CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Generation and analysis of driving scenario trajectories for safety verification of Autonomous Drive

Master's thesis in Computer Science and Engineering

Andreas Demetriou
Henrik Alfsvåg

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Generation and analysis of driving scenario trajectories for safety verification of Autonomous Drive

Andreas Demetriou
Henrik Alfsvåg

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Trajectory generation and analysis with Machine Learning
Andreas Demetriou
Henrik Alfsvåg

© Andreas Demetriou, 2020.
© Henrik Alfsvåg, 2020.

Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering
Advisor: Sadegh Rahrovani, VolvoCars
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Trajectory generation and analysis with Machine Learning
Andreas Demetriou
Henrik Alfsvåg
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Autonomous vehicles are often considered to be the future of transportation. However, many believe fully-autonomous vehicles are still far away from being realized. One of the main reasons is the lack of confidence from a safety level standpoint. While car manufacturers such as Volvo Cars have made great efforts to prove that developed algorithms are safe enough, it is just not feasible to test each new feature in the fields. Luckily, virtual environments address this issue by reproducing traffic situations - scenarios. But even with virtual environments, a lot of data still has to be collected to cover the whole spectrum of possible scenarios. This can be addressed with simulated scenarios. Another approach that is investigated in this thesis is generating scenarios based on real ones. We considered two approaches: Recurrent Conditional Generative Adversarial Nets and a Recurrent Autoencoder with Generative Adversarial Nets. The latter also shows useful properties for scenario analysis. The second problem that we try to tackle in this thesis is the evaluation of our developed models.

Keywords: Generative Adversarial Nets, Recurrent Networks, Autonomous vehicles, Deep Learning.

Acknowledgements

There are a lot of people whom we would like to thank. First, we are grateful to Sadegh Rahrovani, our supervisor from Volvo Cars, who proposed this project and helped us during it by providing many ideas and directions for the development. We are equally grateful to Morteza Haghir Chehreghani, our academic supervisor, who lead all our joint meetings, always helped with any of our concerns and shared his research experience with us. Also, we would like to thank Devdatt Dubhashi for providing feedback in a timely manner. Moreover, we would like to thank Viktor Wänerlöv, Rune Suhr and Richard Brodie for their help in getting us into the project and all the fruitful meetings we had together. We are also thankful to Martin Magnusson for the opportunity to conduct our thesis at the Scenario Analysis Team. Lastly, we want to thank Volvo Cars for providing everything necessary for this work and being well organized even in a time of a world pandemic.

Andreas Demetriou, Gothenburg, June 2020
Henrik Alfsvåg, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

Autonomous vehicles have been a hot topic in recent years. It is easy to see why as the driving behavior of people is far from ideal. Thus, we could see a lot of benefits by taking that erratic part out of the equation. This includes a reduction in emissions by getting rid of phantom traffic jams [1], increasing the lane capacity given superior response times compared to humans as well as reducing the number of accidents by getting rid of intoxicated drivers to name a few [2]. However, autonomous drive (AD) is not foolproof and comes with its own set of challenges. AD can best be described as a 'black box' that generates a set of instructions for the car to follow based on a lot of different inputs from radars, cameras, and LiDARs. Thus, it becomes extremely hard to say with certainty whether or not the outcome will be correct as there are many unknowns. For instance, the measurements from the sensors may contain errors. Furthermore, images captured by the camera are processed using Computer Vision which nowadays is most often based on Deep Learning techniques. Hence, non-obvious errors such as mislabeling can occur [3]. The list can be made long, but the main takeaway is that there are a lot of ambiguities. To make matters worse, AD must outperform human drivers by several orders of magnitude in order to replace them [4]. Therefore, to reach this milestone, the verification of AD through meticulous testing is just as crucial as the actual implementation.

1.1 Context

In order to assess AD safety with confidence, statistical analyses have shown that fully autonomous vehicles would have to be driven for hundreds of millions of kilometers [5, 4]. This is not feasible, particularly in cases when we need to assess different system design proposals or in case of system changes, since the same amount of distance needs to be driven again by the AD vehicle for the verification sign-off.
To address some of the above mentioned issues, an alternative verification approach is proposed known as scenario-based verification. This approach uses a scenario database created by extracting driving scenarios/events (e.g. cut-in, overtaking, etc.) that the AD vehicle is exposed to in naturalistic driving situations [6]. Once such a scenario database has been developed, it can be used for test case generation and verification of the AD functionality in a virtual environment [7].
However, several challenges must be addressed in order to create a reliable scenario database. As previously mentioned, a huge amount of data is still needed, and physical collection of real scenarios is not a reasonable option. Thus, to expand the database, one potential solution can be generation of realistic scenarios. Sce-

nario extraction is another challenge with a scenario-based approach, as it requires properly labeled data. Scenarios are extracted from time series (sequence of the ego-vehicle states) which in turn are processed data collected by sensors of the AD vehicle. To the best of our knowledge, scenario extraction, in general, can be addressed with two approaches: either a clustering approach or an explicit rule-based approach [8, 9, 10]. In general, clustering is preferable as explicit rules are very likely to miss outliers. In addition, explicit rules require expert domain knowledge and they become harder to formulate when the dimensionality of data increases.

1.2 Problem

The main goal of the thesis is to generate realistic scenario trajectories that can be used for verification. To describe a trajectory we consider two features: relative latitude and longitude position of the tracked vehicle which are obtained by radar on the ego-vehicle as schematically showed in Fig. 1.1. There are three major components to achieving this goal (presented further as subsections).

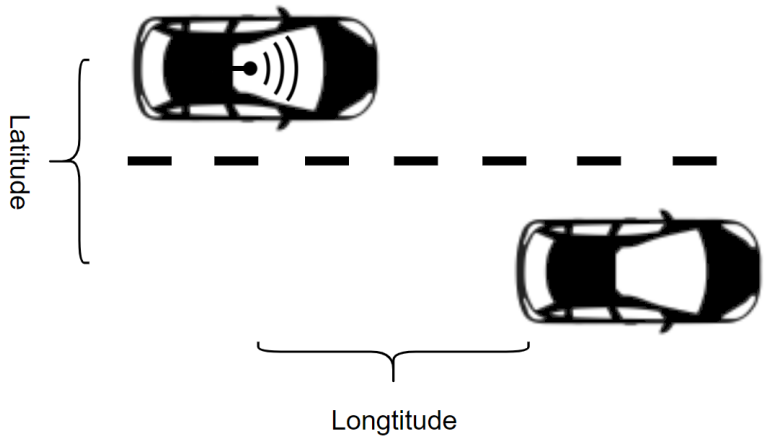


Figure 1.1: Relative position of tracked vehicle.

1.2.1 Data

We are provided with data collected by Volvo Cars Corporation. This data consists of information about the ego-vehicle and its surroundings such as detected objects, information about the road, etc. In order to train our models we first need to extract the scenario trajectories from the data. As was previously mentioned, there are different ways to accomplish this.

1.2.2 Models for generation

While the generation task can be addressed with different Machine Learning techniques, the one considered in this work is Generative Adversarial Networks (GANs) as they have shown great results in tasks related to generation of synthetic data. More specifically, two recurrent architectures will be investigated since the data has

sequential nature: recurrent GANs and recurrent autoencoder combined with GANs. Moreover, recurrent autoencoder produces an embedding of a trajectory that can be used for further analysis such as clustering trajectories of different scenarios or outlier detection.

1.2.3 Evaluation of the models

The final problem to tackle is the evaluation of the models. While this may seem trivial at a first glance, there are no go-to evaluation metrics when dealing with generated data. For example, when it comes to image generation it is very common to perform a visual inspection and determine whether or not it looks good enough. In our case, it is possible to visually inspect the generated trajectories to initially see if it is going in the right direction, however, it is insufficient when determining if they are satisfactory.

1.3 Structure of this report

This report is divided into several chapters. The Theory chapter describes the necessary theoretical background for this project. The Methods chapter explains the application of described theory and different ideas to the defined problems and the steps taken in order to overcome challenges that have emerged. The Result chapter presents the final results and comparisons between the implemented models. Finally, the Discussion and Conclusion chapter presents the discussion about our results together with some proposals for future work.

2
Theory

This chapter introduces the theory that is used in this project. This is a general overview of concepts and approaches, so for deeper understanding provided citations can be investigated. The specific application of the described theory is presented in the Methods chapter.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of Neural Networks (NNs) that is frequently used for processing sequential data [11]. The main difference compared to classical feed-forward NNs is that the neurons in RNNs can be dependent on its previous state. This enables RNNs to process sequential data such as time-series. As this architecture processes each input individually while at the same time taking its internal state determined by the output of the previous input into account, it is well suited for the variable length input.
Due to these recurrent connections, backpropagation that is typically used to train NNs [12] can not be applied directly to RNNs. To address this issue they are unfolded: the recursive structure of RNNs is converted into a sequential structure with repeated steps as presented in Fig. 2.1.

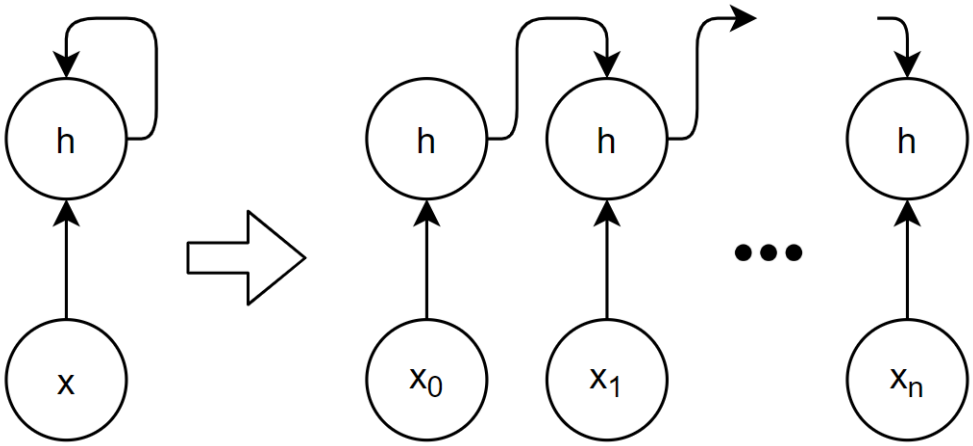


Figure 2.1: Unfolding of RNNs.

While unfolding allows backpropagation to be applied to RNNs, the vanishing/ exploding gradient problem [13] arises as a consequence of unfolding the network. While this can happen to deep feed-forward networks as well, it is much less likely as

their weights are independent for each layer as can be seen when applying backpropagation (which calculates the gradient using the chain rule) to a deep feed-forward NNs: $w_1 \cdot \alpha_1 \cdot w_2 \cdot \alpha_2 \cdot \ ... \cdot w_n \cdot \alpha_n$.
On the other hand, when RNNs are unfolded, a new copy of the network is created for each unrolling (time step in our case) which means that they share the same weights: $w \cdot \alpha_1 \cdot w \cdot \alpha_2 \cdot \ ... \cdot w \cdot \alpha_n$. So, as the gradient is calculated using the chain rule, the first layer in a n layer network will have the same weight multiplied with itself n times. Thus, the earlier the layer is, the more it will be affected by the weight and if it is small it will decrease exponentially and quickly go towards zero which means that the weights there will hardly update (vanishing gradient). On the other hand, if the weight is large, the rate of change will grow exponentially and end up being too big. As a result the training will be unstable (exploding gradient).

2.1.1 LSTM

Since we are dealing with time-series it is important to be able to deal with long-term dependencies as it is crucial to remember things that happened a while back in order to predict the future. One solution is long short-term memory (LSTM) [14] which is a RNN architecture that is designed to avoid problems with exploding and vanishing gradients. The LSTM unit is composed of two states: cell(2.1e) and hidden(2.1f); and three gates: forget(2.1a), input(2.1b) and output(2.1c), that are calculated as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.1a}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.1b}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.1c}$$
$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.1d}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{2.1e}$$
$$h_t = o_t \circ tanh(c_t) \tag{2.1f}$$

.
Here, W and U are weight matrices and b is a bias vector, c_0 and h_0 are initialized with zeros. This structure of LSTM units help to regulate how much old information should be kept and how much new information should be taken in.

2.1.2 Teacher Forcing

Teacher Forcing is a technique used to train RNNs. The idea of this method is to change the output of the cell (out_i) with the ground-truth value (p_i)[15] as presented in Fig. 2.2. If the output from the first cell out_1 is incorrect and is used as an input to the second cell, then it is very likely that out_2 will be incorrect as well. So we increase the chances for the prediction to be correct by replacing out_1 with p_1. This is of course only done during training to stabilize and speed up the process as cells further down the line do not have to wait until all cells leading up to them have adapted to the same extent.

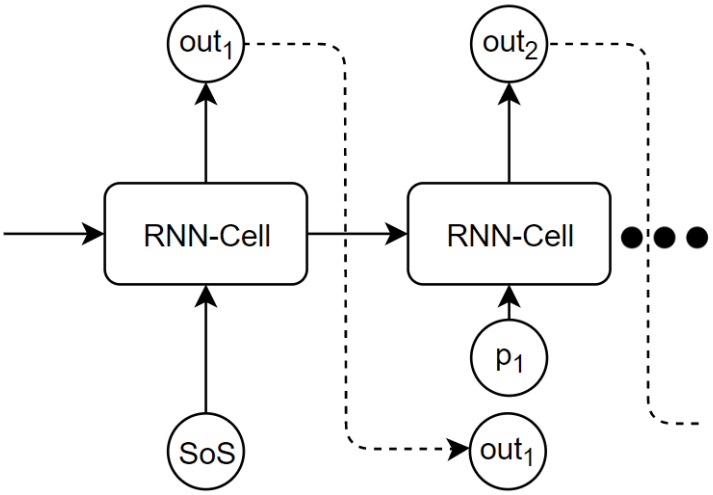


Figure 2.2: Schematic explanation of Teacher Forcing: out_1 is replaced by ground truth p_1.

2.2 Autoencoders

Autoencoders are NNs that are trained to reproduce the input. On the one hand, they are unsupervised since no labels for the data are required, on the other hand, they are trained in a supervised manner since the input itself is used as a label. The structure of an autoencoder could intuitively be split into two parts: an encoder and a decoder, as presented in Fig. 2.3.

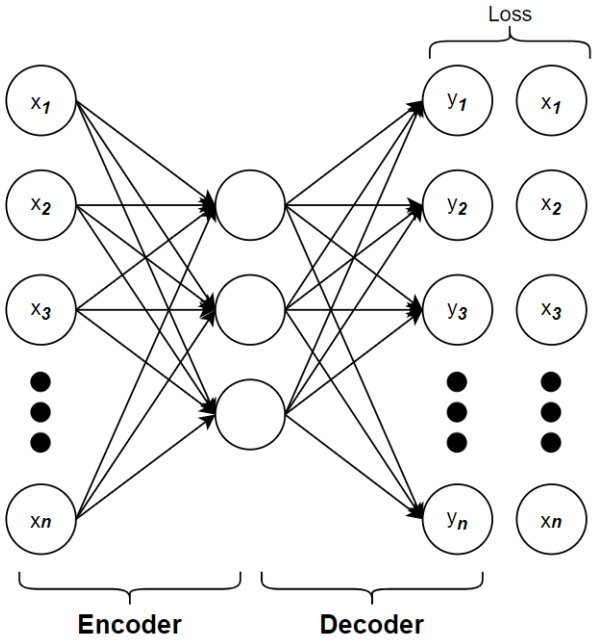


Figure 2.3: Simple architecture of an autoencoder.

The architecture of an autoencoder could be more complex, however, the main idea stays the same: the encoder part is used to learn a compressed (latent space)

representation of the data while the decoder part should be able to reconstruct the data from this latent space representation, minimizing the loss between original and reconstructed samples. Since the latent space has lower dimensions, it will not always be able to reconstruct the data precisely. It is instead approximated and only the most essential information is kept, hence, autoencoders got a denoising property.

2.3 Distance between probability distributions

There are several metrics to measure distance or similarity between probability distributions, however, we consider three of them as only those are related to this project: Kullback–Leibler divergence (KLd) [16], Jensen–Shannon divergence (JSd) [17] and Earth Movers Distance (EMD) [18].

2.3.1 Kullback–Leibler and Jensen–Shannon divergences

The Kullback–Leibler divergence compares two probability distributions by the following formula:

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx \tag{2.2}$$

The KLd would thus be optimal at zero when $p(x) = q(x), \forall x$. The major disadvantage of this metric is its asymmetry. If for instance p(x) is close to zero and q(x) is not, then q's effect is nullified. If we instead would calculate $D_{KL}(q\|p)$ at the same point, it would go to infinity. Moreover, as both distributions are equally important, we can not just pick one over the other. The Jensen–Shannon divergence addresses these issues.
The JSd is based on the KLd as shown in Eq. 2.3. Clearly, JSd is symmetric and it is also bounded by [0, log2].

$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q\|\frac{p+q}{2}) \tag{2.3}$$

2.3.2 Earth Movers Distance

The Earth Movers Distance is named this way as it is can be seen as the minimum cost for moving a pile of dirt in the shape of one probability distribution to form the shape of another. For better understanding consider two one-dimensional histograms P and Q:

$$P_1 = 5, P_2 = 4, P_3 = 2, P_4 = 1$$
$$Q_1 = 3, Q_2 = 5, Q_3 = 4, Q_4 = 0$$

In this simple case, we can calculate the cost that it would take to match P_i and Q_i at each step as δ_i. Moreover, the dirt that may have been shoveled over at the previous time step must be taken into account and the formula is thus: $\delta_{i+1} = \delta_i + P_i - Q_i$. By applying this to our example we get get:

$$\delta_0 = 0$$
$$\delta_1 = \delta_0 + P_1 - Q_1 = 0 + 5 - 3 = 2$$
$$\delta_2 = \delta_1 + P_2 - Q_2 = 2 + 4 - 5 = 1$$
$$\delta_3 = \delta_2 + P_3 - Q_3 = 1 + 2 - 4 = -1$$
$$\delta_4 = \delta_3 + P_4 - Q_4 = -1 + 1 - 0 = 0$$

The final EMD cost is: $\sum |\delta_i| = 4$

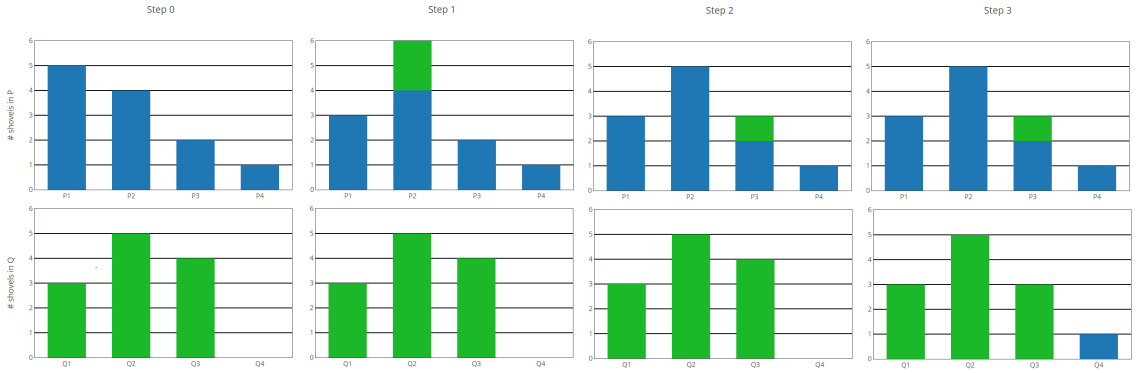


Figure 2.4: The steps taken to match distributions P and Q.

In general, the EMD is formulated as follows:

$$\text{EMD}(a, b) = \inf_{\gamma \sim \Pi(a,b)} \mathbb{E}_{(x,y)\sim\gamma}[|x - y|] \tag{2.4}$$

where $\Pi(a, b)$ is set of all joint probability distributions between a and b. A single joint distribution $\gamma \in \Pi(a, b)$ denotes the dirt transportation plan for that particular distribution. Thus, to make a look like b we get: $\sum_x \gamma(x, y) = b(y)$. As $\gamma(x, y)$ denotes the amount of dirt moved we multiple it by the distance $|x - y|$ to get the cost. The same applies the other way around so we also get $\sum_y \gamma(x, y) = a(x)$. Therefore, we end up with $\sum_{x,y} \gamma(x, y)|x-y| = \mathbb{E}_{x,y\sim\gamma}[|x-y|]$. Out of all $\gamma \in \Pi(a, b)$ the one representing the lowest cost (infimum) is the EMD.

2.4 GANs

The Generative Adversarial Networks (GAN) was introduced in 2014 by Goodfellow et al. [19]. A GAN consists of two major components. Firstly, a neural network called the generator (G). The goal of G is to generate samples close to the target domain while using random noise as input in order to create diversity. Secondly, another neural network called the discriminator (D) is used. The D is fed samples from the target domain (real) and samples obtained from the G (fake). The main objective is thus to tell them apart. While both neural networks have their own loss functions, they are correlated. Hence, the idea is that they will feed of each other and progressively improve until the generated samples look indistinguishable from

the target samples making it impossible for the discriminator to tell them apart. This can be formalized as the following two-player minimax game with the value function $V(D, G)$,

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z)))] \tag{2.5}$$

where p_{data} - distribution of real data x and p_z - distribution of random noise z.
There are several problems connected with GANs. One of the most prominent issue is *mode collapse*, a situation where the generator learns to produce only a limited range of samples [20].

2.4.1 Wasserstein GANs

The original GANs training process implicitly tries to minimize JSd. Wasserstein GAN (WGAN) introduces a modification that instead minimizes EMD (also called Wasserstein distance) [21].
Unfortunately, it is not feasible to calculate inf in Eq.2.4 as it requires every possible joint distribution between P_{real} and P_{fake} to be explored. Thus, the formula was altered based on the Kantorovich-Rubinstein duality [22] to:

$$W(P_{real}, P_{fake}) = \frac{1}{K} \sup_{|f|_L \leq K} \mathbb{E}_{x \sim P_{real}}[f(x)] - \mathbb{E}_{x \sim P_{fake}}[f(x)]. \tag{2.6}$$

Here f must satisfy $|f|_L \leq K$ to be K-Lipschitz continuous. More specifically, K-Lipschitz continuous is a real-valued function $f : \mathbb{R} \to \mathbb{R}$ where there exists a real constant $K \geq 0$ such that for all $x_1, x_2 \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{2.7}$$

Intuitively, this ensures that the slope of f is confined by K. So, if we have a set of K-Lipschitz continuous functions $\{f_w\}_{w \in W}$ parameterized by w and where W are the set of all possible weights, then we get:

$$W(P_{real}, P_{fake}) = \max_{w \in W} \mathbb{E}_{x \sim P_{real}}[f_w(x)] - \mathbb{E}_{z \sim p_z(z)}[f_w(G_\theta(z))] \tag{2.8}$$

Unlike original GANs, there is no longer a discriminator telling real and fake samples apart. It can instead be seen as a critic that gets trained with a fixed $g_\theta(z)$ to convergence in order to obtain f_w, which can then be used to compute the Wasserstein distance and by applying backpropagation g_θ can be fine-tuned. Following the definition of Wasserstein distance, the lower it gets the closer the probability distributions get to each other.
However, this derivation only works if the K-Lipschitz of f_w is maintained during training. Originally, Arjovsky et al. proposed weight clipping [21]. It means that after each gradient update, the weights w are constrained to lie within [-x,x] by clipping them to fit.
Gulrajani et al. explored weight clipping further [23] and found the critic to be biased towards very simple approximations far from optimality. As a result they

found the weights of the gradient to veer towards the extremes of the weight clippings outer boundary allowing for the same vanishing/exploding gradient problem as the gradient propagated through the network. The proposed solution introduces an additional gradient penalty (GP) to the loss function calculated by Eq.2.9. Thus, this version of WGAN with improved training procedure is called WGAN-GP.

$$\lambda \mathbb{E}_{x \sim P_x}[(\|\nabla_x D(x)\|_2 - 1)^2] \tag{2.9}$$

2.4.2 Recurrent GANs

While the original GANs consist of multi-layer perceptrons, the same idea with a confrontation between generator and discriminator can be applied to different types of networks including RNNs. Recurrent GANs for continuous sequential data was proposed by Mogren and showed promising results in music generation [24]. As in the original GANs, the generator is fed with a random input z. For Recurrent GANs z is passed for each recurrent unit. The discriminator is fed with real and generated sequences and each recurrent unit outputs a probability whether a sequence is real or fake.

2.4.3 Conditional GANs

In generic GANs it is impossible to control the type of samples that are generated. For instance, if we assume that there is a generic GAN trained on the MNIST(handwritten digit database) [25], then it would not be possible to generate a specific digit. Conditional GANs is a very straightforward and logical extension of generic GANs that address this issue.
The modifications from the original GANs are as follows: conditional labels are passed to the generator and discriminator with random input z and generated/real samples respectively [26]. Then the equation 2.5 evolves to:

$$\min_G \max_D (G, D) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x, c)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z, c), c))] \tag{2.10}$$

Thus, the discriminator learns to distinguish fake and real samples with respect to their labels. In case of the digit generation example, assume there is a generated digit '9' that looks indistinguishable from a real '9'. After training, the discriminator's probability for this sample would be different based on the label: indistinguishable with label '9', but low probability being real for any other label. The generator also adjusts to these settings and starts to generate samples based on labels.
Note that just like generic GANs, conditional GANs can be based on different types of networks including RNNs.

2.5 Clustering

Clustering is an unsupervised method for arranging data into groups with similar properties, so called clusters. However, the definition of 'similar' properties varies

from application to application. Thus, it is not easy to define. For example, in low dimensional space the Euclidean distance can be a metric for the similarity between points.

2.5.1 K-means

One basic and straightforward clustering algorithm is K-Means. As an input it takes N observations $\{x_1, x_2, ..., x_n\}$ and K - a predefined number of clusters. First, K centers (or centroids) are initialized. Based on the initialization method, centers can be random points in a defined space or randomly picked observations. After centroids initialization, each observation is linked to the closest centroid by the defined distance metric, then all centroids are recomputed to be the mean of each cluster (each cluster is defined by all observations linked to a specific centroid). This process is iterative until some end criteria, e.g. centroids are no longer updated. The output of the algorithm is K centers $\{\mu_1, \mu_2, ..., \mu_k\}$, where each μ_i is the mean of the i^{th} cluster [27].
K-means has several disadvantages. For instance, estimating the number of clusters can be a problem in and of itself. In addition, K-means assumes a spherical distribution of the observations, that is not always the case.

2.5.2 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is, as the name suggests, a density based clustering algorithm. This means that it creates clusters where the density of observations is high. To determine density, DBSCAN requires two parameters: ϵ-radius from a considered point p and $minPoints$ - the minimal number of observations required in a ϵ-neighborhood to form a cluster. The algorithm starts by visiting an arbitrary observation: if there exist more than $minPoints$ observations in the ϵ-neighborhood, the cluster is formed. Then, all previously clustered observations within the ϵ-neighborhood are added to the cluster as well. This process continues until the remaining observations are insufficient to expand the current cluster. The same procedure is then done for unvisited observations, until all points are assigned to clusters or marked as noise [28].
Since DBSCAN is based only on the density of observations, it is a good fit for the clusters with different shapes and sizes. However, it is not the best fit for the clusters with different densities.

2.5.3 Dimensionality reduction

The aforementioned clustering algorithms are based on a distance metric, for example, the commonly used Euclidean distance. However, as the dimensionality of data increases, the distances between observations become less and less meaningful due to the so called curse of dimensionality [29]. Thus, the performance of the clustering algorithms also decreases in high dimensional space. To address this issue there are several techniques to map observations from higher to lower dimensions called Dimensionality Reduction [30]. In this work, we consider Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). Note

that the aforementioned autoencoders are also sometimes used for dimensionality reduction.

2.5.3.1 Principal Component Analysis

PCA is a linear technique to map data to lower dimensions by maximizing the variance in lower dimensions. Let X be $m*n$ matrix, where m is number of samples in the dataset and n - number of features. Then, by applying PCA, X will be decomposed as follows [31]:

$$X = \bar{X} + TP' + E \tag{2.11}$$

Here, $\bar{X}$ is a mean vector, P' is a projection matrix to a A-dimensional space ($1 \leq A \leq n$) and T can be considered as coordinates in the new space. Matrix E contains deviations between original data and projections. Finding P' can be performed with Singular Value Decomposition (SVD) for $X - \bar{X}$.
SVD decomposes matrix M_{m*n} into three matrices U_{m*m}, Σ_{m*n} and V_{n*n}, such that [32]:

$$M = U\Sigma V \tag{2.12}$$

Note that Σ is a singular-value diagonal matrix $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$, such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$.

2.5.3.2 t-Distributed Stochastic Neighbor Embedding

The t-Distributed Stochastic Neighbor Embedding (t-SNE) is a method to project high-dimensional data onto 2d or 3d space. While PCA tries to keep as many meaningful properties of data as possible within a lesser amount of features, t-SNE uses another approach. With t-SNE, the mapping is based on probabilistic distances: objects that are close in the original dimension are mapped to nearby points in 2d or 3d space with high probability [33]. This is done by the following steps: firstly, the probability distribution over pairs of high-dimensional observations is constructed as

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{2.13}$$

where σ_i is the variance of the Gaussian that is centered on datapoint x_i. Secondly, the probability distribution over the points in the lower-dimensions is computed (Eq. 2.14) minimizing the Kullback-Leibler divergence (Eq. 2.2) between p and q.

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_k||^2)} \tag{2.14}$$

2.6 Evaluation

A big issue with GANs is to find evaluation metrics to determine if the results are satisfactory [34]. Common evaluation metrics are the Kullback-Leibler divergence

[16] or the improved Jensen-Shannon divergence [17]. However, both of these measure how the probability distribution of two datasets differ from one another. Thus, it is not a metric that can be used to definitively conclude that two datasets are similar. For instance, a dataset consisting of generated images could on average be identical to a dataset consisting of real images while being constructed of pixels that do not correlate very well with each other, rendering images that look like garbage. When dealing with time-series this becomes even more apparent since in addition to the aforementioned disadvantages, the time component is not taken into consideration. Therefore, instead of comparing whole sets against each other it seems like a better idea to do pairwise comparisons between time series instead. The euclidean distance is one metric that can achieve this, however, as it does one-to-one comparisons of the points that make up the time series, it can not compare time series of different lengths.

2.6.1 Dynamic time warping

Dynamic Time Warping (DTW) is an algorithm that finds an optimal alignment between two time-series by using a set of constraints [35]. If A is an alignment $A = a_k = (i_k, j_k)$, where i_k and j_k are indexes of the first and second time series at k alignment step, then constraints are formulated as follows[36]:

- $i_{k-1} \leq i_k$ and $j_{k-1} \leq j_k$ - monotonicity ensures any segment is aligned only once,
- $i_k - i_{k-1} \leq 1$ and $j_k - j_{k-1} \leq 1$ - continuity ensures all points are included in alignment,
- $i_1 = 1, i_k = n$ and $j_1 = 1, j_k = m$, where n and m are lengths of 1st and 2nd time-series - boundary conditions ensure neither of time series are partially considered,
- $i_k - j_k \leq r$ with $r > 0$ - warping window ensures that no segments longer than r are skipped in the alignment.

An example of such an alignment can be found in Fig. 2.5. However, DTW computes in quadratic time and it is thus only realistically viable with shorter sequences and/or smaller data sets. This limitation can be overcome by using FastDTW proposed by Salvador et al. [37]. This is a powerful approximation algorithm that sheds the time complexity down to linear time.

2.6.2 The Hungarian Algorithm

The Hungarian algorithm solves the assignment problem between two datasets. Consider a square matrix C, where $C_{i,j}$ is the cost of linking points i and j. The Hungarian algorithm provides a matching that ensures the minimum total sum of distances

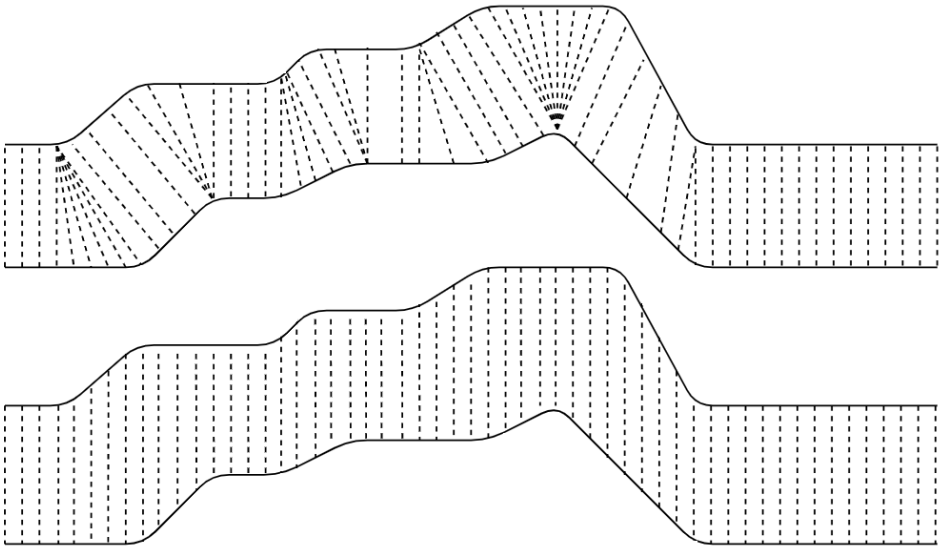


Figure 2.5: Example of sequence aligment with DTW(up) and Euclidean(down).

between paired instances [38]. It can be formulated as follows:

$$\begin{aligned} &\min \sum_{i=1}^{N} \sum_{j=1}^{N} C_{i,j} * X_{i,j} \\ &\text{with} \begin{cases} \sum_{j=1}^{N} X_j = 1, \forall i = \overline{1,N} \\ \sum_{i=1}^{N} X_i = 1, \forall j = \overline{1,N} \end{cases}, \\ &\text{where } X_{i,j} = \begin{cases} 1, \text{if i and j are paired} \\ 0, \text{otherwise} \end{cases} \end{aligned} \tag{2.15}$$

Thus, as one application it can be used to match samples from two sets pairwise (if the distance metric is defined). In our case we are using normalized sum of distances (between paired samples obtained by DTW) as the similarity metric for sets.

3
Methods

This chapter describes three key parts of our work. First, the data and its extraction. Secondly, the generative models:

- An Autoencoder combined with GANs for latent space representation (AE-GAN),
- Recurrent Conditional GANs (RC-GAN).

Finally, our approach to evaluate the developed models.

3.1 Data

As was mentioned in the section 1.2, we are only interested in the relative latitude and longitude position of the tracked vehicles in this thesis. There are a couple of reasons for this where the main one is that these two parameters provide the most essential piece of information for what constitutes a scenario. In addition, other important features such as velocity and acceleration could be derived from them. Moreover, data visualisation is simple with two dimensions.

3.1.1 Pre-processing

The objects' trajectories were extracted from the raw-data (sensor measurements). These trajectories vary in length from 1 second up to 1 hour and have a sampling rate of 10 Hz. The length depends on how long the object was tracked by the ego-vehicle in the field of view (FoV). Thus, it is plausible that the same vehicle gets tracked more than once if it was to get obstructed and then come back into FoV. However, as we are only interested in the trajectories, it does not matter which vehicle produced them. The abstract scheme of this process is presented in Fig. 3.1.

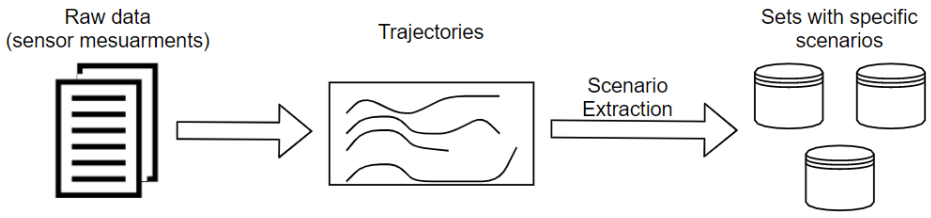


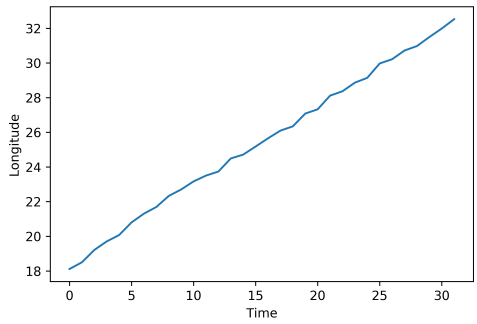
Figure 3.1: Scheme of full extraction process, starting from raw data to the trajectory datasets with specific scenario

3.1.2 Cut-in extraction

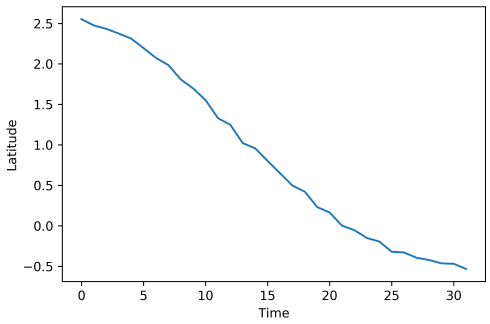
The specific scenario which we have focused on for generation is cut-in. This scenario was chosen as it is of high interest from a safety verification perspective and strikes a good balance between simplicity and complexity. It would make little sense to attempt a very complex scenario (such as double lane change) as not much could be said about the limitations of GANs for generation if it did not succeed. Moreover, it is challenging to create a dataset with the required amount of samples from those complex scenarios as they happen rarely. Cut-ins, on the other hand, while occurring less often rather than drive-bys, still occurs often enough to create a robust dataset.

We have noticed that there are many different definitions of what constitutes a cut-in. We have defined it as a vehicle that approaches the ego-vehicle from the left lane and then overtakes the ego-vehicle by switching to its lane. More specifically, our definition of a cut-in also requires the vehicle to stay in front of the ego-vehicle for at least 2 seconds.

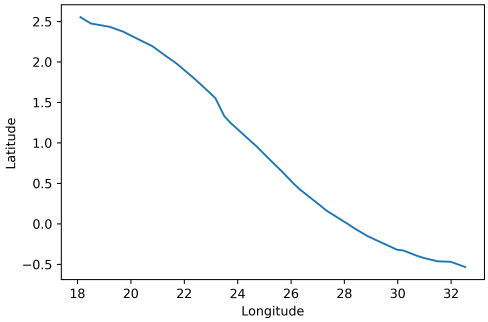
Our extraction approach is explicit rule-based with respect to the provided definition. As was mentioned in the introduction, clustering is also an option and seems to have several benefits over the explicit rule-based approach. However, as we are only dealing with two parameters and a scenario which can be defined easily, we deemed explicit rules to be sufficient.



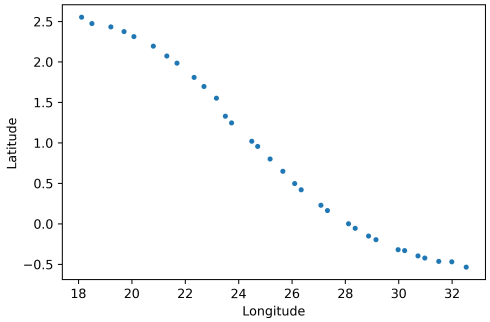
(a) Longitudinal distance with respect to time



(b) Lateral distance with respect to time



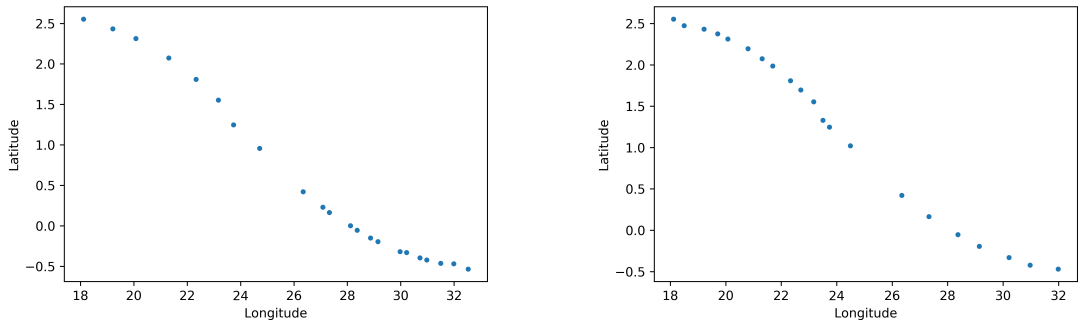
(c) Lateral distance with respect to longitudinal distance



(d) Lateral distance with respect to longitudinal distance

Figure 3.2: This figure presents the extracted cut-in in different forms.

An example of an extracted cut-in is presented in Fig. 3.2. Note that plotting trajectories as a line as shown in Fig. 3.2(c) has the disadvantage of 'eliminating' the time component. This issue is more pronounced in Fig. 3.3 where two different trajectories are presented. While plotting them as a line can make certain trajectories look more similar than they actually are, we find it necessary as it becomes extremely cluttered otherwise when multiple trajectories overlap.



(a) This trajectory is faster in the beginning

(b) This trajectory is faster in the end

Figure 3.3: Similar but different trajectories, point plots visualise the difference. Those trajectories plotted as lines are hard to distinguish.

To provide a better understanding of the extracted data, Fig. 3.4 shows 100 randomly picked trajectories extracted with the explicit rule-based approach. It is clear that the distribution is not even as there are more samples in the 20-60 meters longitudinal region while only a few are seen past 100 meters. Another observation that can be made is that the majority of trajectories have a trend to increase in longitudinal distance over time. However, there are several samples where longitudinal distance decreases instead. This can be interpreted as cut-ins where the tracked vehicle accelerates/decelerates with relation to the ego-vehicle. Apart from varying in aggressiveness, they also vary in length. Trajectories from 3 to 7 seconds are considered in this work as the vast majority of the data is within this region. Given the sampling rate of 10Hz that correlates to 30-70 time steps.

3.2 Variable Length

One of the main issues in training is to handle the variable length input/output. One approach is to train the model with padding[39]. To apply padding, a pad token has to be defined. For instance, in natural language processing (NLP) it is common to use word embedding and use zero vectors as a pad token[40]. Unfortunately, it is not a trivial task to define a pad token in case of real coordinates as any pair of real numbers is a realistic point in space. A possible solution is to pad sequences with the last point, however, it does not seem like a feasible approach in our case due to a high variation in length: the shortest sequence after padding would contain more than 57% 'padding' points.

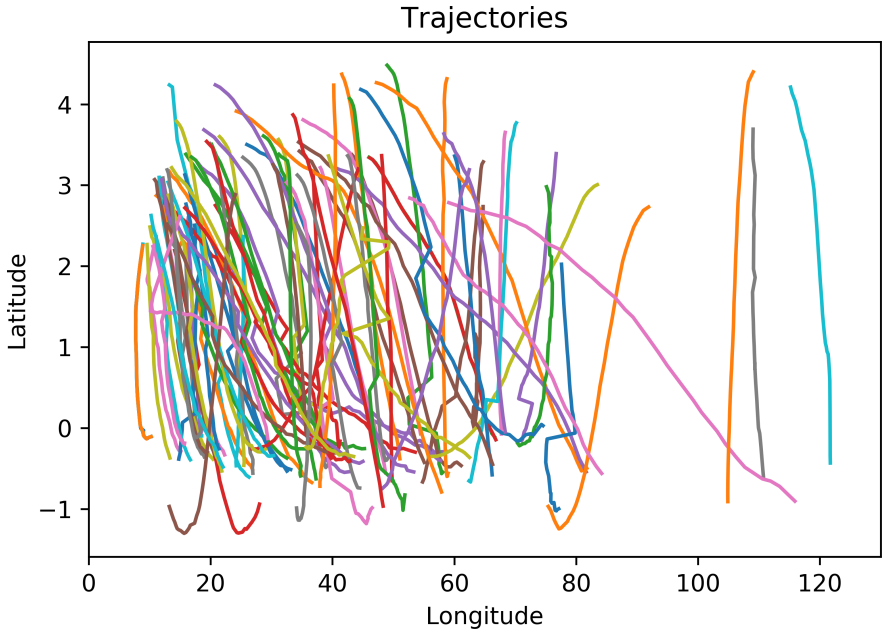


Figure 3.4: This figure presents 100 randomly picked extracted trajectories.

Moreover, post-processing to remove the 'padding' points afterwards would be necessary. For example, if the last n points of a generated sample are the same they should be assumed to be padding points and removed. Unfortunately, this is a problem in itself as the definition of 'same' is tricky since some noise is always added in the generation. In addition, this problem would increase in complexity if more features besides lateral and longitudinal position were to be added.
One potential solution could be to feed the sequences to the model one by one. However, this will greatly impact the performance, so we choose to gather sequences with the same lengths into batches. This is done in the following way: each trajectory is stored in a dictionary with its length as a key. Once the dictionary with all trajectories is created, it is split into batches. We do not state that it is the most effective solution, but this process is done once before training a model so it does not affect the performance noticeably. The next steps depend on the generative model's architecture.

3.3 Autoencoder with GANs

Our first approach is similar to the one proposed in [41] and consists of these steps:

- Train the autoencoder for the trajectories, obtain an encoder and a decoder.
- Encode all the trajectories with the encoder.
- Crete a tool to estimate the length of a trajectory based on its latent space representation.
- Train GANs for the latent space representation.
- Decode generated latent space representations with the decoder using the estimated length.

3.3.1 Autoencoder

The architecture of the autoencoder used in this thesis is presented in Fig. 3.5. It consists of an encoder and a decoder that are both RNNs based on LSTM units. Each LSTM-Cell could be anywhere from one to multiple layers deep (also called stacked-LSTM). The input to a LSTM-Cell in the encoder is $p_1..p_n$ (where p_i consists of lateral and longitudinal position at time step i) paired with the hidden state from the previous cell. For the first cell the hidden vector is initialized to 0. For the decoder the first hidden vector is the output of the last cell from the encoder and the first input is (0,0) which is described as SoS (Start of Sequence) in Fig. 3.5.
During training, for each consecutive cell in the decoder the input is taken from the original sequence using teacher forcing. The output of the decoder is the sequence $\hat{p_1}, \hat{p_2}...\hat{p_n}$. This sequence is compared with the original one to calculate the loss using mean squared error (MSE) and the weights are updated through backpropagation.

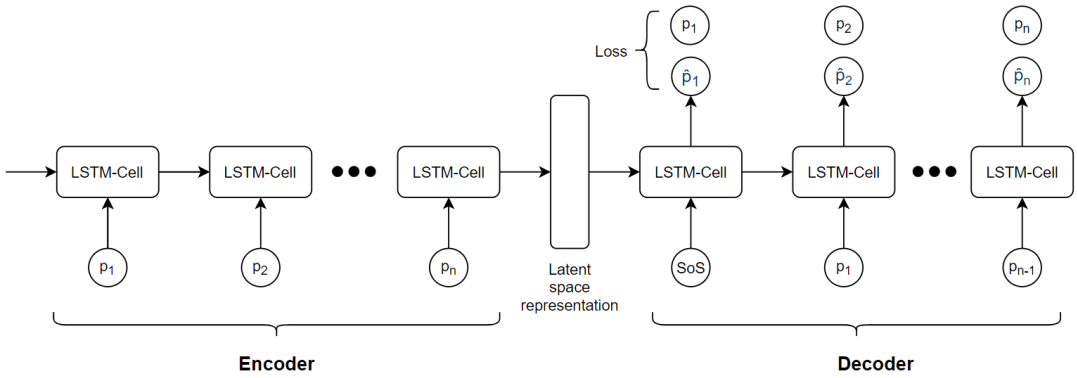


Figure 3.5: Schematic structure of recurrent autoencoder trained with teacher forcing.

Once the autoencoder is trained, the encoder can compress a trajectory to the latent space representation from which it can later be reconstructed using the decoder. Note that latent space representation is of fixed size, thus, trajectories of any length are represented within the same dimensions.

3.3.2 Length Estimator

The decoder is used to obtain sequences from the latent vector representation. However, it is essential to know the length for each encoded trajectory in order to properly decode it from latent space. We did not opt to use any type of padding as we assume that it is possible to estimate the length of a sequence based on its latent space representation by training a separate feed-forward NN. When the sequences are encoded to latent space, their lengths are stored as well. Thus, two sets are created: X - set of latent space representations and Y - set of actual lengths for all encoded sequences. With these sets the task of length estimation from latent space can be considered as a supervised regression task which can be solved using the aforementioned feed-forward architecture. The full reconstruction process of trajectories from latent space representation is presented in Fig. 3.6.

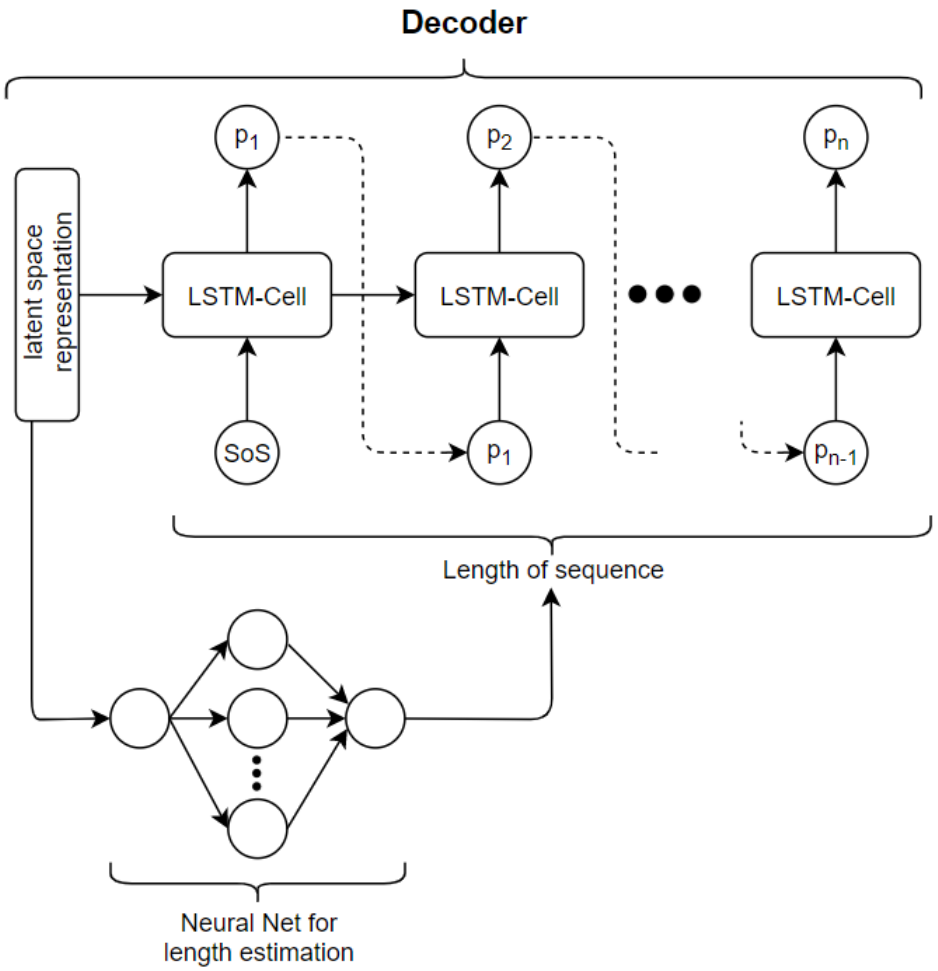


Figure 3.6: Structure of decoder combined with length estimator to decode trajectory from latent space representation

3.3.3 GANs for latent vector representation

Once the autoencoder is trained, the task of generating sequences starts by generating new latent space representations from the existing ones. Even though it seems reasonable to implement both the generator and the discriminator as standard fully-connected NNs, the authors of the original paper [41] use the ResNet architecture [42] to mitigate the problems related to gradient instability. With that said, we decided to try both approaches and see for ourselves which one works the best. For the training procedure we also experiment with both approaches: original GANs and WGAN-GP

3.3.4 Clustering

Clustering is not the main focus of this thesis, however, clustering time-series is an interesting and complex task as the length of sequences consists of different lengths and the distance between time-series is hard to define. There are different approaches available such as clustering based on DTW distance, clustering with hidden markov models (HMM) or a mixture of hidden markov models (MHMM)[43, 44]. However, there are no definitive solution. So, since we already have a fixed-size vector represented in latent space obtained from the autoencoder, we figured we could pursue this a bit and see if we could obtain better results. It should be noted that a Variational AE may be a better fit, since it also regulates the distribution in latent space [45]. However, this is out of scope for this work.
To check the assumption we use the following approach. First, three different types of scenarios are extracted with explicit rules: cut-in as well as left- and right drive-by. Since drive-by occurs much more frequently, the cut-in set contains less trajectories compared to the drive-by set. Thus, we use oversampling for the cut-in set and undersampling for drive-by set. Secondly, the autoencoder is trained and all trajectories gets encoded. The obtained latent space representation is a high dimensional vector, so it can be challenging to cluster it directly with distance based algorithms such as K-means or DBSCAN. Thus, we apply dimensionality reduction techniques in the form of Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE). Finally, we analyze the outcome by clustering algorithms and plots (where appropriate).

3.4 Recurrent Conditional GANs

Our second approach consists of Recurrent GANs (RecGans) as they have already shown promising results for generating time-series in several applications such as music generation [24], real-valued medical data [46] and sensor error modelling [47]. We consider the following architecture: both the generator and discriminator are RNNs based on LSTM cells. For the discriminator we choose a bidirectional architecture as we assume this improves the performance of the discriminator, but at the same time does not make it too complex to outperform the generator and ruin the training process. For each time step i a LSTM-Cell in the generator takes random input and the hidden vector from the previous cell (for the first cell the previous

hidden vector is initialized to 0) in order to generate p_i. The sequence $p_1 ... p_n$ forms the final trajectory.
RecGANs can also be conditional. With Recurrent Conditional GANs (RC-GANs), a condition is passed as input into each cell for both the discriminator and generator. As shown by Hyland et al. the condition can simply be concatenated to the input [46]. We adapt this architecture by using the length of time-series as a condition. The architecture can be seen in Fig. 3.7.

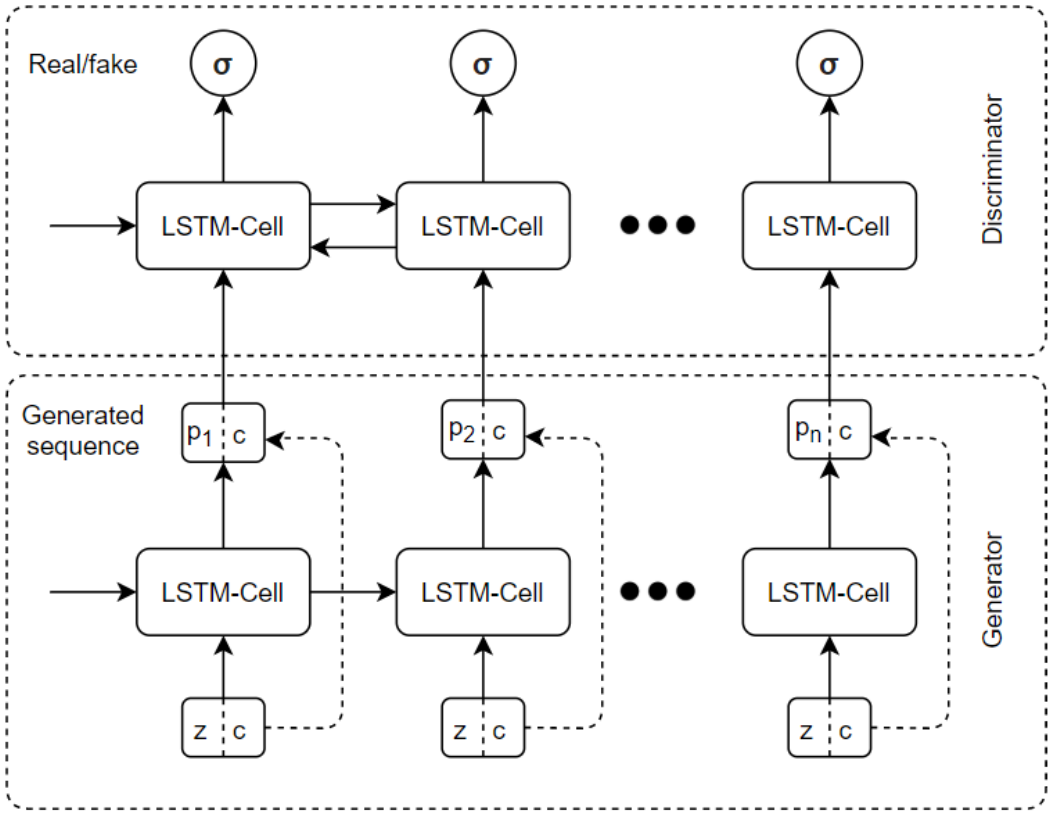


Figure 3.7: Schematic structure of Recurrent Conditional GANs

3.5 Evaluation of generative models

One important and challenging task is to choose proper metrics to evaluate the quality of the generated trajectories [34]. We find it reasonable to evaluate the results via visualization initially to see if it is going in the right direction. However, as the results improve it becomes harder to tell exactly how good they are. Thus, it is important to consider quantitative evaluation metrics as well such that one can objectively quantify the similarity of the generated trajectories with the original ones. This is not trivial though as many different approaches for evaluating GANs have been proposed over the years and none have been objectively agreed upon as the definitive one. Borji [48] provides an overview and summary of many of them. Basically, the key takeaway is to be able to answer the following questions:

- Are samples realistic (e.g. similar to the training data)?

		Set 1			
		tr_1	tr_2	...	tr_n
Set 2	tr'_1	$dist(tr'_1, tr_1)$	$dist(tr'_1, tr_2)$	...	$dist(tr'_1, tr_n)$
	tr'_2	$dist(tr'_2, tr_1)$	$dist(tr'_2, tr_2)$	...	$dist(tr'_2, tr_n)$
	...	...	...	$\ddots$	...
	tr'_n	$dist(tr'_n, tr_1)$	$dist(tr'_n, tr_2)$	...	$dist(tr'_n, tr_n)$

Table 3.1: Pairwise distances between trajectories from two sets, here tr stands for trajectory.

- Does the model cover all modes in the given set?

A commonly used approach to measure similarities between time series is DTW. Note that even if we consider DTW as a metric, it only measures the similarity between individual samples, not the whole set of trajectories. Thus, to compare sets of trajectories, we start by building a table with pairwise distances between each sample from the two sets as shown in Table 3.1.

To address the aforementioned questions we will consider two directions when analyzing Table 3.1.

3.5.1 Matching + Coverage

In this approach, each sample for the generated set (GS_M) gets matched with the closest sample from the real set(RS_N). The matching criterion is defined by the following formula:

$$matching = \frac{\sum_i^M \min_j \left(dist(GS_i, RS_j)\right)}{M} \tag{3.1}$$

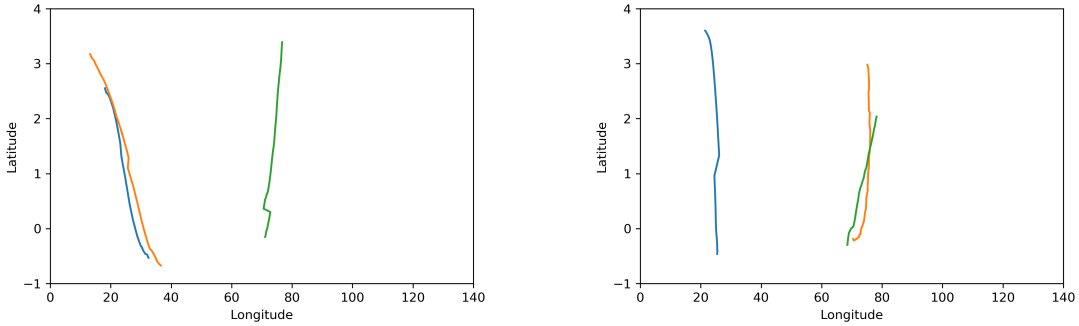
This metric is similar to the Reconstruction Loss used by Xiang et al. [49] and this type of metric is also considered as the precision by some [34]. Even if reasonable results are achieved, it does not necessarily indicate that the model performs well since many generated samples can be linked to the same real sample. If this occurs it means that the coverage of the model is low. Thus, we also measure the coverage as the amount of samples used in matching divided by the total number of samples, or formally:

$$coverage = \frac{|argmin_j(dist(GS_i, RS_j)), \forall i = \overline{1, M}|}{N} \tag{3.2}$$

However, even the combination of these metrics have disadvantages. For instance, if there are two (or more) similar samples in the real set and many generated samples are matched to one of the real samples then the coverage decreases, but that does not necessarily mean that the model performs poorly. Note that sets we are dealing with are diverse, thus, we consider GS_M and RS_N with $M > N$, in our experiments we used $M = 4 * N$

3.5.2 One-to-one Matching with the Hungarian algorithm

The second approach consists of the Hungarian algorithm which is a one-to-one matching algorithm. Applying it to the Table 3.1 gives the matching for each sample from the first set to the second set. This matching ensures that the sum of distances of the paired samples is minimal. One major disadvantage of this approach is the different sample distributions between sets. A simple example of this case is presented in Fig. 3.8. Both sets contain real trajectories, however, the described metric would not produce the best results due to different number of samples in different regions.



(a) Visualisation of Set 1 **(b)** Visualisation of Set 2

Figure 3.8: Two sets consisting of three real trajectories with different region distributions

We assume that for the bigger sets this problem has less of an impact as the generated data conforms towards the original distribution. To address this issue we analyze the amount each matching contributes as shown in Table 3.2. This table is built with the aforementioned approach for the sets presented in Fig. 3.8. It seems reasonable for us to exclude 1623.34, however, some consensus must be reached as we could otherwise make a model that performs extremely poorly look good by throwing away most of the pairings.

Table 3.2: Pairwise distances between trajectories from two simple sets presented in Fig. 3.8. Highlighted values form the one-to-one matching with the lowest sum.

	tr_1	tr_2	tr_3
tr'_1	101.75	1857.09	**1623.34**
tr'_2	**188.82**	2008.17	1950.79
tr'_3	1863.69	**43.83**	43.10

It is not very clear how to define a threshold for the discardment. To address this issue we spilt the real dataset into two subsets of 100 random samples each after which we apply the Hungarian algorithm to these two subsets and plot the obtained matching distances as shown in Fig. 3.9. This gives us a baseline of what to expect ideally from the generative models.

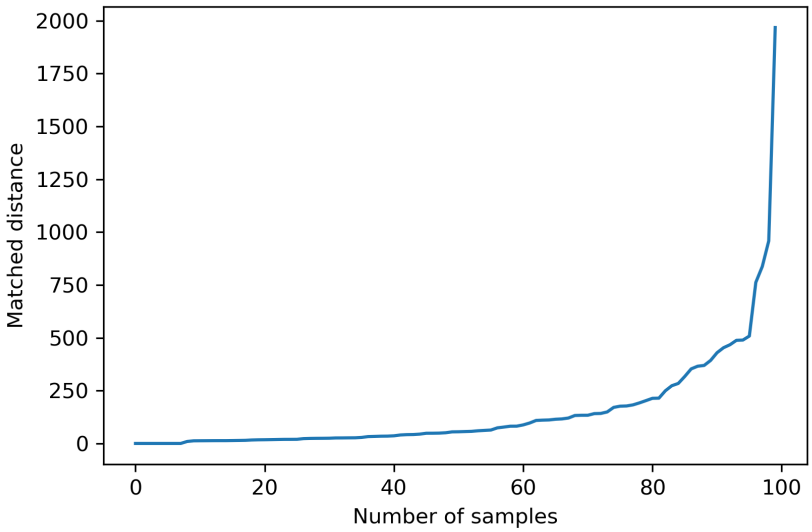


Figure 3.9: This graph presents matched distances for comparing two sets consisting of hundred real trajectories each.

From the graph presented in Fig. 3.9 it is seen that the distance between matched samples slowly grows until about 70%, then it grows faster from 70% to 90%, and then there is a drastic increase from 90% onwards. These last 10% is the distribution difference that we try to describe with Fig. 3.8. Thus, this graph is used for further analysis of the generated sets.

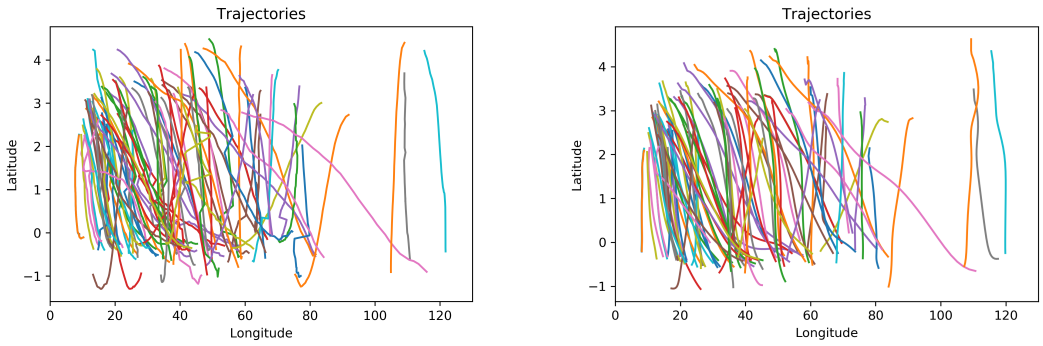
4
Results

This section presents the obtained results. Before evaluating the models with the previously outlined metrics, we examined them with visual inspection. Thus, this section presents plots with generated trajectories and the outcome of the proposed metrics.

4.1 Autoecnoder with GANs

4.1.1 Autoencoder

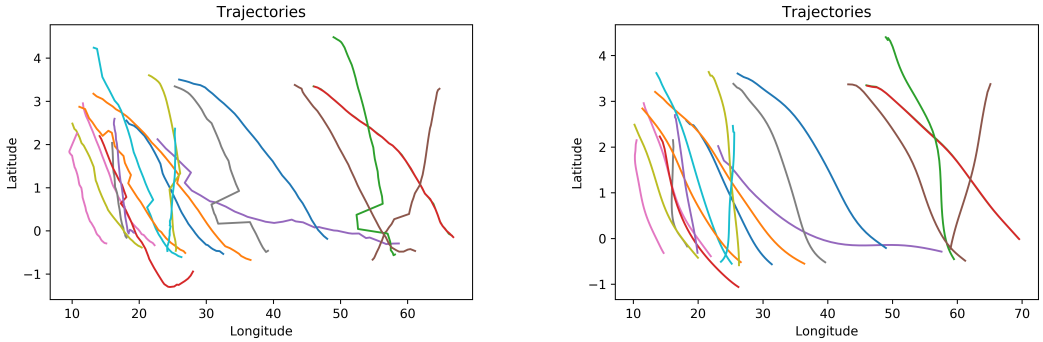
The autoencoder is trained with reconstruction loss, specifically MSE between the original and reconstructed trajectory. Thus, a low score by the loss function entails that our autoencoder performs well. We split the dataset with trajectories into a training and validation set and kept training the autoencoder until the loss for the validation set stabilized and obtained the same loss score as the training set. As a sanity check, we ensured that the autoencoder worked as expected by plotting 100 randomly picked trajectories before and after we encoded them and brought them back as can be seen in Fig. 4.1.



(a) Real Trajectories **(b)** Reconstructed Trajectories

Figure 4.1: 100 real trajectories(a) and their reconstruction by autoencoder(b)

An autoencoder has a characteristic that denoise the input, in case of a recurrent autoencoder, this property manifest itself in smoother trajectories after being processed by the autoencoder as shown in Fig. 4.2.
The final goal of this work was to generate trajectories from 3 to 7 seconds long. However, we started our experiments with shorter trajectories to understand potential pitfalls. We performed two experiments: the first one with trajectories from 3



(a) Real Trajectories **(b)** Reconstruction of real trajectories

Figure 4.2: Visualisation of the denoising property of a recurrent autoencoder

to 5 seconds and the second experiment with 3 to 7 second trajectories. For the first experiment we used a 2 layer LSTM-cell with a hidden state of 32 since increasing it up to 64 does not introduce a great improvement. These settings seemed sufficient and produced satisfactory results. However, for the second experiment with 3 to 7 seconds trajectories, a hidden size of 32 produced worse results from visual inspection. The performance was improved by increasing the hidden size to 64 as seen in Table 4.1. Note that close loss values for trajectories with different lengths does not necessarily mean that the autoencoder's performance are on the same level since the mean is calculated with respect to the different amount of points. Thus, we suggest to compare loss values only for the same sets and increase the depth of the model until no further improvement is introduced instead of comparing different sets.

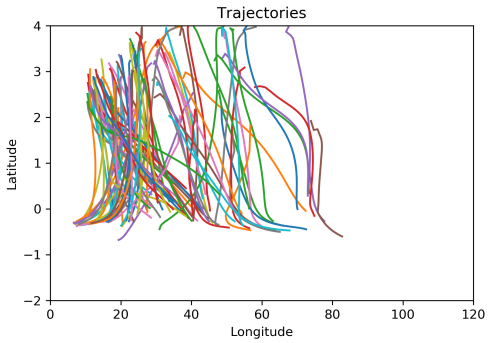
Table 4.1: Comparison between a 32 and 64 sized hidden state for two sets consisting of trajectories between 3 to 5 seconds (I) and 3 to 7 seconds (II)

Size of hidden state	Val.Loss I	Train Loss I	Val.Loss II	Train Loss II
32	0.0633	0.0652	0.0648	0.0637
64	0.0619	0.0620	0.0469	0.0469
128	0.0601	0.0610	0.0451	0.0471

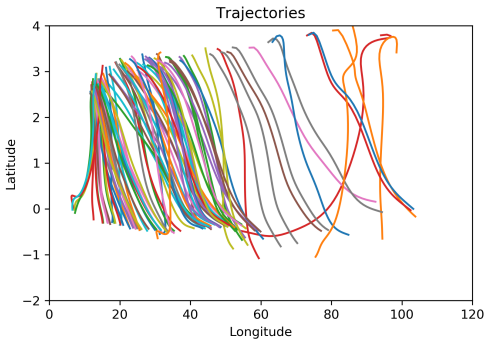
4.1.2 GANs for latent space representation

There is no objective function to evaluate GANs during training for latent space vectors. The adversarial nature of the training process means that the losses of G and D will fluctuate, but apart from a stable state it does not provide any useful information regarding the state of the GAN. Thus, we also plot some generated trajectories during training to get a better understanding of the current state and to detect a potential mode-collapse.
As with the autoencoder, we started our experiments with 3 to 5 seconds trajectories. For this set, we used standard fully-connected feed-forward NNs in both the Generator and Discriminator. A set of 100 trajectories reconstructed from the generated latent space representation is shown in Fig. 4.3(a). Even though some

trajectories do not look realistic, the general trends are the same as in the original dataset.
In the second experiment with 3 to 7 seconds trajectories, the dimension of a generated vector doubles as hidden size increases from 32 to 64. The GANs architecture from the first experiment has not shown any reasonable results, thus, we perform experiments with ResNet architecture for G and D and WGAN-GP. Unfortunately, ResNet architecture did not introduce any great improvement. However, the standard fully-connected NNs produces better results with WGAN-GP, at least from a visual inspection. Examples of generated trajectories are presented in Fig. 4.3(b). Quantitative metrics for comparison are presented in the comparison section.



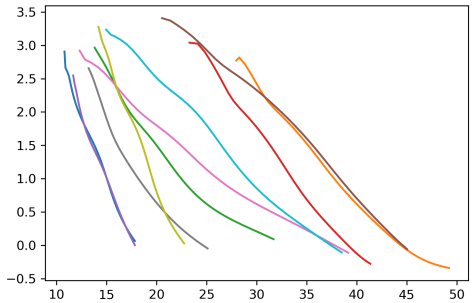
(a) Example of 100 trajectories from 3 to 5 seconds generated with AEGAN



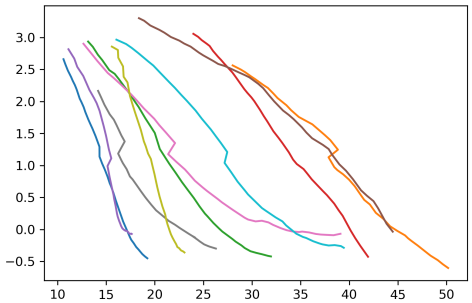
(b) Example of 100 trajectories from 3 to 7 seconds generated with AE-WGAN-GP

Figure 4.3: Trajectories generated with AE-GANs and AE-WGAN-GP

As described in the Methods section, for evaluation, a table with pairwise distances is built (Table 3.1) and with this table it is a trivial task to find the closest samples from two disjoint sets. This can be seen in Fig. 4.4 where the trajectories generated by AE-GAN that matches the real ones to most are shown.



(a) Generated trajectories with AE-GAN.



(b) Real trajectories closest to the trajectories presented in (a).

Figure 4.4: This figure presents the 10 closest samples from the set generated with AE-GAN(a) and real set(b).

4.1.3 Clustering

Clustering is performed on latent space representations of the trajectories obtained from the autoencoder. As said in the Methods section we extracted three types of trajectories, the result of this extraction process is presented in Fig. 4.5. Therefore, we expect the clustering algorithm to either find these three clusters or additional ones as sub-clusters could be found with respect to some hidden property that was not taken into account with our explicit-rules.

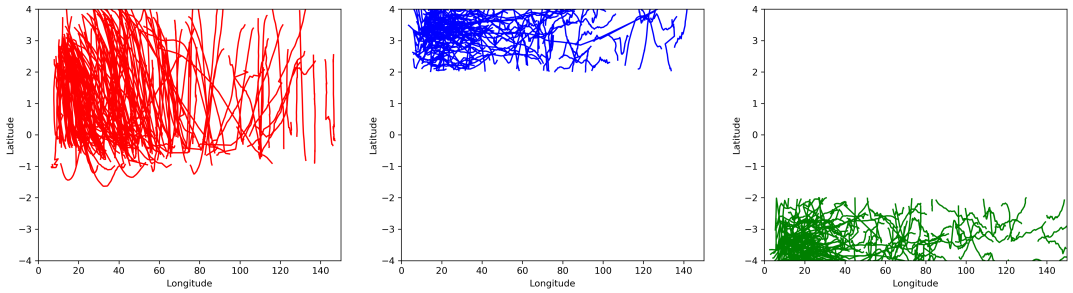
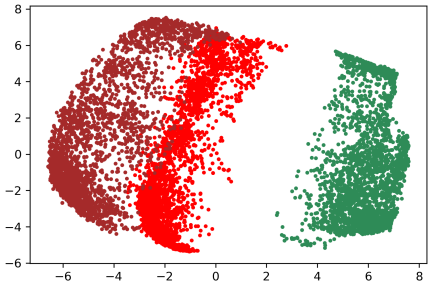
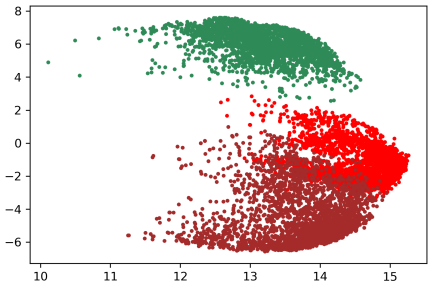


Figure 4.5: Three types of explicit-rule extracted trajectories: cut-ins, left- and right drive-by.

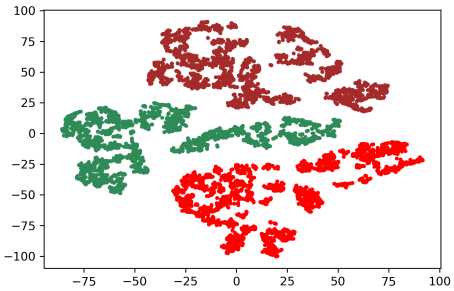
We apply three dimensionality reduction techniques: PCA, SVD and t-SNE. The only difference between SVD and PCA is that PCA is centering the data by mean value. The results are presented in Fig. 4.6. Neither PCA nor SVD transformed data can be clustered easily in 2d (into three clusters), however, PCA produced more promising results.
By applying SVD (either directly or after centering) we obtain a diagonal matrix Σ with singular values and based on them it is possible to calculate a percentage of variance introduced by each component. This information can help to find an optimal number of principal components to capture enough information from the original data to distinguish clusters while at the same time avoiding the curse of dimensionality.
Based on Fig. 4.6 we find dimensionality reduction produced by t-SNE to be the most reasonable to perform clustering. The results of DBSCAN with $\epsilon = 9$ and $minNeighbors = 25$ are presented in Fig. 4.7a, 5 clusters were found. Note that originally, t-SNE was developed for visualisation and can produce some misleading results [50], however there are cases when t-SNE produces satisfactory embedding for clustering[51] as in our case. With that said, we also investigate PCA for clustering as an alternative direction in case t-SNE were to perform worse for more complex tasks (e.g. more scenario types, more features, longer sequences etc.).
We use PCA with 4 principal components (that covers 75% of variance) and then apply clustering. As it is impossible to plot results in 4d, a 2d representation of trajectories obtained from t-SNE can be used to plot the results. While K-means does not produce any feasible results in this 4d space (we assume there is no spherical distribution needed by K-means), reasonable results are achieved when applying DBSCAN instead. This can be seen in Fig. 4.7b, where 4 clusters are found. It is important to note that some points are labeled as noise (corresponding trajectories are potential outliers).



(a) PCA

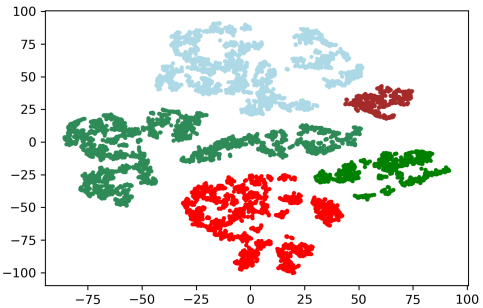


(b) SVD

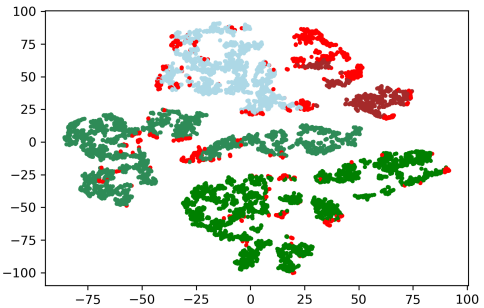


(c) t-SNE

Figure 4.6: This figure presents latent space representation of trajectories with ground-truth labels: green - right drive by, brown - left, red - cut-in



(a) DBSCAN applied to t-SNE with $\epsilon = 9$ and $minNeighbors = 25$.

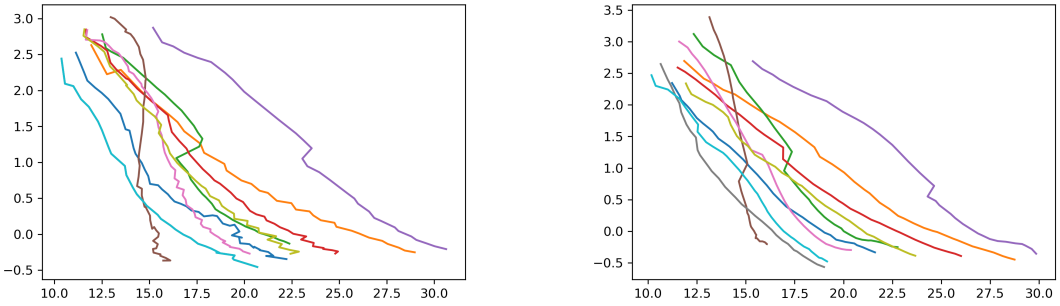


(b) DBSCAN applied to PCA(components=4) with ϵ=0.8 and $minNeighbors = 20$. Red labels mean noise.

Figure 4.7: Results of DBSCAN clustering of 2d obtained with t-SNE(a) and 4d obtained with PCA: points are plotted from t-SNE, but labels from 4d clustering PCA(b)

4.2 Recurrent Conditional GANs

Just like in the case with GANs for latent space representation, there is no objective loss function to evaluate during training. Thus, we examine losses of G and D in the same manner and periodically plot generated samples to get a grasp if the training performs adequately.
As for RC-GAN, we present 10 generated trajectories with minimal distance to the real ones and 100 randomly picked trajectories generated with RC-GANs in Fig. 4.8 and Fig. 4.9 respectively. Clearly, RC-GANs capture the major trends of cut-ins.

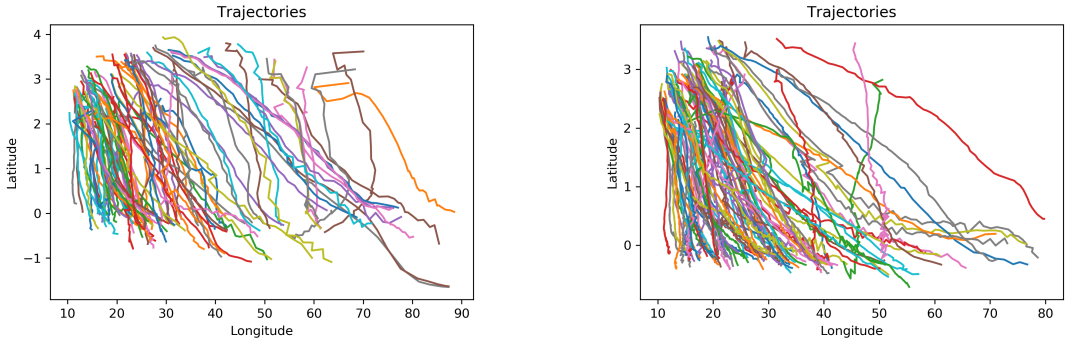


(a) Generated trajectories with RC-GAN.

(b) Real trajectories closest to the trajectories presented in (a).

Figure 4.8: This figure presents the 10 closest samples from the set generated with RC-GAN(a) and real set(b).

The developed RC-GANs is conditional with respect to the length of the trajectory, so, it is possible to generate trajectories with a specified length. This condition works as expected and can be seen in Fig. 4.9: all trajectories end up in region 0 (from a lateral perspective), that means those are complete cut-ins. For example, there are no trajectories that are just truncated halfway after 3 seconds.



(a) Trajectories generated with specified length from 3 to 4 seconds

(b) Trajectories generated with specified length from 6 to 7 seconds

Figure 4.9: Trajectories generated with RC-GANs.

4.3 Comparison

First off, from the visual inspection, both models perform adequately, e.g. capture the trends and distribution of the trajectories: the majority of the generated trajectories start and finish in reasonable areas, and like in the real dataset there are more trajectories generated closer to the ego-vehicle and less further away. Moreover, there are generated trajectories both with acceleration and deceleration but as in the original set the vast majority are cut-ins with acceleration (increasing longitudinal position throughout time). Samples generated with RC-GAN look more noisy compared to samples from AE-GAN which is to be expected.
To examine the proposed metrics: Matching+Coverage and Hungarian, we first apply them to the real set to obtain a baseline. Each experiment is done 5 times and an average is taken as a final outcome. We also present the minimum and maximum values out of those experiments to provide an order of variance. The results are presented in Table 4.2 and Table 4.3 for the first and the second metric respectively. Note that experiments for RC-GAN and AE-WGAN-GP are performed for 3 to 7 seconds trajectories while results for AE-GAN are only for 3 to 5 seconds givin its limitations.
Based on the Matching+Coverage RC-GAN is the best model since it has both the highest coverage and lowest matching among generated sets. The matching metric is even lower than for the real set which may seem suspicious at first glance. However, we explain this by the lower coverage: calculating the average for around 60% of the best matched samples in the real set would produce a lower value.

Table 4.2: This table presents results for Matching and Coverage metric. Min, Max and Average is out of 5 experiments.

Set	Matching			Coverage		
	Min	Max	Average	Min	Max	Average
Real Set (Baseline)	39.78	45.46	43.28	0.85	0.89	0.875
RC-GAN	29.08	31.93	30.37	0.63	0.68	0.66
AE-GAN	48.85	56.13	53.31	0.39	0.45	0.42
AE-WGAN-GP	39.04	52.65	44.70	0.54	0.59	0.56

The results for the second proposed metric that is based on one-to-one matching (Hungarian) are far from ideal - the best result for AE-WGAN-GP is 229.91 which is more than 3 times higher compared to the real set. Yet, this behavior was assumed in the description of this metric in the Method section. To obtain more useful information out of one-to-one matching the distances between matched samples are plotted in Fig. 4.10 (as proposed in the description of this approach).
We can clearly see similar behavior between the generated and real set, however, the scale of these graphs are different and matched distances for generated sets explodes more compared to the real set. We assume this behavior is a combination of two factors: firstly, as described in the method section, the distribution of trajectories are different; secondly, the non-realistic samples produced by our generative models increase the growth even more. From Fig. 4.10 it is clearly seen that none of the

Table 4.3: This table presents results for Hungarian distance. Min, Max and Average is out of 5 experiments.

SET	Hungarian			Hungarian (75%)		
	Min	Max	Average	Min	Max	Average
Real Set (Baseline)	62.01	84.58	75.2	29.84	40.14	36.10
RC-GAN	330.38	500.66	430.33	121.14	151.83	138.56
AE-GAN	330.3	424.59	358.69	121.94	141.74	130.02
AE-WGAN-GP	159.15	284.36	229.91	63.47	102.68	79.32

plots explodes until 150 out of 200 samples, thus, we also compute an average of matched distances for the first 75% of samples and those are shown in Table 4.3.

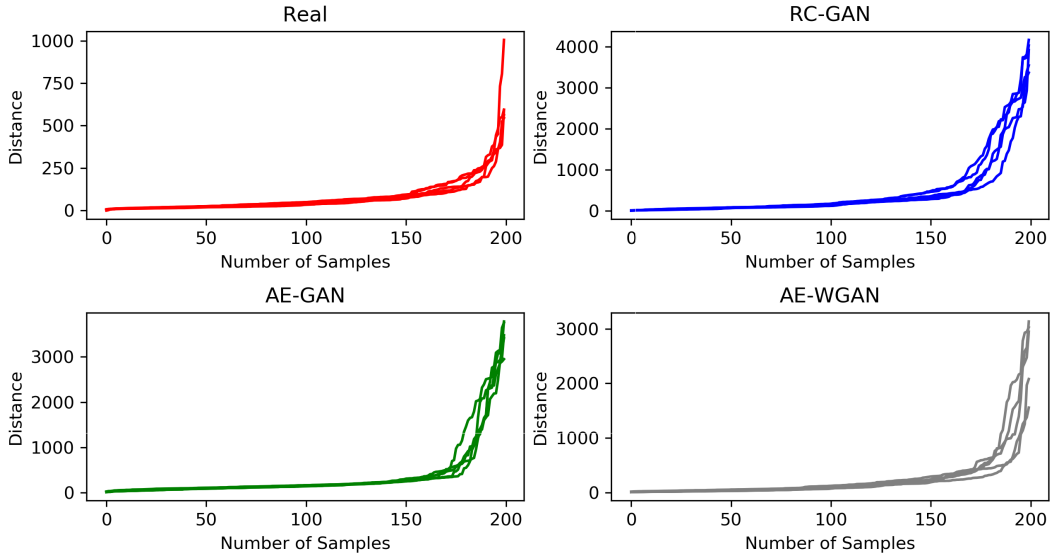


Figure 4.10: Distance between matched samples in one-to-one matching: Red - real, RC-GAN - blue, AEGAN - green, AEWGAN - grey.

5

Discussion and Conclusion

This section presents a discussion about the conducted project and the obtained results. It also contains several directions for future work.

5.1 Discussion

Both proposed approaches share the same property: they generate some samples that are indistinguishable from real ones (based on the proposed metrics), but at the same time some generated samples are far from realistic. There is at least one intuitive reason for this which is that there are minorities and outliers in the original dataset and it would be too ambitious to assume that a generative model could capture it all. Moreover, we assume these outliers make the discriminator less confident about real and generated samples, e.g. it is harder for the discriminator to tell real and generated samples apart when they are outliers, but this is just our intuitive explanation. The suggested usage of the model would be to take only samples that are close to the real ones which we found to be possible with our defined metrics.

In the case of the increasing complexity of the trajectories, such as more features or longer trajectories, AE-GAN makes it easier to identify the interfering factors, if any, due to the 2 steps involved in this approach. For example, in this work, AE-GAN with original GAN produced satisfactory results for trajectories from 3 to 5 seconds, however, to generate trajectories from 3 to 7 seconds we replaced the original GAN with WGAN-GP, since the autoencoder part of the model performed adequately for this set. At the same time, RC-GAN covers different modes of trajectories better as can be seen both from a visual inspection and proposed metrics, so, it is also worth considering as a meaningful model.

While the clustering performs quite well, the results should be taken with a grain of salt. More complex classifications would have to be carried out in order to determine if this is a feasible way to obtain different scenarios. In this regard we only carried out the first step to see if it is possible and while it looks promising, these scenarios are quite basic. Furthermore, the clustered scenarios are very distinctive from one another and did not contain any additional noise as they were obtained already through explicit rules. That may beg the question, then why bother with this clustering approach? As we mentioned earlier explicit rules will only get you so far, for more complex scenarios and higher dimensions, it is no longer feasible. So with that said, it is promising that everything that was labeled did not contain a single miss classification. In addition, the scenarios that were not classified did

differ from the other ones for one reason or another. It might simply be due to poor performance, but those trajectories may also differ from their peers and could be interesting to investigate further. These "outliers" would not be caught with explicit rules.

5.2 Conclusion

The goal of this thesis was to investigate different possibilities for generation of driving scenario trajectories, more specifically, cut-ins. For these purposes two generative approaches were considered: AE-GAN (with AE-WGAN-GP extension) and RC-GAN. Both of them perform adequately. Moreover, both models work with trajectories of different length. It is possible due to the proposed architectures and approach with aggregating trajectories with the same length into batches. Also, another challenge occurred during the project - the evaluation of the proposed models. Thus, two evaluation metrics were proposed that can help to improve those models and/or the development of new models. Proposed models showed satisfactory results based on these metrics but did not achieve the same level as comparison between real sets. From this we conclude two things: proposed metrics are useful and the developed models should be improved upon or other approaches can be considered and compared against them.
The implemented autoencoder also has useful properties of trajectories' embedding. We showed it by clustering three different types of trajectories. Thus, this clustering method can be used in the scenario extraction process.

5.3 Future Work

Both proposed frameworks have shown promising results, however, we believe that there is room for improvement. For example, all hyper-parameters were tuned by a small grid search that can be improved with different techniques, such as Bayesian optimization or Reinforcement Learning.
One of the proposed models (RC-GAN) is conditional, since it was a necessity to handle variable length input, but a combination of an autoencoder with GANs is not conditional. One straightforward direction for improvement could be to modify this architecture to be conditional as well. This should be possible to accomplish since the length of each trajectory in latent space is stored and could be used as a label.
In this work we considered a trajectory as a relative lateral and longitudinal position of the tracked object. However, there are other parameters collected by sensors such as speed, acceleration and many others for both tracked vehicles and the ego-vehicle itself. Thus, creating a generative model with more features is of high interest. During this work, we have seen potential for scalability in the proposed models. Moreover, if correlations between parameters are found they could be used as another metric to determine the quality of the generated scenarios. For example, a relation is found within real trajectories then it should be found in the generated trajectories as well.

The extraction process of different trajectories was performed with explicit rules, however, this is not the best approach due to a variety of reasons. This includes bad scalability for additional features as well as the increasing chances to miss outliers as the scenarios becomes more complex. Thus, clustering based on the autoencoder can be investigated more for the extraction process. As the autoencoder for clustering is trained, the whole trajectory can be processed with a sliding-window to extract parts clustered as scenarios of interest. Moreover, this approach can be compared with MHMM proposed by Martisson et al. [9] used for the same purposes.

Bibliography

[1] J. Kowszun, "Jamitons: Phantom traffic jams." *School Science Review*, vol. 95, no. 350, pp. 53–61, 2013.

[2] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[3] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[4] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[5] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[6] P. Themann, A. Pütz, A. Zlocki, and L. Eckstein, "Database of relevant traffic scenarios as a tool for the development and validation of automated driving," in *Proc. of International symposium on Advanced Vehicle Control AVEC*, vol. 16, 2016, pp. 489–496.

[7] B. Kim, Y. Kashiba, S. Dai, and S. Shiraishi, "Testing autonomous vehicle software in the virtual prototyping environment," *IEEE Embedded Systems Letters*, vol. 9, no. 1, pp. 5–8, 2016.

[8] X. Li, W. Hu, and W. Hu, "A coarse-to-fine strategy for vehicle motion trajectory clustering," in *18th International conference on pattern recognition (ICPR'06)*, vol. 1. IEEE, 2006, pp. 591–594.

[9] J. Martinsson, N. Mohammadiha, and A. Schliep, "Clustering vehicle maneuver trajectories using mixtures of hidden markov models," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3698–3705.

[10] J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic, "Discovering clusters in motion time-series data," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1. IEEE, 2003, pp. I–I.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[12] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[13] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[16] S. Sherman, "Review: Solomon kullback, information theory and statistics," *Bull. Amer. Math. Soc.*, vol. 66, no. 6, p. 472, 11 1960. [Online]. Available: https://projecteuclid.org:443/euclid.bams/1183523751

[17] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, June 2004, pp. 31–.

[18] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[20] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[22] S. T. Rachev *et al.*, "Duality theorems for kantorovich-rubinstein and wasserstein functionals," 1990.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[24] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[27] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[28] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[29] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[30] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, no. 66-71, p. 13, 2009.

[31] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[32] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," in *Linear Algebra*. Springer, 1971, pp. 134–151.

[33] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[34] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in neural information processing systems*, 2018, pp. 700–709.

[35] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, 1994, pp. 359–370.

[36] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[37] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[38] L. Ramshaw and R. E. Tarjan, "On minimum-cost assignments in unbalanced bipartite graphs," *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, 2012.

[39] M. Dwarampudi and N. Reddy, "Effects of padding on lstms and cnns," *arXiv preprint arXiv:1903.07288*, 2019.

[40] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[41] D. Donahue and A. Rumshisky, "Adversarial text generation without reinforcement learning," *arXiv preprint arXiv:1810.06640*, 2018.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[43] J. Hu, B. Ray, and L. Han, "An interweaved hmm/dtw approach to robust time series clustering," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3. IEEE, 2006, pp. 145–148.

[44] T. W. Liao, "Clustering of time series data—a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[45] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[46] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," *arXiv preprint arXiv:1706.02633*, 2017.

[47] H. Arnelid, E. L. Zec, and N. Mohammadiha, "Recurrent conditional generative adversarial networks for autonomous driving sensor modelling," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 1613–1618.

[48] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[49] S. Xiang and H. Li, "On the effects of batch and weight normalization in generative adversarial networks," *arXiv preprint arXiv:1704.03971*, 2017.

[50] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/misread-tsne

[51] G. C. Linderman and S. Steinerberger, "Clustering with t-sne, provably," *SIAM Journal on Mathematics of Data Science*, vol. 1, no. 2, pp. 313–332, 2019.