44	6.08	5.470	0.28	5.10	5.84
$\sigma_{l'}$	7.996	0.85	6.66	9.11	7.591	0.61	6.82	8.50
IPA	0.326	0.04	0.28	0.38	0.278	0.11	0.15	0.47
	TeacherF(0). Pre-Encoding(0).				TeacherF(0). Pre-Encoding(1).			
	μ	σ	min	max	μ	σ	min	max
mADE	3.275	0.09	3.14	3.37	3.289	0.10	3.11	3.41
mFDE	4.865	0.12	4.68	4.99	4.903	0.25	4.57	5.13
$\sigma_{l'}$	9.657	0.64	8.86	10.4	10.05	1.50	8.81	12.8
IPA	0.367	0.07	0.28	0.48	0.236	0.10	0.15	0.43

Table 5.10: SDVAE: ablation study. Teacher forcing and pre-encoding in the Crossroad dataset. Each configuration is run five times and statistics of four metrics are reported.

These results encourage the idea of a pre-embedding. If the patches are previously encoded, then the training of SDVAE would be faster. In order to do this, an AE is pre-trained to do the encodings. In light of the results shown in Figure 5.13, the AE is trained using data augmentation. During training, each patch is randomly rotated. After the rotation, the last dimension is normalized to have a distribution over both categories and the KL divergence loss is used. In the Figure 5.14, we can

see that this data augmentation process facilitates training. There is no validation dataset, and the last weights are kept for further testing.

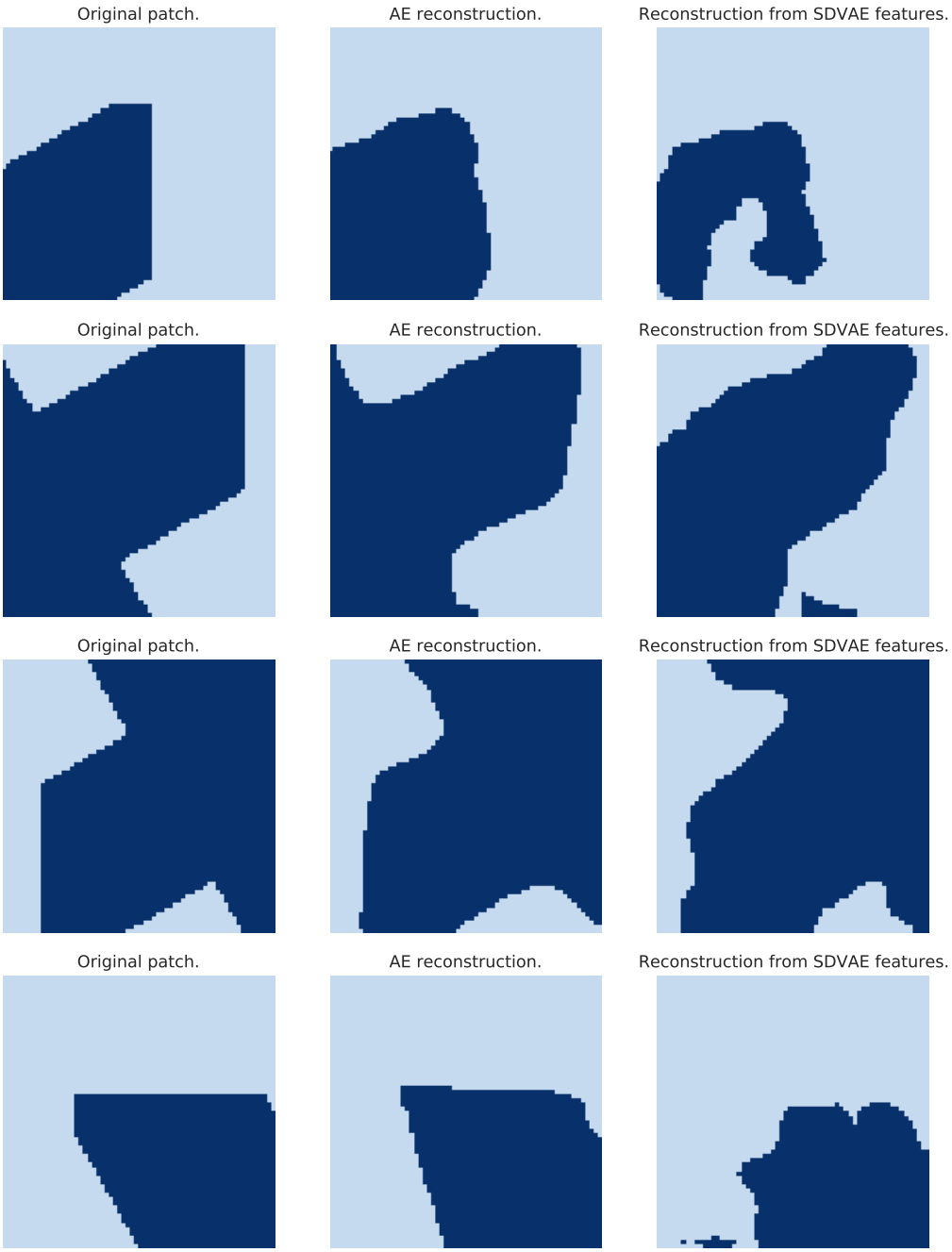


Figure 5.15: Original patches and their reconstruction from two AE.

With the pre-trained AE, the patches are mapped to $\mathbb{R}^{32}$ before training and those vectors are incorporated to the SDVAE model. The results are shown and compared with the previous ones in Table 5.10.

In terms of the mADE and mFDE metrics, the models trained without teacher forcing are pretty similar; in average, the predictions are a bit more diverse and there are less predictions on invalid positions. On the other hand, the model trained with teacher forcing improves in all the metrics when a pre-encoding is done.

5.2.4 VAE: Ablation Study

In the same way as in Subsection 5.1.2, an ablation study is done to have a better insight of how teacher forcing and the inclusion of ground truth data (trying to improve the sampling in training) affect the VAE model performance. Also, SS is incorporated to the ablation study. Since the SS pre-encoding worked well for the SDVAE model (Subsection 5.2.3) and since it is computationally cheaper than extracting features from scratch (Subsection 5.2.2), in this Subsection only the pre-encoding strategy is followed to incorporate SS to the VAE model.

	TeacherF(1). Y(1). SS(0).				TeacherF(1). Y(0). SS(0).			
	μ	σ	min	max	μ	σ	min	max
mADE	5.702	0.18	5.37	5.87	6.438	0.21	6.04	6.59
mFDE	9.932	0.36	9.39	10.5	12.01	0.60	10.8	12.5
$\sigma_{l'}$	14.02	0.55	13.0	14.6	12.92	1.70	11.6	16.2
IPA	4.523	0.32	4.11	4.93	5.267	1.20	4.29	7.68
	TeacherF(0). Y(1). SS(0).				TeacherF(0). Y(0). SS(0).			
	μ	σ	min	max	μ	σ	min	max
mADE	5.929	0.19	5.57	6.11	10.05	0.28	9.68	10.4
mFDE	10.49	0.45	9.82	11.0	23.01	0.84	21.8	23.9
$\sigma_{l'}$	14.40	1.10	13.3	16.0	3.867	0.25	3.52	4.17
IPA	3.663	0.51	2.93	4.36	2.924	0.43	2.55	3.59

Table 5.11: 100VAE: ablation study. Teacher forcing and ground truth encoding in the Crossroad dataset without SS. Each configuration is run five times.

In the Table 5.11, the results without SS are presented and in the Table 5.12, the results with SS are presented. All the metrics are computed with 100 predictions (100VAE). In both tables, it can be seen that training with teacher forcing helps to get better values for mADE and mFDE, specially when the ground truth is not used. Using the ground truth to train the model leads to better results as well. It is possible to see that the model that uses only SS (bottom right of Table 5.12) works better than all the models from Table 5.11.

	TeacherF(1). Y(1). SS(1).				TeacherF(1). Y(0). SS(1).			
	μ	σ	min	max	μ	σ	min	max
mADE	4.373	0.21	4.08	4.64	4.432	0.27	4.09	4.90
mFDE	7.524	0.39	7.02	7.92	7.977	0.67	7.08	9.06
$\sigma_{l'}$	2.067	0.46	1.51	2.72	2.376	1.10	0.90	4.07
IPA	0.852	0.08	0.74	0.97	1.115	0.41	0.69	1.68
	TeacherF(0). Y(1). SS(1).				TeacherF(0). Y(0). SS(1).			
	μ	σ	min	max	μ	σ	min	max
mADE	4.417	0.20	4.12	4.72	5.010	0.09	4.91	5.15
mFDE	7.576	0.42	7.06	8.23	9.839	0.28	9.52	10.2
$\sigma_{l'}$	2.723	0.32	2.22	3.21	1.137	0.07	1.05	1.26
IPA	0.551	0.03	0.49	0.59	0.739	0.23	0.37	1.07

Table 5.12: 100VAE: ablation study. Teacher forcing and ground truth encoding in the Crossroad dataset with SS. Each configuration is run five times and statistics of four metrics are reported.

When SS is included (Table 5.12), the four metrics indicate that the model improves. An important observation is that even when using 100 predictions, this model is not able to reach the performance of the SDVAE model in the mADE and mFDE metrics, which uses only 20 predictions (Table 5.10). Comparing the diversity on VAE and SDVAE trajectories ($\sigma_{l'}$), it seems that the SDVAE model is more efficient in capturing multi-modality. Qualitatively, this idea is reinforced through the Figure 5.16, where extreme $\sigma_{l'}$ cases are shown. It can be seen that the case with a maximum value of $\sigma_{l'}$ in a run is a turn case and that only one mode is captured.

I watched the 20 cases with highest values of $\sigma_{l'}$ in the five runs of the VAE model and in none of them I found a case where two modes are captured.

In the Figure 5.17, some good turn cases are shown, the same shown in the case of the SDVAE model. VAE with semantics is capable of detecting, in some cases, when the pedestrian is going to turn. Comparing with the VAE model, the predictions are concentrated in a smaller and cleaner region. Maybe there are latent variables that lead to another mode, but if there is no easy way to find them with a normal sampling process, which is a negative point for this model.

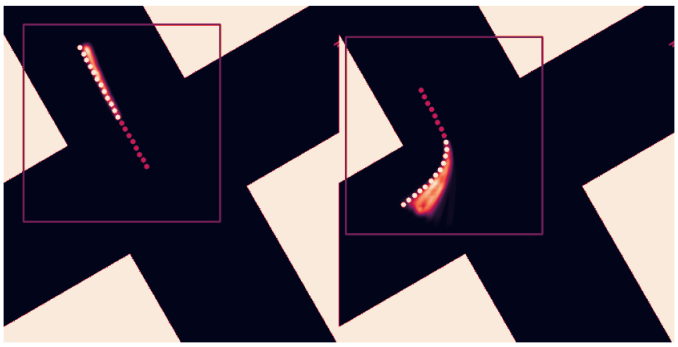


Figure 5.16: Extreme cases for $\sigma_{l'}$ with a 100VAE model trained with teacher forcing. Minimum on the left and maximum on the right.

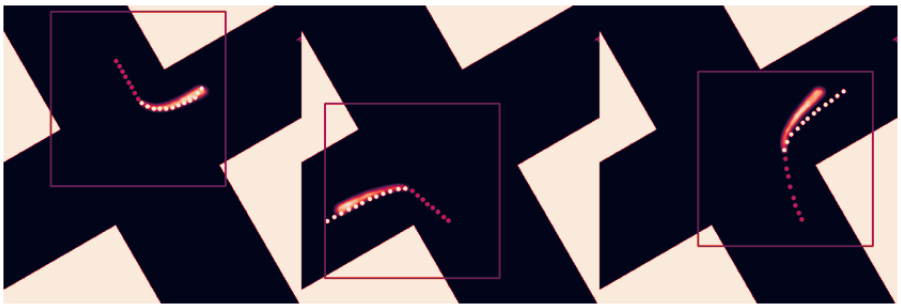


Figure 5.17: Good turn cases in the case of a 100VAE model.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

According to the results presented in the last Chapter, we may conclude the following:

- In all the experiments, the results show that training the VAE model with teacher forcing, or incorporating the ground truth data to learn $q(z|x, y)$ to do the sampling instead of $q(z|x)$, helps to get better results or, at least, does not affect the performance.

- In all the experiments, in contrast with the VAE, the results indicate that training the SDVAE without teacher forcing is the best option.

- Incorporating extra features is not always straightforward, as we saw in the case of the ETH-UCY dataset. In that case, the use of APG can be a distraction for the model and make the results worse. Despite that failure, we saw that the SDVAE model can successfully extract features from the patches in order to improve the predictions in the Crossroad dataset. We also saw that it is possible to improve the VAE and SDVAE models results by doing a pre-encoding of the patches as well.

- According to the obtained results, the SDVAE model has more potential than the VAE model.

- Both models that we presented are able to get competitive results on the ETH-UCY benchmark, specially the SDVAE model.

6.2 Future Work

I consider important to complement this work, in the future, with some of the following points:

- Testing the models on more datasets and evaluating whether the first two points from the conclusions (last Section) remain verified.
- Improving our SDVAE model and get a deeper insight on it. For example, we are looking into modifying the model to be able to rank the predictions. Another potential path is to incorporate an attention model in the observed trajectories or features. On the other hand, an important limitation on this model is the fixed number of prediction samples, hence it would be good to modify it to get as many samples as desired.
- Improving our VAE model. A potential path is to study the new VAE model [27], and see why it behaves better than traditional structures. Then we could try to apply it in this context.
- Studying the models properties from a more theoretical perspective.

Bibliography

[1] Adobe (2020). Adobe mixamo. https://www.mixamo.com/#/. Accessed: 2020-02-22.

[2] Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. (2016). Social lstm: Human trajectory prediction in crowded spaces. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971.

[3] Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

[4] Doersch, C. (2016). Tutorial on variational autoencoders.

[5] Forsyth, D. A. and Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference.

[6] Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep ensembles: A loss landscape perspective.

[7] Giuliari, F., Hasan, I., Cristani, M., and Galasso, F. (2020). Transformer networks for trajectory forecasting.

[8] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[9] Gupta, A., Johnson, J., Fei-Fei, L., Savarese, S., and Alahi, A. (2018). Social GAN: socially acceptable trajectories with generative adversarial networks. *CoRR*, abs/1803.10892.

[10] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[11] Ivanovic, B. and Pavone, M. (2018). Modeling multimodal dynamic spatiotemporal graphs. *CoRR*, abs/1810.05993.

[12] Kosaraju, V., Sadeghian, A., Martín-Martín, R., Reid, I. D., Rezatofighi, S. H., and Savarese, S. (2019). Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. *CoRR*, abs/1907.03395.

[13] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[14] Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks.

[15] Lee, N., Choi, W., Vernaza, P., Choy, C., Torr, P., and Chandraker, M. (2017). Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2165–2174.

[16] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867.

[17] Liang, J., Jiang, L., Murphy, K., Yu, T., and Hauptmann, A. (2020). The garden of forking paths: Towards multi-future trajectory prediction. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[18] Liang, J., Jiang, L., Niebles, J. C., Hauptmann, A. G., and Fei-Fei, L. (2019). Peeking into the future: Predicting future person activities and locations in videos. *CoRR*, abs/1902.03748.

[19] Pfeiffer, M., Paolo, G., Sommer, H., Nieto, J. I., Siegwart, R., and Cadena, C. (2017). A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments. *CoRR*, abs/1709.08528.

[20] Ridel, D., Deo, N., Wolf, D., and Trivedi, M. (2019). Scene compliant trajectory forecast with agent-centric spatio-temporal grids.

[21] Sadeghian, A., Kosaraju, V., Sadeghian, A., Hirose, N., and Savarese, S. (2018). Sophie: An attentive GAN for predicting paths compliant to social and physical constraints. *CoRR*, abs/1806.01482.

[22] Salzmann, T., Ivanovic, B., Chakravarty, P., and Pavone, M. (2020). Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data.

[23] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

[24] Shi, X., Chen, Z., Wang, H., Yeung, D., Wong, W., and Woo, W. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214.

[25] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*.

[26] Unity3D (2020). Unity3D. www.unity.com. Accessed: 2020-02-22.

[27] Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder.

[28] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[29] Wang, H., Raj, B., and Xing, E. P. (2017). On the origin of deep learning. *CoRR*, abs/1702.07800.

[30] Yamaguchi, K., Berg, A. C., Ortiz, L. E., and Berg, T. L. (2011). Who are you with and where are you going? In *CVPR 2011*, pages 1345–1352.

[31] Zhang, P., Ouyang, W., Zhang, P., Xue, J., and Zheng, N. (2019). SR-LSTM: state refinement for LSTM towards pedestrian trajectory prediction. *CoRR*, abs/1903.02793.