Probabilistic Parametric Curves for Sequence Modeling

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc.

Ronny Hug

aus Freudenstadt

Tag der mündlichen Prüfung:	16.12.2021
Erster Gutachter:	Prof. Dr.-Ing. Jürgen Beyerer
Zweiter Gutachter:	Prof. Dr.-Ing. Marco Huber

Abstract

Representations of sequential data are commonly based on the assumption that observed sequences are realizations of an unknown underlying stochastic process. Usually, the determination of such a representation is construed as a learning problem and yields a sequence model. In this context, the model must be able to capture the multi-modal nature of the data, without blurring between single modes. For modeling the underlying stochastic process, commonly used neural network-based approaches either learn an implicit representation by using stochastic inputs or units, or learn to parameterize a probability distribution. As such, these models usually incorporate Monte Carlo or other approximation techniques in order to perform parameter estimation and probabilistic inference. This even holds true for regression-based approaches based on Mixture Density Networks, which still require Monte Carlo simulation for performing multi-modal inference. Thus, a research gap in fully regression-based approaches for parameter estimation and probabilistic inference emerges.

Towards this end, this thesis proposes a probabilistic extension to Bézier curves ($\mathcal{N}$-Curves), as a basis for effectively modeling continuous-time stochastic processes with a bounded index set. The proposed stochastic process model is denoted as the $\mathcal{N}$-Curve model and is based on Mixture Density Networks (MDN) and Bézier curves with Gaussian random variables as control points. Taking an MDN-based approach is in line with recent attempts to address the problem of quantifying uncertainty as a regression problem and yields a generic model, which is generally applicable as a basic model for probabilistic sequence modeling. Key advantages of the model include the ability of generating smooth multi-mode predictions in a single inference step, which avoids the need for Monte Carlo simulation. Further, being based on Bézier curves, the model can, in theory, be scaled up to high dimensional sequence data by embedding the control points in a high dimensional space. In order to approach theoretical limitations imposed by the restriction to a bounded index set, a conceptual extension to the $\mathcal{N}$-Curve model, capable of modeling infinite stochastic processes, is presented. Essential properties of the proposed approach and its extension are illustrated by several toy examples considering a sequence synthesis task.

With the original $\mathcal{N}$-Curve model being sufficient for most real-world applications, a thorough evaluation is conducted on different multi-step sequence prediction tasks for evaluating the capabilities of the model applied to real-world data. First, the model is evaluated against commonly used generic probabilistic sequence models on a human trajectory prediction task, proving the capabilities of the $\mathcal{N}$-Curve model, as the model outperforms other the models in this comparison. A qualitative evaluation investigates the behavior of the model in a prediction context. Further, difficulties in assessing the performance of probabilistic sequence models in a multi-modal setting are discussed. In addition, the model is applied to a human motion prediction task, assessing the claimed scalability of the model to higher-dimensional data. In this task, the model outperforms commonly used simple and neural network-based baselines and performs on par with different state-of-the-art models on several occasions, proving its capabilities in this higher-dimensional example. Further, difficulties in covariance estimation and the smoothing property of the $\mathcal{N}$-Curve model are discussed.

Kurzfassung

Repräsentationen sequenzieller Daten basieren in der Regel auf der Annahme, dass beobachtete Sequenzen Realisierungen eines unbekannten zugrundeliegenden stochastischen Prozesses sind. Die Bestimmung einer solchen Repräsentation wird üblicherweise als Lernproblem ausgelegt und ergibt ein Sequenzmodell. Das Modell muss in diesem Zusammenhang in der Lage sein, die multimodale Natur der Daten zu erfassen, ohne einzelne Modi zu vermischen. Zur Modellierung eines zugrundeliegenden stochastischen Prozesses lernen häufig verwendete, auf neuronalen Netzen basierende Ansätze entweder eine Wahrscheinlichkeitsverteilung zu parametrisieren oder eine implizite Repräsentation unter Verwendung stochastischer Eingaben oder Neuronen. Dabei integrieren diese Modelle in der Regel Monte Carlo Verfahren oder andere Näherungslösungen, um die Parameterschätzung und probabilistische Inferenz zu ermöglichen. Dies gilt sogar für regressionsbasierte Ansätze basierend auf Mixture Density Netzwerken, welche ebenso Monte Carlo Simulationen zur multi-modalen Inferenz benötigen. Daraus ergibt sich eine Forschungslücke für vollständig regressionsbasierte Ansätze zur Parameterschätzung und probabilistischen Inferenz.

Infolgedessen stellt die vorliegende Arbeit eine probabilistische Erweiterung für Bézierkurven ($\mathcal{N}$-Kurven) als Basis für die Modellierung zeitkontinuierlicher stochastischer Prozesse mit beschränkter Indexmenge vor. Das vorgestellte Modell, bezeichnet als $\mathcal{N}$-Kurven – Modell, basiert auf Mixture Density Netzwerken (MDN) und Bézierkurven, welche Kurvenkontrollpunkte als normalverteilt annehmen. Die Verwendung eines MDN-basierten Ansatzes steht im Einklang mit aktuellen Versuchen, Unsicherheitsschätzung als Regressionsproblem auszulegen, und ergibt ein generisches Modell, welches allgemein als Basismodell für die probabilistische Sequenzmodellierung einsetzbar ist.

Ein wesentlicher Vorteil des Modells ist unter anderem die Möglichkeit glatte, multi-modale Vorhersagen in einem einzigen Inferenzschritt zu generieren, ohne dabei Monte Carlo Simulationen zu benötigen. Durch die Verwendung von Bézierkurven als Basis, kann das Modell außerdem theoretisch für beliebig hohe Datendimensionen verwendet werden, indem die Kontrollpunkte in einen hochdimensionalen Raum eingebettet werden. Um die durch den Fokus auf beschränkte Indexmengen existierenden theoretischen Einschränkungen aufzuheben, wird zusätzlich eine konzeptionelle Erweiterung für das $\mathcal{N}$-Kurven – Modell vorgestellt, mit der unendliche stochastische Prozesse modelliert werden können. Wesentliche Eigenschaften des vorgestellten Modells und dessen Erweiterung werden auf verschiedenen Beispielen zur Sequenzsynthese gezeigt.

Aufgrund der hinreichenden Anwendbarkeit des $\mathcal{N}$-Kurven – Modells auf die meisten Anwendungsfälle, wird dessen Tauglichkeit umfangreich auf verschiedenen Mehrschrittprädiktionsaufgaben unter Verwendung realer Daten evaluiert. Zunächst wird das Modell gegen häufig verwendete probabilistische Sequenzmodelle im Kontext der Vorhersage von Fußgängertrajektorien evaluiert, wobei es sämtliche Vergleichsmodelle übertrifft. In einer qualitativen Auswertung wird das Verhalten des Modells in einem Vorhersagekontext untersucht. Außerdem werden Schwierigkeiten bei der Bewertung probabilistischer Sequenzmodelle in einem multimodalen Setting diskutiert. Darüber hinaus wird das Modell im Kontext der Vorhersage menschlicher Bewegungen angewendet, um die angestrebte Skalierbarkeit des Modells auf höherdimensionale Daten zu bewerten. Bei dieser Aufgabe übertrifft das Modell allgemein verwendete einfache und auf neuronalen Netzen basierende Grundmodelle und ist in verschiedenen Situationen auf Augenhöhe mit verschiedenen State-of-the-Art-Modellen, was die Einsetzbarkeit in diesem höherdimensionalen Beispiel zeigt. Des Weiteren werden Schwierigkeiten bei der Kovarianzschätzung und die Glättungseigenschaften des $\mathcal{N}$-Kurven – Modells diskutiert.

Acknowledgements

Throughout the past few years I have received a great deal of support and assistance in my goal of pursuing my PhD.

First of all, I would like to thank my dissertation advisor Prof. Dr.-Ing. Jürgen Beyerer for his guidance and feedback, which were invaluable to complete this thesis. I am grateful to Prof. Dr.-Ing. Marco Huber, Prof. Dr. Wolfgang Karl, Prof. Dr.-Ing. Jörg Henkel, and Prof. Dr. Gregor Snelting for agreeing to be part of my examination committee. In particular, I would like to thank Prof. Dr.-Ing. Marco Huber for his interest in my work and for being in the committee as a second advisor.

I want to acknowledge my colleagues of the *Object Recognition Department* at the *Fraunhofer Institute of Optronics, System Technologies and Image Exploitation* for their constant assistance. Our conversations were essential for solving various challenges surrounding my thesis. In particular, I would like to thank Dr. Stefan Becker for his ongoing encouragement and our joint research providing a strong foundation for parts of this thesis. Special thanks go to Dr. Wolfgang Hübner and Dr. Michael Arens for providing excellent conditions, assistance and feedback to prepare this thesis.

Lastly, I want to thank my family and friends for their continuous support and encouragement throughout the years. In particular, I want to thank Melanie and Maximilian Hemgesberg for inspiring conversations about my work, and finally my wonderful wife Simone for her unconditional support and our little son Leon for being the little sunshine that he is.

Contents

Notation

This chapter introduces the notation and symbols which are used in this thesis.

General notation

Scalars	italic Roman and Greek lowercase letters	a, α
Sets	bold calligraphic Roman uppercase letters	$\boldsymbol{\mathcal{D}}$
Sequences	calligraphic Roman uppercase letters	$\mathcal{S}$
Vectors	bold Roman lowercase letters	$\mathbf{x}$
Matrices	bold Roman uppercase letters	$\mathbf{R}$
Random variables	italic Roman uppercase letters	X

Probability Distributions

$\mathcal{N}$	Gaussian normal distribution
Ξ	Gaussian mixture distribution
$p(\cdot), q(\cdot)$	probability density functions
$p_\theta(\cdot), q_\phi(\cdot)$	probability density functions parameterized by θ, ϕ

Numbers, Indexing and Conventions

$\mathbb{N}$	natural numbers

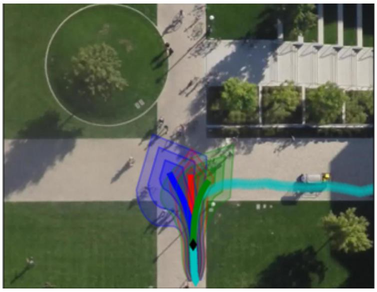
$\mathbb{N}_0$	natural numbers including zero (non-negative integers)
$\mathbb{R}$	real numbers
$\mathbb{R}_0^+$	non-negative real numbers (including zero)
t	indexing for points in time
i, j, k	indexing for objects, measurements and points
N,M,K	quantities
$\mathbf{0}$	zero vector
$\mathbf{I}$	identity matrix

Parametric Curves and Sequence Modeling

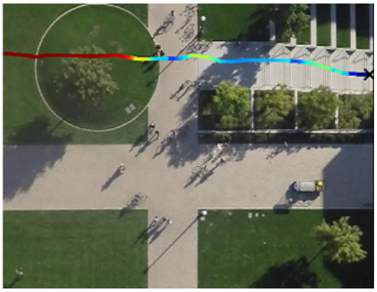
$\mathcal{G}_T$	stochastic process with index set T
T, T_N	index sets
$\mathscr{P}, \mathscr{P}_{\mathcal{N}}$	set of (Gaussian) curve control points
$B_{\mathscr{P}}(t), B_{\mathcal{N}}(t)$	(probabilistic) parametric curve function
$b_{i,N}(t)$	Bernstein polynomials
ψ, Ψ	$\mathcal{N}$-Curve (mixture)
$\mu^{\psi}(t)$	$\mathcal{N}$-Curve mean function
$\Sigma^{\psi}(t)$	$\mathcal{N}$-Curve covariance function
π	mixture weights
$p_t^{\psi}(x)$,$p_t^{\Psi}(x)$	curve point probability density function
$\tilde{t}$	meta-time index
t_c	curve-time index
$m(t)$	meta-time mapping
$m_c(t)$	curve-time mapping
$\mathscr{D}$	(training/test) dataset
$\mathbf{v}$	sequence encoding
θ	model parameters

1 Introduction

Sequential data, or rather timely ordered information, arises in the context of many different applications, like for example risk assessment in autonomous driving or in data-driven behavior analysis. In general, it is possible to reduce the majority of such use-cases to more abstract inference tasks, like sequence prediction. With real-world data being subject to noise and detection or annotation errors, the use of a probabilistic sequence model is favorable, as such models also take uncertainty in the data into account.



(a) Trajectory prediction



(b) Out-of-distribution detection

Figure 1.1: Exemplary sequence modeling tasks on different levels of abstraction: 2D trajectory prediction in a constraint setting and out-of-distribution detection built upon the derived probabilistic sequence model when training the prediction model. Both tasks can contribute to a superordinate risk assessment application. The prediction task (1.1a) is concerned with future trajectory prediction (red, green and blue distributions) given an observed trajectory (solid cyan). In such a structured environment, a sequence model is learned, which is capable of capturing statistically relevant paths through the given scene. As the sequence model provides a model for the underlying data distribution, out-of-distribution detection can be performed given a trajectory (1.1b). In this example, moving on the pathway is valid under the model, but moving onto the grass is highly unlikely. The validity under the model is color-coded from red (not valid) to blue (valid). Figure 1.1b is taken from [Har17].

The determination of such a probabilistic sequence model is commonly layed out as a learning problem, where the model parameters are estimated from given data samples. This formulation as a learning problem goes along with the current dominance of deep learning approaches in a range of different fields related to sequential data. However, working with uncertainties and associated probability distributions, most current deep learning-based approaches for probabilistic sequence modeling rely on the calculation of intractable probability density functions. Because of that, variational or sample-based approximations are generally required during training and inference in such models. Although there exist regression-based approaches, which try to avoid the need for such expensive approximations during training, they still require Monte Carlo methods for inference.

Following this, a common ground for current sequence modeling approaches can be observed in their need for Monte Carlo methods during inference. Thus, a research gap in regression-based approaches for multi-modal probabilistic inference emerges.

Towards this end, the primary goal of this thesis revolves around the formulation of a fully regression-based probabilistic sequence model. In addition, common drawbacks of existing models should be avoided, i.e.

1 The necessity of Monte Carlo or approximate Bayesian methods during either training or inference

2 Common sequential approaches to sequence generation are often uncontrolled and are prone to artifact generation,

Following this, this thesis proposes a probabilistic extension for parametric curves for use in probabilistic sequence modeling and provides an implementation of the resulting model based on regression neural networks. The motivation for basing the approach on parametric curves is driven by the following expectations: First, modeling full curves enables *instant* multi-step inference without iteration and the need for Monte Carlo methods. Further, generated sequences are constrained by the underlying parametric curves. This, in turn, is expected to help stabilizing training. In addition, artifact generation during inference should be mitigated. Finally, modeling a stochastic process in terms of a probabilistic parametric curve yields a compact representation of said stochastic process.

1.1 Contributions

In compliance with the aforementioned primary objectives, the main contributions provided in this thesis revolve around a novel probabilistic sequence model, built on a probabilistic extension to parametric curves. As such, the contributions can be ascribed to three categories: *theory*, *algorithms* and *evaluation*.

Theory: A probabilistic extension to Bézier curves and Bézier splines capable of modeling multi-modal stochastic processes is derived. In this extension, the Bézier curve's control points are assumed to be Gaussian, thus inheriting the stochasticity to the curve points by linear combination, resulting in a model for a continuous-time stochastic process. Discrete-time stochastic processes can be represented by discretizing such a probabilistic curve. Multi-modality is achieved, by combining multiple probabilistic curves into a mixture.

Algorithms: A learning- and regression-based approach for applying these probabilistic parametric curves in different sequence modeling tasks, specifically synthesis and prediction, is proposed. The approach is based on a Mixture Density Network, which outputs the parameters for (a mixture of) probabilistic parametric curves. This enables multi-step sequence generation without iteration or the need for Monte Carlo methods. Several toy examples assess different aspects and qualities of the approach.

Evaluation: An extensive evaluation of the proposed model is provided for the task of human trajectory prediction on real-world datasets. In addition, the common approach to evaluation in human trajectory prediction is examined with an attempt to provide insight into the suitability of the methodology for different task setups. Emphasis is put especially on commonly used performance measures. Finally, scalability of the approach is proven in a higher-dimensional scenario given by human motion prediction.

Additional contributions to the field of human trajectory prediction exceeding the topical scope of this thesis are given by:

1 A learning-based normalization for sequential data, which can be used as a data preprocessing method [Hug20b].

2 An approach for estimating the complexity of a given benchmark dataset accompanied by a ranking of commonly used datasets [Hug21].

3 A complementary benchmark, aiming at a fine-grained evaluation of trajectory prediction models, using a hierarchy of tasks [Hug20a].

1.2 Outline

The thesis is structured as follows: Chapter 2 provides a brief overview on the most common probabilistic sequence models most state-of-the-art deep learning models are built upon. This background chapter also serves the purpose of supporting the aforementioned claim for the revealed research gap. Chapter 3 provides the derivation of a probabilistic extension for Bézier curves and Bézier splines, including discussions on choices made for the approach and comparisons with related probabilistic sequence models. Closely connected to Chapter 3 is the proposed implementation of the probabilistic curve model given in Chapter 4. Besides implementation details, e.g. the structure of the model, several toy examples are provided, assessing different aspects and qualities of the model. Chapter 5 provides a real-world evaluation of the proposed model, using a low-dimensional and a higher-dimensional task, given by human trajectory prediction and human motion prediction. Finally, Chapters 6 and 7 conclude the thesis and give hints to potential future research directions.

2 Sequence Modeling

In the context of machine learning, the task of *sequence modeling* is, in general, concerned with determining (stochastic) models able to represent, process and generate sequential data from a given data basis. When uncertainties about the data are taken into account, the sequence model aims to provide an either implicit or explicit representation of the underlying probability distribution.

To enable a more nuanced view on sequence modeling, this general task can be subdivided into three closely related sub-tasks, namely sequence *encoding*, *synthesis* and *prediction*. While sequence encoding is concerned with reducing a given sequence into a compact representation, e.g. a single vector, sequence synthesis and prediction aim at generating sequential data. Sequence synthesis, on the one hand, is concerned with the generation of sequences according to an underlying probability distribution, potentially conditioned on a specific input. On the other hand, sequence prediction combines both tasks by first requiring to encode a given input sequence (the *observation*) in order to generate a prediction for future data points of the observed sequence. As such, sequence prediction can be regarded as a variant of conditional sequence synthesis, where the synthesis model is conditioned on another sequence. Most applications in the context of sequence modeling can be ascribed to at least one of these three more general inference tasks. A schematic of each task is given in Figure 2.1.

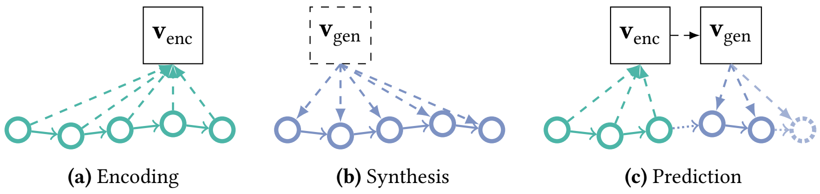

$\mathbf{v}_{\text{enc}}$
$\mathbf{v}_{\text{gen}}$
$\mathbf{v}_{\text{enc}}$
$\mathbf{v}_{\text{gen}}$
(a) Encoding
(b) Synthesis
(c) Prediction

Figure 2.1: Schematic of sequence modeling sub-tasks sequence encoding, synthesis and prediction. As an example, a sequence of 2D points is considered. In sequence encoding, the sequence model takes in a given sequence and encodes it into a specific representation, e.g. a vector $\mathbf{v}_{\text{enc}}$. A sequence synthesis model optionally takes a specific input, e.g. a vector $\mathbf{v}_{\text{gen}}$, and generates a sequence. Sequence prediction combines the two, as the sequence prediction model needs to encode a sequence it is given (green), in order to synthesize a continuation of that sequence (blue).

With the prevalence of noise and uncertainties in real-world data, statistical sequence models are employed for tackling either of the sequence modeling tasks. For determining a statistical sequence model, it is assumed that each sequence $\mathcal{S} = \{\mathbf{x}_t\}_{t \in T}$ in a specific dataset is a realization of an unknown stochastic process $\mathcal{G}_T = \{X_t\}_{t \in T}$ with index set T and random variables X_t following some probability distribution. Typically, T either corresponds to $\mathbb{N}_0$, $\mathbb{R}_0^+$ or some interval $[a, b]$, indicating a discrete-time, continuous-time or finite (continuous-time) stochastic process, respectively. Commonly, these statistical sequence models are either *probabilistic sequence models* or *stochastic process models*. While the latter are themselves variants of stochastic processes (e.g. *Gaussian processes* [Ras06]), probabilistic sequence models process and generate probability distributions, thus providing a model for the underlying stochastic process. Thereby, the probabilistic sequence model itself can be either probabilistic or even deterministic.

Following this, this thesis focuses on learning-based probabilistic sequence models for (conditional) sequence synthesis, including sequence prediction. A sequence model then generates a distribution over the sequence to be synthesized instead of a single (maximum likelihood) sample. The remainder of this section provides an overview of the most important probabilistic models in this context. Given the prevalence of deep learning-based models among current state-of-the-art approaches, the overview is limited to such models only. For an overview of machine learning models beyond deep learning, e.g.

state space models, such as recursive bayesian estimators [Sär13] or autoregressive models, like the autoregressive moving-average model [Box15], the reader may be referred to comprehensive surveys on the topic, e.g. [Rud20b]. Although this survey focuses on a prediction task, most mentioned models are more universally applicable.

2.1 Neural Sequence Processing

To preface the overview, it is important to mention that deep learning-based probabilistic sequence models are commonly built around an underlying neural sequence model, which is in charge of processing sequences at hand. While feed-forward networks (e.g. the Multilayer Perceptron [Mur91]) can be used in a setting of fixed length sequences or when applying a sliding window approach, dedicated sequence models are usually preferred. Common choices for the underlying sequence model are *Recurrent Neural Networks* (abbrev.: RNN, [Rum86]) and their variants, *Temporal Convolutional Networks* (abbrev.: TCN, [Bai18]) and *Transformer Networks* (abbrev.: TF, [Vas17]).

2.1.1 Recurrent Neural Networks

Recurrent Neural Networks are feed-forward networks with additional recurrent connections along the time axis, enabling it to iteratively process sequences and carry information about past inputs. As such, RNNs, and especially its *Long Short-Term Memory* (abbrev.: LSTM, [Hoc97]) and *Gated Recurrent Unit* (abbrev.: GRU, [Cho14]) variants, are widely used. While vanilla RNNs are prone to gradient-related problems during training, especially vanishing gradients [Pas13], aforementioned variants incorporate gating mechanism to cope with such problems. From an operational point of view, RNNs are usually build as either *1-to-1* or *sequence-to-sequence* (abbrev.: seq2seq, sometimes also denoted as encoder-decoder, [Sut14]) RNNs. On the one hand, a 1-to-1 RNN processes a given sequence one element at a time and generates an output at each time step. This approach is generally applicable. Opposed to that, seq2seq RNNs are more tailored towards conditional sequence synthesis, where a given sequence is encoded first using an encoder RNN. The resulting encoding is then decoded by another RNN – the decoder – in order to generate an output sequence. Overall, both variants yield comparable performance considering a range of sequence modeling tasks, with the GRU performing slightly better in many cases [Chu14][Joz15]. However, when the network is built as a sequence-to-sequence model, the LSTM outperforms the GRU variant [Bri17].

As a final note, due to RNNs employing an autoregressive structure, i.e. using their own output at time t as input at time $t + 1$ during inference, techniques for managing the network input during training should be discussed. The most commonly used approach is given by the *teacher forcing* approach [Goo16]. Teacher forcing is a technique for training recurrent neural networks, that, at time t uses the ground truth $\mathbf{x}_t$ as input, rather than the model's output from the previous time step $\hat{\mathbf{y}}_{t-1}$. As such, the actual network input signal is replaced with a *teacher* signal. This approach helps reaching convergence faster, at the cost of the network only eventually learning to cope with its own imperfect output. A way to tackle this problem, is to start the training process using teacher forcing and then slowly transitioning into an auto-conditioning scheme, where the actual network output is fed back in the subsequent time step [Ben15].

2.1.2 Temporal Convolutional Networks

Temporal Convolutional Networks are a special variant of *Convolutional Neural Networks* (abbrev.: CNN, [LeC95]) for sequential data, popularized by the *WaveNet* model in the context of audio synthesis [Oor16]. The model consists of dilated causal convolutions. While *dilated convolutions* [Hol90] are incorporated in order to capture long range dependencies, *causal convolutions* [Oor16] ensure that the temporal order of a given sequence is taken into account. An advantage of the TCN over the RNN is its inherent parallelism on the one hand and a more stable training on the other hand. As the TCN processes multiple time steps at once instead of sequentially, convolutions can be done in parallel. A more stable training of the TCN can be attributed to more stable gradients. On the downside, the TCN is less flexible in processing sequences of variable length. Although it is possible to process variable length sequences by sliding the convolutional kernels, the *memory* of the model is limited by the filter kernel's width and the dilation rate, whereas the RNN may, in theory, establish dependencies up to the first sequence element. Looking at a range of different sequence modeling tasks, the TCN is able to outperform LSTM and GRU models [Bai18] or at least perform similar [Bec18].

2.1.3 Transformer Networks

Transformer Networks originated from the field of natural language modeling as a replacement for the commonly used RNN-based sequence-to-sequence models. Since its emergence, Transformers also gained traction in other application domains, most notably speech processing, where Transformers consistently outperform RNN-based models [Kar19, Wan21]. Compared to RNNs, which process sequences recursively, the Transformer aims to get rid of recurrence and always considers the entire input sequence. As such, the most important concept Transformers are build around are *positional encoding* and *attention*. While the positional encoding enriches input sequence elements with information about their position within the given sequence, the attention mechanism is in charge of determining which parts of the input sequence are of importance for the calculation of each element in the target sequence. Further, using an attention mechanism, enriched information is available for sequence generation, when compared to RNN-based sequence-to-sequence models, where the sequence decoder is only provided with an encoded representation of the input sequence. Besides that, Transformers are in general more stable during training, but also seem to be more prone to overfitting, which indicates problems with generalization [Zey19]. Additionally, in its original formulation, the Transformer model is restricted to fixed-length sequences. This restriction, is tackled by the *Transformer-XL* extension [Dai19], which re-introduces a notion of recurrence and extends on the positional encoding concept.

2.2 Probabilistic Sequence Models

This section provides an overview of deep learning-based probabilistic sequence models commonly used as a basis for task-specific model adaptations. These models can roughly be put into three categories: *Bayesian*, *regression-based* and *transformative* approaches. For each category, the most relevant representatives are examined.

2.2.1 Bayesian Approaches

In deep learning-based Bayesian approaches, a deterministic neural network is turned into a probabilistic model by treating all its parameters as random variables. A prominent example for this class of approaches is given by *Bayesian Neural Networks* (abbrev.: BNN, [Bis95]). In these models, inference and parameter estimation are built around Bayes' theorem. As such, the neural network outputs an arbitrary predictive probability distribution

$$p(\mathbf{y}|\mathbf{x},\mathcal{D}) = \int_{\theta} p(\mathbf{y}|\mathbf{x},\theta')p(\theta'|\mathcal{D})d\theta' \tag{2.1}$$

by propagating the units' output distributions through the network. For parameter estimation, the posterior distribution

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}_y|\mathcal{D}_x,\theta)p(\theta)}{\int_{\theta} p(\mathcal{D}_y|\mathcal{D}_x,\theta')p(\theta')d\theta'} \propto p(\mathcal{D}_y|\mathcal{D}_x,\theta)p(\theta) \tag{2.2}$$

of the network parameters, given a set of data samples, needs to be determined. Here, θ denotes the model parameters, $\mathcal{D}$ the training dataset split into input data $\mathcal{D}_x$ and target data $\mathcal{D}_y$, and $\mathbf{x}$ and $\mathbf{y}$ denote specific input and target vectors, respectively. Due to intractable probability distributions arising from non-linear transformations, usually either Monte Carlo methods [Nea92] or approximate inference is required for both training and inference. Common techniques used for approximate inference include variational inference (also known as *Bayesian Backpropagation*, [Blu15]), inference based on expectation propagation [Her15] and Monte Carlo Dropout [Gal16]. In order to extend BNNs for probabilistic sequence modeling, *Bayesian Recurrent Neural Networks* (abbrev.: BRNN, [For17]) were introduced. For the BRNN, the variational Bayesian Backpropagation scheme is adapted for Backpropagation Through Time [Wer90].

In summary, BNN-based approaches provide a fully probabilistic framework for sequence modeling. In addition to that, major advantages of such models are also given by their robustness to overfitting and the ability to provide information about model uncertainty. As a drawback, such models are difficult to train, due to the requirement of approximate inference making the training computationally more intensive and potentially less stable. Further, the need for approximate inference also yields a significant computational overhead when generating predictions. As a final note, considering the need for approximate inference, the *Bayesian Perceptron* [Hub20] is worth mentioning. The Bayesian Perceptron is a specific novel probabilistic formulation of the Perceptron [Ros58], which provides closed-form parameter propagation and estimation, thus eliminating the need for Monte Carlo Methods and approximate inference. However, a recurrent extension for sequence modeling building on this approach is not yet available.

Despite not being probabilistic sequence models according to the definition given earlier in this section, Gaussian process models are worth mentioning in the context of deep learning-based Bayesian approaches. This is due to their corresponding relationship, in that the function computed by a deep neural network is a function drawn from a Gaussian process [Lee18]. Conversely, a GP corresponds to a neural network with an infinite number of units in its hidden layer [Nea96, Wil97].

Gaussian Processes (GP) and Gaussian process regression [Ras06] provide a well-established model for probabilistic sequence modeling and especially prediction. Given a collection of sample points of a non-linear function $f(\cdot) : \mathbb{R}^m \rightarrow \mathbb{R}$, a mean function $m(\cdot)$ and a covariance function $k(\cdot,\cdot)$ (*kernel*), the GP yields a multivariate Gaussian prior probability distribution over function space. The Gaussian distribution can be used to determine a conditional predictive distribution over the next element in a sequence given preceding observations. By embedding this 1-step prediction model into a sequential Monte Carlo simulation, multiple time steps can be predicted [Ell09]. *Deep Gaussian Processes* (abbrev.: DGP, [Dam13]) extend on the GP framework in order to constitute non-Gaussian, and therefore more complex, models. A DGP is a hierarchy of multiple GPs using non-linear mappings between each layer of the hierarchy. However, the resulting probability densities are intractable and thus require an approximate solution, which can be achieved e.g. by variational approximation [Cam15]. A special case of the DGP, which implements an autoregressive structure comparable to that of an RNN, is given by *Recurrent Gaussian Processes* (abbrev.: RGP, [Mat16]). Here, the priors of latent variables in each hidden layer follow an autoregressive structure. Following this, a recurrent variational approximation scheme, which uses a state space model-based approach, is introduced for inference. Besides having a computation intensive inference scheme, GP-based approaches grant good control over generated sequences, by explicitly modeling the kernel functions, thus controlling the prior over functions representable by the model through a regularization over the entire value range. This gives an advantage over most competing neural network-based approaches that generate sequences in a mostly unconstrained fashion. It should be noted, however, that GP-based approaches are rarely used in most application domains currently dominated by deep learning-based models.

2.2.2 Regression-based Approaches

One of the main areas of application for neural networks is given by regression tasks, due to their ability to learn arbitrary mappings from a given domain into a targeted co-domain. As such, neural networks can be used for probabilistic modeling when treating the task of uncertainty estimation as a regression problem. The neural network is then in charge of learning a mapping from a given set of samples onto the parameters of a probability distribution estimating the generating distribution. Following this, the negative data log likelihood is optimized during training:

$$\mathcal{L} = -\log p_\theta(\mathbf{x}). \tag{2.3}$$

The most widely used regression-based neural network for probabilistic modeling is given by *Mixture Density Networks* (abbrev.: MDN, [Bis94, Bis06]), which map the output of their last layer onto the parameters of a mixture distribution. The prevalent choice for the mixture distribution is given by the Gaussian distribution, although Laplace distributions have also been used with Mixture Density Networks [Bra19]. For building a probabilistic sequence model using MDNs, a common choice is the *Recurrent Mixture Density Network* (abbrev.: R-MDN) model as proposed in [Gra13]. Here, an MDN is stacked on top of an LSTM network. The recurrent structure is then used for encoding the observed sequence as well as for generating predictions.

Compared to Bayesian approaches, using a deterministic model, such regression-based approaches are generally much simpler in terms of inference and computational cost, while still generating probabilistic output. On the downside, these approaches only give a point-estimate for the parameters of a preset target probability distribution. This limits the modeling capabilities of the approach and also does not allow to make assumptions about model uncertainty in a direct way. Drawbacks specific to R-MDNs are on the one hand given by the fact that generating multi-modal probabilistic predictions generally requires expensive Monte Carlo simulation [Hug18]. On the other hand, MDNs are prone to mode collapse [Mak19], where the model collapses into generating only slight variations of a single mode.

A more detailed introduction to MDNs is given in Section 4.1.

2.2.3 Transformative approaches

Transformative approaches *transform* samples of a simple probability distribution into a sample-based representation of a more complex probability distribution. As such, transformative approaches combine deterministic neural networks with stochastic inputs in order to define a generative model. The most important models in this category are given by *Variational Autoencoders* and *Generative Adversarial Networks*.

Variational Autoencoders (abbrev.: VAE, [Kin14]) are a class of deep generative models with latent variables. In latent variable models, the unknown generating distribution $p(\mathbf{x})$ is modeled in terms of latent variables $\mathbf{z} \sim p_\theta(\mathbf{z})$ with prior distribution $p_\theta(\mathbf{z})$. According to Bayes' theorem, $p(\mathbf{x})$ and $p_\theta(\mathbf{z})$ are linked by the *mappings* $p_\theta(\mathbf{x}|\mathbf{z})$ (*likelihood*) and $p_\theta(\mathbf{z}|\mathbf{x})$ (*posterior*), which are computed explicitly. Following this, the generally intractable posterior needs to be approximated. For this approximation, the VAE follows a variational inference approach, approximating $p_\theta(\mathbf{z}|\mathbf{x})$ with the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$. Putting each part of the latent variable model together, in VAEs $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ are defined in terms of deterministic neural networks, denoted as the *recognition model* and the *generative model*. The networks are arranged similar to autoencoders [Hin94], with the latent space as the bottleneck. As a consequence, the generative process of the VAE works by transforming a set of latent variable samples $\mathbf{z}_i \sim p_\theta(\mathbf{z})$ drawn from the prior distribution $p_\theta(\mathbf{z})$ using the generative model $p_\theta(\mathbf{x}|\mathbf{z})$. The prior distribution $p_\theta(\mathbf{z})$ is commonly defined as $\mathcal{N}(\mathbf{0},\mathbf{I})$. Training the VAE is made possible by using the variational lower bound (also known as the evidence lower bound)

$$\log p(\mathbf{x}) \geq -\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] \tag{2.4}$$

in conjunction with the *reparameterization trick* [Kin14], which enables joint gradient-based training of the entire network. It should be noted, that *Normalizing Flows* [Rez15] can be used in a VAE in order to replace the learned approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$. Normalizing Flows are a chain of *invertible* mappings, that can be used to transform samples of one probability distribution into another. In the context of VAEs, Normalizing Flows provide a framework for building a more flexible and complex variational approximation of the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ through an iterative procedure [Kin16, Hua18].

In order to extend on the concept of VAEs for sequence modeling, two approaches have emerged: the *Seq2seq Conditional VAE* and the *Variational Recurrent Neural Network*. *Seq2seq Conditional VAEs* [Bow16] build on the concept of conditional VAEs (abbrev.: CVAE, [Soh15]), which employ a conditional generating distribution $p(\mathbf{x}|\mathbf{v})$ conditioned on some input $\mathbf{v}$. This results in the conditional latent prior $p_\theta(\mathbf{z}|\mathbf{v})$ and conditional mappings

$p_\theta(\mathbf{x}|\mathbf{z},\mathbf{v})$ and $q_\phi(\mathbf{z}|\mathbf{x},\mathbf{v})$. Following this, for sequence modeling, each sample $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z},\mathbf{v})$ resembles a full sequence. Further, in the case of sequence prediction, a given observed sequence needs to be encoded into $\mathbf{v}$, in order to condition the CVAE's generative model on the given sequence. Hence, in seq2seq CVAEs, a CVAE is combined with a seq2seq RNN, where the RNN encoder is used to encode the observation into $\mathbf{v}$ and the RNN decoder resembles $p_\theta(\mathbf{x}|\mathbf{z},\mathbf{v})$, generating the target sequence from $\mathbf{v}$ and $\mathbf{z}$. Following this, the seq2seq CVAE generates a distribution over sequences of a specific length. Opposed to the composite approach of seq2seq CVAEs, the *Variational Recurrent Neural Network* (abbrev.: VRNN, [Chu15]) explicitly models the dependencies between latent variables of subsequent time steps. Following this, the VRNN embeds an RNN into a CVAE, which is at time t conditioned on the RNN's previous hidden state $\mathbf{h}_{t-1}$. For sequence synthesis, the VRNN then operates as a 1-to-1 model, generating a sequence of probability distributions rather than a probability distribution over sequences.

To summarize, VAE-based probabilistic sequence models provide comparable modeling capabilities to Bayesian approaches, while eliminating expensive approximations during inference, due the transformative approach. On the downside, because of imperfect reconstructions due to the injected noise when generating samples, training results can become less consistent.

Generative Adversarial Networks (abbrev.: GAN, [Goo14]) are another type of generative model, learning an *implicit*[1] model of the unknown generating distribution $p(\mathbf{x})$. While the generative model component in GANs is very similar to that of VAEs in that samples of a simple distribution are transformed into a sample-based representation of a more complex distribution using a deterministic neural network, the network structure and approach to estimating the parameters of the generative model is vastly different. In order to bypass the need to solve or approximate an intractable posterior distribution, GAN training is framed as a supervised learning problem, using a combination of two neural networks: the generative model itself (denoted as *generator*) and

[1] Implicit density models do not compute $p(\mathbf{x})$, but allow sampling from the underlying distribution using the model.

a *discriminator*. Both models are jointly trained playing a zero-sum game, where the generator tries to generate samples from the unknown data distribution $p(\mathbf{x})$, which the discriminator is incapable of classifying as real or fake. As such, the generator G and the discriminator D play the two-player minimax game

$$\mathcal{L} = \min_G \max_D \underbrace{\left(\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right]\right)}_{V(G,D)} \tag{2.5}$$

with value function $V(G,D)$. The generator's stochastic input distribution $p(\mathbf{z})$ is commonly defined as the multivariate standard Gaussian $\mathcal{N}(\mathbf{0},\mathbf{I})$.

Similar to VAE-based approaches, the GAN can be extended for probabilistic sequence modeling by combining a conditional variant of the GAN [Mir14] with seq2seq RNNs [Yu17, Gup18]. Following this, the conditional generator $G(\mathbf{z},\mathbf{v})$ is combined with a seq2seq RNN and the conditional discriminator $D(\mathbf{x},\mathbf{v})$ is combined with an RNN sequence encoder. With the similarities to the VAE in the generative model, GANs provide similar benefits without the need for variational inference during training. Despite this, GANs tend to be hard to train because of vanishing gradient problems and the GANs proneness to mode collapse. While these problems are addressed by variations of the GAN building on the Wasserstein distance [Arj17, Gul17] or by incorporating multiple versions of the discriminator into the generator's loss function [Met17], balancing parameter updates between the generator and the discriminator still poses a challenging problem, as the discriminator converges faster than the generator on many occasions [Ham20].

2.3 Placement of this Thesis

Looking at the overview of commonly used (probabilistic) sequence models for handling sequential data under uncertainty, a research gap in regression-based approaches for multi-modal probabilistic inference is revealed. Following this, this thesis aims to provide a fully regression-based probabilistic sequence model with respect to model training and inference using the model.

The targeted placement of this thesis among other sequence models is given in Table 2.1.

Table 2.1: Targeted placement of this thesis among other (probabilistic) sequence models.

Model	**Approach**			
	Regression		**(Approximate) Bayesian**	
	Training	**Inference**	**Training**	**Inference**
BNN			✗	✗
DGP			✗	✗
VAE			✗	✗
GAN			✗	✗
R-MDN	✗			✗[a]
This thesis	✗	✗		

[a] In a multi-modal setting

3 Concept

Throughout this chapter, a probabilistic sequence model for representing stochastic processes is formulated, which aims to avoid the necessity of Monte Carlo approaches. To achieve this, first and foremost a sequence model for fixed-length sequences will be introduced in Section 3.1, covering the most application-relevant case. This covers continuous-time as well as discrete-time stochastic processes with bounded index set in an unimodal and a multi-modal setting. The model is then extended for the representation of infinite stochastic processes in Section 3.2 by lifting some conceptual limitations present in the former variant of the model.

The general idea behind the proposed probabilistic sequence model is to circumvent Monte Carlo sampling. Therefore, the model needs to represent full sequences instead of iteratively building them. Following this, a probabilistic extension to a certain type of parametric curves, Bézier curves in this case, is derived, granting a suitable representation of sequential data in arbitrary dimensions. The probabilistic sequence model is then built on these probabilistic Bézier curves.

3.1 The $\mathcal{N}$-Curve Model

This section proposes a Bézier curve defined by stochastic control points capable of describing a continuous-time stochastic process $\mathcal{G}_T = \{X_t\}_{t \in T}$ on a closed range with Gaussian random variables $X_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and index set $T = [0, 1]$. This concept is further extended for modeling random variables following a Gaussian mixture distribution.

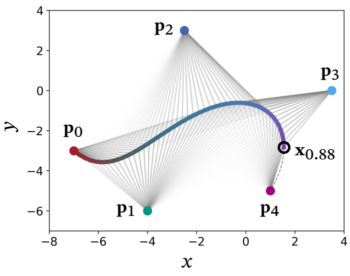
Starting with plain Bézier curves, a Bézier curve (e.g. [Pra02, Far02]) of degree N_{deg}

$$B_{\mathscr{P}}(t) = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t)\mathbf{p}_i \tag{3.1}$$

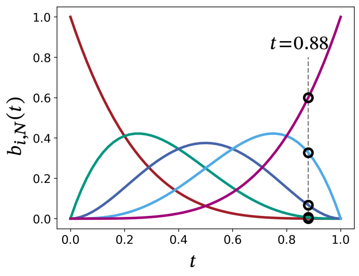
is a blended curve constructed as a linear combination of $N_{\text{deg}} + 1$ d-dimensional *control points* $\mathscr{P} = \{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{N_{\text{deg}}}\}$ using the Bernstein basis polynomials [Lor13]

$$b_{i,N}(t) = \binom{N}{i}(1-t)^{N-i}t^i \tag{3.2}$$

as blending functions. The Bernstein basis polynomials are non-negative and satisfy $\sum_i b_{i,N}(t) = 1$. Each curve point $\mathbf{x}_t = B_{\mathscr{P}}(t)$ is determined by the curve's positional parameter $t \in [0,1]$, where $t = 0$ corresponds to $\mathbf{p}_0$ and $t = 1$ to $\mathbf{p}_{N_{\text{deg}}}$, respectively. The positional parameter can also be interpreted as a *time* parameter when looking at the curve points as a timely-ordered sequence of points. An example for a 2-dimensional Bézier curve of degree $N_{\text{deg}} = 4$ for $t \leq 0.88$ with corresponding Bernstein basis polynomials is depicted in Figures 3.1a and 3.1b, respectively.



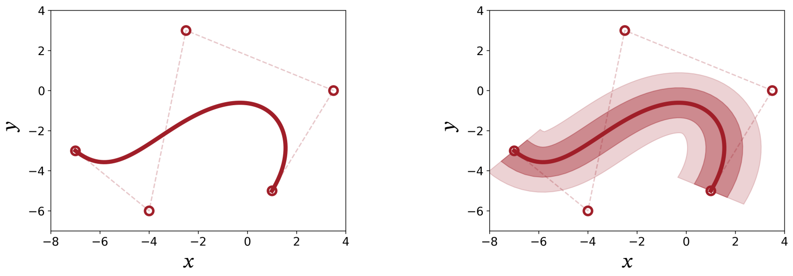
(a) Exemplary 2-dimensional Bézier curve of degree $N_{\text{deg}} = 4$ for $t \leq 0.88$.



(b) Bernstein basis polynomials $b_{i,4}(t)$ for 4'th degree blending.

Figure 3.1: Illustrating the connection between the Bernstein basis polynomials and Bézier curve construction. The Bernstein polynomial values control the weighting of control points for calculating curve points. The colors of the control points in figure (a) are associated with the weight curve of the same color in figure (b). Weights of control points for each curve point are dependent on the positional parameter t. Figure (a) shows a curve constructed up to $t = 0.88$, the remainder is indicated as a dashed line. Corresponding weights for $t = 0.88$ are indicated by circular markers in figure (b).

Considering the objective of modeling a stochastic process, the curve points along this parametric curve have to be stochastic. A schematic of such a probabilistic Bézier curve is illustrated in Figure 3.2. Here, Figure 3.2a illustrates a discrete 2-dimensional Bézier curve as the starting point. Figure 3.2b indicates uncertainty associated with each curve point as a shaded region around the curve. It has to be noted, that this presentation of uncertainty is for illustration purposes only. Uncertainties of multiple time steps are overlayed while only considering uncertainty orthogonal to the actual curve. Thus, it does not reflect the real probability distribution when integrating over the positional parameter.



(a) Exemplary 2-dimensional Bézier curve of degree $N_{\text{deg}} = 4$.

(b) Schematic extension of a Bézier curve incorporating curve point uncertainty.

Figure 3.2: Illustration of the starting point ((a): discrete Bézier curve) and goal ((b): probabilistic Bézier curve) of this section for a Bézier curve of degree $N_{\text{deg}} = 4$. Uncertainty associated with each curve point is indicated by a shaded region around the curve, representing σ and 2σ regions. Note: The presentation of uncertainty is for illustration purposes only and does not reflect a real probability distribution integrated over the curve's positional parameter.

In order to define a probabilistic extension for Bézier curves, such that generated curve points are stochastic and follow some probability distribution, it is necessary for the control points to be stochastic as well. This is due to every curve point being a linear combination of the curve's control points. Thus, an important question is given by the choice of a suitable probability distribution for the control points. A common choice is given by the Gaussian distribution, which is commonly used in machine learning and statistics due to its mathematical properties. On the one hand, the popularity of the Gaussian distribution can be explained through the *central limit theorem* [And10], which states that the sum of independent random variables converges towards a Gaussian distribution. Further, among all real-valued distributions with a given mean and variance, the Gaussian distribution is the *distribution of maximum entropy* [Con04]. On the other hand, the most notable mathematical property for defining a probabilistic Bézier curve is the fact that the linear combination of Gaussian random variables is again Gaussian.

Following this, for describing a stochastic process $\mathcal{G}_T$ in terms of a parametric curve, each curve point should follow a Gaussian distribution. Thus, a *Gaussian Bézier curve* ψ, denoted as $\mathcal{N}$-Curve, is proposed. The $\mathcal{N}$-Curve extends on Equation (3.1) and defines the control points $\mathscr{P}_{\mathcal{N}} = \{P_0, P_1, ...P_{N_{\text{deg}}}\}$ to follow a Gaussian distribution with $P_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)\ \forall P_i \in \mathscr{P}_{\mathcal{N}}$. The set of mean vectors is denoted as $\mu_{\mathscr{P}} = \{\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_{N_{\text{deg}}}\}$ and the set of covariance matrices $\Sigma_{\mathscr{P}} = \{\boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1, ..., \boldsymbol{\Sigma}_{N_{\text{deg}}}\}$, respectively. Thus, the $\mathcal{N}$-Curve is defined by a tuple $\psi = \mathscr{P}_{\mathcal{N}} = (\mu_{\mathscr{P}}, \Sigma_{\mathscr{P}})$. As curve points are defined through a linear combination of the control points, the stochasticity is inherited from the control points to the curve points $\{X_t\}_{t\in[0,1]}$. This is due to the fact, that for $\mathbf{A}X + \mathbf{B}Y$ with $X \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ and $Y \sim \mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$ follows¹

$$\mathbf{A}X + \mathbf{B}Y \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu}_x + \mathbf{B}\boldsymbol{\mu}_y, \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^T + \mathbf{B}\boldsymbol{\Sigma}_y\mathbf{B}^T).$$

Thus, the curve function

$$B_{\mathcal{N}}(t, \psi) = (\mu^{\psi}(t), \Sigma^{\psi}(t)) \tag{3.3}$$

with

$$\mu^{\psi}(t) = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t)\boldsymbol{\mu}_i \tag{3.4}$$

and

$$\Sigma^{\psi}(t) = \sum_{i=0}^{N_{\text{deg}}} \left(b_{i,N_{\text{deg}}}(t)\right)^2 \boldsymbol{\Sigma}_i, \tag{3.5}$$

defines the parameters of a (multivariate) Gaussian probability distribution for each $t \in [0, 1]$. Each d-dimensional curve point X_t then follows the respective Gaussian distribution $\mathcal{N}(\mu^{\psi}(t), \Sigma^{\psi}(t))$ at index t. The Gaussian probability

¹ Following the definition as provided in *The Matrix Cookbook* [Pet08].

density at index t is given by

$$
\begin{aligned}
p_t^{\psi}(\mathbf{x}) &= p\left(\mathbf{x}|\mu^{\psi}(t), \Sigma^{\psi}(t)\right) \\
&= \mathcal{N}\left(\mathbf{x}|\mu^{\psi}(t), \Sigma^{\psi}(t)\right) \\
&= \frac{1}{|2\pi\Sigma^{\psi}(t)|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}\left(\mathbf{x}-\mu^{\psi}(t)\right)^{\top}\left(\Sigma^{\psi}(t)\right)^{-1}\left(\mathbf{x}-\mu^{\psi}(t)\right)\right\}.
\end{aligned}
\tag{3.6}
$$

An example for a Gaussian curve point constructed from Gaussian control points for $t = 0.5$ is depicted in Figure 3.3. The intermediate control point P_1 influences X_t the most, which leads X_t to adopt a skewed covariance ellipse. Due to the covariance matrices being interpolated, the other control points, P_0 and P_2, contribute to X_t by making the covariance ellipse more spherical.

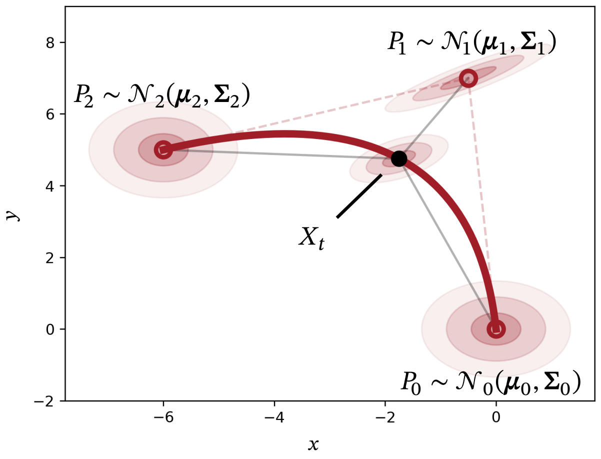
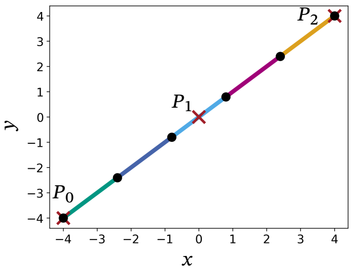


Figure 3.3: Example for a Gaussian curve point X_t on an $\mathcal{N}$-Curve with 3 Gaussian control points for $t = 0.5$. The covariance matrix of X_t's Gaussian distribution is a combination of the control point covariance matrices.

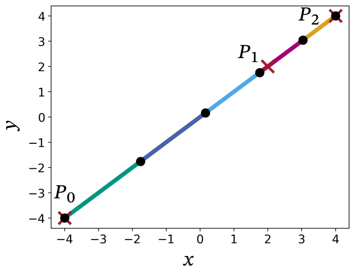
So far, a stochastic process model for a continuous index set $T = [0, 1]$ has been defined. In contrast to this, many real-world applications require discrete-time stochastic processes handling sequential data. For handling such use-cases, the $\mathcal{N}$-Curve model can be used to model Gaussian distributions at N discrete points in time with $B_{\mathcal{N}}(t, \psi)$ using N equally distributed values for t, yielding a discrete subset

$$T_N = \left\{\frac{v}{N-1} | v \in \{0, ..., N-1\}\right\} = \{t_1, ..., t_N\} \tag{3.7}$$

of the index set T. Thus, each process index (curve parameter) $t_i \in T_N$ corresponds to its respective sequence index at time $i \in \{1, ..., N\}$. It has to be noted, that using equidistant values for t does not necessarily result in equidistant curve points. The distribution of the curve points along the curve depends on the positions of the control points. This is illustrated in Figure 3.4.



(a) Example for equidistant curve points ($\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 \approx 2.3$).



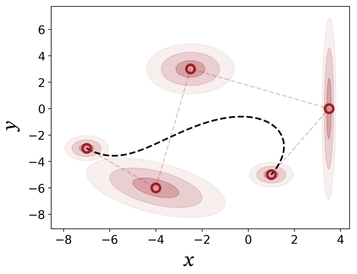
(b) Example for decreasing distance between curve points ($\Delta_1 \approx 3.2$, $\Delta_2 \approx 2.7$, $\Delta_2 \approx 2.3$, $\Delta_4 \approx 1.8$ and $\Delta_5 \approx 1.4$).

Figure 3.4: Illustration of the impact of control point positioning on the distribution of curve points along the curve. Figures (a) and (b) show two exemplary Bézier curves with one shifted control point and identical shape. Note that the shape is not impacted by shifting P_1, as it lies on the straight line between P_0 and P_2. Curve points (black circular markers) are calculated using the same discrete index set $T_{N=6} = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ of equally distributed values for t.

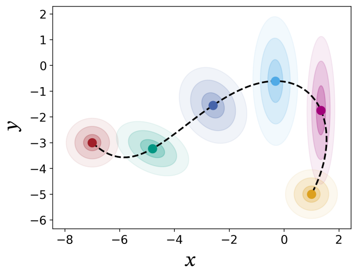
Finally, the Gaussian random variable X_{t_i} at time i is given by

$$X_{t_i} \sim \mathcal{N}(B_{\mathcal{N}}(t_i, \psi)) = \mathcal{N}(\mu^{\psi}(t_i), \Sigma^{\psi}(t_i)) \tag{3.8}$$

with $P_0 = X_{t_0}$ and $P_{N_{\text{deg}}} = X_{t_N}$ as exact start and end conditions. Figure 3.5 depicts a 2-dimensional example for an $\mathcal{N}$-Curve with 5 control points. The mean curve and control points with respective covariance ellipses are shown in Figure 3.5a. Gaussian random variables X_t along the $\mathcal{N}$-Curve given different values for t are illustrated in Figure 3.5b. The influence of the most dominant control point for each curve point X_t is clearly visible in the covariances, adapting towards respective control point covariances. Note that the parametric curve interpolates the mean vectors of all Gaussian distributions through time.



(a) Gaussian control points and continuous mean curve of a 2-dimensional $\mathcal{N}$-Curve.



(b) Gaussian random variables along the $\mathcal{N}$-Curve for $t \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

Figure 3.5: Example for modeling a finite discrete-time stochastic process using an $\mathcal{N}$-Curve. The stochastic process consists of random variables corresponding to a discrete subset of $T = [0,1]$.

As a final aspect to consider, the $\mathcal{N}$-Curve model can easily be extended for modeling multi-modal stochastic processes. While Gaussian probability distributions are a sufficient representation for unimodal sequence data, many real-world problems require a multi-modal representation. For this, a common approach is to use a Gaussian mixture probability distribution

$$\Xi\left(\{\pi_k\}_{k\in\{1,\dots K\}}, \{(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}_{k\in\{1,\dots K\}}\right), \tag{3.9}$$

defined by K weighted Gaussian components and probability density function

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right), \text{ with } \sum_{k=1}^{K} \pi_k = 1 \text{ and } \pi_k \geq 0. \tag{3.10}$$

In the same way, the concept of $\mathcal{N}$-Curves can be extended to a mixture $\Psi = (\pi, \{\psi_1, ..., \psi_K\})$ of K weighted $\mathcal{N}$-Curves with normalized weights $\pi = \{\pi_1, ..., \pi_K\}$. The stochastic curve points X_t at index $t \in T$ then follow a Gaussian mixture distribution

$$X_t \sim \Xi\left(\pi, \{B_{\mathcal{N}}(t, \psi_k)\}_{k \in \{1,...,K\}}\right). \tag{3.11}$$

Accordingly, the probability density at $t \in T$ induced by Ψ is given by

$$p_t^{\Psi}(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} | \mu^{\psi_k}(t), \Sigma^{\psi_k}(t)), \tag{3.12}$$

with $\mu^{\psi_k}(t)$ and $\Sigma^{\psi_k}(t)$ given by the k'th $\mathcal{N}$-Curve, i.e. $(\mu^{\psi_k}(t), \Sigma^{\psi_k}(t)) = B_{\mathcal{N}}(t, \psi_k)$. Following this, each stochastic curve point can be multi-modal and each mode of the modeled stochastic process follows a separate $\mathcal{N}$-Curve. As such, the $\mathcal{N}$-Curve mixture provides the evolution of X_t along multiple paths through time. An example for an $\mathcal{N}$-Curve mixture is depicted in Figure 3.6.

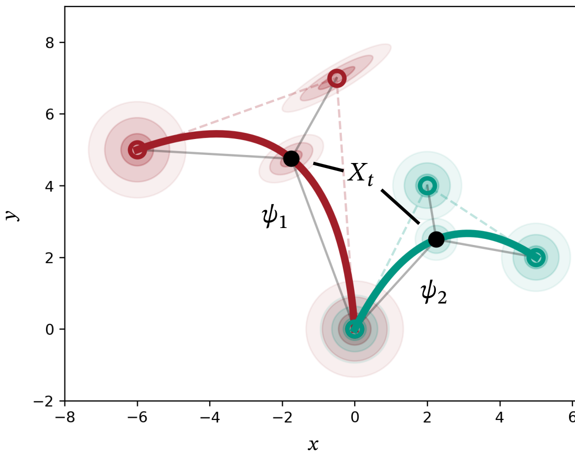
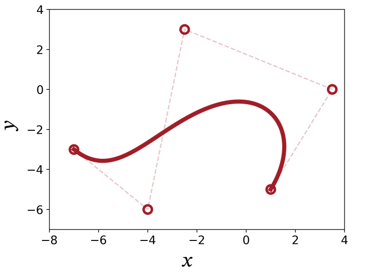


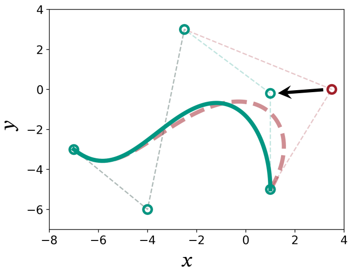
Figure 3.6: Example for a multi-modal stochastic curve point X_t for $t = 0.5$ given by an $\mathcal{N}$-Curve mixture consisting of 2 $\mathcal{N}$-Curves ψ_1 and ψ_2. Both ψ_1 and ψ_2 are defined by 3 Gaussian control points. The curve point X_t follows a 2-component Gaussian mixture distribution.

3.1.1 Rationale behind choosing Bézier curves

Among related parametric curves with alternative formulations (e.g. Pythagorean-hodograph curves [Far08]) or basis polynomials (e.g. Lagrange bases [War79, Jef88] or the power basis [Sto89]), Bézier curves are the most widely used type of blended curves in various fields, especially in computer-aided design (e.g. [Fit14]), animation (e.g. [Haa18, Izd20]) and path planning (e.g. [Jol09, Tha19]). Besides their popularity, Bézier curves offer some valuable properties for the $\mathcal{N}$-Curve model. First and foremost, Bézier curves are numerically stable, as well as easy to calculate, control and manipulate. Every control point contributes to every curve point, which makes curve construction more intuitive and reasonable. An example for how the manipulation of a single control point impacts the entire curve is given in Figure 3.7.



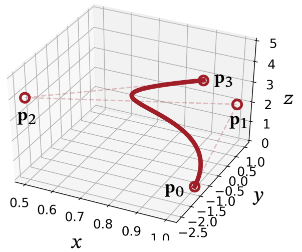
(a) 2-dimensional Bézier curve of degree 4.



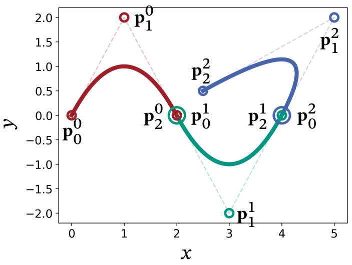
(b) Impact of relocating a single (intermediate) control point.

Figure 3.7: Illustration of global control, i.e. every control point affects every curve point, in Bézier curves. The initial curve is depicted in red and the modified curve in green.

In addition, Bézier curves provide a compact representation of the entire curve in terms of a set of control points. This, in turn, allows the description of a whole sequence of random variables, requiring only few stochastic control points. Further, Bézier curves can be scaled up to higher dimensions easily by increasing the dimensionality of the control points. Besides that, Bézier curves provide a commonly used building block for splines, which are segmented curves consisting of multiple parametric curves. The ability to combine Bézier curves into splines is relevant for a recurrent extension of the $\mathcal{N}$-Curve model as discussed in Section 3.2. Figure 3.8 provides basic examples for scalability and Bézier splines.



(a) Exemplary 3-dimensional Bézier curve of degree 4.



(b) Exemplary Bézier spline with 3 Bézier curve segments.

Figure 3.8: Basic examples for Bézier curve scalability (3.8a) and Bézier curves as a building block for Bézier splines consisting of Bézier curve segments (3.8b). Scalability is illustrated by increasing the control point dimension to 3, resulting in a 3-dimensional Bézier curve.

In the context of regression-based deep learning approaches to probabilistic sequence modeling, a model based on Bézier curves is expected to have a positive impact on the training and inference process. Due to the modeled mean sequence being constraint by an underlying parametric curve and the omission of an iterative generation approach, the generation of outliers within the sequence can be avoided. This, in turn, reduces the effect of error propagation present in iterative approaches under the presence of outliers.

3.1.2 A potential caveat: Non-linear Covariance Blending

When combining control points into curve points, the control points are weighted using the Bernstein basis polynomials (see Equations (3.4) and (3.5)). While the control point mean vectors are linearly interpolated when calculating a curve point mean vector, non-linear weighting is introduced for the covariance matrices in Equation (3.5), due to the control points being Gaussian random variables. This, in turn, leads to an effect that prevents the $\mathcal{N}$-Curve model from maintaining a constant variance along the curve. Instead it is scaled down for curve points with $0 < t < 1$, resulting in a *squeezing effect*.

This effect is easiest to see, taking an $\mathcal{N}$-Curve with 2 control points, resembling a straight line, as an example. Setting the variance of both control points to 1, the variance of intermediate points is parabolic because of the non-linear covariance weighting. This is illustrated in Figure 3.9.

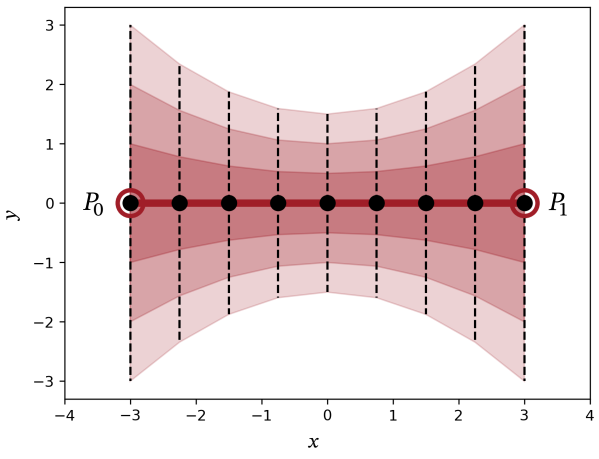
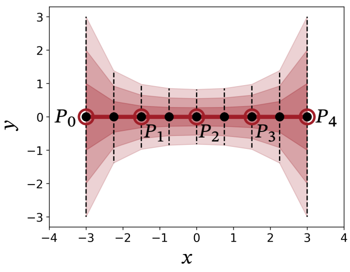


Figure 3.9: Illustration of non-linear variance interpolation using a simple 1-dimensional $\mathcal{N}$-Curve with 2 control points. Both control points P_0 and P_1 have a variance of 1. The shaded region around the curve depicts 1, 2 and 3 times the variance for each curve point. It can be seen, that the evolution of the variance along the curve is parabolic.

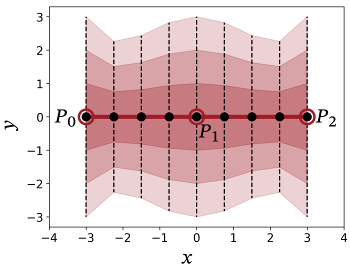
Recalling Equation (3.5), in $\Sigma^{\psi}(t) = \sum_{i=0}^{N} \left(b_{i,N}(t)\right)^2 \mathbf{\Sigma}_i$, the Bernstein coefficients need to be squared when blending the covariance matrices, due to

$$\begin{aligned} \mathrm{cov}[aX] &= \mathbb{E}[(aX - \mathbb{E}[aX])(aX - \mathbb{E}[aX])^T] \\ &= a^2 \cdot \mathrm{cov}[X]. \end{aligned} \tag{3.13}$$

Following this and $b_{i,N}(t) < 1$ for $0 < t < 1$, the normalization property $\sum_{i=0}^{N} \left(b_{i,N}(t)\right)^2 < 1$ does not hold. Obvious attempts to mitigate this effect involve the addition of intermediate control points or an adjustment of intermediate control point variances. First, adding intermediate control points with constant variance does only amplify the squeezing effect, due to the increasing number of weights being involved, leading to $\sum_{i=0}^{M} \left(b_{i,M}(t)\right)^2 < \sum_{i=0}^{N} \left(b_{i,N}(t)\right)^2$ for $M > N$. Second, adjusting intermediate control point variances can only mitigate the squeezing effect for selected curve points, making it at least viable for discrete-time stochastic processes in theory. The effect of both approaches on the variance along the curve is depicted in Figure 3.10.



(a) Exemplary 1-dimensional $\mathcal{N}$-Curve using 3 intermediate control points with variance 1.



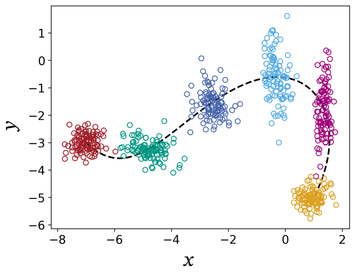
(b) Exemplary 1-dimensional $\mathcal{N}$-Curve using one intermediate control point with increased variance.

Figure 3.10: Illustration of different approaches trying to mitigate the squeezing effect in $\mathcal{N}$-Curves due to non-linear variance blending. In both subfigures, a simple 1-dimensional $\mathcal{N}$-Curve is depicted. Respective first and last control points have a variance of 1. The shaded region around the curve depicts 1, 2 and 3 times the variance for each curve point. It can be seen, while adding multiple intermediate control points with the same variance amplify the squeezing effect, increasing intermediate control point variances can help mitigate the effect.

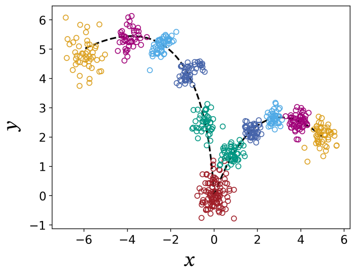
Although, in theory, this effect seems like a major drawback of the $\mathcal{N}$-Curve model, especially in the continuous-time case, it is less relevant in practice, due to the prevalence of discrete sequence data. In addition, real-world data is commonly suspect to noise, which makes the constant variance case discussed in this subsection less likely to appear. In order to provide more insight into this effect and its impact in the context of sequence modeling, it is discussed further in the context of experiments conducted on real-world data in Section 5.1.5.4.

3.1.3 The $\mathcal{N}$-Curve Model as a Generative Model

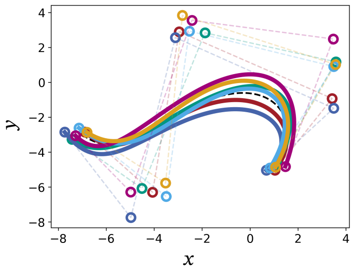
For modeling a stochastic process, the $\mathcal{N}$-Curve model provides a Gaussian probability distribution for each point in time. At the same time, it provides a probability distribution over parametric curves. Following this, the $\mathcal{N}$-Curve model can be used as a generative model to either generate samples at specific points in time or to generate (continuous) realizations of the stochastic process itself. The latter can be achieved by sampling a set of Bézier curve control points from an $\mathcal{N}$-Curve ψ, or an $\mathcal{N}$-Curve mixture Ψ, respectively. In the case of $\mathcal{N}$-Curve mixtures, a specific $\mathcal{N}$-Curve to draw a sample from is randomly selected according to the weight distribution π in a first step. A set of samples drawn from an $\mathcal{N}$-Curve and a mixture of $\mathcal{N}$-Curves is depicted in Figure 3.11.



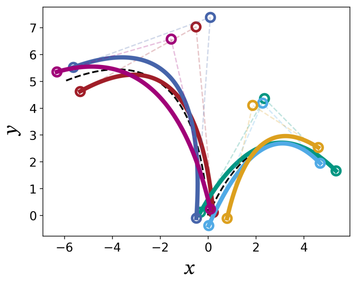
(a) Samples for X_t along an $\mathcal{N}$-Curve for $t \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.



(b) Samples for X_t along an $\mathcal{N}$-Curve mixture for $t \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.



(c) Bézier curves sampled from an $\mathcal{N}$-Curve.



(d) Bézier curves sampled from an $\mathcal{N}$-Curve mixture.

Figure 3.11: Illustration of the $\mathcal{N}$-Curve model as a generative model for generating data for specific points in time along the curve ((a) and (b)) and for generating full Bézier curves according to the $\mathcal{N}$-Curve (mixture) control points ((c) and (d)).

3.1.4 Connection to Gaussian Processes

Generally speaking, Gaussian processes are a form of stochastic processes, where the joint distribution of all stochastic variables $\{X_t\}_{t \in T}$ is a multivariate Gaussian distribution. The joint distribution is obtained using an explicit mean function and covariance function, commonly referred to as the kernel of the Gaussian process (see also Section 2.2.1). Due to the joint distribution being Gaussian, each individual stochastic variable, either obtained through marginalization or conditioning, is again Gaussian. Following this, a fundamental similarity between the $\mathcal{N}$-Curve model and Gaussian processes can be observed, in that the $\mathcal{N}$-Curve model provides a model for a stochastic process $\{X_t\}_{t\in T}$ comprised of Gaussian random variables $X_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Thus, the question arises, if the underlying $\mathcal{N}$-Curves are a special case of Gaussian processes using an implicit covariance function.

Following the definition of Gaussian processes [Mac03, Ras06], an $\mathcal{N}$-Curve would be classified as a Gaussian process, if for any finite subset $\{t_1, ..., t_k\}$ of T, the joint probability density $p(X_{t_1}, ..., X_{t_k})$ of corresponding random variables is Gaussian. This property is referred to as the *GP property* in the following and can be shown to hold true for $\mathcal{N}$-Curves, as these are, in fact, an alternative formulation for Gaussian processes with specific mean and covariance functions.

In order to prove the GP property holding true for $\mathcal{N}$-Curves, first recall, that an $\mathcal{N}$-Curve ψ is defined in terms of a set $\mathcal{P}_\mathcal{N}$ of N_{deg} *independent* d-dimensional Gaussian control points $P_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, which are defined as column vectors, i.e.

$$P_i = \begin{pmatrix} P_1^i \\ \vdots \\ P_d^i \end{pmatrix}. \tag{3.14}$$

Using these control points, a sequence of Gaussian probability distributions along the corresponding $\mathcal{N}$-Curve has been defined (see Equation (3.3)). As an alternative to this approach, the control points can also be stacked into the $((N_{\text{deg}} + 1) \cdot d \times 1)$ control point random vector

$$\mathbf{P} = \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_{N_{\text{deg}}} \end{pmatrix}, \tag{3.15}$$

consisting of independent Gaussian random variables, which is itself jointly Gaussian. Further, a $(N \cdot d \times (N_{\text{deg}} + 1) \cdot d)$ transformation matrix

$$\mathbf{C} = \begin{pmatrix} \mathbf{B}_{0,N_{\text{deg}}}(t_1) & \dots & \mathbf{B}_{N_{\text{deg}},N_{\text{deg}}}(t_1) \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{0,N_{\text{deg}}}(t_N) & \dots & \mathbf{B}_{N_{\text{deg}},N_{\text{deg}}}(t_N) \end{pmatrix}, \tag{3.16}$$

with

$$\mathbf{B}_{i,N_{\text{deg}}}(t) = b_{i,N_{\text{deg}}}(t)\mathbf{I}_d, \tag{3.17}$$

where $\mathbf{I}_d$ is the d-dimensional identity matrix, can be derived using the Bernstein polynomials $b_{i,N_{\text{deg}}}(t_i)$ with $t_i \in T_N$ (see Equation (3.7) for T_N), in order to map the control point random vector $\mathbf{P}$ onto a random vector consisting of N d-dimensional Gaussian curve points, i.e.

$$\mathbf{X} = \mathbf{C} \cdot \mathbf{P} = \begin{pmatrix} \mathbf{B}_{0,N_{\text{deg}}}(t_1) \cdot P_0 + \cdots + \mathbf{B}_{N_{\text{deg}},N_{\text{deg}}}(t_1) \cdot P_{N_{\text{deg}}} \\ \vdots \\ \mathbf{B}_{0,N_{\text{deg}}}(t_N) \cdot P_0 + \cdots + \mathbf{B}_{N_{\text{deg}},N_{\text{deg}}}(t_N) \cdot P_{N_{\text{deg}}} \end{pmatrix} = \begin{pmatrix} X_1 \\ \vdots \\ X_N \end{pmatrix}. \tag{3.18}$$

As $\mathbf{X}$ is obtained through a linear transformation of a Gaussian random vector, it is jointly Gaussian as well. As a consequence, the corresponding probability density function $p(\mathbf{X}) = p(X_1, ..., X_N)$ is a Gaussian probability density, thus the GP property holds.

Next, the mean function and Gaussian process kernel induced by a given $\mathcal{N}$-Curve will be derived. For simplicity, only the 1-dimensional case is regarded, which is also the common use case of Gaussian processes. Following this, the control points P_i are defined by the mean value μ_i and variance σ_i^2. While the mean function is equal to that of the $\mathcal{N}$-Curve itself (see Equation (3.4)), the kernel $k_{\mathcal{P}_{\mathcal{N}}}(t,t')$ for two curve points $X = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t)P_i$ and $Y = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t')P_i$ at indices t and t' with $t,t' \in [0,1]$, and respective mean values $\mu_X = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t)\mu_i$ and $\mu_Y = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t')\mu_i$ is defined as follows:

$$
\begin{aligned}
k_{\mathcal{P}_{\mathcal{N}}}(t_i,t_j) &= \mathbb{E}[(X-\mu_X)(Y-\mu_Y)] \\
&= \mathbb{E}\left[\left(\sum_{k=0}^{n} b_{k,n}(t_i)P_k - \mu_X\right)\left(\sum_{k=0}^{n} b_{k,n}(t_j)P_k - \mu_Y\right)\right] \\
&= \mu_X\mu_Y + \mathbb{E}\left[\sum_{k=0}^{n}\left(\sum_{k'=0}^{n} b_{k,n}(t_i)b_{k',n}(t_j)P_kP_{k'}\right)\right] \\
&\quad - \mathbb{E}\left[\sum_{k=0}^{n} b_{k,n}(t_i)\mu_Y P_k\right] - \mathbb{E}\left[\sum_{k=0}^{n} b_{k,n}(t_j)\mu_X P_k\right] \\
&= \mu_X\mu_Y + \mathbb{E}\left[\sum_{k=0}^{n} b_{k,n}(t_i)b_{k,n}(t_j)P_k^2\right] \\
&\quad + \mathbb{E}\left[\sum_{k=0}^{n}\left(\sum_{k'=0,k'\neq k}^{n} b_{k,n}(t_i)b_{k',n}(t_j)P_kP_{k'}\right)\right] \\
&\quad - \mu_Y\sum_{k=0}^{n} b_{k,n}(t_i)\mu_k - \mu_X\sum_{k=0}^{n} b_{k,n}(t_j)\mu_k.
\end{aligned}
$$

By applying $\mathbb{E}[P_i \cdot P_j] = \mathbb{E}[P_i] \cdot \mathbb{E}[P_j]$, which follows from the independence of the control points, and $\mathbb{E}[P_i^2] = \mathrm{Var}[P_i] + (\mathbb{E}[P_i])^2$, follows the closed-form solution

$$
\begin{aligned}
k_{\mathcal{P}_{\mathcal{N}}}(t_i,t_j) &= \mu_X\mu_Y + \sum_{k=0}^{n} b_{k,n}(t_i)b_{k,n}(t_j)(\sigma_k^2 + \mu_k^2) \\
&\quad + \sum_{k=0}^{n}\left(\sum_{k'=0,k'\neq k}^{n} b_{k,n}(t_i)b_{k',n}(t_j)\mu_k\mu_{k'}\right) \\
&\quad - \mu_Y\sum_{k=0}^{n} b_{k,n}(t_i)\mu_k - \mu_X\sum_{k=0}^{n} b_{k,n}(t_j)\mu_k.
\end{aligned}
\tag{3.19}
$$

It can be noted, that the diagonal elements of a covariance matrix obtained by $k_{\mathcal{P}_{\mathcal{N}}}(t,t')$ correspond to the interpolated covariances of the given $\mathcal{N}$-Curve as defined in Equation (3.5).

Now, with the $\mathcal{N}$-Curve model covering a specific subset of Gaussian processes, the main commonalities and differences between both formulations will be discussed briefly. Although both, Gaussian processes and the $\mathcal{N}$-Curve model target a distribution over functions, or rather parametric curves in the case of $\mathcal{N}$-Curves, there is a key differences worth mentioning, in that both approaches provide a different perspective on the task of distribution modeling. While Gaussian processes pursue a bottom-up approach, especially in Gaussian process regression [Ras06], $\mathcal{N}$-Curves provide a top-down approach. As such, in Gaussian process regression, the relation between Gaussian "curve points" is modeled explicitly using the covariance function. Then treating these curve points as part of a partitioned joint distribution ensures the GP property. In the $\mathcal{N}$-Curve model, the distribution over functions is achieved by modeling the curve-defining control points stochastically, which dictate the relation between curve points implicitly. Thereby, the GP property follows from the correlation between curve points, which emerges from geometric constraints given by the underlying Bézier curve, i.e. the curve points being linear transformations of the same set of stochastic control points.

In order to conclude this short section on the connection between $\mathcal{N}$-Curves and Gaussian processes, a few illustrations are given, which compare commonly used Gaussian process kernels with different $\mathcal{N}$-Curve kernels. After that, a simple toy example, depicting the calculation of the posterior distribution given a few observation of a target function, is provided. For these examples, zero mean Gaussian processes are considered only.

Figure 3.12 illustrates a *radial basis function* (abbrev.: RBF, [Gör19]) kernel

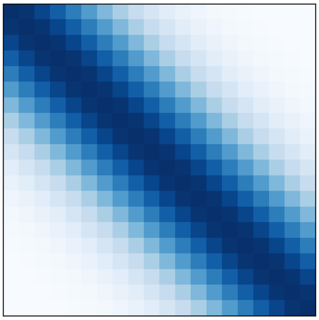
$$k^{\mathrm{rbf}}_{\sigma,l}(t,t') = \sigma^2 \exp\left(-\frac{\|t-t'\|^2}{2l^2}\right), \tag{3.20}$$

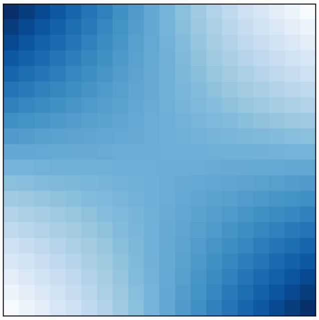
with $\sigma = 1$ and $l = 0.25$, a linear kernel [Gör19]

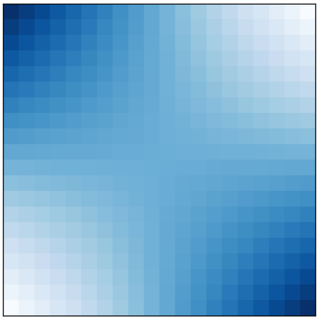
$$k^{\mathrm{lin}}_{\sigma,\sigma_b,c}(t,t') = \sigma_b^2 + \sigma^2(t-c)(t'-c), \tag{3.21}$$

with $\sigma = \sigma_b = c = 0.5$, and two $\mathcal{N}$-Curve kernels $k_{\mathcal{P}_1}(t,t')$ and $k_{\mathcal{P}_2}(t,t')$. $\mathcal{P}_1$ consists of two unit Gaussians, i.e. $\mathcal{N}(0,1)$, and $\mathcal{P}_2$ consists of 9 zero mean

Gaussian control points with standard deviations $\sigma_0 = \sigma_8 = 1$, $\sigma_1 = \sigma_7 = 1.25$, $\sigma_2 = \sigma_6 = 1.5$, $\sigma_3 = \sigma_5 = 1.75$ and $\sigma_4 = 2$. The standard deviation increases towards the center of the control point set, in order to cope with non-linear blending (see also Section 3.1.2). For each kernel, the covariance matrix has been calculated for 20 equally spaced values ranging from 0 to 1.


(a) RBF kernel $k^{\text{rbf}}_{\sigma=1,l=0.25}(t,t')$.


(b) Linear kernel $k^{\text{lin}}_{\sigma=0.5,\sigma_b=0.5,c=0.5}(t,t')$.


(c) $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_1}(t,t')$.

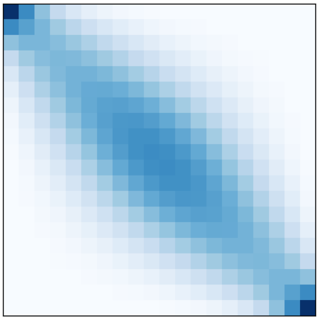
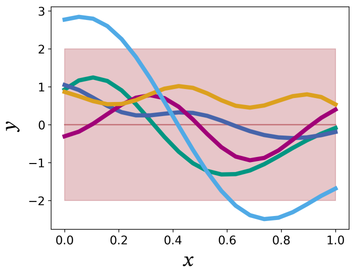

(d) $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_2}(t,t')$.

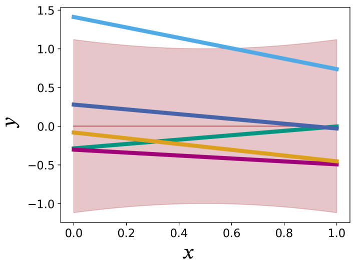
Figure 3.12: Covariance matrices for 20 equally spaced values in $[0,1]$ obtained by using different Gaussian process kernels.

When comparing the covariance matrices in figures (b) and (c), it can be seen, that the results from the kernel based on a linear $\mathcal{N}$-Curve with unit Gaussian control points look similar to those based on the given linear kernel. In fact, the covariance matrix calculated with $k_{\mathcal{P}_1}$ is equale to the covariance matrix calculated with $k^{\mathrm{lin}}_{\sigma=0.5,\sigma_b=0.5,c=0.5}(t,t')$ when normalizing its values to $[0,1]$. On the other hand, the covariance matrix obtained with $k_{\mathcal{P}_2}$ (figure (d)), which is derived from a more complex $\mathcal{N}$-Curve, tends to be more comparable to the covariance matrix calculated with $k^{\mathrm{rbf}}_{\sigma=1,l=0.25}(t,t')$ (figure (a)).

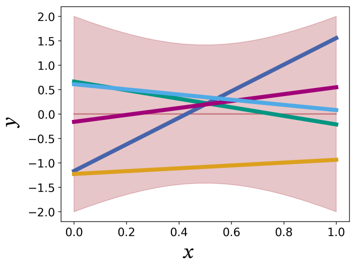
In combination with a mean vector, $\mathbf{0}$ in this case, each covariance matrix defines a prior distribution for a Gaussian process. Following this, Figure 3.13 depicts sample functions drawn from each prior distribution, again showing the parallels between the kernels.



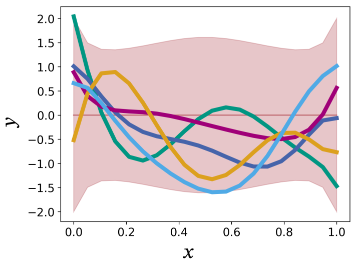
(a) RBF kernel $k^{\mathrm{rbf}}_{\sigma=1,l=0.25}(t,t')$.



(b) Linear kernel $k^{\mathrm{lin}}_{\sigma=0.5,\sigma_b=0.5,c=0.5}(t,t')$.



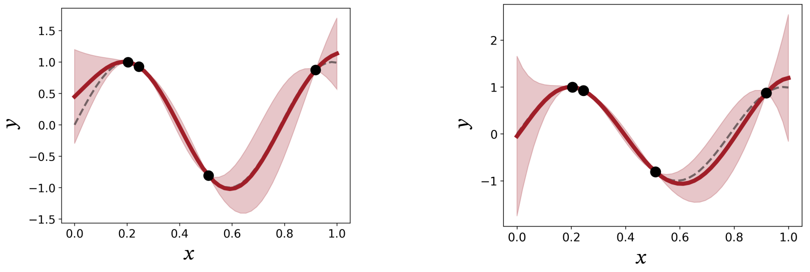
(c) $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_1}(t,t')$.



(d) $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_2}(t,t')$.

Figure 3.13: Samples drawn from prior distributions using different Gaussian process kernels. The 2σ region around the mean value is depicted as a red shaded area.

Finally, the Gaussian processes defined by the RBF kernel $k^{\text{rbf}}_{\sigma=1,l=0.25}(t,t')$ and the $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_2}$ are used to approximate $f(x) = \sin(8x)$ on $[0,1]$ using 4 observed data points. Using these data points, the posterior distribution of each Gaussian process can be calculated, which ideally tends to fit the targeted function with an increasing number of observed data points. The posterior distributions for both Gaussian processes are depicted in figure 3.14.



(a) RBF kernel $k^{\text{rbf}}_{\sigma=1,l=0.25}(t,t')$. **(b)** $\mathcal{N}$-Curve kernel $k_{\mathcal{P}_2}(t,t')$.

Figure 3.14: Posterior distributions of Gaussian processes obtained by using different kernels given 4 data points (circular markers) of a sine function (dashed line). The 2σ region around the mean value is depicted as a red shaded area.

3.2 Modeling Infinite Stochastic Processes

The $\mathcal{N}$-Curve model as presented in the previous Section 3.1 is viable for most real-world applications, which are generally concerned with sequences of fixed or at least bounded length. Thus, using Bézier curves as a basis, the representable curve complexity suffices the requirements of given sequential data. However, apart from the practicality of the $\mathcal{N}$-Curve model, there exists a conceptual limitation for modeling continuous-time stochastic processes $\{X_t\}_{t\in T}$. This limitation is given by the bounded index set $T = [0,1]$, which is imposed by the use of a Bézier curve basis. Because of this, infinite continuous-time stochastic processes, i.e. with $T = \mathbb{R}^+_0$, cannot be represented by the $\mathcal{N}$-Curve model. Further, infinite discrete-time stochastic processes $\{X_t\}_{t\in T_N}$, which model open-ended sequences and are realized using a discrete subset T_N of T, are also affected by this limitation in a more subtle way. Although the number of control points of an $\mathcal{N}$-Curve is fixed, it is still possible to extract an infinite number of curve points with infinitesimal distance between subsequent curve points. However, as a sequence becomes longer, it generally also expands in space. Thus a potentially more complex underlying Bézier curve, i.e. a curve of higher degree, is required for achieving an accurate approximation. While there is no theoretical limit to the number of control points defining a Bézier curve, the approximation quality may suffer from an increasing number of control points in practice. This is due to the increased number of concurring control points, each contributing to every curve point (*global control*).

In the context of parametric curves, a common approach to tackle increasing curve complexity in terms of length and shape is the use of segmented curves. Here, simpler curves of fixed degree are stitched together in order to form a more complex curve, granting *local control* over curve segments. Thus, the number of segments can be increased as required, without affecting the entire curve. In the context of Bézier curves, such a curve is then called a *composite Bézier curve* or *Bézier spline* [Reb21]. Following this, a Bézier spline of degree N_{deg} is defined in terms of a sequence of N_c Bézier curve segments $\mathcal{P} = \{\mathscr{P}_1, ..., \mathscr{P}_{N_c}\}$, where each segment is defined by its own set of control points $\mathscr{P}_i = \{P_0^i, ..., P_{N_{\text{deg}}}^i\}$. Further, at least C^0 continuity, i.e. $P_{N_{\text{deg}}}^i = P_0^{i+1}$, holds for subsequent Bézier curve segments.

If necessary, additional smoothness requirements, e.g. C^1 or C^2 continuity, can be added. Under C^1 continuity, subsequent curve segments have identical tangents at the control point joining both segments. C^2 continuous curves additionally have identical curvature at this point [Bar95]. Under C^0 continuity, Bézier splines grant local control, i.e. the curve can be altered on a per segment basis without affecting other segments. This flexibility is restricted when enforcing C^1 or C^2 continuity, as neighboring control points of subsequent curve segments become dependent on one another. Geometrically, C^1 continuity can be enforced by making the second last control point

$P^i_{N_{\text{deg}}-1}$ of a curve segment and the second control point P^{i+1}_1 of the subsequent curve segment collinear. For C^2 continuity, these control points additionally have to have the same (euclidean) distance from the joining control point $P^i_{N_{\text{deg}}} = P^{i+1}_0$. Following this, local control can be granted to some degree by using more than 4 control points in a segment. Examples for C^0, C^1 and C^2 continuous segment intersections are given in Figure 3.15.

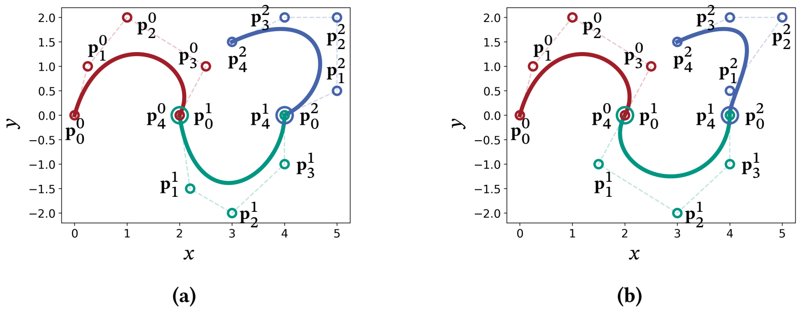


Figure 3.15: Examples for Bézier splines consisting of three segments with varying continuity constraints at segment intersections. Figure (a) depicts a C^0 continuous Bézier spline and (b) depicts a Bézer spline meeting C^2 continuity at the intersection of the first two segments and C^1 continuity at the intersection of the second and third segment.

3.2.1 The Meta-time $\mathcal{N}$-Curve Model

Given the aforementioned conceptual limitations, the goal is to extend the $\mathcal{N}$-Curve model for infinite stochastic processes and open-ended sequences, i.e. a stochastic process with index[1] $t \in \mathbb{R}^+_0$. For this purpose, the concept of splines is incorporated into the model, thus combining $\mathcal{N}$-Curve segments into a more complex probabilistic spline. In order to model an indefinite number of curve segments, a notion of control point evolution is introduced, by defining the set of Gaussian control points as a function $\mathcal{P}_{\mathcal{N}}(t)$ of time t. Due

[1] The index t will again be interpreted as *time* for a more intuitive derivation of this extension.

to each function value then defining an entire curve segment covering multiple time steps, a (potentially) asynchronous timeline emerges. This timeline is denoted as *meta-time* with index $\tilde{t}$ in the following. Subsequent values of $\tilde{t}$ then yield subsequent $\mathcal{N}$-Curve segments on the probabilistic spline modeling the stochastic process. This probabilistic spline will be denoted as a *meta-time $\mathcal{N}$-Curve* in the following. With the control point function representing a sequence of Gaussian control point sets, the meta-time $\mathcal{N}$-Curve is defined as a sequence of connected $\mathcal{N}$-Curve segments $\psi = \{\{P_0^{\tilde{t}}, ..., P_{N_{\text{deg}}}^{\tilde{t}}\}\}_{\tilde{t}}$, indexed by meta-time $\tilde{t} \in \mathbb{N}_0$. For associating a point in time t with the corresponding $\mathcal{N}$-Curve segment, the *meta-time mapping* $m : t \rightarrow \tilde{t}$ onto the meta-timeline is introduced. In addition, with t (and $\tilde{t}$) now exceeding the index range of an $\mathcal{N}$-Curve, another mapping $m_c : t \rightarrow t_c$ onto the *curve-time* parameter $t_c \in [0, 1]$ is introduced in order to access the exact curve point on a curve segment. This mapping is denoted as the *curve-time mapping*. Note, that m_c is technically defined as $m_c : (t, \tilde{t}) \rightarrow t_c$, but as $\tilde{t}$ is derived from t through m, the additional parameter can be omitted. Finally, in the context of meta-time $\mathcal{N}$-Curves, C^0 continuity is given by matching the mean vectors and covariance matrices at the intersection of subsequent $\mathcal{N}$-Curve segments. Aforementioned geometric restrictions for C^1 and C^2 continuity only apply to control point mean values. Figure 3.16 gives an illustration of the different timelines and the basic idea of the model extension.

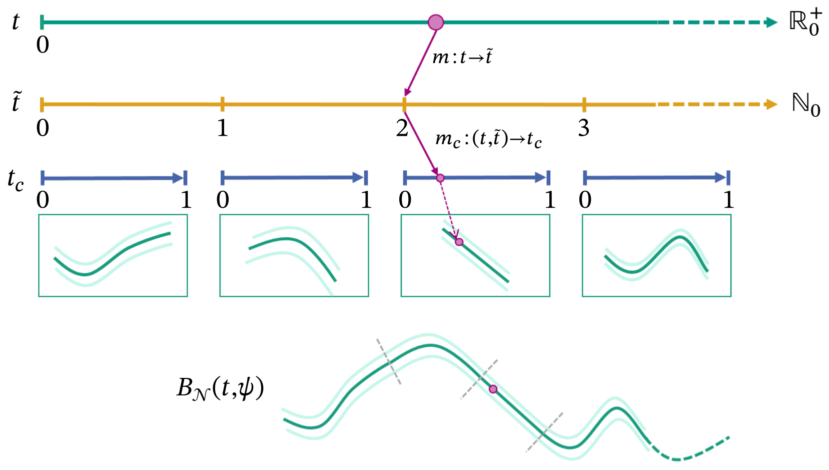


Figure 3.16: Illustration of the proposed extension of the $\mathcal{N}$-Curve model, basing stochastic process modeling on probabilistic Bézier splines instead of single Bézier curves, thus allowing to model infinite stochastic processes. Interpreting the stochastic process index t as time, multiple connected timelines emerge, namely meta-time $\tilde{t}$ and curve-time t_c. Given a specific point in time t, the corresponding $\mathcal{N}$-Curve segment is determined by the mapping $m(t)$ and the specific point on the segment by $m_c(t)$. An exemplary resulting probabilistic spline $B_\mathcal{N}(t, \psi)$ is depicted at the bottom.

With the introduced timeline mappings, the original formulation of the $\mathcal{N}$-Curve model can be extended into the *meta-time $\mathcal{N}$-Curve model* as depicted in Table 3.1. Here, only the definition of the extended $\mathcal{N}$-Curve is provided, as the derivation of other formulas building on the curve definition, e.g. the curve point probability density function (Equation (3.6)), is not directly affected by these changes. Further, exemplary definitions are provided for both the meta-time and the curve-time mapping. Potential definitions for these mappings are discussed in Section 3.2.2.

Table 3.1: Overview of changes made to the $\mathcal{N}$-Curve model in order to derive the meta-time $\mathcal{N}$-Curve model extension. For completeness, examples for the meta-time and curve-time mappings are provided.

Targeted stochastic process	
$\mathcal{N}$-Curve	$\mathcal{G}_T = \{X_t\}_{t\in[0,1]}$
Meta-time $\mathcal{N}$-Curve	$\mathcal{G}_T = \{X_t\}_{t\in\mathbb{R}_0^+}$
Curve (segment) definition	
$\mathcal{N}$-Curve	$\psi = \{P_0, ..., P_{N_{\text{deg}}}\}$
Meta-time $\mathcal{N}$-Curve	$\psi = \{\{P_0^{\tilde{t}}, ..., P_{N_{\text{deg}}}^{\tilde{t}}\}\}_{\tilde{t}\in\mathbb{N}_0}, \psi(\tilde{t}) = \{P_0^{\tilde{t}}, ..., P_{N_{\text{deg}}}^{\tilde{t}}\}$
Curve points (see Equation (3.3))	
$\mathcal{N}$-Curve	$B_{\mathcal{N}}(t, \psi) = \left(\mu^{\psi}(t), \Sigma^{\psi}(t)\right)$
Meta-time $\mathcal{N}$-Curve	$B_{\mathcal{N}}(t, \psi) = \left(\mu^{\psi(m(t))}(m_c(t)), \Sigma^{\psi(m(t))}(m_c(t))\right)$
Mean function (see Equation (3.4))	
$\mathcal{N}$-Curve	$\mu^{\psi}(t) = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t)\boldsymbol{\mu}_i$
Meta-time $\mathcal{N}$-Curve	$\mu^{\psi(\tilde{t})}(t_c) = \sum_{i=0}^{N_{\text{deg}}} b_{i,N_{\text{deg}}}(t_c)\boldsymbol{\mu}_i, \boldsymbol{\mu}_i \in \psi(\tilde{t})$
Covariance function (see Equation (3.5))	
$\mathcal{N}$-Curve	$\Sigma^{\psi}(t) = \sum_{i=0}^{N_{\text{deg}}}(b_{i,N_{\text{deg}}}(t))^2\boldsymbol{\Sigma}_i$
Meta-time $\mathcal{N}$-Curve	$\Sigma^{\psi(\tilde{t})}(t_c) = \sum_{i=0}^{N_{\text{deg}}}(b_{i,N_{\text{deg}}}(t_c))^2\boldsymbol{\Sigma}_i, \boldsymbol{\Sigma}_i \in \psi(\tilde{t})$
Meta-time mapping	
$\mathcal{N}$-Curve	-
Meta-time $\mathcal{N}$-Curve	$\tilde{t} = m(t) = \left\lfloor \frac{t}{a} \right\rfloor$
Curve-time mapping	
$\mathcal{N}$-Curve	-
Meta-time $\mathcal{N}$-Curve	$t_c = m_c(t) = t - m(t)$

On a final note, the meta-time $\mathcal{N}$-Curve model can be used in the context of multi-modal stochastic processes by following the same approach as described for the original $\mathcal{N}$-Curve model using a mixture of meta-time $\mathcal{N}$-Curves. In this case, each mode of a stochastic process representation follows a separate meta-time $\mathcal{N}$-Curve. The mixture weight distribution is defined on a per segment basis, i.e. $\pi(\tilde{t}) = \{\pi_1^{\tilde{t}}, ...\pi_K^{\tilde{t}}\}$ is given at meta-time $\tilde{t}$. Following this, a potential benefit of meta-time $\mathcal{N}$-Curves is given by the fact, that it is possible to alter the curve weights in each meta-time step. This allows for more control about the number of required mixture components on a per-segment basis.

3.2.2 Mapping Functions

In the context of the meta-time $\mathcal{N}$-Curve model, two mappings have been defined, namely the meta-time mapping and the curve-time mapping, connecting the introduced timelines. As there possibly exists a wide range of options for defining either mapping, this section provides exemplary mapping definitions, which are expected to be relevant for an actual implementation of the model in the context of different sequence modeling tasks.

Meta-time mapping: Moving along the meta-timeline yields a sequence of $\mathcal{N}$-Curve segments along a probabilistic spline. In this context, the meta-time mapping maps a given point in time t onto the meta-timeline, i.e. a natural number $\tilde{t} \in \mathbb{N}_0$, in order to determine the corresponding $\mathcal{N}$-Curve segment. Following this, the goal is to define a consistent meta-time mapping from t onto the set of natural numbers.

The first possible definition of this mapping is given by a *fixed interval mapping*

$$m_a(t) = \left\lfloor \frac{t}{a} \right\rfloor. \tag{3.22}$$

In this case, the meta-time $\tilde{t}$ is advanced at a fixed rate as t increases, thus traversing an infinite sequence of (distinct) $\mathcal{N}$-Curve segments. While this definition may result in premature segment changes, an interesting special case is given for $a = 1$, where the resulting meta-time $\mathcal{N}$-Curve resembles a probabilistic spline with segments connected at their endpoints. Besides resulting in a well-defined segmented probabilistic curve, this further allows a straightforward definition of C^1 and C^2 continuous segment intersections via control point placement.

In cases, where periodic repetitions are expected in sequential data, another definition is given by a *periodic mapping*

$$m_{a,b,p}(t) = \left\lfloor \frac{2 \cdot a}{\pi} \arcsin\left(\sin\left(\frac{2 \cdot \pi}{p} \cdot t\right)\right) + b \right\rfloor, \tag{3.23}$$

which can be based for example on a triangle wave. Here, t is mapped onto the same sequence of $\mathcal{N}$-Curve segments periodically, thus repeating the same segment sequence over and over. The repetition frequency is controlled by p. An alternative to the periodic mapping is given by a *modulo reset mapping*

$$m_{a,k}(t) = \left\lfloor \frac{t}{a} \right\rfloor \mod k \tag{3.24}$$

with $k \geq 1$, which repeats the same segment sequence after every k meta-time steps. This mapping is basically built on a sawtooth wave. Similar to the fixed interval mapping, these two definitions can be parameterized to yield an endpoint-connected probabilistic spline.

Curve-time mapping: After determining the current $\mathcal{N}$-Curve segment at time t, the current position within this segment needs to be determined. For this, a curve-time mapping needs to be defined, mapping t onto curve-time $t_c \in$ [0,1]. As this mapping is highly dependent on the specific definition of the meta-time mapping m, an exemplary curve-time mapping compliant with the given variants of m, which result in an endpoint-connected spline, is provided. All these variants map a given time t in a way, that a segment intersection occurs whenever t is a multiple of 1. Following this, the difference between the value of t for the first and last segment point is exactly 1 and intermediate values are in $[0, 1]$. As such, the curve-time mapping can be defined as

$$m_c(t) = t - m(t). \tag{3.25}$$

Learned mapping: On a final note, it is also possible to learn both mappings in a constraint optimization setting. A potential benefit of this can be given by a resulting efficient re-use of few base segments, especially when the mapping can be conditioned on additional domain-specific input. On the downside, depending on the constraints defined during optimization, spline properties regarding C^1 and C^2 continuity might be lost.

3.2.3 Modeling discrete-time stochastic processes

As a last point, this section provides some insight into how the concept of meta-time $\mathcal{N}$-Curves can be applied for modeling discrete-time stochastic processes using different mapping variants. For this, only mapping variants resulting in an endpoint-connected probabilistic spline are considered, being expected to be the most relevant in an application context.

Recall in the $\mathcal{N}$-Curve model, a discrete index set $T_N \subset T$ is extracted from the continuous index set $T = [0,1]$ using a predetermined sequence length N (see also Section 3.1). Now, with $T = \mathbb{R}_0^+$ and sequences being unbounded in length, it is necessary to define the sequence length N_{seg} covered by each $\mathcal{N}$-Curve segment along a meta-time $\mathcal{N}$-Curve. Following this, the difference between subsequent stochastic process indices $t_i \in T_N$ and $t_{i+1} \in T_N$ is dictated by N_{seg} through $\Delta t = \frac{1}{N_{\text{seg}}-1}$. Using this formulation, $N = N_{\text{seg}}$ is a necessary condition in the original $\mathcal{N}$-Curve model. Opposed to that, in the meta-time $\mathcal{N}$-Curve model it is possible to have $N > N_{\text{seg}}$, which ultimately allows modeling open-ended sequences through generating a stream of $\mathcal{N}$-Curve segments. The discrete index set T_N can now be re-formulated as

$$T_N = \left\{ \sum_{i=0}^{v} \Delta t \,|\, v \in \{0, ..., N-1\} \right\}. \tag{3.26}$$

The meta-time mapping $m(t_i)$ is now required to fill the additional role of determining how many $\mathcal{N}$-Curve segments are required to model a sequence of length N given N_{seg}. In the case of the fixed interval mapping

$$m_{a=1}(t_i) = \lfloor t_i \rfloor,$$

the N^{th} element of a sequence lies on the $m_{a=1}(t_N)$'th $\mathcal{N}$-Curve segment. Thus, the meta-time $\mathcal{N}$-Curve model needs to generate

$$m_{a=1}(t_N) = \frac{N}{N_{\text{seg}}}$$

segments compliant with given continuity constrains. With $N \to \infty$, this results in an indefinite stream of distinct $\mathcal{N}$-Curve segments, connected at their endpoints. Looking at periodic mappings, e.g. the modulo reset mapping

$$m_{a=1,k}(t_i) = \lfloor t_i \rfloor \mod k,$$

the meta-time $\mathcal{N}$-Curve model needs to generate a maximum of k segments which are referenced in a loop via $m_{a=1,k}$. The maximum number of segments is required when $N > (k-1) \cdot N_{\text{seg}}$. For $N \to \infty$, this results in an indefinite repetition of the same k $\mathcal{N}$-Curve segments. Finally, in both cases, specific random variables along the meta-time $\mathcal{N}$-Curve are determined according to m_c and the corresponding $\mathcal{N}$-Curve segment.

3.3 Summary

The main contributions of this chapter are twofold. First, a probabilistic extension to Bézier curves ($\mathcal{N}$-Curves) was introduced, which models sequences of Gaussian probability distributions along a parametric curve. Thereby, an $\mathcal{N}$-Curve is defined in terms of stochastic control points. Further, it has been shown that $\mathcal{N}$-Curves are a special case of Gaussian processes. Second, a model building on mixtures of $\mathcal{N}$-Curves was presented, which enables the modeling of multi-modal stochastic processes. Using the $\mathcal{N}$-Curve model and its meta-time variant, finite and infinite, as well as discrete-time and continuous-time stochastic processes can be modeled.

4 Proposed Implementation

This chapter provides an implementation for the $\mathcal{N}$-Curve model (see Section 3.1), based on Mixture Density Networks (MDN). Therefore, Section 4.1 first extends on the introduction of Mixture Density Networks as given in Section 2.2.2 and then proceeds to define $\mathcal{N}$-Curve Mixture Density Networks (abbrev.: $\mathcal{N}$-MDN). The definition of the $\mathcal{N}$-MDN is accompanied with several toy examples, exploring the capabilities of the $\mathcal{N}$-Curve model.

In addition to the $\mathcal{N}$-MDN, this chapter provides a proof of concept for the conceptual extension of the $\mathcal{N}$-Curve model, given by the meta-time $\mathcal{N}$-Curve model as described in Section 3.2. As such, a recurrent extension of the $\mathcal{N}$-MDN is introduced and briefly evaluated on multiple toy examples in Section 4.2.

4.1 $\mathcal{N}$-Curve Mixture Density Networks

Defining a fully regression-based probabilistic sequence model is one of the main objectives pursued in this thesis. Following this, an MDN for learning the parameters of an $\mathcal{N}$-Curve mixture (see Section 3.1) from discrete sequence data is proposed. An MDN is a feed-forward neural network

$$\Phi(\mathbf{v}) = (\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k\in\{1,\ldots,K\}}|\mathbf{v}), \tag{4.1}$$

that takes an input vector $\mathbf{v}$ and maps it onto the parameters of a d-dimensional, K-component Gaussian mixture distribution. In order to ensure that the MDN generates a valid set of mixture parameters, the partitioned network output

$$\hat{y} = (\tilde{\pi}_1, ..., \tilde{\pi}_K, \tilde{\boldsymbol{\mu}}_1, ..., \tilde{\boldsymbol{\mu}}_K, \tilde{\boldsymbol{\sigma}}_1, ..., \tilde{\boldsymbol{\sigma}}_K, \tilde{\boldsymbol{\rho}}_1, ..., \tilde{\boldsymbol{\rho}}_K),$$

with

$$\tilde{\pi}_k \in \mathbb{R}, \tilde{\boldsymbol{\mu}}_k \in \mathbb{R}^d, \tilde{\boldsymbol{\sigma}}_k \in \mathbb{R}^d \text{ and } \tilde{\boldsymbol{\rho}}_k \in \mathbb{R}^{\frac{d^2-d}{2}}$$

is further transformed to meet parameter value requirements, i.e.

$$\pi_k = \text{softmax}(\tilde{\pi}_1, ..., \tilde{\pi}_K)_k,$$

such that $\sum_k \pi_k = 1$ and

$$\begin{aligned}
\boldsymbol{\mu}_k &= \tilde{\boldsymbol{\mu}}_k \\
\sigma_{k,i} &= f_\sigma(\tilde{\sigma}_{k,i}) > 0 \ \forall i \in \{1, ..., d\} \\
\rho_{k,j} &= f_\rho(\tilde{\rho}_{k,j}) \in [-1, 1] \ \forall j \in \left\{1, ..., \frac{d^2-d}{2}\right\}.
\end{aligned}$$

Note that the covariance matrices $\boldsymbol{\Sigma}_k$ are calculated from the standard deviations and correlations in order to ensure positive definiteness. For the transformations f_σ and f_ρ, there are several relevant options to consider. The original formulation [Bis94] employed

$$f_\sigma(x) = \exp(x) \text{ and } f_\rho(x) = \tanh(x) \tag{4.2}$$

to transform σ and ρ into respective value ranges. Both of these functions, however, can lead the MDN into having numerical issues during training. In the case of f_σ, the exponential function yields instable optimization results for large input values due to its exponential growth. To cope with this, a shifted version of the *Exponential Linear Unit* (abbrev.: *ELU*, [Cle15, Gui17])

$$f_\sigma(x) = \text{ELU}(1,x) + 1 \tag{4.3}$$

with

$$\mathrm{ELU}(\alpha,x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{4.4}$$

and the *softplus* function (also referred to as *SmoothReLU*, [Dug01, Glo11, Iso17])

$$f_\sigma(x) = \mathrm{softplus}(x) = \ln(1 + e^x) \tag{4.5}$$

are commonly used in MDNs. These functions are similar to the exponential function for negative and small positive input values, but transition into a linear function for larger input values. From an optimization point of view, the softplus function may be preferred over the ELU, as the latter is non-continuous in its derivatives [Sch20b]. Regarding the correlations ρ, using the tanh function for f_ρ can lead to vanishing gradients. Thus the *softsign* function [Glo10, Iso17]

$$f_\rho(x) = \mathrm{softsign}(x) = \frac{x}{1 + |x|} \tag{4.6}$$

may be used instead, despite having more complex derivatives [Sza21]. A schematic for an MDN generating a 2-dimensional Gaussian mixture distribution is depicted in Figure 4.1.

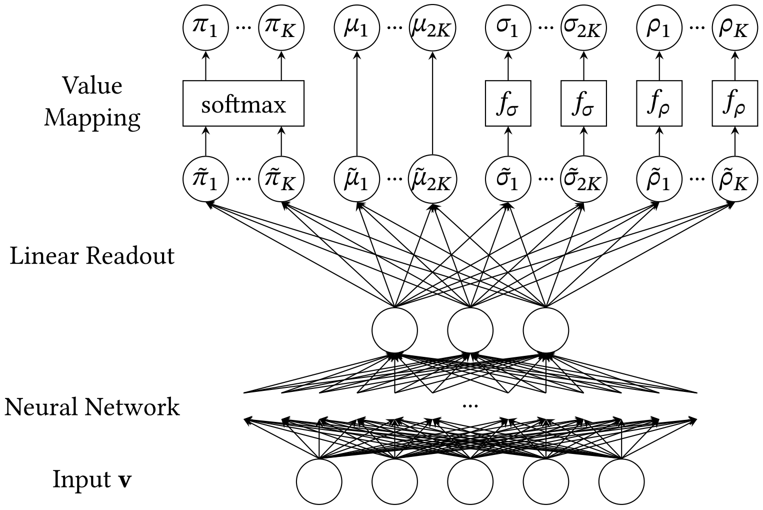

π_1 ··· π_K
μ_1 ··· μ_{2K}
σ_1 ··· σ_{2K}
ρ_1 ··· ρ_K
Value Mapping
softmax
f_σ
f_σ
f_ρ
f_ρ
$\tilde{\pi}_1$ ··· $\tilde{\pi}_K$
$\tilde{\mu}_1$ ··· $\tilde{\mu}_{2K}$
$\tilde{\sigma}_1$ ··· $\tilde{\sigma}_{2K}$
$\tilde{\rho}_1$ ··· $\tilde{\rho}_K$
Linear Readout
Neural Network
Input **v**

Figure 4.1: Schematic of a Mixture Density Network generating a 2-dimensional Gaussian mixture distribution. The outputs of a (feed-forward) neural network are linearly transformed and mapped onto respective parameters value ranges in order to determine the parameters, $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k\in\{1,...,K\}}$, of a Gaussian mixture distribution. The covariance matrices $\boldsymbol{\Sigma}_k$ are given in terms of the standard deviations $\sigma_{k,i}$ and the correlations $\rho_{k,j}$. For illustration purposes, the mean vector values $\mu_{k,1}$ and $\mu_{k,2}$, as well as the standard deviations $\sigma_{k,1}$ and $\sigma_{k,1}$ for each mixture component are not displayed separately.

Following this, Mixture Density Networks can be adapted easily to output the parameters of a K-component $\mathcal{N}$-Curve mixture with $\mathcal{N}$-Curves of degree N_{deg} by generating the parameters $\{\psi_k\}_{k\in\{1,...,K\}} = \{(\boldsymbol{\mu}^k_{\mathcal{P}}, \boldsymbol{\Sigma}^k_{\mathcal{P}})\}_{k\in\{1,...,K\}}$ for all $K \cdot (N_{\text{deg}} + 1)$ stochastic control points and the respective K curve weights $\{\pi_k\}_{k\in\{1,...,K\}}$. Advantages of using an MDN for learning the $\mathcal{N}$-Curve mixture parameters, rather than other algorithms (e.g. *Expectation Maximization* [Dem77]), are twofold. First, MDNs allow its output distribution to be conditioned on arbitrary inputs. Thus, the MDN provides an easy approach to learn and process conditional $\mathcal{N}$-Curve mixtures, allowing the model to be used in a conditional inference framework. Second, the MDN can be incorporated easily into (almost) any neural network architecture without the need to control the gradient flow. Besides that, there are two notable drawbacks of MDNs to consider, namely mode collapse due to overfitting and instabilities during training [Mak19]. In the context of MDNs, mode collapse commonly refers to a problem, where the MDN puts all weight on a single low-variance component, regardless of the number of available components. While instabilities during training can be mitigated by choosing appropriate functions for f_σ and f_ρ, mode collapse is expected to be reduced by setting MDNs in the context of $\mathcal{N}$-Curves, as modeling parametric curves instead of single points is expected to yield more distinct modes.

The most commonly used loss function for training an MDN is the *negative log-likelihood* [Bis94], which can also be adapted for training the $\mathcal{N}$-Curve Mixture Density Network from discrete sequence data. Let $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_1, ..., \mathcal{S}_M\}$ be a set of M realizations of a stochastic process with $\mathcal{S}_j = \{\mathbf{x}_1^j, ..., \mathbf{x}_N^j\}$ where each $\mathbf{x}_i^j$ with $i \in \{1, ..., N\}$ is a sample value for the respective random variable X_{t_i} at time i for $t_i \in T_N$ (see Section 3.1). In order to simplify the training procedure, Gaussian random variables along the $\mathcal{N}$-Curve are treated as if they were independent. Following this, the joint probability of the samples $\mathbf{x}_i^j$ in a sequence $\mathcal{S}_j$ along an $\mathcal{N}$-Curve ψ factorizes into the unnormalized Gaussian density

$$p^\psi(\mathcal{S}_j) = p^\psi(\mathbf{x}_1^j, ..., \mathbf{x}_N^j) = \prod_{i=1}^{N} p_{t_i}^\psi(\mathbf{x}_i^j). \tag{4.7}$$

This is exploited when defining the loss function. It should be noted, that this simplification can be justified, as the correlation between these Gaussian random variables is enforced implicitly by the underlying $\mathcal{N}$-Curve and thus by the stochastic control points, which are estimated during training.

For a single sequence $\mathcal{S}_j$ and an $\mathcal{N}$-Curve $\psi = \Phi(\mathbf{v})$, the loss function is then defined by the negative (unnormalized) log-likelihood

$$\begin{aligned}\mathcal{L} &= -\log\ p^{\psi}(\mathbf{x}_1^j, \ldots, \mathbf{x}_N^j) \\ &= -\log\left(\prod_{i=1}^{N} p_{t_i}^{\psi}(\mathbf{x}_i^j)\right) \\ &= -\sum_{i=1}^{N} \log\ p_{t_i}^{\psi}(\mathbf{x}_i^j) \\ &= -\sum_{i=1}^{N} \log\ p(\mathbf{x}_i^j | \mu^{\psi}(t_i), \Sigma^{\psi}(t_i))\end{aligned} \tag{4.8}$$

of the sequence given an input vector $\mathbf{v}$. Therefore, the loss for a set of M sequences $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$ is defined as

$$\mathcal{L} = \sum_{j=1}^{M}\left(-\sum_{i=1}^{N} \log\ p(x_i^j | \mu^{\psi}(t_i), \Sigma^{\psi}(t_i))\right). \tag{4.9}$$

Equation (4.9) can easily be extended for $\mathcal{N}$-Curve mixtures. Given an $\mathcal{N}$-Curve mixture $\Psi = \Phi(\mathbf{v})$, the likelihood of a single training sequence $\mathcal{S}_j$ is now calculated as the weighted linear combination of the likelihood of $\mathcal{S}_j$ for each ψ_k (see Equation (4.7)):

$$p^{\Psi}(\mathcal{S}_j) = \sum_{k=1}^{K} \pi_k p^{\psi_k}(\mathcal{S}_j). \tag{4.10}$$

Thus, the loss for a set $\mathcal{S}$ of M sequences can be defined as

$$\begin{aligned}\mathcal{L} &= \frac{1}{M}\sum_{j=1}^{M} -\log \sum_{k=1}^{K} \pi_k p^{\psi_k}(\mathcal{S}_j) \\ &= \frac{1}{M}\sum_{j=1}^{M} -\log \sum_{k=1}^{K} \pi_k \prod_{i=1}^{N} p_{t_i}^{\psi}(\mathbf{x}_i^j) \\ &= \frac{1}{M}\sum_{j=1}^{M} -\log \sum_{k=1}^{K} \exp\left(\log \pi_k + \sum_{i=1}^{N} \log\left(p_{t_i}^{\psi}(\mathbf{x}_i^j)\right)\right).\end{aligned} \tag{4.11}$$

Then, the $\mathcal{N}$-Curve Mixture Density Network (abbrev.: $\mathcal{N}$-MDN) can be trained using a standard gradient descent policy. Most commonly, momentum-based gradient descent optimizers are employed. Popular choices include *Adam* [Kin15] and *RMSprop* [Rud16]. It should be noted, that from an optimization point of view, it is preferred to use the mean of the likelihoods when long sequences or many samples should be processed, as the sum of negative log likelihoods may result in large loss values and thus a less stable optimization. Further, it is recommended to output the mean vectors relative to the last element of the input sequence instead of their absolute values. That way, the $\mathcal{N}$-MDN learns a residual mapping, which have proven to be easier to optimize and yield more accurate results [He16][Hug17]. Finally, the loss function $\mathcal{L}$ given in Equation (4.11) is arranged in a way, such that the *log-sum-exp trick* [Pre07] can be applied. This trick prevents arithmetic underflow by offsetting the values in the exponent, according to

$$\log\left\{\sum_i \exp\{z_i\}\right\} = \log\left\{\sum_i \exp\{z_i - z_{\max}\}\right\} + z_{\max}.$$

In this implementation of the $\mathcal{N}$-Curve (mixture) model, the input vector $\mathbf{v}$ allows the model to be used in either a conditional or a non-conditional inference setting. Examples for both settings include sequence prediction (conditional) and the estimation of the data generating distribution given some dataset (non-conditional). Regarding the conditional case, the stochastic process, and thus the $\mathcal{N}$-Curve mixture, depends on some input sequence $\mathcal{S}$. This sequence may be encoded into the input vector $\mathbf{v}$ using some encoder $\mathrm{Enc}(\mathcal{S})$. A common choice for sequence encoders are recurrent neural networks and variants, such as the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU). In the non-conditional case, $\mathbf{v}$ can be set to some constant value. While, technically, this also gives a conditional $\mathcal{N}$-Curve mixture, there is no variation in $\mathbf{v}$, resulting in a constant mixture.

Subsections 4.1.1 – 4.1.4 provide several toy examples using synthetically generated data to showcase different features and the functionality of the proposed implementation of the $\mathcal{N}$-Curve model, based on Mixture Density Networks. In order to remove as much complexity as possible, the input vector $\mathbf{v}$ is set to be constant, thus creating a non-conditional sequence synthesis setting. Following this, the $\mathcal{N}$-MDN learns to generate an $\mathcal{N}$-Curve mixture, which estimates the underlying stochastic process generating the provided data. For all of the toy examples, a *PyTorch* [Pas19] implementation of the $\mathcal{N}$-MDN is used. The model is trained using the Adam optimizer with the learning rate set to 0.01. All other parameters are left at PyTorch defaults. With low-dimensional sequence datasets being small in size, there is no need to perform batch optimization, as the entire dataset fits into memory. As such, the entire training dataset is processed with each iteration of training. During training, the model is assumed to have reached convergence, when the training loss stagnates for 10 iterations.

4.1.1 Estimating $\mathcal{N}$-Curve mixtures from noisy data

For testing the $\mathcal{N}$-Curve Mixture Density Network's capability of learning the parameters of an $\mathcal{N}$-Curve mixture from noisy sequence data, a simple experiment is conducted. To enable a proper visualization of the results, this example uses 2-dimensional data. Following this, an experiment is set up as follows:

1 Define an arbitrary 2-component $\mathcal{N}$-Curve mixture Ψ_{gt} with 5 Gaussian control points per component by defining the weights π, as

well as the mean vectors $\mu^k_{\mathscr{P}}$ and covariance matrices $\Sigma^k_{\mathscr{P}}$ for the control points $\mathscr{P}^k_{\mathcal{N}}$ of each $\mathcal{N}$-Curve ψ_k. The weights are set to $\pi = \{0.75, 0.25\}$ resulting in a biased training dataset.

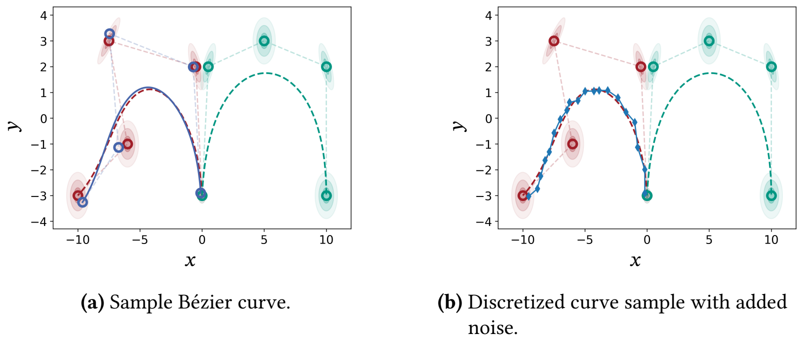
2 Draw a set of $M = 1000$ Bézier curves from Ψ_{gt}.

3 Determine the discrete index set $T_{N=20}$ consisting of 20 (arbitrary, but fixed) evenly distributed values for $t \in [0, 1]$ (see also Section 3.1) in order to discretize each of the M Bézier curves into a set $\mathcal{S}$ of sample sequences.

4 Apply Gaussian noise to each sample sequence in order to create a more realistic training dataset.

5 Train the $\mathcal{N}$-MDN using $\mathcal{S}$ and check if the network is capable of reconstructing Ψ_{gt}.

In this way, the stochastic control points defining the $\mathcal{N}$-Curve mixture have to be estimated indirectly through a set of sample sequences. The ground truth $\mathcal{N}$-Curve mixture Ψ_{gt} and a sample sequence is depicted in Figure 4.2.



(a) Sample Bézier curve.

(b) Discretized curve sample with added noise.

Figure 4.2: Ground truth $\mathcal{N}$-Curves ψ_1 and ψ_2 (red and green) starting at $(0, 0)$ alongside a sampled Bézier curve (blue). Both, (a) and (b), show the mean curve and the control points with covariance ellipses for both $\mathcal{N}$-Curves. In (a) the Bézier curve is illustrated as sampled from ψ_1, while (b) shows the discretized version of the sample curve with Gaussian noise applied.

The training dataset $\mathcal{S}$ and the evolution of the parameters defining the estimated $\mathcal{N}$-Curve mixture Ψ_{pred} over several training iterations are depicted in Figure 4.3. In order to make the quality of the estimation more comprehensible, Figures 4.3c – 4.3f show the deviation of the estimated from the actual parameters. The deviations are defined per control point P_i^k as

$$\begin{aligned}
\Delta\mu &= \|\boldsymbol{\mu}_{\text{pred},i}^k - \boldsymbol{\mu}_{\text{gt},i}^k\|_2 \\
\Delta\sigma &= \boldsymbol{\sigma}_{\text{pred},i}^k - \boldsymbol{\sigma}_{\text{gt},i}^k \\
\Delta\rho &= \rho_{\text{pred},i}^k - \rho_{\text{gt},i}^k.
\end{aligned} \tag{4.12}$$

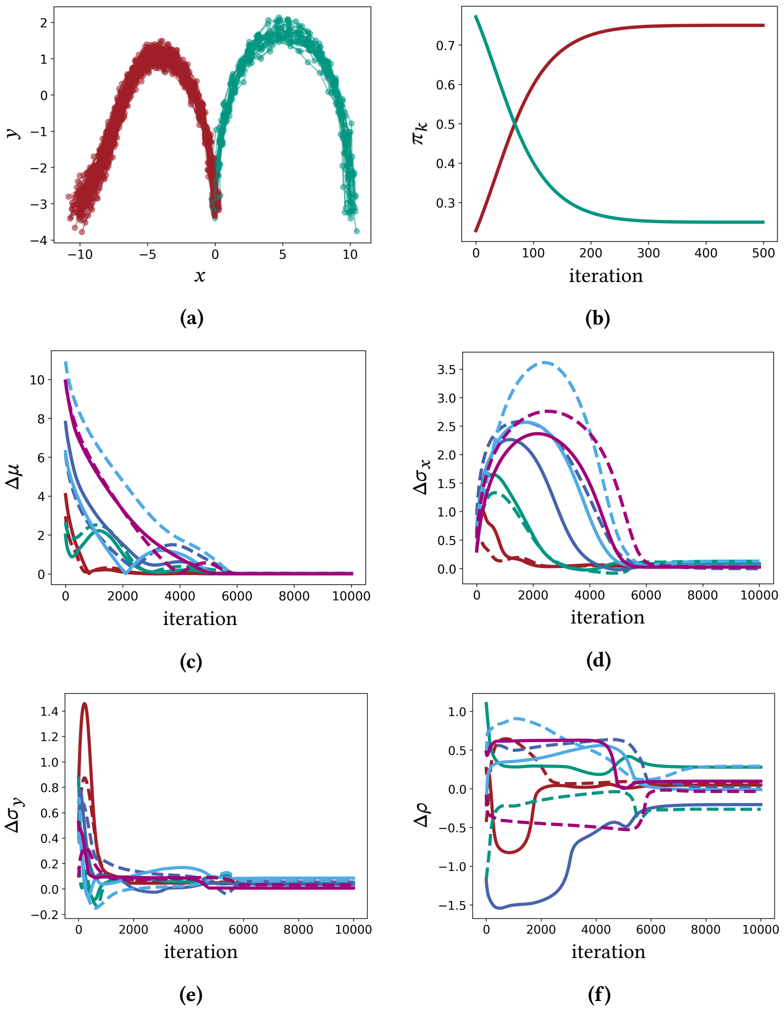
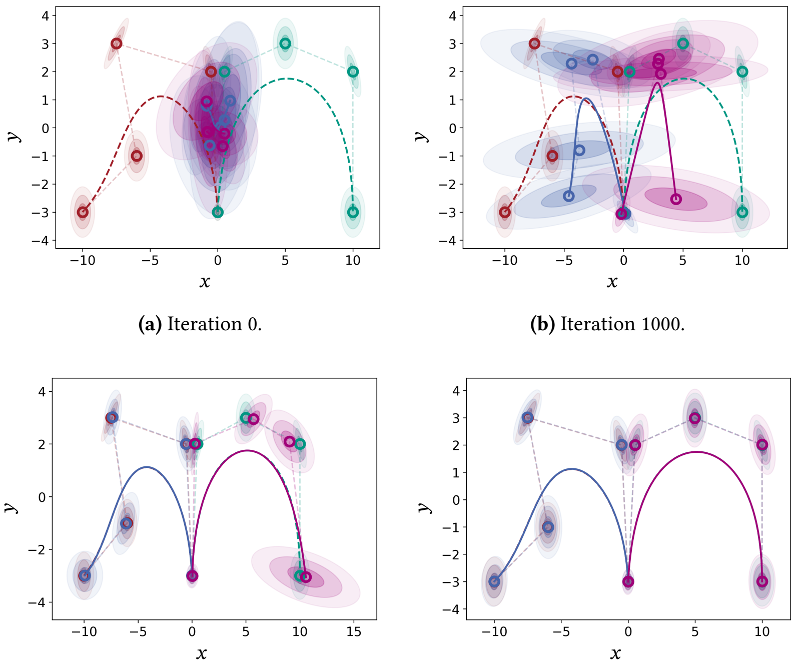


Figure 4.3: Training dataset (a) and estimated mixture parameters over the course of the training ((b) – (f)). (c): Euclidean distance between the estimated mean vectors and the ground truth. (d) – (f): Signed difference between estimated and ground truth standard deviations and correlations, revealing the occurrence of under- and over-estimation. In (c) – (f), values corresponding to the first and second mixture component are shown as a solid and dashed line, respectively. The depicted colored lines correspond to the control points $\{P_0, P_1, P_2, P_3, P_4\}$ of each $\mathcal{N}$-Curve in the mixture.

Looking at Figures 4.3b and 4.3c, it can be seen that the $\mathcal{N}$-MDN is well capable of estimating the actual component weights and control point mean vectors. On the other hand, the model seems to over-estimate the standard deviations slightly ($\Delta\sigma \approx 0.1$) and there appears to be a rather large discrepancy between the estimated and actual correlation values. A possible explanation for these discrepancies can be found when looking at the covariance matrix from a geometric point of view. In general, the covariance matrix not only controls the amount of dispersion in data drawn from a corresponding Gaussian distribution, but also the orientation of the principal axes of dispersion. As such, the covariance matrix can be interpreted as a linear transformation defined by a rotation matrix $\mathbf{R}$ and a diagonal scaling matrix $\mathbf{S}$, such that $\mathbf{\Sigma} = \mathbf{RSSR}^{-1}$[1]. In the 2-dimensional case, the data dispersion can be visualized by an (rotated) ellipse. The orientation of this ellipse is controlled by the covariance matrix off-diagonal element $\Sigma_{0,1} = \Sigma_{1,0} = \sigma_x\sigma_y\rho$. Following this, when either σ is over-estimated, the error in ellipse orientation can be compensated by adapting the correlation ρ. Besides that, there likely exist multiple similar solutions to the covariance matrix defined by different correlation values generating a similar set of samples. As such, the $\mathcal{N}$-MDN only finds a locally optimal solution.

Finally, Figure 4.4 depicts the Ψ_{pred} at different stages during training, illustrating the process of estimating Ψ_{gt}. It can be seen, that the position of the control points is estimated well. Further, the orientation preservation assumption can be confirmed looking at Figure 4.4d. The orientation of the estimated covariance ellipses is similar to the real ellipses, but the variances are slightly over-estimated.

[1] The transformation matrices can be obtained by an *Eigendecomposition*.



(a) Iteration 0.

(b) Iteration 1000.

(c) Iteration 5000.

(d) Iteration 10000.

Figure 4.4: Ground truth $\mathcal{N}$-Curve mixture (red and green) and mixture estimated from noisy sequence data (blue and purple) after 0 (random initialization), 1000, 5000 and 10000 iterations of training.

4.1.2 Handling heteroscedastic data

This example examines the capabilities of the $\mathcal{N}$-Curve model in handling heteroscedastic data, i.e. a stochastic processes with varying variance between time steps. Using heteroscedastic data especially allows the examination of the modeling accuracy when varying the number of $\mathcal{N}$-Curve control points. In contrast to the previous toy example, this and further examples are only concerned with curve points along the estimated $\mathcal{N}$-Curve (mixture), representing the actual stochastic process.

In order to keep the experiment as simple as possible, the dimensionality of the data is set to 1. Thus, a dataset consisting of 1000 sample sequences is generated using an unimodal, time-discrete stochastic process $\mathcal{G}_T = \{X_t \sim \mathcal{N}(\mu_i, \sigma_i)\}_{t \in T_{N=11}}$ with mean values moving along a parabolic curve and corresponding standard deviations changing between consecutive process indices t_i and t_{i+1}. The stochastic process alongside sampled process realizations is given in Figure 4.5.

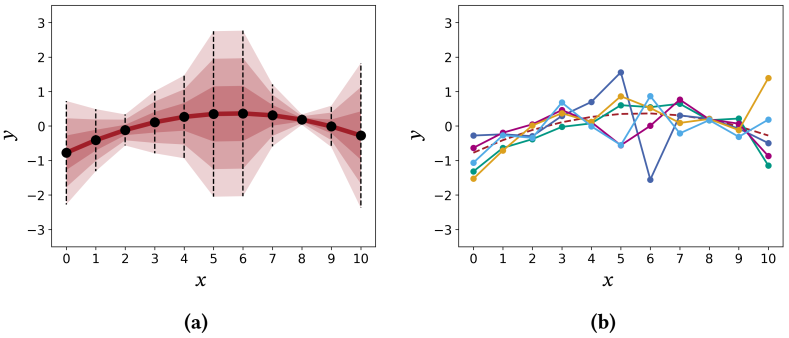


Figure 4.5: Illustration of the training samples drawn from a heteroscedastic stochastic process. (a): Ground truth discrete-time stochastic process $\mathcal{G}_T = \{X_t\}_{t \in T_{N=11}}$. Standard deviations ($\sigma$, 2σ and 3σ) along $\mathcal{G}_T$ are illustrated as a shaded region around the mean curve. 3σ for each X_t is indicated by a horizontal dashed line. (b): Sample sequences drawn from $\mathcal{G}_T$.

Estimated $\mathcal{N}$-Curves with 5 and 15 control points generated by an $\mathcal{N}$-MDN are depicted in Figure 4.6. It can be seen, that the $\mathcal{N}$-MDN learns a smooth mean curve and compensates variation in noise using the variance of the control points. With an increasing number of control points, an increasing number of variations in input noise can be compensated. This, however, comes at the cost of a less accurate mean curve, due to the increasing degree of the underlying polynomial curve. Note that the $\mathcal{N}$-Curves still model a time-continuous stochastic process, despite being given discrete data. Intermediate values are interpolated according to the Bernstein polynomials (see also Section 3.1).

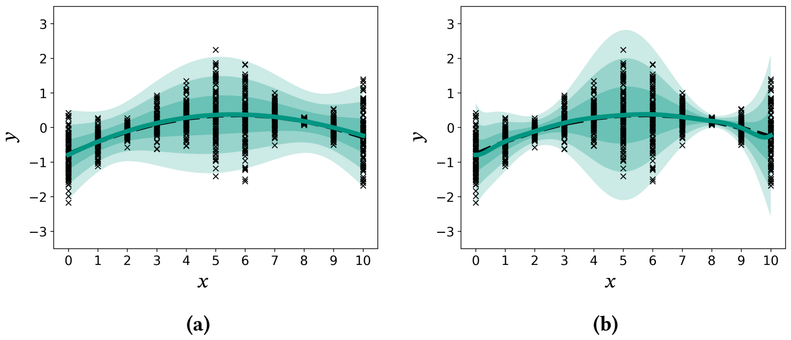


Figure 4.6: Approximations of $\mathcal{G}_T$ using $\mathcal{N}$-Curves with 5 (a) and 15 (b) control points as generated by an $\mathcal{N}$-MDN trained with noisy sequence data. Using more control points increases the accuracy in modeling the variance of the stochastic process at the cost of a less accurate mean curve. For reference, the training sequences (without connections between subsequent points) are illustrated using black cross markers.

4.1.3 Presence of superfluous mixture components

When dealing with multi-modal sequence data, the actual number of modes is usually unknown. Following that, this toy example is concerned with the impact of superfluous $\mathcal{N}$-Curve mixture components on the $\mathcal{N}$-MDN training, as well as the resulting mixture model. For this experiment, a bimodal discrete-time stochastic process $\mathcal{G}_T = \{X_t\}_{t_i \in T_{N=10}}$ with constant variance is defined for generating a training dataset. Here, each random variable of the process follows a bimodal Gaussian mixture distribution and realizations of the process follow one of two paths with equal probability. The stochastic process alongside process realizations is depicted in Figure 4.7.

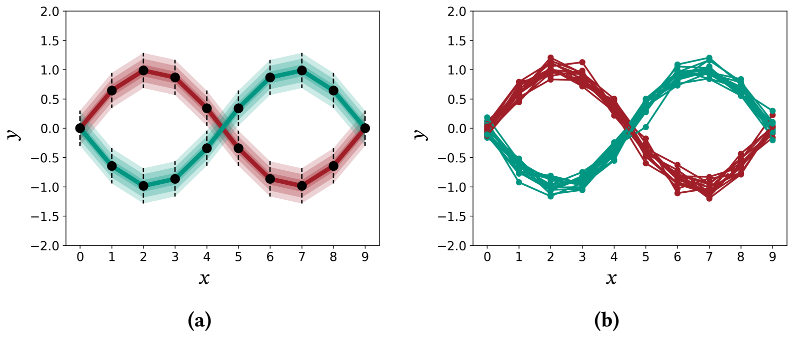
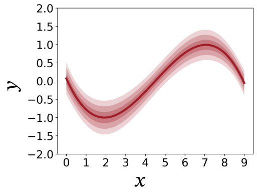
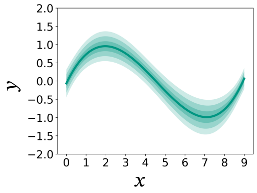


Figure 4.7: Illustration of the training samples drawn from a multi-modal stochastic process. (a): Ground truth discrete-time stochastic process $\mathcal{G}_T = \{X_t\}_{t \in T_{N=10}}$. Standard deviations ($\sigma$, 2σ and 3σ) along $\mathcal{G}_T$ are illustrated as a shaded region around the mean curves. 3σ for each X_t is indicated by a horizontal dashed line. (b): Sample sequences for both modes drawn from $\mathcal{G}_T$.

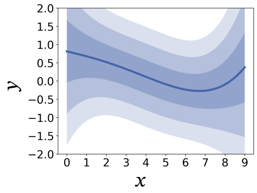
A training dataset consisting of a set of 100 realizations of the aforementioned stochastic process is now used to estimate an $\mathcal{N}$-Curve mixture with $K = 6$ components, i.e. 4 superfluous components, with 5 control points each. Preferably, in the resulting model, the weights of all 4 unnecessary components are driven towards 0 and the remaining $\mathcal{N}$-Curves model the two modes of the stochastic process. The resulting $\mathcal{N}$-Curve mixture components after training the $\mathcal{N}$-MDN are depicted in Figure 4.8. The components are ordered in descending order by their associated mixture weight π_k.



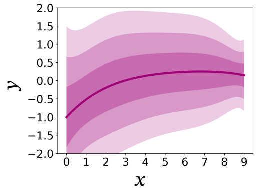
(a) $\pi_1 = 0.4992$



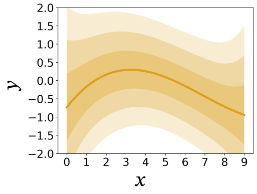
(b) $\pi_2 = 0.4991$



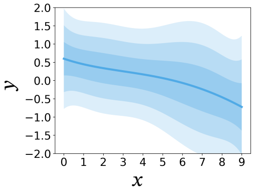
(c) $\pi_3 = 0.0008$



(d) $\pi_4 = 0.0005$



(e) $\pi_5 = 0.0002$



(f) $\pi_6 = 0.0002$

Figure 4.8: Estimated $\mathcal{N}$-Curve mixture with $K = 6$ components generated by an $\mathcal{N}$-MDN. Two components ((a) and (b)) represent $\mathcal{G}_T$ with accurate weighting. The weights of superfluous components is driven towards 0 during training. The shape of these components is thus not further optimized at some point (see figures (c) – (f)).

Looking at the estimated $\mathcal{N}$-Curve mixture, the $\mathcal{N}$-MDN behaves as expected. During training it learns the weight distribution rather fast, leading to superfluous components being not further optimized in their shape. This can be seen in Figure 4.9a. The remaining non-zero components (4.8a and 4.8b) accurately model $\mathcal{G}_T$ with minor over-estimation of the variances. Stripping away the superfluous components, the resulting $\mathcal{N}$-Curve mixture is depicted in Figure 4.9b.

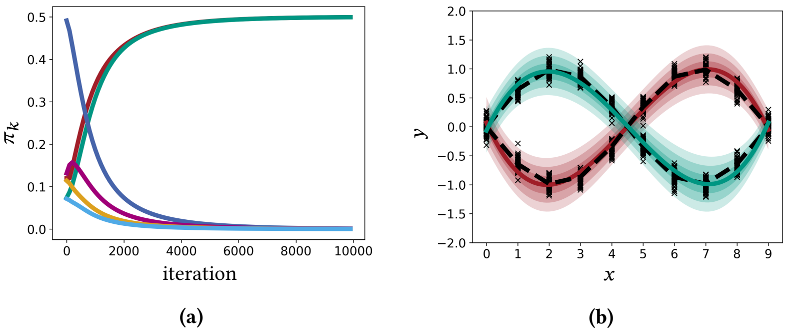


Figure 4.9: Evolution of the weight distribution during training (a) and the resulting approximation of $\mathcal{G}_T$ after removing low-weight components (b). The coloring in (a) matches the components depicted in the previous figure (4.8). For reference, the training sequences (without connections between subsequent points) are illustrated using black cross markers in figure (b).

As a final note, purposefully choosing $K > K_{\text{real}}$ and relying on the optimization driving superfluous components towards 0 might lead to several similar non-zero components when processing more complex datasets. In this case, there are several possibilities to cope with this when required. The first, and most straightforward approach, is implementing a post-processing step, which collapses similar components into one by accumulating their weights and averaging the curves. With respect to the training phase, proper regularization could be employed, trying to enforce sparsity. The most commonly used sparsity-inducing regularization is given by the L_1 norm [Ng04] applied to the mixture weight distribution. Lastly, the determination of K itself can be approached from a different perspective by trying to implement the idea of the Infinite Gaussian Mixture Model [Ras99]. The basic idea of the infinite GMM is applying a Bayesian modeling approach to model the mixture parameters. As such, the mixture weights are modeled using a Dirichlet prior distribution. While this approach removes the problem of choosing an appropriate value for K, it also introduces more complexity into the $\mathcal{N}$-MDN and its training. Additionally, estimating the parameters of an infinite GMM usually involves Monte Carlo methods or variational inference, making such an extension of the $\mathcal{N}$-MDN counteract the intuition of designing a regression-based probabilistic sequence model.

4.1.4 Comparison with SMC inference

One design choice for the $\mathcal{N}$-Curve model is to move multi-step inference into the training phase, thus allowing for an instant prediction of several time steps. This is opposed to sequential monte carlo (SMC) approaches, which model the transition between subsequent time steps and perform iterative inference. In this experiment, the performance of the $\mathcal{N}$-MDN implementation of the $\mathcal{N}$-Curve model is compared to an exemplary SMC approach. For this comparison, an LSTM-MDN model embedded into a particle filtering cycle [Hug18], denoted as *ParticleLSTM* in the following, is used for generating an approximation for a discrete-time stochastic process. As the ParticleLSTM expects discrete inputs and outputs a Gaussian mixture probability distribution, a new set of samples, also called *particles*, needs to be drawn from the mixture distribution after each inference step. This is comparable to the resampling step of particle filters [Dou09] and serves two purposes. First and foremost, this approach keeps the number of particles constant, tackling exponential growth of particles when using a brute force approach. Second, it enables the propagation of a sample-based representation of a probability distribution through time using an LSTM-MDN.

For comparing the performance of the $\mathcal{N}$-MDN and the ParticleLSTM in approximating a time-discrete stochastic process $\mathcal{G}_T$ from noisy sequence data, a training dataset is sampled from $\mathcal{G}_T$. In order to keep this experiment clear and easier to evaluate, $\mathcal{G}_T = \{X_t\}_{t \in T_{N=5}}$ is defined to be an unimodal stochastic process with finite index set $T_{N=5} = \{0, 0.25, 0.5, 0.75, 1\}$. The stochastic process and a training dataset sampled from $\mathcal{G}_T$ are depicted in Figure 4.10.

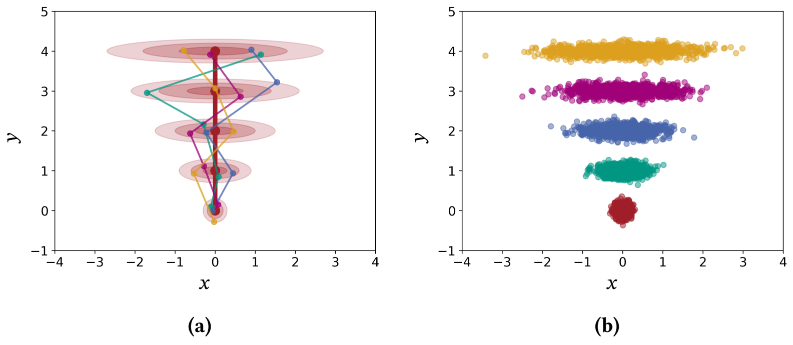
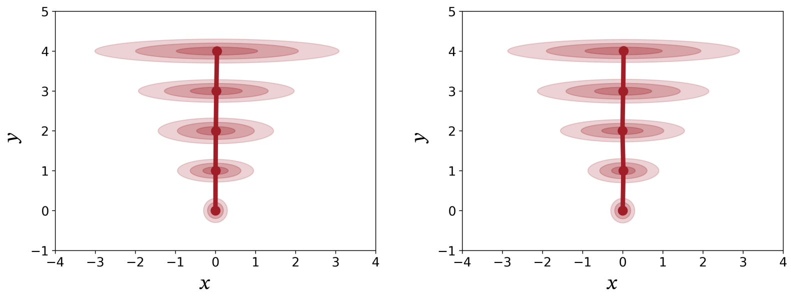


Figure 4.10: Illustration of the training dataset for comparing the $\mathcal{N}$-MDN with an exemplary SMC approach. (a): Ground truth discrete-time stochastic process $\mathcal{G}_T = \{X_t\}_{t \in T_{N=5}}$ alongside sample sequences. (b): Training data sampled from $\mathcal{G}_T$. Sequential connections are left out in this illustration in order to provide a cleaner illustration.

Using this training dataset, both models are trained until convergence is reached. In this experiment, the $\mathcal{N}$-MDN generates a 1-component $\mathcal{N}$-Curve mixture with 3 control points and the ParticleLSTM uses N_p particles for generating its prediction. An approximation of $\mathcal{G}_T$, given $\mathbf{v} = \mathbf{0}$ as a constant input, is then generated as follows: In case of the $\mathcal{N}$-MDN, a single pass through the network yields the parameters of an $\mathcal{N}$-Curve. Accessing this curve at $t \in \{0, 0.25, 0.5, 0.75, 1\} =: T_{N=5}$ gives the stochastic variables $\{X_t\}_{t \in T_{N=5}}$ approximating $\mathcal{G}_T$. For the ParticleLSTM, passing N_p copies of $\mathbf{v}$ through the network generates a Gaussian mixture distribution Ξ_1 approximating X_1. Next, N_p samples are taken from Ξ_1 and fed into the network again in order to approximate X_2. This process is repeated for retrieving X_3, X_4 and X_5. The resulting approximations of $\mathcal{G}_T$ are depicted in Figure 4.11.



(a) $\mathcal{N}$-MDN using $K = 1$ component with 3 control points.

(b) ParticleLSTM using $N_p = 1000$ particles.

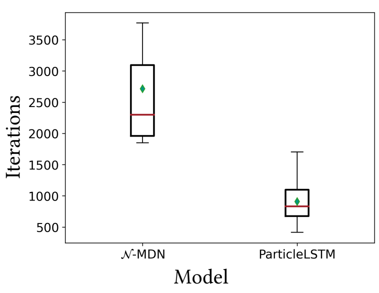
Figure 4.11: Resulting approximations of the stochastic process $\mathcal{G}_T$. Both approaches yield similar results with slight differences in the variances, especially for X_3 and X_4.

It can be seen, that both approaches lead to similar results. The most notable differences can be observed looking at the variances, where the $\mathcal{N}$-MDN yields slightly less accurate results, which can be attributed to the use of a compact representation with only 3 control points. The difference in variances is most visible for X_3 and X_4, where the non-linear weighting of control point covariances yield a minor under-estimation of the actual variance values (see also Section 3.1.2). On the other hand, the mean curve generated by the ParticleLSTM is less stable, which is most likely due to the uncontrolled and stochastic nature of the iterative approach.

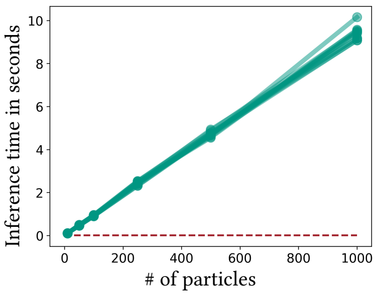
While the comparison of the resulting approximations confirms the viability of the $\mathcal{N}$-Curve model as an alternative to SMC approaches, additional aspects should be considered. By design, the $\mathcal{N}$-Curve model moves the task of multi-step inference into the training phase, thus eliminating the need for Monte Carlo simulation during inference. Following this, Figure 4.12 depicts the differences in training time (4.12a), inference time (4.12b), memory usage (4.12c) and accuracy (4.12d) in order to reveal the impact of this design choice on these aspects. In order to provide a measure for the approximation accuracy, the error E in terms of the euclidean distance of vectors combining mean and standard deviation values is averaged over all time steps, i.e.

$$E = \frac{1}{5}\sum_{t=1}^{5} \left\| \begin{pmatrix} \hat{\mu}_{t,x} & \hat{\mu}_{t,y} & \hat{\sigma}_{t,x} & \hat{\sigma}_{t,y} \end{pmatrix}^{\top} - \begin{pmatrix} \mu_{t,x} & \mu_{t,y} & \sigma_{t,x} & \sigma_{t,y} \end{pmatrix}^{\top} \right\|_2 . \tag{4.13}$$

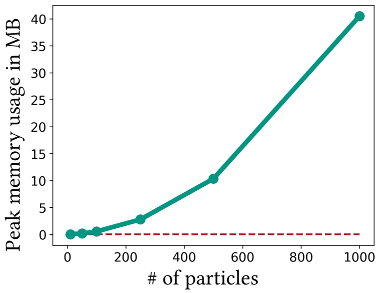
Here, $\hat{\cdot}$ represents estimated values. In this formulation, the error function aggregates the square differences of each factor. Although this approach ignores the actual semantics of the mean and variance values, it should provide a viable estimate for the accuracy of all approximated factors, due to their common minimum error value and comparable squared value ranges. Note that for this comparison, the reference values $\mu_{t,x}$, $\mu_{t,y}$, $\sigma_{t,x}$ and $\sigma_{t,y}$ are calculated from the training dataset, as these are likely to differ slightly from the actual ground truth values due to the training data being sampled randomly. Further, for the ParticleLSTM, the statistics are provided for an increasing amount of particles used for inference. Training and inference of each configuration is performed 10 times, in order to generate more reliable results.



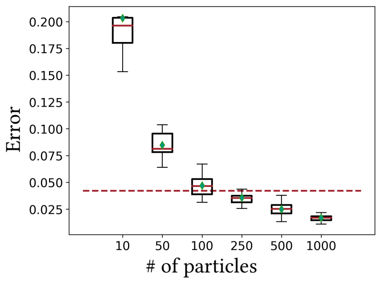
(a) Training iterations required to reach convergence.



(b) Time required to infer an approximation of $\mathcal{G}_T$ after training. Each line corresponds to one repetition of the inference procedure.



(c) Peak GPU memory usage during inference. Memory usage is consistent for each repetition of the inference procedure.



(d) Error in the estimate of $\mathcal{G}_T$. The error is unitless due to the comparison of vectors composed of mean and standard deviation values.

Figure 4.12: Comparison of the $\mathcal{N}$-MDN and the ParticleLSTM focusing on different aspects related to training and inference when increasing the number of particles used for inference. Inference statistics are provided for 10 repetitions in order to show the consistency of the SMC approach. In figures (b) – (d), the red line indicates the $\mathcal{N}$-MDN baselines, the ParticleLSTM is compared to. In figures (a) and (d), the green diamond markers indicate the mean values. The error in (d) is given by the average deviation of the estimated mean and standard deviation values from a reference provided by the training data.

Looking at Figure 4.12, the impact on the depicted factors is as expected. Due to moving the multi-step inference into the training phase, more iterations are required to reach convergence when compared to the ParticleLSTM, which only needs to learn the transition between subsequent time steps (see 4.12a). On the other hand, multi-step inference is achieved using a single pass through the $\mathcal{N}$-MDN, resulting in faster inference. The time required to approximate $\mathcal{G}_T$ using the ParticleLSTM scales linearly with the number of particles N_p (see 4.12b). At the same time, the memory usage grows with an increasing N_p. Looking at Figure 4.12c memory usage even grows superlinear due to the heavy computations being implemented to run on the GPU, where vectorization is required. This, in turn, has higher memory demand, especially for parallelized particle re-sampling. As expected, Figure 4.12d shows that the accuracy of the approximation increases with higher n_p, surpassing the accuracy of the $\mathcal{N}$-MDN approximation at some point. Ultimately, it depends on the specific use case whether faster but slightly less accurate or slower but more accurate inference is more important.

4.2 Proof of Concept: Recurrent $\mathcal{N}$-Curve Mixture Density Networks

In order to provide a proof of concept for the conceptual extension of the $\mathcal{N}$-Curve model, the meta-time $\mathcal{N}$-Curve model (see Section 3.2), an approach which is capable of generating a steady stream of $\mathcal{N}$-Curve segments is required. Additionally, dependencies between subsequent segments along a generated probabilistic spline need to be taken into account, especially in C^1 or C^2 continuity cases. Following this, an autoregressive approach is well-suited for this implementation. With combinations of recurrent neural networks and Mixture Density Networks being a state-of-the-art sequence model (see also Section 2.2.2), an LSTM network will be combined with the $\mathcal{N}$-MDN (see Section 4.1) for this proof of concept. The resulting model is denoted as *recurrent $\mathcal{N}$-MDN*. The recurrent $\mathcal{N}$-MDN operates on the meta-timeline and targets the generation of an endpoint-connected probabilistic spline. This restricts the timeline mappings presented in Section 3.2.2 to special cases without overlapping curve segments. A schematic of the model architecture is given in Figure 4.13.

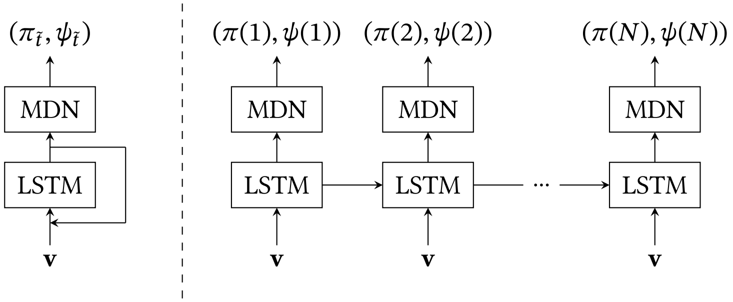


Figure 4.13: Schematic of the recurrent $\mathcal{N}$-MDN illustrating the architecture with a loop (left side) and unrolled over N steps of meta-time (right side). Outputs a $\mathcal{N}$-Curve segment at each meta-time step.

As illustrated, the same model input $\mathbf{v}$ is used in each meta-time step. The reason for this is given by the fact, that feeding generated $\mathcal{N}$-Curve segments back into the model would require a way of encoding an $\mathcal{N}$-Curve into $\mathbf{v}$. Instead, by using constant input, the model relies on its recurrent connection for evolving its output over time. As a technical detail, the model is designed to generate a stream of residuals, i.e. Gaussian control point mean vectors are always given as offsets to preceding control points. This has multiple advantages. First, as mentioned in Section 4.1, using residuals instead of absolute values is more stable during training and inference, as the target domain is more restricted. Second, by defining segment control points in terms of previous control points, it is easier to take geometric restrictions into account for enforcing C^1 or C^2 continuity as required. Regarding the loss function employed during training of the recurrent $\mathcal{N}$-MDN, the negative log-likelihood as defined in Equation (4.11) in Section 4.1 can be directly translated to this extension. This is due to the extraction of sequences from a meta-time $\mathcal{N}$-Curve working basically in the same way as in $\mathcal{N}$-Curves.

In summary, using an autoregressive model provides a straightforward approach for implementing the meta-time $\mathcal{N}$-Curve model, with the capability of infinite sequence generation. On the downside it should be mentioned, that the calculation of the $\mathcal{N}$-Curve segment at meta-time $\tilde{t}$ requires the calculation of all preceding segments. This is, however, also necessary independent of the specific approach when C^1 continuity is required, as it introduces dependencies between subsequent segments. Although this approach re-introduces a notion of iterative generation, constrains imposed by the underlying parametric curve, which is a spline in this case, remain. Also, with every stochastic process mode now being modeled by a probabilistic spline, the need for Monte Carlo simulation is still avoided. Thus, the presented approach mostly complies with the objectives formulated in Chapter 1. The only exception is given by multi-step sequence generation beyond N_{seg} steps, which requires an iterative approach instead of being instantaneous.

4.2.1 Toy Examples

This section provides a brief evaluation of the capabilities of the meta-time $\mathcal{N}$-Curve model through different toy examples. Similar to the toy examples given in Section 4.1, the examples in this section focus on non-conditional sequence synthesis. Thus $\mathbf{v} = \mathbf{0}$ is set as the constant input of the recurrent $\mathcal{N}$-MDN for each meta-time step. This reduces the information processed by the recurrent $\mathcal{N}$-MDN to the information passed over time through the recurrent connection. Further, $\mathcal{N}$-Curve segments will be defined by 5 Gaussian control points. Due to working with discrete-time ground truth stochastic processes (see also Section 3.2.3), each segment is defined to cover sub-sequences of length $N_{\text{seg}} = 20$.

Three scenarios are considered for comparing the meta-time $\mathcal{N}$-Curve model to the original $\mathcal{N}$-Curve model. The targeted discrete-time stochastic processes $\mathcal{G}_T^i$ for each scenario are depicted in Figure 4.14. For training, a set of $M = 200$ realizations is sampled from each stochastic process. The curve-time mapping defined in Section 4.14 is used for all examples.

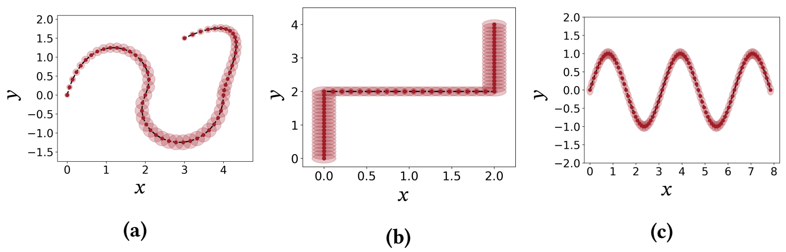


Figure 4.14: Ground truth discrete-time stochastic processes $\mathcal{G}_T^i$ with $i \in \{a,b,c\}$ providing different scenarios for exploring the capabilities of the meta-time $\mathcal{N}$-Curve model. The index sets[a] are defined as $T = T_{N=58}$ for scenarios (a) and (b), and $T = T_{N=96}$ for scenario (c), thus covering sequences consisting of 58 and 96 elements, respectively. While $\mathcal{G}_T^a$ is subject to varying variance between time steps, $\mathcal{G}_T^b$ and $\mathcal{G}_T^c$ have constant variance. For all figures, Gaussian random variables along the stochastic process' mean curve are given with corresponding 2σ covariance ellipses.

[a] The discrete index set notation T_N follows the definition provided in Section 3.2.3.

The first scenario covers the case of a stochastic process with a complex mean curve, in terms of length and shape. Following this, a parametric curve requires an increased number of control points to approximate the sequence properly. For this example, a fixed interval mapping with $a = 1$ is used and C^1 continuity is enforced. With the training dataset consisting of sequences of length $N = 58$ and each segment of the meta-time $\mathcal{N}$-Curve covering $N_{\text{seg}} = 20$ elements, the recurrent $\mathcal{N}$-MDN will generate 3 segments. Thus, the segmented curve is defined by a total of 13 control points due to subsequent segments having one control point in common. Following this, the $\mathcal{N}$-Curve model in comparison is defined with an equal number of 13 control points. The estimated original and meta-time $\mathcal{N}$-Curves generated by a respective $\mathcal{N}$-MDN and recurrent $\mathcal{N}$-MDN are depicted in Figure 4.15.

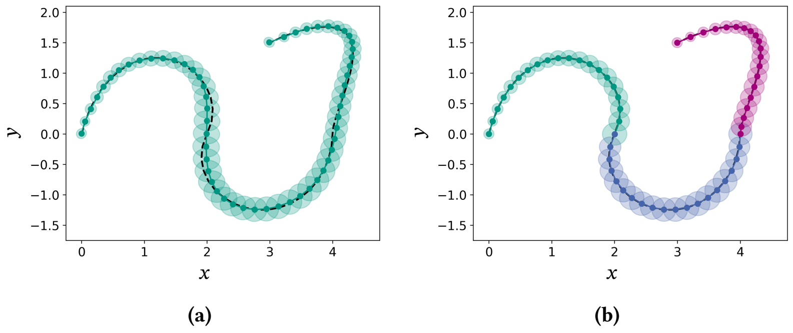


Figure 4.15: Approximations of $\mathcal{G}_T^a$ using an $\mathcal{N}$-Curve (a) and a meta-time $\mathcal{N}$-Curve (b) as generated by an $\mathcal{N}$-MDN and a recurrent $\mathcal{N}$-MDN trained with noisy sequence data. For the meta-time $\mathcal{N}$-Curve, a fixed interval mapping is used and C^1 continuity is enforced. In (b) $\mathcal{N}$-Curve segments are highlighted by using different colors. 2σ covariance ellipses are provided for all Gaussian random variables along the mean curve.

While the meta-time $\mathcal{N}$-Curve approximates the mean curve perfectly, the estimated $\mathcal{N}$-Curve deviates slighty at around $x = 2$ and $x = 4$, averaging out a curved shape. A slight over-estimation of the variance at the beginning and end of the approximation can be observed for both models. Besides both models performing quite similar in their generated approximation, the $\mathcal{N}$-MDN took 8 times more iterations to reach convergence compared to recurrent $\mathcal{N}$-MDN. This observation can most likely be attributed to single $\mathcal{N}$-Curves of higher degree being harder to fit to given data due to the global control property.

The second scenario covers a stochastic process with its mean curve including sharp edges. In general, such curves cannot be represented by Bézier curves due to their smoothness property. Using a segmented curve, on the other hand, allows such edges by only targeting C^0 continuity. Additionally, by using segments of lower degree, Gibbs phenomenon [Jer13] can be circumvented. Apart from the training dataset, the setup for this scenario is similar to the first scenario. The resulting approximations are depicted in Figure 4.16.

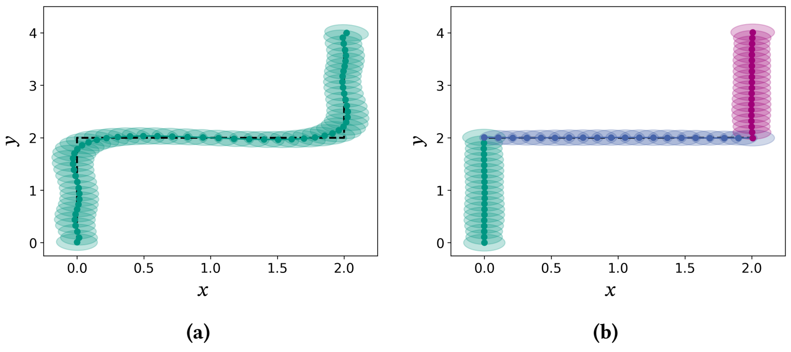


Figure 4.16: Approximations of $\mathcal{G}_T^b$ using an $\mathcal{N}$-Curve (a) and a meta-time $\mathcal{N}$-Curve (b) as generated by an $\mathcal{N}$-MDN and a recurrent $\mathcal{N}$-MDN trained with noisy sequence data. For the meta-time $\mathcal{N}$-Curve, a fixed interval mapping is used. No smoothness constrains are applied. In (b) $\mathcal{N}$-Curve segments are highlighted by using different colors. 2σ covariance ellipses are provided for all Gaussian random variables along the mean curve.

As expected, the estimated $\mathcal{N}$-Curve is unable to replicate the target mean curve, but still provides a close approximation. As the $\mathcal{N}$-Curve is of higher degree, Gibbs phenomenon is quite noticeable in this example. Besides minor fluctuations in the variances, the estimated meta-time $\mathcal{N}$-Curve is accurate with respect to its mean curve.

The third and final scenario regards a stochastic process whose mean curve follows a sine wave. Because of the periodicity, a modulo reset mapping with $a = 1$ and $k = 2$ will be used for the meta-time $\mathcal{N}$-Curve. Further, C^1 continuity is enforced. Note that k can only be assigned an appropriate value due to knowledge about the structure of the targeted mean curve. As such, $\mathcal{G}_T^c$ can be approximated with a meta-time $\mathcal{N}$-Curve, which repeats the same two segments as many times as required. Following this, an $\mathcal{N}$-Curve with 9 control points will be estimated for comparison. The results are depicted in Figure 4.17.

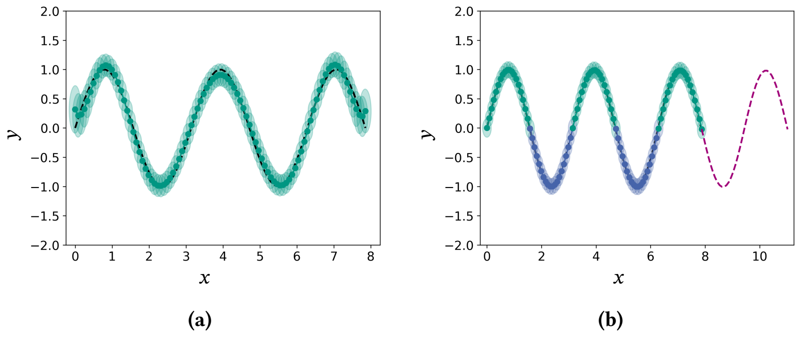


Figure 4.17: Approximations of $\mathcal{G}_T^c$ using an $\mathcal{N}$-Curve (a) and a meta-time $\mathcal{N}$-Curve (b) as generated by an $\mathcal{N}$-MDN and a recurrent $\mathcal{N}$-MDN trained with noisy sequence data. For the meta-time $\mathcal{N}$-Curve, a modulo reset mapping is used and C^1 continuity is enforced. In (b) $\mathcal{N}$-Curve segments are highlighted by using different colors. 2σ covariance ellipses are provided for all Gaussian random variables along the mean curve.

Looking at the estimated $\mathcal{N}$-Curve first, the approximation of $\mathcal{G}_T^c$ is quiet accurate apart from the beginning and ending portions. On the other hand, the meta-time $\mathcal{N}$-Curve alternates between the two learned segments in order to achieve a close approximation of $\mathcal{G}_T^c$. Further, in such scenarios, the curve could be continued indefinitely, as indicated in Figure 4.17b (purple curve).

4.2.2 Handling underdetermined areas

As the meta-time $\mathcal{N}$-Curve model is based on segmented curves being calculated iteratively using an autoregressive approach, the existence of underdetermined areas provides an aspect worth discussing. Such underdetermined areas are defined as segments within a meta-time $\mathcal{N}$-Curve, which are not well estimated during training. The main causes for this are given by either areas being sparely covered by the training dataset or insufficient model capacity. Besides the model output within these segments being less stable and reliable in an application context, it also affects subsequent segments due to error propagation in the autoregressive model structure. As a general approach for coping with underdetermined areas, a fallback mechanism can be integrated into the model. As such, the sequence model can rely on a basic, domain-specific sequence model, which covers regions of high model uncertainty. Besides the need of a potentially handmade fallback model, the model uncertainty needs to be measured using such an approach. Following this, a brief overview of techniques for measuring model uncertainty is given. An evaluation of the practicality of the presented techniques for the meta-time $\mathcal{N}$-Curve is thereby left out, being beyond the scope of this thesis

A prominent approach to measuring model uncertainty is given by transforming a network into a Bayesian neural network (see also Section 2.2.1), which is implemented using Monte Carlo Dropout [Gal16, Gal17, Ken17]. In these variants of Bayesian neural networks, dropout [Sri14] is applied in conjunction with multiple passes through the network in order to generate a distribution over the network's output. This distribution can then be used to measure the model's uncertainty by correlating it to the variance in the generated distribution. A downside of employing such an approach is that a given sequence model needs to be transformed into a Bayesian neural network, thereby also inheriting their potentially unwanted properties and problems. An alternative to Bayesian neural networks is given by *Prior Networks* [Mal18]. While Bayesian neural networks implicitly model distributional uncertainty, Prior networks provide an explicit model for model uncertainty. This is achieved by parameterizing a prior distribution over predictive distributions. Thus, the Prior network approach also requires changes to a given model. Opposed to that, an ensemble approach [Lak17, Hua17] can be pursued, avoiding the need to change the model at hand. Here, an ensemble of the same model is trained. As the training process itself is usually stochastic, the resulting ensemble consists of several models generating slightly different outputs for the same input. As such, using the entire ensemble, a distribution similar to that of a Monte Carlo Dropout Bayesian neural network can be generated.

4.3 Summary

Overall, this chapter first provided a detailed introduction to Mixture Density Networks, focusing on their general structure and how their output is generated. Following this, $\mathcal{N}$-Curve Mixture Density Networks were defined as a regression-based implementation for the $\mathcal{N}$-Curve model, which enables multi-step inference only requiring a single forward pass through the network. Using synthetically generated data, several toy examples show the model's capability of learning stochastic control points from noisy sequence data and explore the model's behavior and capabilities under different circumstances. Finally, a comparison with an SMC-based approach was performed, depicting the advantages of $\mathcal{N}$-Curve Mixture Density Networks during inference in terms of memory usage and inference time.

Additionally, a proof of concept for an implementation of the meta-time $\mathcal{N}$-Curve model was presented. The presented model relies on an autoregressive structure in order to enable the representation of infinite stochastic processes. In comparison to $\mathcal{N}$-Curve Mixture Density Networks, toy examples on synthetically generated data indicate greater flexibility in terms of modeling capabilities at the cost of requiring a more complex neural network model, which is more expensive in terms of computation time during inference.

5 Evaluation

This evaluation focuses on the $\mathcal{N}$-Curve model and the $\mathcal{N}$-MDN implementation as proposed in Sections 3.1 and 4.1. For this, the $\mathcal{N}$-Curve model is applied to two sequence prediction tasks. In both tasks, evaluated models need to represent a stochastic process describing $N_{\text{obs}} + N_{\text{pred}}$ time steps, such that given N_{obs} observations of a process realization, the remaining N_{pred} steps can be inferred.

In the first part of the evaluation, the $\mathcal{N}$-Curve model is applied to the task of human trajectory prediction (Section 5.1). On the one hand, this task provides easy to interpret and visualize results. As such, it gives a good foundational evaluation of the general viability of the model. Further, although being simple in terms of data dimensionality, the task provides a lot of complexity with human trajectory prediction being a highly multi-modal problem. In this regard, human trajectory prediction provides an appropriate task for evaluating the capabilities of the model.

Following the evaluation of the viability and capabilities of the $\mathcal{N}$-Curve model, its claimed capability of being scalable to arbitrary dimensions is assessed. For this, the model is applied to the task of human motion prediction. This task provides a high-dimensional example, being concerned with modeling sequences of human poses (Section 5.2).

It is worth mentioning, that the meta-time $\mathcal{N}$-Curve model (Section 3.2), and thus the recurrent $\mathcal{N}$-MDN (Section 4.2), are excluded from this evaluation. There are two main reasons for this. First, the toy examples in Section 4.2 indicate that the meta-time $\mathcal{N}$-Curve model gains an edge over the original $\mathcal{N}$-Curve model for very long sequences only. However, common evaluations conducted on real-world data are usually restricted to rather short time horizons, i.e. short sequences. Second, the meta-time $\mathcal{N}$-Curve model is first and foremost a conceptual extension to the $\mathcal{N}$-Curve model, lifting some more domain-specific limitations.

For convenience, the following notation, extending on the notation used in previous chapters, is introduced for the scope of the evaluation. A dataset is denoted as $\mathfrak{D} = \{\mathcal{X}_1, ..., \mathcal{X}_M\}$ and consists of M sequences of fixed length $N = N_{\text{obs}} + N_{\text{pred}}$ with $\mathcal{X}_i = \{\mathbf{x}_1^i, ..., \mathbf{x}_N^i\}$. Each dataset can further be divided into a training and test dataset, such that $\mathfrak{D} = \mathfrak{D}_{\text{train}} \cup \mathfrak{D}_{\text{test}}$, with $\mathfrak{D}_{\text{train}} \cap \mathfrak{D}_{\text{test}} = \emptyset$. Finally, each sequence $\mathcal{X}_i$ in a dataset $\mathfrak{D}$ is divided into an observed $\mathcal{X}_{i,\text{obs}}$ (N_{obs} time steps) and target $\mathcal{Y}_i$ (N_{pred} time steps) portion

$$\begin{aligned}\mathcal{X}_i &= \mathcal{X}_{i,\text{obs}} \cup \mathcal{Y}_i \\ &= \Big\{ \underbrace{\mathbf{x}_1^i, ..., \mathbf{x}_{N_{\text{obs}}}^i}_{\text{observation}}, \underbrace{\mathbf{x}_{N_{\text{obs}}+1}^i, ...\mathbf{x}_{N_{\text{obs}}+N_{\text{pred}}}^i}_{\text{target}} \Big\} \\ &= \Big\{ \underbrace{\mathbf{x}_1^i, ..., \mathbf{x}_{N_{\text{obs}}}^i}_{\text{observation}}, \underbrace{\mathbf{y}_1^i, ..., \mathbf{y}_{N_{\text{pred}}}^i}_{\text{target}} \Big\},\end{aligned}$$

where the target portion is to be predicted.

5.1 Long-term Human Trajectory Prediction

With the emergence of autonomous driving and advances in the field of automated video surveillance, the task of human trajectory prediction gained a significant amount of research interest in recent years. A trajectory is defined as a sequence of locations in a regarded scene, with some velocity profile attached to it. Predictions are then performed on sequences consisting of subsequent 2D image coordinates or 3D world coordinates, generated by e.g. a detection-tracking pipeline. Generally speaking, human trajectory prediction can be subdivided in a number of more specific tasks, depending on the time horizon for prediction, the point of view of recording and camera motion. Each of these aspects impacts observations, and such the respective prediction models, in a different way.

Time Horizon: In autonomous systems, the prediction task can be divided into short-term (0.5 up to 2 seconds) and long-term (up to 20 seconds) trajectory prediction [Rud20b]. While short-term predictions are mainly used for immediate decisions, such as collision avoidance, long-term prediction impacts the long-term behavior of an autonomous system, e.g. by influencing its path planning component. Considering short-term prediction, linear models combined with local collision avoidance approaches, e.g. the *social force model* [Hel95] or *Optimal Reciprocal Collision Avoidance* (abbrev.: ORCA, [Van11]), are generally well suited. In the context of human trajectories, ORCA yields more realistic motion patterns [Kot21]. In long-term trajectory prediction, the trajectory shape is greatly influenced by the surrounding static environment and interactions with other pedestrians. The extent of influence is highly dependent on the ground resolution and annotation rate of a given dataset [Hug21], as well as the pedestrian density.

Point of view: Most commonly, trajectory prediction datasets are recorded from a bird's eye view (*top view*), an elevated viewpoint with a tilted camera (*tilted view*) or from a camera positioned on the ground, e.g. mounted to a car (*frontal view*). While top view and surveillance datasets yield complete trajectories, occlusions occur frequently in frontal view datasets. As a consequence, prediction models need to be able to cope with missing inputs when working with frontal view datasets. In addition, constant velocity trajectories are distorted in frontal view datasets due to perspective distortion.

Camera motion: For top view and surveillance datasets, static cameras are a common choice. As such, recorded trajectory data complies with the static observed scene, potentially resulting in decision points, e.g. junctions, at specific locations. With frontal view datasets, identical trajectories change in shape with the ego-motion of the camera, when mounted to a car. In such cases, datasets are usually transformed into an ego-motion compensated reference frame (e.g. [Sch13]).

Following this, the following evaluation focuses on long-term trajectory prediction using bird's eye view data, giving a suitable task for learning-based sequence prediction models. Given an observed trajectory $\mathcal{X}_{\text{obs}} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_{\text{obs}}}\}$ consisting of N_{obs} observed positions of a person, the subsequent N_{pred} future positions need to be predicted.

5.1.1 Dataset Overview

With the rising interest in the topic of human trajectory prediction, a number of datasets has emerged. These datasets are most often created from annotated videos, recorded from a specific point of view. An overview of commonly used human trajectory datasets, categorized by the respective point of view, is given in Table 5.1.

Table 5.1: Non-exhaustive list of datasets appropriate for human trajectory prediction. Note that the list of top view datasets also includes datasets providing real-world positions instead of image coordinates. As data of persons moving on a flat plane is recorded, these datasets are similar to top view datasets, except trajectory points are given in a 3D reference frame with constant elevation.

Point of View	**Datasets**
Top view	BIWI Walking Pedestrians [Pel09]
	Crowds by Example [Ler07]
	Stanford Drone Dataset [Rob16]
	Edinburgh Forum [Maj09]
	inD Dataset [Boc19]
	Thör Dataset [Rud20a]
	CITR [Yan19]
	DUT [Yan19]
Tilted view	Grand Central [Yi15]
	PETS 2009 [Fer09]
	VIRAT [Oh11]
	Town Center [Ben11]
	WILDTRACK [Cha18]
Frontal view	KITTI [Gei13]
	nuScenes [Cae20]
	JAAD [Ras17b, Ras17a]
	Daimler [Sch13]

In the context of long-term human trajectory prediction, top view and surveillance datasets are preferred due to the lack of occlusions and perspective distortions. In addition, data recorded from a static scene imposes a structure onto the dataset, which yields well-defined walking paths and decision points, exposing the multi-modal nature of human trajectory prediction.

Finally, the most commonly used datasets for long-term human trajectory prediction include the BIWI Walking Pedestrians (abbrev.: *biwi*), Crowds by Example (abbrev.: *crowds*) and the Stanford Drone (abbrev.: *sdd*) datasets. These datasets further consist of 2, 4 and 8 scenes, respectively. In the following, these scene datasets will be referred to as *dataset:scene*, e.g. *biwi:eth*. In the case of the Stanford Drone Dataset, multiple, partially overlapping[1], recordings of the same scene are provided. The specific recording will be indicated by a number added to the scene dataset abbreviation, e.g. *sdd:hyang00*.

5.1.2 State-of-the-art Human Trajectory Prediction Models

Looking at state-of-the-art deep learning-based sequence prediction models for long-term human trajectory prediction, these models can be divided into aggregating and holistic models. Holistic models, on the one hand, model the entire observed scene including all pedestrians by using a spatio-temporal graph network, where each object in the scene is a unique node (e.g. [Moh20, Sal20]). Opposed to that, the more prevalent aggregating models have separate processing pipelines for each type of input, which are fused together at some point. For this class of models, a modular meta-architecture revolving around an underlying base sequence model can be defined, covering the main components of each model. Additional types of inputs, also referred to as *additional cues*, are discussed later in this section. A schematic of this meta-architecture is depicted in Figure 5.1.

[1] In the sense of the observed real-world scenery.

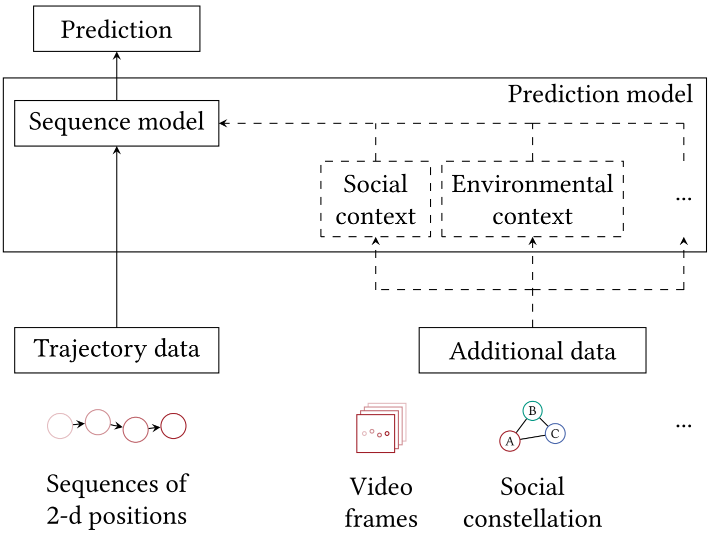


Figure 5.1: Schematic of a meta-architecture for aggregating trajectory prediction models. Such models contain at least some sequence model and optional building blocks for processing additional cues, such as social or environmental context.

Each aggregating prediction model at least consists of a base sequence model, which encodes input trajectories, the *observation*, and generates either single trajectories or probabilistic predictions. Taking a range of state-of-the-art deep learning-based prediction models into consideration, these models can be boiled down to few base sequence models. An overview of commonly used base sequence models is depicted in Table 5.2. Note that due to the existence of many similar models, only representative examples for each base sequence model are featured. For a comprehensive overview of existing human trajectory prediction approaches, the reader may be referred to recent surveys, e.g. [Rud20b]. Further, no distinction is made for variants of the same base model, as these most commonly only differ slightly. Finally, endpoint-conditioned prediction models (e.g. [Kit12, Man20]) are excluded from this overview, as the endpoint is assumed to be unknown in the context of this evaluation.

Table 5.2: Overview of commonly used base sequence models in human trajectory prediction alongside representative prediction models. The most frequently used sequence models are given by *Recurrent Mixture Density Networks* (abbrev.: R-MDN, [Gra13]), variants of *Generative Adversarial Networks* (abbrev.: GAN, [Goo14]) and *Variational Autoencoders* (abbrev.: VAE, [Kin14]) combined with a sequence to sequence model [Sut14], as well as *Transformers* [Vas17]. Transformers are only recently being studied in the context of human trajectory prediction. It could be noted, that *Temporal Convolutional Networks* (abbrev.: TCN, [Oor16, Bai18]) are excluded from this overview, as these are rarely used and yield similar performance to LSTM networks [Bec18].

Base Model	**Model**	**Additional cues**	
		Social	**Environmental**
R-MDN	Social LSTM [Ala16]	✓	✗
	ParticleLSTM [Hug18]	✗	✗
	Social Attention [Vem18]	✓	✗
GAN	Social GAN [Gup18]	✓	✗
	SoPhie [Sad19]	✓	✓
	Social Ways [Ami19]	✓	✗
VAE	DESIRE [Lee17]	✓	✓
	LSTM-BMS [Bha18]	✗	✗
	DAG-Net [Mon21]	✓	✗
Transformer	STAR [Yu20]	✓	✗
	TF [Giu21]	✗	✗
	AgentFormer [Yua21]	✓	✗

With reference to the introductory section on sequence modeling (see Chapter 2), each of the base models listed in Table 5.2 provides certain benefits for the task of human trajectory prediction. R-MDN and Transformer models on the one hand are purely regression-based and thus easier to train. Additionally, these models can be used to output an explicit probability distribution over future trajectories by parameterizing a Gaussian mixture model. This, however, comes at the cost of a more difficult approach to generate multi-modal predictions. When parameterizing a Gaussian mixture model, the model can for example be embedded into a particle filter cycle [Hug18]. Another approach construes the trajectory prediction problem as a classification task, where possible future predictions are covered by different classes [Giu21]. Opposed to that, VAE and GAN are probabilistic models providing an implicit model of the data distribution. Both models employ a generator network processing a stochastic input in addition to the encoded observation. As a consequence, these models provide a straightforward approach to generating multi-modal predictions by sampling.

In recent years, an increasing number of approaches emphasize the use of additional cues. The most common additional cues are given by the social context, i.e. neighboring pedestrians, and the environmental context, i.e. static scene elements. Established approaches for incorporating social context are commonly based on either grid-based pooling (e.g. [Ala16, Gup18]), graph attention (e.g. [Vem18, Kos19, Hua19]) or graph convolution (e.g. [Moh20]). Environmental context, on the other hand, is usually given by an encoding of some reference image or video frame generated by a Convolutional Neural Network (CNN).

As the $\mathcal{N}$-Curve model introduced in this thesis provides an alternative model for the underlying base sequence model, such additional cues will not be considered in the following quantitative and qualitative evaluation. Consequently, the performance of the $\mathcal{N}$-Curve model is compared with the aforementioned base sequence models. It should be noted, that when taking away the additional cue components, most state-of-the-art models collapse onto their underlying base sequence models, thus justifying a comparison based on these base sequence models.

5.1.3 Evaluation Setup

In order to provide a comprehensive evaluation of the $\mathcal{N}$-Curve model in the context of long-term human trajectory prediction and in comparison with commonly used sequence models, the tasks of unimodal and multi-modal trajectory prediction are considered. Therefore, the current standard approach to evaluation in the literature is extended by using additional datasets and performance measures, as it does not cover the task of multi-modal trajectory prediction. Further, the evaluation will be performed on each selected dataset in isolation, due to the removal of additional cues for the sequence models. Without such additional cues, long-term prediction requires well-defined decision points tied to static locations in the observed scene, in order to capture relevant walking paths. This is especially true for non-goal-driven prediction approaches, as regarded in this evaluation. As a consequence, pooling together unrelated datasets into a common reference frame cannot be justified and thus datasets are evaluated in isolation.

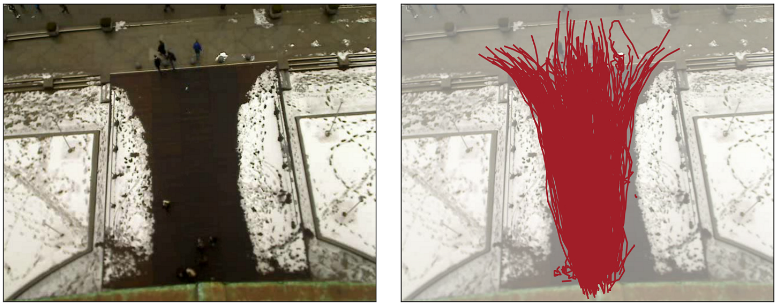
5.1.3.1 Selected Datasets

Under these conditions and in compliance with the current standard evaluation approach, the *biwi:eth*, *biwi:hotel*, *crowds:zara01* and *crowds:zara02* datasets are selected. The *crowds:students* dataset is left out, as it focuses heavily on human-human interaction and as such does not provide well-defined walking paths or decision points. As these datasets provide rather simple scene geometry, additional scenes are taken from the Stanford Drone Dataset. In order to keep the evaluation more concise, scene datasets with varying complexity [Ami20, Hug21] are considered. Thus, the *sdd:bookstore03* and *sdd:hyang00* datasets are included in the evaluation. For these datasets, only pedestrian trajectories are considered[1]. Table 5.3 and Figures 5.2 and 5.3 depict samples from the datasets and relevant statistical details.

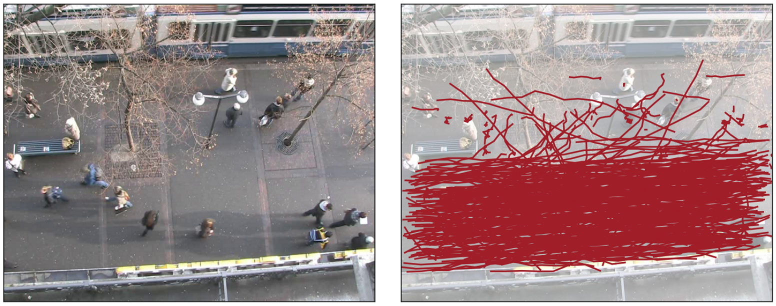
Table 5.3: Statistical details of human trajectory datasets selected for evaluation. It should be noted, that the number of trajectories can deviate from those given in the literature, as trajectories lying outside the image boundary after projection are dismissed. The trajectory length denotes the number of points defining a specific trajectory.

Dataset	**Image Resolution**	**# Trajectories**	**Average Trajectory Length**
biwi:eth	640x480	354	15.47 ± 8
biwi:hotel	720x576	378	17.22 ± 12.16
crowds:zara01	720x576	128	34.27 ± 17.11
crowds:zara02	720x576	204	47.09 ± 72.30
sdd:bookstore03	1322x1079	260	42.37 ± 40.14
sdd:hyang00	1455x1925	285	60.39 ± 44.99

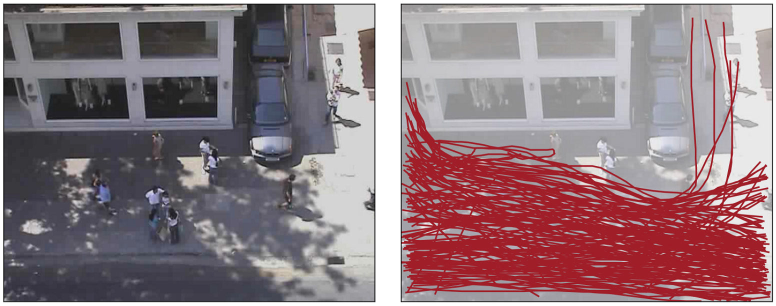
[1] The Stanford Drone dataset provides trajectory data for a multitude of different agent types, including for example pedestrians, bikers and skateboarders.



(a) biwi:eth

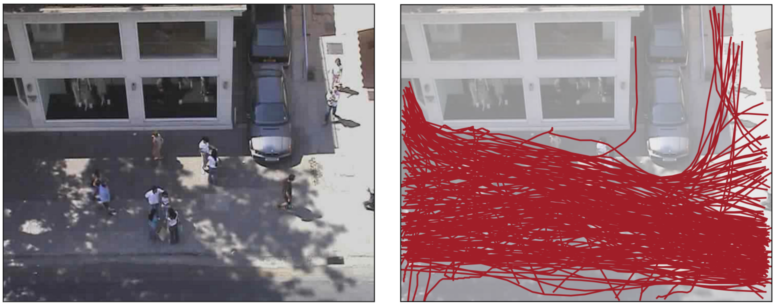


(b) biwi:hotel

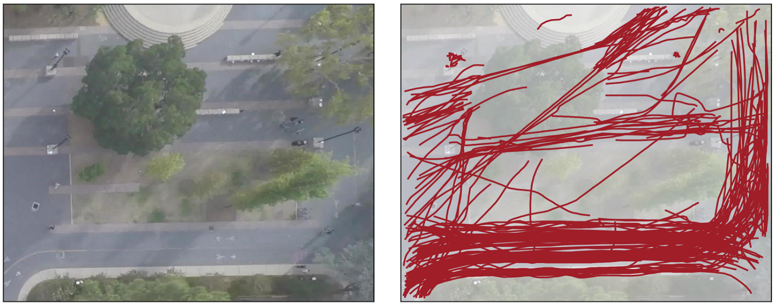


(c) crowds:zara01

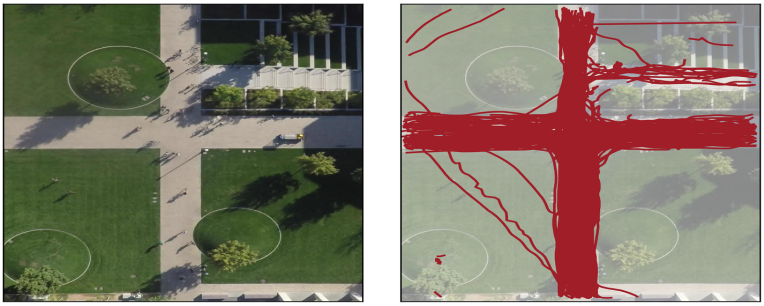
Figure 5.2: Overview of human trajectory datasets selected for evaluation. Sub-figures depict a reference image of the recorded scenery (left) and the overlayed dataset (right). Note: For illustration purposes, the image and data scale is aligned for all datasets, for the actual image resolutions see Table 5.3.



(a) crowds:zara02



(b) sdd:bookstore03



(c) sdd:hyang00

Figure 5.3: Overview of human trajectory datasets selected for evaluation. Sub-figures depict a reference image of the recorded scenery (left) and the overlayed dataset (right). Note: For illustration purposes, the image and data scale is aligned for all datasets, for the actual image resolutions see Table 5.3.

Preprocessing: At first, all datasets originally provided in world space coordinates are projected into image space using homographies. In the literature, the annotation frequency of the datasets is usually set to 2.5 annotations per second, which equals to the annotation rate of the BIWI Walking Pedestrians dataset. Thus, the annotation frequency of all datasets included in the evaluation is adjusted accordingly. Further, the evaluation is conducted on trajectories of a fixed length $N = N_{\text{obs}} + N_{\text{pred}}$ (see also Section 5.1.3.5). Following this, all (sub-)trajectories of a given length are extracted from each respective dataset in order to provide training and test datasets. Trajectories shorter than the given length are not considered for evaluation. As a final data preprocessing step, trajectories of non-moving or slow-moving persons are filtered out, as statistical models are worse in modeling trajectories of slow-moving persons, because their behavior becomes less predictable [Has19]. Thus, the dataset-dependent required minimum speed[1] is calculated heuristically for a given dataset $\mathcal{D}$ containing all M possible (sub-)trajectories of length N:

$$\begin{aligned} s_{\text{min}} &= \frac{\max_i m_{\text{speed}}(i) - \min_i m_{\text{speed}}(i)}{M} \\ &\text{with} \\ m_{\text{speed}}(i) &= \frac{1}{N-1} \sum_{t=2}^{N} \|\mathbf{x}_t^i - \mathbf{x}_{t-1}^i\|_2. \end{aligned} \tag{5.1}$$

Here, $m_{\text{speed}}(i)$ denotes the average speed within the i'th trajectory $\mathcal{X}_i \in \mathcal{D}$.

[1] The average euclidean distance between subsequent trajectory points.

5.1.3.2 Performance Measures

In the standard evaluation approach, the designated performance measures are given by the *Average Displacement Error* (abbrev.: *ADE*) and the *Final Displacement Error* (abbrev.: *FDE*), defined as

$$\mathrm{ADE} = \frac{1}{M \cdot N_{\mathrm{pred}}} \sum_{i=1}^{M} \sum_{t=1}^{N_{\mathrm{pred}}} \|\hat{\mathbf{y}}_t^i - \mathbf{y}_t^i\|_2 \tag{5.2}$$

and

$$\mathrm{FDE} = \frac{1}{M} \sum_{i=1}^{M} \|\hat{\mathbf{y}}_{N_{\mathrm{pred}}}^i - \mathbf{y}_{N_{\mathrm{pred}}}^i\|_2 \tag{5.3}$$

for a given prediction horizon N_{pred}, a set $\boldsymbol{\mathcal{Y}} = \{\mathcal{Y}_1, ..., \mathcal{Y}_M\}$ of M ground truth trajectories $\mathcal{Y}_i = \{\mathbf{y}_1^i, ..., \mathbf{y}_{N_{\mathrm{pred}}}^i\}$ and corresponding predictions $\hat{\mathcal{Y}}_i = \{\hat{\mathbf{y}}_1^i, ..., \hat{\mathbf{y}}_{N_{\mathrm{pred}}}^i\}$ generated by a given prediction model. The ADE is then defined by the average L2 distance between the ground truth and a corresponding predicted trajectory, while the FDE is defined by the L2 distance between the final ground truth and predicted trajectory points after the prediction horizon. In the case of probabilistic sequence models, which generate a predictive distribution $p(\mathbf{y}_{\{1,...,N_{\mathrm{pred}}\}}|\mathbf{x}_{\{1,...,N_{\mathrm{obs}}\}})$, $\hat{\mathcal{Y}}_i$ corresponds to a maximum likelihood prediction given the probabilistic output of the model.

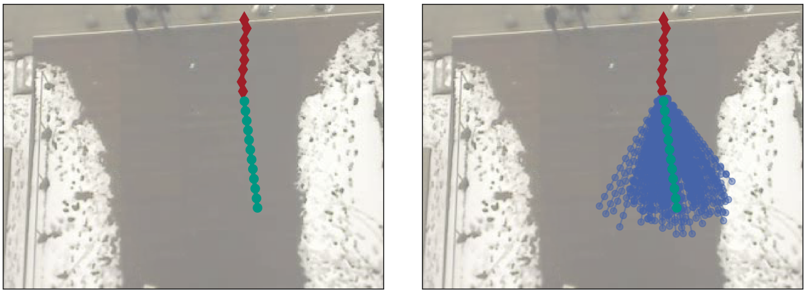
As the ADE and FDE do not provide an adequate measure for assessing the quality of (multi-modal) probabilistic predictions, another performance measure is required for this case. Due to the actual ground truth probability distribution for each time step being unknown, a common choice is given by the *Negative (data) Log-Likelihood* (abbrev.: *NLL*, e.g. [Bha18, Iva19]), defined as

$$\mathrm{NLL} = \frac{1}{M \cdot N_{\mathrm{pred}}} \sum_{i=1}^{M} \sum_{t=1}^{N_{\mathrm{pred}}} -\log p(\mathbf{y}_t^i|\cdot). \tag{5.4}$$

Here, $p(\mathbf{y}_t^i|\cdot)$ denotes the predictive distribution for the t'th trajectory position as generated by the probabilistic sequence model. Note that the conditional part of this distribution is not given explicitly, as it varies between different models (see Section 5.1.3.5). It is worth mentioning, that sometimes an oracle measure (e.g. [Lee17]) is used as a sample-based substitute for the NLL. This measure does, however, introduce another hyperparameter, which is why the NLL is preferred in the context of this evaluation.

5.1.3.3 Baselines

In order to provide reference values for comparison, a simple baseline is given for each performance measure. In the case of the ADE and FDE, a simple prediction model is given by a linear extrapolation calculated from a respective observed trajectory. Here, the relative offset $\delta_i = \mathbf{x}_{N_{\text{obs}}}^i - \mathbf{x}_{N_{\text{obs}}-1}^i$ of the two most recent observations is projected N_{pred} steps into the future, as these positions are assumed to have the most impact on the future trajectory [Sch20a]. In the case of the NLL measure, a sample-based prediction can be generated for each future position by using a *shotgun* approach [Paj18]. In this approach, multiple future trajectories are generated by randomly altering the direction and scale of the relative offset δ_i before projection. The altered offset for each future trajectory is then given by $\mathbf{R}_\alpha \cdot \delta_i \cdot s$ with $\alpha \sim \mathcal{N}(0, \sigma_\alpha)$, $s \sim \mathcal{N}(1, \sigma_s)$ and the matrix $\mathbf{R}_\alpha$ describing a rotation by α degrees. This yields a unimodal probabilistic prediction with a fixed variance for each predicted time step. In the following, $\sigma_\alpha = 15°$ and $\sigma_s = 0.1$ are used. An exemplary prediction using both approaches is depicted in Figure 5.4.



(a) Linear extrapolation **(b)** Shotgun

Figure 5.4: Exemplary predictions generated by a linear extrapolation model and a shotgun model. Predicted samples generated by the shotgun model around the mean linear prediction (green) are depicted in blue.

In addition to these two baselines, a simple LSTM baseline is provided. This mainly has two reasons. On the one hand, the LSTM model is an integral component of multiple sequence models included in the evaluation. On the other hand, it is a widely used baseline next to the linear extrapolation approach.

5.1.3.4 Implementation Details

This section gives a brief overview on implementation details for the sequence models in comparison. The implementations are based on existing approaches, which provide a publicly available implementation. These implementations are adapted to use a common data pipeline. If necessary, components for processing additional cues, such as social context, are removed. The list of approaches the implementations are based on alongside adaptations made is given in Table 5.4.

Table 5.4: Sequence model implementations adapted for use in this evaluation.

Model	**Based on**	**Adaptations**
R-MDN	ParticleLSTM [Hug18], Own implementation	-
GAN	Social GAN [Gup18], Original implementation[a]	Removed social context Data pipeline
VAE	LSTM-BMS [Bha18], Original implementation[b]	PyTorch re-implementation Data pipeline
Transformer	TF [Giu21], Original implementation[c]	Data pipeline

[a] https://github.com/agrimgupta92/sgan
[b] https://github.com/apratimbhattacharyya18/CGM_BestOfMany
[c] https://github.com/FGiuliari/Trajectory-Transformer

The remainder of this section provides some implementation details regarding the prediction models in comparison, including the $\mathcal{N}$-MDN. For each model, respective loss functions, training details and the type of output as generated by the model is depicted. Additionally, a simplified structure illustration is given for each model. These illustrations also serve the purpose of highlighting relevant hyperparameters of each model. The values chosen for each hyperparameter and relevant general information is given at the end of this section.

$\mathcal{N}$-MDN: For the task of human trajectory prediction, the $\mathcal{N}$-MDN is set into a conditional setting. Thus, the MDN's input vector $\mathbf{v}$ (see Section 4.1) needs to hold information about the observed trajectory, in order to condition the MDN's output upon the observation. In accordance with a wide range of human trajectory prediction models, an LSTM network is used for encoding the observed trajectory. The conditional $\mathcal{N}$-MDN is illustrated in Figure 5.5.

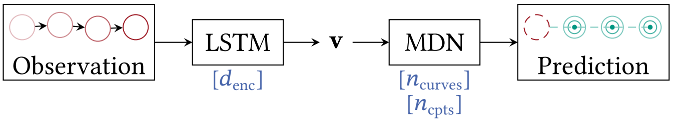


Figure 5.5: Simplified illustration of the conditional $\mathcal{N}$-MDN. Relevant hyperparameters depicted in blue are given by the hidden state dimension d_{enc} of the LSTM encoder, as well as the number of $\mathcal{N}$-Curves n_{curves} output by the MDN. Each generated $\mathcal{N}$-Curve is defined by n_{cpts} Gaussian control points.

As $\mathcal{N}$-Curves model entire trajectories, an $\mathcal{N}$-Curve can be used to either only model the future trajectory, or to model the observed trajectory together with the future trajectory. Both options will be considered in the evaluation.

R-MDN: The SMC-based R-MDN variant used in this evaluation belongs to the group of 1-to-1 sequence models (see Section 2.1.1), processing one trajectory point at a time. As such, the model takes a discrete trajectory point as input and outputs the parameters of a Gaussian mixture distribution modeling the next trajectory point. In order to enable the model to generate a multi-modal prediction, multiple points are sampled from the output distribution and fed back into the model. To prevent exponential growth of samples, subsequent output distributions are combined and re-sampled [Hug18]. A schematic of this model is given in Figure 5.6

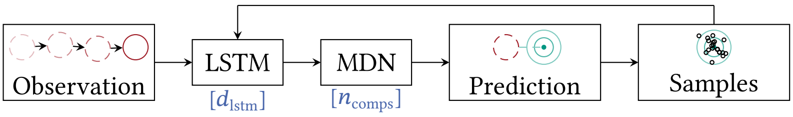


Figure 5.6: Simplified illustration of the SMC-based R-MDN. Relevant hyperparameters depicted in blue are given by the hidden state dimension d_{lstm} of the LSTM network and the number of mixture components n_{comps} generated by the MDN.

During training, the commonly used teacher forcing approach (see Section 2.1.1) is used, as the model generates its prediction sequentially. With the model generating a sequence of conditional mixture distributions, the optimization is based on the negative log-likelihood loss

$$\mathcal{L} = \sum_{t=2}^{N_{\text{obs}}+N_{\text{pred}}} -\log p(\mathbf{x}_t|\mathbf{x}_{t-1}, ..., \mathbf{x}_1), \tag{5.5}$$

for a given training sample trajectory $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_{obs}+N_{\text{pred}}}\}$.

GAN: For applying a GAN in the context of human trajectory prediction, a sequence processing unit must be incorporated into the model. According to [Gup18], a sequence-to-sequence LSTM (see Sections 2.1.1 and 2.2.3) is built into the generator network and another LSTM encoder is built into the discriminator network. The GAN encodes the observed trajectory and then adds a random noise vector to the encoded representation in order to sequentially generate a prediction. By performing multiple passes through the decoder network using different noise vectors, a sample-based distribution of future trajectories is generated. A simplified illustration of the GAN is depicted in Figure 5.7.

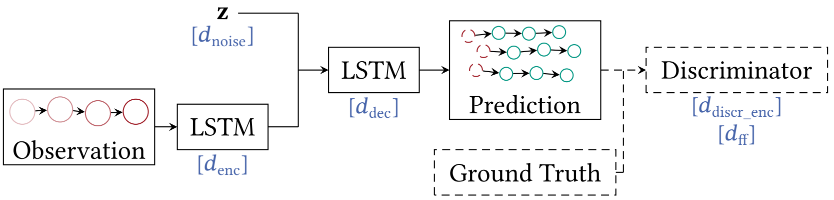


Figure 5.7: Simplified illustration of the GAN. Relevant hyperparameters depicted in blue are given by the dimensionality of the noise vector d_{noise}, the hidden dimension of the generator's LSTM encoder d_{enc} and decoder d_{dec}, as well as the discriminator's LSTM encoder $d_{\text{discr\_enc}}$ and feed forward network d_{ff}. The discriminator part (dashed boxes) are only used during training. The noise vector $\mathbf{z}$ is sampled from $\mathcal{N}(\mathbf{0},\mathbf{I})$.

Opposed to the R-MDN, an auto-conditioning approach (see Section 2.1.1) is employed during training. For the loss calculation, K samples $\{\hat{\mathcal{Y}}_1, ..., \hat{\mathcal{Y}}_K\}$

with $\hat{\mathcal{Y}}_i = \{\hat{\mathbf{y}}_1^i, \ldots, \hat{\mathbf{y}}_{N_\text{pred}}^i\}$ are generated. The loss function consists of a variety loss

$$\mathcal{L}_\text{variety} = \min_i \sum_{t=1}^{N_\text{pred}} \|\hat{\mathbf{y}}_t^i - \mathbf{y}_t\|_2 \tag{5.6}$$

combined with the GAN adversarial loss

$$\mathcal{L} = \mathbb{E}_{\mathbf{y}\sim p_\text{data}(\mathbf{y})}\left[\log D(\mathbf{y})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right]. \tag{5.7}$$

D and G denote the discriminator and generator networks, respectively. The variety loss is intended to encourage the GAN to generate diverse future trajectory predictions for the same observed trajectory.

VAE: Similar to the GAN extension, a sequence-to-sequence LSTM is built into the VAE in order to enable sequence processing. Further, prediction generation works similar to the GAN model by adding a random vector to an encoded representation of an observed trajectory in order to generate multiple future trajectories. A schematic of the VAE is depicted in Figure 5.8.

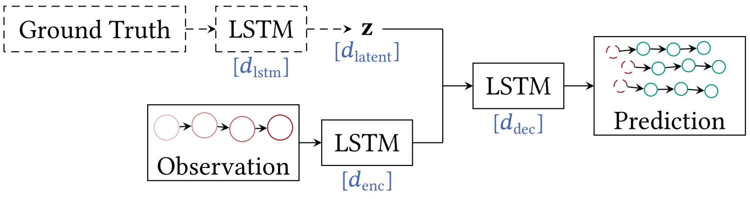


Figure 5.8: Simplified illustration of the VAE. Relevant hyperparameters depicted in blue are given by the dimensionality of the latent space d_latent and of the hidden state of the LSTM encoder d_enc and decoder d_dec. In addition, the hidden state dimension d_lstm of the auxiliary LSTM encoder only active during training gives another hyperparameter. The random vector $\mathbf{z}$ is sampled from $\mathcal{N}(\mathbf{0},\mathbf{I})$ during inference, while the parameters of the Gaussian distribution are determined by the auxiliary LSTM encoder during training.

During training, the LSTM decoder takes the encoded observation with the added random vector as input for every prediction step. In this way, neither teacher forcing, nor auto-conditioning schemes are necessary. At the same time, the model entirely relies on the progressing internal LSTM state to generate an appropriate prediction. Similar to the GAN, K samples are generated for the loss calculation. Here, the loss function consists of a variation of the VAE's standard ELBO (evidence lower bound) loss

$$\mathcal{L} = \max_i \{\log p(\mathcal{Y}|\mathbf{z}_i, \mathcal{X}_{\text{obs}})\} - \log T - D_{\text{KL}}(q(\mathbf{z}|\mathcal{X})||p(\mathbf{z}|\mathcal{X}_{\text{obs}}), \tag{5.8}$$

with $\mathbf{z}_i \sim q(\mathbf{z}|\mathcal{X})$. This *best of many samples* variant of the ELBO contains a variety loss component, comparable to that of the GAN implementation.

Transformer: Although the implementation chosen for this evaluation does not provide a probabilistic prediction model, it is considered in this comparison, as it provides a strong contender to the established LSTM networks built into many human trajectory prediction models. It is an attention-based sequence model, consisting of an encoder, which encodes the entire observed trajectory into a single vector, and a decoder, which sequentially generates one trajectory point at a time, given the encoding. A schematic of this model is depicted in Figure 5.9.

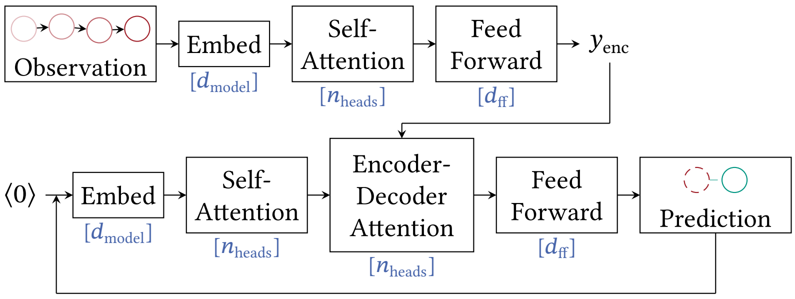


Figure 5.9: Simplified illustration of the Transformer. Relevant hyperparameters depicted in blue are given by the model dimension d_{model}, the number of attention heads n_{heads} and the dimension of the feed forward network d_{ff}. The encoder (top) and decoder (bottom) networks share the same hyperparameters. In the decoder network, $\langle 0 \rangle$ denotes a *start of sequence token* used as input for the initial prediction step.

Similar to the R-MDN, a teacher forcing approach is used during training. As the model generates a single future trajectory, the optimization can be based on the L_2 loss function

$$\mathcal{L} = \sum_{t=1}^{N_{\text{pred}}} \|\hat{\mathbf{y}}_t - \mathbf{y}_t\|_2. \tag{5.9}$$

Hyperparameters: The model's hyperparameters are determined by running a grid search around the parameters provided by the respective authors, using those being most consistent across all datasets. For hyperparameters yielding similar model results, the parameterization given by the respective authors is favored. Output-related parameters for the $\mathcal{N}$-MDN and R-MDN are defined separately in Section 5.1.4. A list of chosen hyperparameters for each model is given in Table 5.5.

Table 5.5: Overview of chosen hyperparameters for each model in comparison.

Model	**Hyperparameters**
$\mathcal{N}$-MDN	$d_{\text{enc}} = 128$
	$n_{\text{cpts}} = 5$
R-MDN	$d_{\text{lstm}} = 256$
GAN	$d_{\text{noise}} = 8$
	$d_{\text{enc}} = 32$
	$d_{\text{dec}} = 48$
	$d_{\text{discr\_enc}} = 32$
	$d_{\text{ff}} = 64$
VAE	$d_{\text{lstm}} = 128$
	$d_{\text{latent}} = 64$
	$d_{\text{enc}} = 48$
	$d_{\text{dec}} = 48$
Transformer	$d_{\text{model}} = 64$
	$n_{\text{heads}} = 1$
	$d_{\text{ff}} = 256$

General Information: All models are trained using a stochastic gradient descent policy and the ADAM optimizer [Kin15], using either mini-batches or the entire training dataset at once (whichever worked best for the respective model). For prediction, $K = 300$ samples are used for the sample-based predictors R-MDN, VAE and GAN.

5.1.3.5 Evaluation Methodology

For achieving a reliable evaluation, a k-fold cross-validation is performed on each dataset, in order to cope with unfavorable random training and test splits. In the following, $k = 5$ folds are performed, as it gives a good trade-off between error bias and variance [Has09]. In compliance with the goal of measuring the raw single dataset performance, all prediction models are re-trained for each fold. As is common practice, prediction models are tasked to predict $N_{\text{pred}} = 12$ steps (4.8 seconds) into the future, given an observation of $N_{\text{obs}} = 8$ steps (3.2 seconds).

For generating a maximum likelihood prediction, the output of the probabilistic prediction models in comparison need to be processed in different ways. For the R-MDN, instead of propagating a set of particles, the mean vector of the highest weighted mixture component is fed back into the model in each time step. As the GAN and VAE models generate a set of sample trajectories, the mean position for each time step is used. Finally, for the $\mathcal{N}$-MDN, the mean curve of the $\mathcal{N}$-Curve with the highest mixture weight is used.

Looking at the NLL measure, which requires a probability density function generated by each prediction model, sample-based output is processed by applying a kernel density estimation [Sco18] using a Gaussian kernel in order to obtain probability density functions for each time step.

5.1.4 Quantitative Results

For the quantitative evaluation, multiple output-related configurations are considered for the R-MDN and $\mathcal{N}$-MDN models, controlling the number of mixture components and the $\mathcal{N}$-MDN model's output mode (see Section 5.1.3.4). The configurations are depicted in Table 5.6.

Table 5.6: Output-related configurations for the R-MDN and $\mathcal{N}$-MDN models in the evaluation.

Configuration	Description
R-MDN$_a$	Outputs a single-component mixture of Gaussians ($n_{\text{comps}} = 1$).
R-MDN$_b$	Outputs a 3-component mixture of Gaussians ($n_{\text{comps}} = 3$).
$\mathcal{N}$-MDN$_a$	Models the observed and future trajectory, and outputs a single $\mathcal{N}$-Curve ($n_{\text{curves}} = 1$).
$\mathcal{N}$-MDN$_b$	Models the observed and future trajectory, and outputs a mixture of 3 $\mathcal{N}$-Curves ($n_{\text{curves}} = 3$).
$\mathcal{N}$-MDN$_c$	Models the future trajectory and outputs a single $\mathcal{N}$-Curve ($n_{\text{curves}} = 1$).
$\mathcal{N}$-MDN$_d$	Models the future trajectory and outputs a mixture of 3 $\mathcal{N}$-Curves ($n_{\text{curves}} = 3$).

Tables 5.7 – 5.12 summarize the results of the quantitative evaluation, using a per dataset 5-fold cross validation and the ADE, FDE and NLL performance measures. Accordingly, respective averaged performance values with corresponding standard deviation considering all 5 folds are depicted. It should be noted, that the performance values are not comparable across datasets, due to different image and ground resolutions. In order to make values comparable, datasets would need to be projected into 3-dimensional world space. Additionally, a re-sampling of trajectory points can be necessary in order to match motion profiles.

Table 5.7: Quantitative results of all approaches on the *biwi:eth* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	ADE	FDE	NLL
Linear	21.80 ± 1.60	47.81 ± 5.41	-
Shotgun	-	-	10.07 ± 2.37
LSTM	13.44 ± 0.97	26.83 ± 2.52	-
Transformer	19.36 ± 2.16	35.81 ± 3.67	-
R-MDN$_a$	26.12 ± 17.42	46.28 ± 32.50	16.62 ± 3.67
R-MDN$_b$	16.11 ± 3.70	28.70 ± 7.72	2893.14 ± 5760.97
VAE	22.33 ± 1.82	37.45 ± 4.91	8.38 ± 0.16
GAN	11.42 ± 2.18	22.26 ± 4.48	1084.14 ± 988.79
$\mathcal{N}$-MDN$_a$	9.51 ± 0.67	17.41 ± 1.13	$\mathbf{7.25} \pm 0.13$
$\mathcal{N}$-MDN$_b$	$\mathbf{9.17} \pm 1.28$	$\mathbf{17.15} \pm 2.89$	7.30 ± 0.30
$\mathcal{N}$-MDN$_c$	10.23 ± 1.07	18.61 ± 2.88	7.48 ± 0.41
$\mathcal{N}$-MDN$_d$	9.87 ± 1.03	18.28 ± 3.40	7.27 ± 0.17

Table 5.8: Quantitative results of all approaches on the *biwi:hotel* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	ADE	FDE	NLL
Linear	26.65 ± 1.24	51.13 ± 3.29	-
Shotgun	-	-	9.87 ± 1.14
LSTM	17.91 ± 2.16	32.93 ± 4.50	-
Transformer	20.42 ± 1.48	34.35 ± 2.95	-
R-MDN$_a$	24.46 ± 4.92	43.52 ± 8.28	15.52 ± 2.93
R-MDN$_b$	19.01 ± 3.77	33.57 ± 6.71	12.20 ± 2.42
VAE	20.03 ± 4.14	35.61 ± 7.90	8.25 ± 0.31
GAN	15.48 ± 1.80	26.38 ± 3.15	20115.70 ± 38614.18
$\mathcal{N}$-MDN$_a$	16.64 ± 3.10	30.36 ± 7.56	8.26 ± 0.48
$\mathcal{N}$-MDN$_b$	15.46 ± 2.03	27.28 ± 3.75	7.96 ± 0.16
$\mathcal{N}$-MDN$_c$	$\mathbf{13.76} \pm 0.94$	$\mathbf{23.82} \pm 1.90$	7.86 ± 0.18
$\mathcal{N}$-MDN$_d$	15.30 ± 1.98	26.60 ± 4.22	$\mathbf{7.85} \pm 0.20$

Table 5.9: Quantitative results of all approaches on the *crowds:zara01* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	**ADE**	**FDE**	**NLL**
Linear	21.69 ± 0.40	46.98 ± 1.39	-
Shotgun	-	-	10.93 ± 0.89
LSTM	23.71 ± 9.30	49.93 ± 24.18	-
Transformer	27.30 ± 1.64	50.04 ± 2.67	-
R-MDN$_a$	20.15 ± 3.25	38.38 ± 6.77	14.76 ± 4.55
R-MDN$_b$	16.86 ± 0.34	31.18 ± 0.72	13.31 ± 2.26
VAE	19.21 ± 0.74	35.49 ± 1.47	8.24 ± 0.19
GAN	$\mathbf{15.59} \pm 0.41$	$\mathbf{30.11} \pm 0.83$	363.05 ± 456.18
$\mathcal{N}$-MDN$_a$	18.48 ± 0.51	35.97 ± 0.74	8.30 ± 0.03
$\mathcal{N}$-MDN$_b$	19.07 ± 0.68	36.49 ± 1.40	8.21 ± 0.06
$\mathcal{N}$-MDN$_c$	16.60 ± 0.44	32.12 ± 0.84	8.04 ± 0.04
$\mathcal{N}$-MDN$_d$	17.71 ± 0.43	34.17 ± 0.84	$\mathbf{7.95} \pm 0.05$

Table 5.10: Quantitative results of all approaches on the *crowds:zara02* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	**ADE**	**FDE**	**NLL**
Linear	28.08 ± 0.33	60.83 ± 0.73	-
Shotgun	-	-	14.97 ± 1.72
LSTM	34.14 ± 22.66	72.47 ± 52.74	-
Transformer	32.17 ± 2.94	58.25 ± 4.76	-
R-MDN$_a$	24.92 ± 1.98	48.18 ± 3.59	11.99 ± 1.37
R-MDN$_b$	21.72 ± 2.59	41.19 ± 4.65	11.67 ± 1.17
VAE	23.44 ± 0.76	43.45 ± 1.42	9.29 ± 0.27
GAN	$\mathbf{20.01} \pm 0.72$	$\mathbf{39.57} \pm 1.35$	26.33 ± 13.40
$\mathcal{N}$-MDN$_a$	22.34 ± 0.92	43.20 ± 2.03	8.54 ± 0.07
$\mathcal{N}$-MDN$_b$	24.74 ± 1.30	47.29 ± 2.90	8.49 ± 0.06
$\mathcal{N}$-MDN$_c$	20.41 ± 1.07	40.39 ± 2.35	8.33 ± 0.07
$\mathcal{N}$-MDN$_d$	21.41 ± 1.03	41.55 ± 2.45	$\mathbf{7.98} \pm 0.03$

Table 5.11: Quantitative results of all approaches on the *sdd:bookstore03* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	ADE	FDE	NLL
Linear	28.39 ± 0.23	60.21 ± 0.67	-
Shotgun	-	-	18.03 ± 4.00
LSTM	27.91 ± 0.95	56.04 ± 1.75	-
Transformer	52.48 ± 14.88	97.00 ± 26.24	-
R-MDN$_a$	50.02 ± 13.19	93.60 ± 23.37	14.28 ± 2.55
R-MDN$_b$	27.58 ± 4.64	50.11 ± 8.56	23.47 ± 23.88
VAE	30.75 ± 1.04	55.93 ± 1.83	8.88 ± 0.09
GAN	$\mathbf{19.10} \pm 0.58$	35.45 ± 0.96	33.18 ± 4.71
$\mathcal{N}$-MDN$_a$	25.24 ± 1.24	48.28 ± 2.80	9.13 ± 0.05
$\mathcal{N}$-MDN$_b$	25.33 ± 2.34	48.69 ± 5.28	9.06 ± 0.09
$\mathcal{N}$-MDN$_c$	22.31 ± 1.15	41.60 ± 1.58	8.89 ± 0.07
$\mathcal{N}$-MDN$_d$	19.43 ± 0.55	$\mathbf{35.37} \pm 1.10$	$\mathbf{8.46} \pm 0.04$

Table 5.12: Quantitative results of all approaches on the *sdd:hyang00* dataset for a prediction time horizon of $N_{\text{pred}} = 12$ time steps (4.8 seconds). ADE and FDE errors are reported in pixels. Lower is better for all performance measures.

Model	ADE	FDE	NLL
Linear	36.25 ± 0.79	75.93 ± 2.09	-
Shotgun	-	-	16.33 ± 1.30
LSTM	40.52 ± 9.44	85.83 ± 23.63	-
Transformer	119.70 ± 0.68	222.18 ± 1.39	-
R-MDN$_a$	90.86 ± 23.89	168.85 ± 43.22	20.43 ± 4.03
R-MDN$_b$	44.18 ± 2.32	84.29 ± 4.17	13.46 ± 0.74
VAE	41.39 ± 2.53	82.21 ± 5.21	9.38 ± 0.09
GAN	$\mathbf{28.84} \pm 1.11$	$\mathbf{57.56} \pm 2.92$	20.48 ± 5.26
$\mathcal{N}$-MDN$_a$	35.83 ± 1.03	72.74 ± 2.49	9.82 ± 0.05
$\mathcal{N}$-MDN$_b$	37.62 ± 2.57	76.31 ± 6.07	9.80 ± 0.07
$\mathcal{N}$-MDN$_c$	34.30 ± 1.11	69.87 ± 2.37	9.63 ± 0.02
$\mathcal{N}$-MDN$_d$	29.68 ± 1.18	58.69 ± 3.13	$\mathbf{9.11} \pm 0.01$

Model Comparison: It can be seen, that in terms of the average and final displacement errors (ADE and FDE), the $\mathcal{N}$-MDN performs on par with the best performing model in comparison, i.e. the GAN model. At the same time, the $\mathcal{N}$-MDN outperforms every other model in terms of the NLL performance measure, with the VAE being the closest contender. It should be noted, that both, the GAN and the R-MDN model, have a tendency to perform worse in terms of NLL, which can be attributed to these model's weakness to mode collapse (see also Section 4.1). This is discussed in more detail in Section 5.1.5. Among the $\mathcal{N}$-MDN variants, the models only modeling the future trajectory seem to outperform those also modeling the observed trajectory. Further, using multiple components appears to be beneficial for more complex datasets. This can be expected, as these datasets contain multiple decision points, leading to multiple distinct possibilities for future trajectories. Lastly, the Transformer model performs notably worse than the LSTM baseline, which indicates that the model is not optimal for the specific task at hand in its original form and thus may require some adaptations.

Baselines: As expected, the linear prediction model performs quite well in terms of the average and final displacement error. This is due to a substantial amount of (sub-)trajectories in each dataset, commonly around 50 to 60 percent [Hug21], representing a constant linear motion. Similarly, the shotgun baseline is only outperformed by 2 out of 4 models, namely VAE and $\mathcal{N}$-MDN. This is due to the baseline's incapability of modeling multiple modes as required for more complex cases. Further, the variance of the prediction is not adapted to the actual location of the observation in the scene, resulting in under- and overestimation. More sophisticated prediction models not suffering from mode collapse are thus able to outperform this baseline.

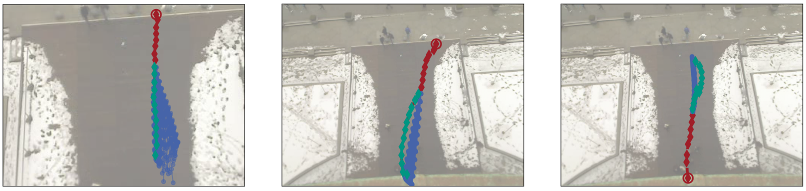
Summary: In summary it can be said, that all probabilistic prediction models perform similar in terms of the presented performance measures, making the choice of model dependent on their respective properties. In this case, the $\mathcal{N}$-MDN may be favored over other models due to it being fully regression-based and thus more stable during training and inference, while at the same time being less computationally heavy during inference.

5.1.5 Qualitative Evaluation

This section provides some insight into the probabilistic models behavior and the evaluation methodology itself. Following this, the problem of mode collapse in R-MDN and GAN models is discussed at first. After that, a qualitative comparison of the probabilistic models considered in this evaluation is provided. Then, the quantitative evaluation of probabilistic prediction models is discussed in more detail, focusing on how to measure probabilistic prediction quality. Finally, different characteristics of the $\mathcal{N}$-Curve model and its $\mathcal{N}$-MDN implementation as discussed in Sections 3.1 and 4.1 are further investigated in the application context using real-world data.

5.1.5.1 R-MDN and GAN: Mode Collapse

The quantitative evaluation revealed that in some cases the R-MDN and GAN models yield large NLL values. While this can be the case because of the model generating bad predictions for certain inputs, this can oftentimes be attributed to both models being vulnerable to mode collapse (MDN: [Mak19], GAN: [Met17]), where the model outputs a narrow prediction due to only generating slight variations of the same sample. Figure 5.10 depicts a well-spread prediction, next to a bad prediction and a prediction indicating a case of mode collapse in order to give a visual example of the latter. In this illustration, exemplary predictions generated by a GAN trained on the biwi:eth dataset are shown.



(a) NLL: 7.70 **(b)** NLL: 2154.72 **(c)** NLL: 7320.78

Figure 5.10: Different predictions (blue) generated by a GAN trained on the biwi:eth dataset, yielding a well-spread (a) and a bad (b) prediction, as well as a prediction indicating a case of mode collapse (c). The observed trajectory is depicted in red. The negative log-likelihood (NLL) is provided for each predicted distribution given the ground truth trajectory depicted in green.

Figure 5.10a gives an example for a spread-out prediction with noticeable bias, which also covers the actual future trajectory. A common failure case is then given in Figure 5.10b, where the model generates a prediction with increasing uncertainty, but misses the actual future trajectory. Finally, 5.10c provides an example of a prediction, which indicates a case of mode collapse. In this example, all samples generated by the model are basically the same, with only minimal variation. While the failure case in Figure 5.10b yields a significantly increased negative log-likelihood, the low variance in the prediction depicted in 5.10c is increasing the error even further.

5.1.5.2 Comparison of Probabilistic Predictions

For a qualitative comparison of the probabilistic prediction models, three examples are taken from the sdd:hyang00 dataset, as it provides a well-structured scenery, where pedestrians mainly stay on designated walking paths. These examples are depicted in Figure 5.11 and cover a range of situations with an increasing number of distinguishable possibilities for future trajectories.

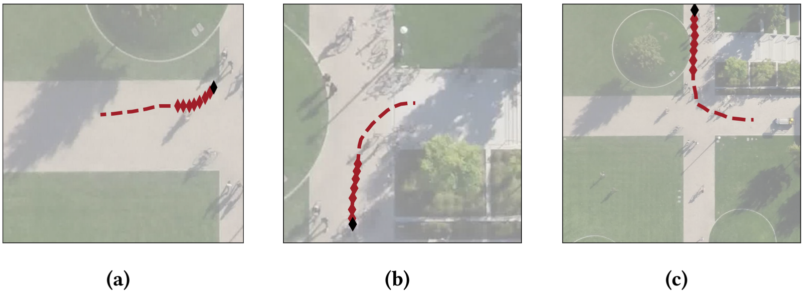
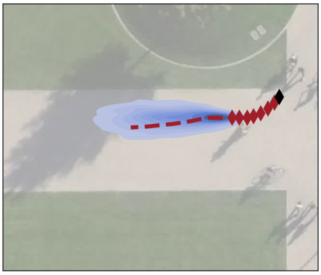


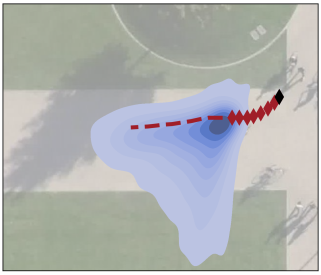
Figure 5.11: Examples (observation depicted with markers) taken from the sdd:hyang00 dataset, providing examples for a straight prediction and multi-modal predictions at different decision points, i.e. junctions, on the pathway.

In the example depicted in Figure 5.11a, prediction of a straight motion is to be expected, as there are no decision points after the observed portion of the trajectory. Example 5.11b provides an observed trajectory, which ends prior to a decision point, where the observed person can either move straight or turn to the right. Although, looking at the data, turning to the right is statistically less likely, the observed trajectory shows a tendency of moving to the right, making both options possible. At last, the example given in Figure 5.11c grants the possibility of a potential trimodal prediction. In this case, however, the number of modes in the prediction is highly dependent on the local neighborhood of the observed trajectory considered during model training, as it influences the target distribution. This is discussed in more detail in Section 5.1.5.3.

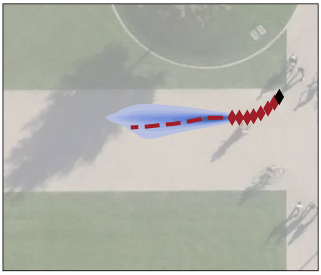
The predictions by each model for each example are depicted in Figures 5.12, 5.13 and 5.14. For the R-MDN and $\mathcal{N}$-MDN models, the R-MDN$_b$ and $\mathcal{N}$-MDN$_d$ variants are used as representatives. Predictions are illustrated as a heatmap calculated from predicted samples of each time step $t \in \{1,...,N_{\text{pred}}\}$.



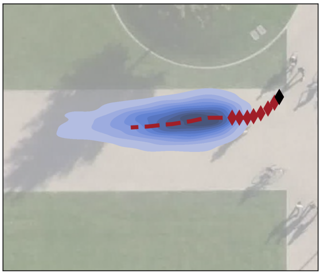
(a) R-MDN



(b) VAE



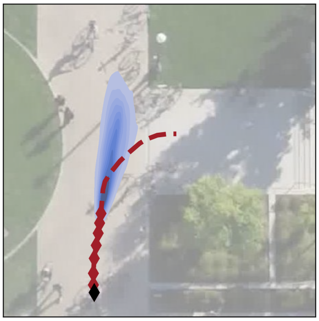
(c) GAN



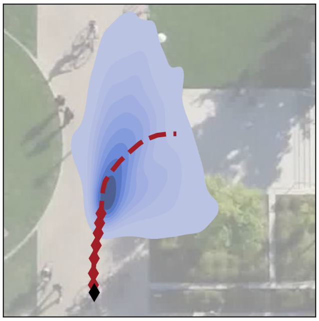
(d) $\mathcal{N}$-MDN

Figure 5.12: Predictions generated by the probabilistic prediction models in comparison for an example trajectory (red markers) taken from the sdd:hyang00 dataset. The actual future trajectory is indicated as a dashed red line.

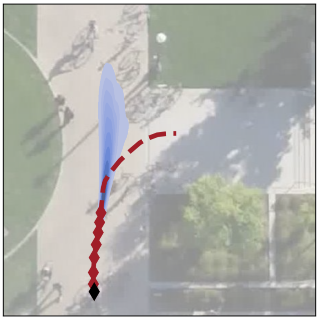
For the first example, all models, with the exception of the VAE, generate a unimodal prediction going straight, as expected. While the R-MDN and the GAN generate comparable results, the $\mathcal{N}$-MDN generated a higher-variance prediction. The VAE on the other hand wrongly predicts another possibility of moving downwards in addition to the straight prediction. This may be caused by the close proximity of the absolute positions to the junction, where moving down is another option. This, in turn, indicates, that the model puts more weight on the observed positions in isolation, rather than to the context.



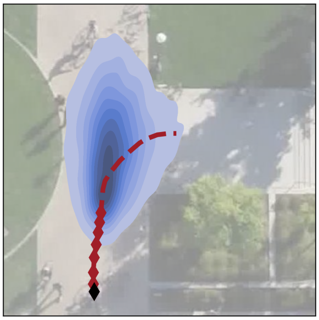
(a) R-MDN



(b) VAE



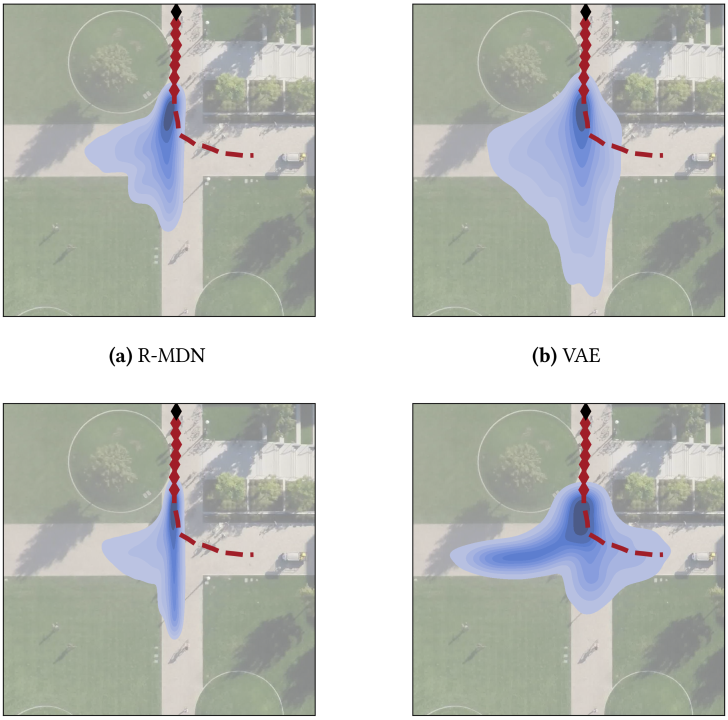
(c) GAN



(d) $\mathcal{N}$-MDN

Figure 5.13: Predictions generated by the probabilistic prediction models in comparison for an example trajectory (red markers) taken from the sdd:hyang00 dataset. The actual future trajectory is indicated as a dashed red line.

The second example shows another form of mode collapse in the predictions of the R-MDN and the GAN, where the statistically less relevant mode is suppressed, thus the model's prediction collapses onto a single mode. Again, the $\mathcal{N}$-MDN and VAE models generate predictions with a higher variance, thereby also covering the possibility of turning to the right. Still, it is visible from the heatmap, that moving straight is the dominant option.



(a) R-MDN **(b)** VAE **(c)** GAN **(d)** $\mathcal{N}$-MDN

Figure 5.14: Predictions generated by the probabilistic prediction models in comparison for an example trajectory (red markers) taken from the sdd:hyang00 dataset. The actual future trajectory is indicated as a dashed red line.

Consistent with the two previous examples, the R-MDN and GAN generate a similar bimodal prediction for the final example, both ignoring the possibility of moving to the right. Ignoring this possibility could be attributed to the observed trajectory being close to the left side of the pathway, making it less likely moving to the right. Combined with the rather low-variance predictions of both models, hinting at a smaller surrounding area being considered for the conditional prediction, trajectories located closer to the right side of the pathway could not have had much influence on the model's output during training. Opposed to that, the VAE and $\mathcal{N}$-MDN models output a trimodal prediction, where the third mode is more defined in the $\mathcal{N}$-MDN's prediction. At the same time, the VAE seems to over-estimate the pedestrian's movement speed when going straight, while the $\mathcal{N}$-MDN rather under-estimates it slightly, when compared to the R-MDN and GAN predictions.

5.1.5.3 Assessing the Quality of Probabilistic Predictions

In the quantitative evaluation section, the negative log-likelihood (NLL) has been used as a measure of the quality of probabilistic predictions generated by the R-MDN, VAE, GAN and $\mathcal{N}$-MDN models. Although the NLL evaluates a predictive distribution generated for a given observation using only a single sample (the actual future trajectory), its application is justified under the assumption, that similar observations result in similar predictive distributions, thus evaluating the entire distribution. At the same time, wrong or superfluous modes in the predictive distribution are not penalized. This is one of the reasons for models, which generate distributions with higher variance, are often scored better. This is also the case for the oracle measure, as it ignores all predictions that are not close to the ground truth [Paj18]. These difficulties in assessing the quality of probabilistic predictions might be a reason for the standard evaluation approach for trajectory prediction models leaving out such a measure, even though most state-of-the-art models are capable of generating probabilistic predictions. Apart from these difficulties, the NLL provides a reliable measure for probabilistic predictions, as it does not require the actual ground truth distribution to be known.

Nonetheless, it would be interesting to compare the probabilistic models under a more sophisticated measure, using an estimation of the conditional ground truth distribution. Thus, this section aims to provide a toy example on a real-world dataset for evaluating the R-MDN, VAE, GAN and $\mathcal{N}$-MDN using the Wasserstein distance [Kol17]

$$W_p(P,Q) = \left(\inf_{\gamma \in \Gamma(P,Q)} \int ||x-y||^p d\gamma(x,y) \right)^{\frac{1}{p}}, \tag{5.10}$$

where P and Q are probability distributions and $\Gamma(P,Q)$ is the set of all joint distributions $\gamma(x,y)$ whose marginals are P and Q, respectively. The Wasserstein distance, originally formulated in the context of optimal transport [Kan39], is preferred over the KL-Divergence [Kul51] and metrics built upon it (e.g. the Jensen-Shannon distance [End03]), as it also takes the metric space into account. As such, it considers the work required to transport the probability mass from a given distribution to a target distribution. Because of this intuition, it is also known as the *Earth Mover's distance* in the 1-dimensional case. For dimensions $d > 1$, there exists no closed form solution for the Wasserstein distance. In this case, a commonly used approximation is given by the sliced Wasserstein distance [Bon15, Kol19].

The following toy example focuses on the evaluation of the endpoint distribution $p(\mathbf{y}_{N_{\text{pred}}}|\cdot)$, N_{pred} steps into the future, generated by each probabilistic prediction model, using the sliced Wasserstein distance. As a first step, the conditional ground truth distribution needs to be determined for each trajectory in the test dataset. This can be achieved by searching the training dataset for trajectories, which are similar to each test dataset trajectory in their respective observed portion. The conditional ground truth distribution can be estimated by applying a Gaussian kernel density estimation, using the endpoints of similar training dataset trajectories. The steps required to determine the conditional ground truth distribution for an exemplary test dataset trajectory $\mathcal{X}_* \in \mathfrak{D}_{\text{test}}$ (Figure 5.15a) are depicted in Figure 5.15.

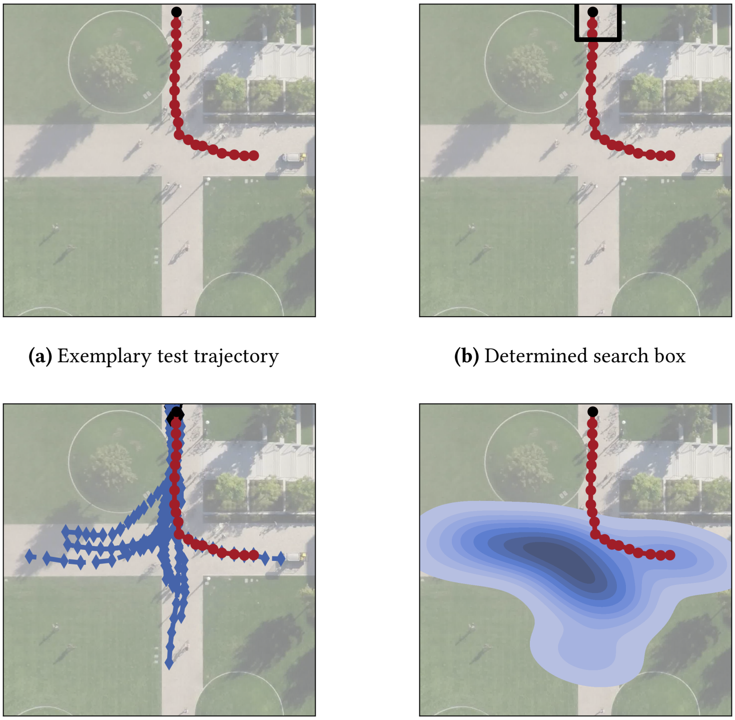

(a) Exemplary test trajectory
(b) Determined search box
(c) Set of similar trajectories
(d) Estimated endpoint distribution

Figure 5.15: Example for estimating a conditional ground truth probability distribution of a trajectory endpoint given a test trajectory's first 8 points as observation. The exemplary test trajectory in this figure starts at the black circular marker. The observed portion of the exemplary test trajectory ends prior to the junction, making a probabilistic prediction of its true endpoint potentially multi-modal.

For finding similar trajectories of $\mathcal{X}_* = \{\mathbf{x}_1^*, ..., \mathbf{x}_{N_{\text{obs}}+N_{\text{pred}}}^*\}$, an axis-aligned rectangular search region around the test trajectory's first point $\mathbf{x}_1^*$ is determined. While the longitudinal expansion e_{long} is calculated to include the first 3 trajectory points, the lateral expansion e_{lat} considers the width of the walking path and is set by hand. This assumes, that there is no bias in the conditional ground truth distribution, if the observed trajectory is closer to either side of the walking path. The resulting search region is depicted in Figure 5.15b.

As a next step, all training dataset trajectories starting within this region are gathered. From this set of trajectories, only those complying with the general movement direction

$$\mathbf{m}^* = \frac{1}{N_{\text{obs}}} \sum_{t=2}^{N_{\text{obs}}} \mathbf{x}_t^* - \mathbf{x}_{t-1}^* \tag{5.11}$$

and speed

$$s^* = \frac{1}{N_{\text{obs}}} \sum_{t=2}^{N_{\text{obs}}} \|\mathbf{x}_t^* - \mathbf{x}_{t-1}^*\|_2 \tag{5.12}$$

of the observed portion of the test trajectory are kept for the distribution estimation. Here, a movement direction deviation θ_{dir} of 10° and a speed deviation Δs of 25% is allowed. The resulting set of similar trajectories $\mathcal{D}_{\text{sim}}^*$ is depicted in Figure 5.15c. Finally, the conditional endpoint distribution estimated from the set of similar trajectories is illustrated in Figure 5.15d.

It has to be noted, that the resulting probability distribution is highly dependent on the considered local neighborhood defined by e_{long} and e_{lat} and the deviation parameters θ_{dir} and Δs. At the same time, it is not quite clear how to choose these values properly. As such, the assumption made for this example might be inaccurate. Further, it is even harder to define these parameters in less structured datasets, making such an evaluation non-viable for large scale evaluations including several datasets. In addition to this aspect, another obstacle is the required amount of trajectories similar to an observed trajectory in question. This is touched upon in more detail later in this section.

Aiming at a comparison of endpoint probability distributions, Figure 5.16 depicts sample-based predictions for the endpoint as generated by the sample-based prediction models.

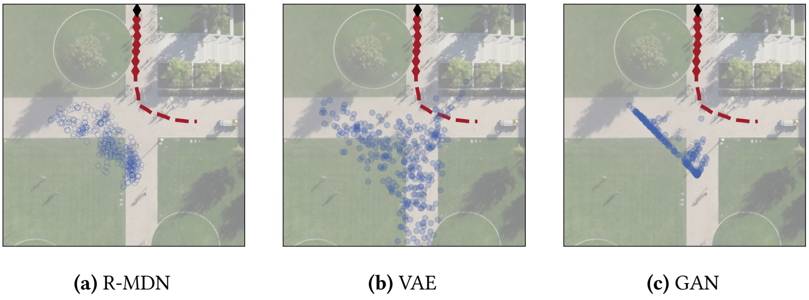
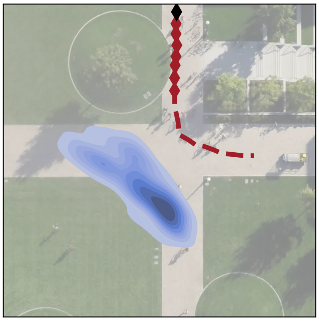
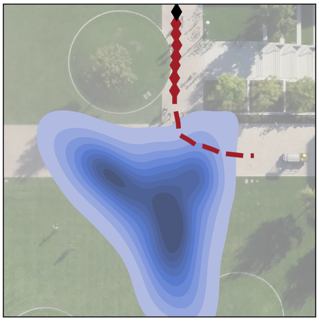
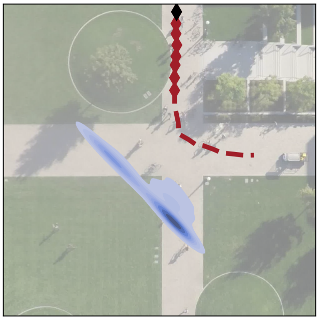


Figure 5.16: Sample-based endpoint predictions as generated by the R-MDN, VAE and GAN models. Regions of high sample-density can be interpreted as modes in the actual probability distribution, whereas the sample-spread indicates the variance.

As described before, a probability density function is estimated from these sample-based predictions by applying a Gaussian kernel density estimation. The resulting probability densities, including the one generated by the $\mathcal{N}$-MDN, are depicted in Figure 5.17. In addition to the probability densities, respective NLL scores given $\mathcal{X}_*$ and Wasserstein distances given the estimated ground truth distribution (see Figure 5.15d) are provided.


(a) R-MDN (NLL: 124.65, Wasserstein: 116.66)


(b) VAE (NLL: 16.61, Wasserstein: 127.24)


(c) GAN (NLL: 200.31, Wasserstein: 144.21)

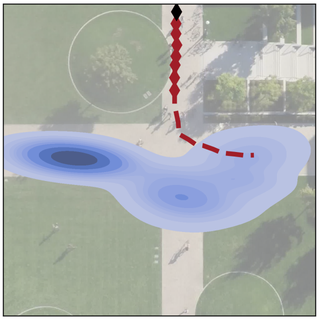

(d) $\mathcal{N}$-MDN (NLL: 11.46, Wasserstein: 91.10)

Figure 5.17: Predicted endpoint probability distributions as generated by the R-MDN, VAE, GAN and $\mathcal{N}$-MDN models. The NLL and Wasserstein distance values between each respective predicted distribution and an estimated ground truth distribution (see Figure 5.15d) are provided.

Following this example on how to calculate the Wasserstein distance for an exemplary trajectory, the same methodology is applied on the first fold test dataset of the sdd:hyang00 dataset. For the evaluation on the whole dataset, a few things have to be noted. First, in practice[1], the Wasserstein distance is calculated on a sample-based representation of the provided probability distributions. Thus, the actual sample-based representations are used for the R-MDN, VAE and GAN models, and an equal amount of samples is drawn from the distribution generated by the $\mathcal{N}$-MDN. Opposed to that, the estimated ground truth distribution is not re-sampled in order to obtain a larger number of samples, as to not distort the actual distribution. Further, test trajectories $\mathcal{X}_i \in \mathfrak{D}_{\text{test}}$ are only considered, if there are at least 30 similar trajectories available, i.e. $|\mathfrak{D}_{\text{sim}}| \geq 30$, according to the aforementioned method. According to [Sil86], at least 19 samples are required in order to calculate an accurate estimation of a bivariate Gaussian density using a kernel density estimation. Due to the ground truth distributions in this evaluation potentially being multi-modal, the number of samples should be increased. At the same time, increasing the number of required samples potentially reduces the number of available test trajectories, when there are not enough similar trajectories available. Using $|\mathcal{X}_{\text{sim}}| \geq 30$, the size of the test dataset is reduced by approximately 45%, thus providing a trade-off between obtaining an accurate ground truth distribution and a reasonable test set size.

Following this, Table 5.13 depicts the mean Wasserstein distance calculated using the sdd:hyang00's first fold test dataset. For comparison, the NLL, as calculated for the quantitative evaluation (see Section 5.1.4), is provided. For completeness, the Wasserstein distance is also provided for the shotgun baseline.

[1] In the context of this thesis, the implementation provided by the *Python Optimal Transport* library [Fla21] is used, which computes a Monte Carlo approximation of the 2-sliced Wasserstein distance.

Table 5.13: Negative Log-Likelihood and Wasserstein distance calculated on the sdd:hyang00's first fold test dataset for each probabilistic prediction model in comparison. In case of the Wasserstein distance, the predicted endpoint distribution is compared with an estimation of the true endpoint distribution for each test trajectory. Lower is better for both measures.

Model	**NLL**	**Wasserstein**
Shotgun	14.97	68.04
R-MDN	14.67	81.25
VAE	9.41	62.68
GAN	30.36	61.47
$\mathcal{N}$-MDN	**9.12**	**51.62**

Looking at the results, the ranking of the probabilistic models is, in parts, consistent with the NLL-based ranking. The shotgun baseline still performed well in this toy example, which is probably due to the presence of many cases, where an unimodal prediction is sufficient. This also supports the shotgun approach' viability as a baseline for probabilistic trajectory prediction. Besides that, a major difference is the GAN performing notably better under the Wasserstein distance, which is likely to be attributed to the Wasserstein distance not penalizing lower variance predictions as is the case for the NLL. Still, the $\mathcal{N}$-MDN outperforms the other models in terms of the quality of the probabilistic prediction. The results further indicate being more stable under the use of the Wasserstein distance.

In summary, this toy example supports the viability of the proposed $\mathcal{N}$-Curve approach in the context of human trajectory prediction. Further, it is suggested, that the NLL can pose a viable performance measure for probabilistic prediction, but it needs to be accompanied with a qualitative evaluation in order to investigate on the reasonability of the predictions in terms of their variance. Finally, in cases, where the ground truth data distribution is available, e.g. when using synthetically generated datasets, the Wasserstein distance may be preferred over the NLL.

5.1.5.4 $\mathcal{N}$-MDN: Additional Examples

This section focuses on few real-world examples, addressing different characteristics of the $\mathcal{N}$-Curve model and its $\mathcal{N}$-MDN implementation, namely the suppression of superfluous mixture components (see Section 4.1.3), modeling different speeds using multiple mixture components and the squeezing effect (see Section 3.1.2). In addition, the most common failure case occurring when using the $\mathcal{N}$-MDN is presented.

Starting off with superfluous component suppression, the prediction of a 3-component $\mathcal{N}$-MDN for an exemplary trajectory taken from the biwi:eth is depicted in Figure 5.18. In this example, it can be seen, that in the model's output, two components have been suppressed, by assigning them a weight of $\pi_k \approx 0$, leaving a single component for the prediction. This complies with the desired behavior in this situation, as all persons moving towards the university come together at the entrance. Additionally, all persons in the dataset move with the same speed on average, making multi-modal prediction only necessary in situations with multiple distinct possible future trajectories.

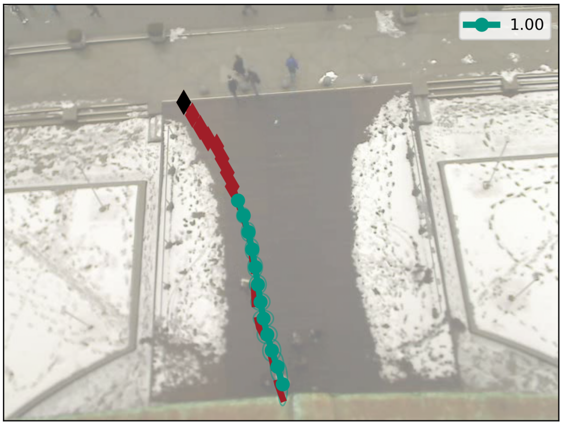


Figure 5.18: Exemplary prediction of a 3-component $\mathcal{N}$-MDN, where 2 $\mathcal{N}$-Curves were suppressed in favor of a single $\mathcal{N}$-Curve responsible for the prediction.

On rare occasions, a low-weight, not well-optimized component appears in a generated prediction. This is the most common failure case when using the $\mathcal{N}$-MDN and is closely connected to the presence of superfluous components. An example for this taken from the crowds:zara02 dataset is given in Figure 5.19. In this example, the component depicted in green unexpectedly branches out and reduces in speed greatly. While both incidents are valid under the presence of other nearby pedestrians being part of a collision avoidance behavior, both actions combined are more likely to be an optimization artifact, where a mixture component receives no more support from training samples from some point onwards during training. Although the model is not exposed to specific multi-agent data, isolated trajectories still reflect this behavior.

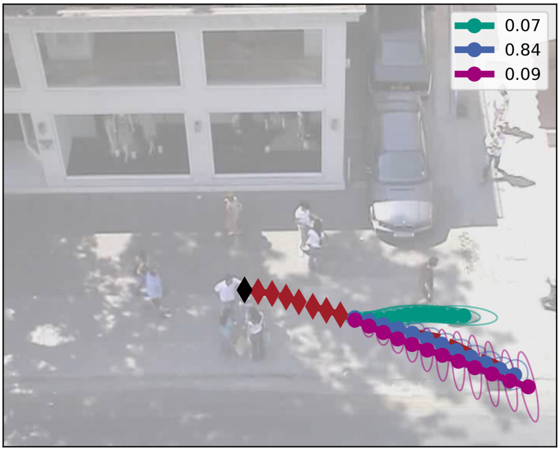


Figure 5.19: Exemplary prediction of a 3-component $\mathcal{N}$-MDN revealing a common failure case of the prediction including a low-weighed not well-optimized mixture component (green).

Besides using multiple mixture components in a prediction for modeling distinct future trajectories, these components can also be used to model similar future trajectories, but at different speeds. Figure 5.20 gives an example taken from the crowds:zara02 dataset, where the $\mathcal{N}$-MDN uses all of its 3 mixture components for modeling different speed versions of the same future trajectory in terms of its movement direction and curvature. As mentioned before, having a higher person density than for example the biwi datasets, it may be more likely to deviate from the average movement speed, in order to prevent collisions with other pedestrians. This, in turn, likely causes the multi-modal prediction covering different speeds.

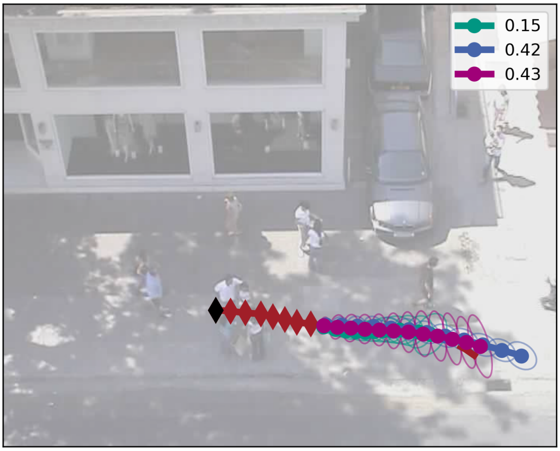


Figure 5.20: Exemplary prediction of a 3-component $\mathcal{N}$-MDN, where each $\mathcal{N}$-Curves in the mixture models another version of the same trajectory, using a different movement speed.

Finally, open questions concluding Section 3.1.2, include whether the squeezing effect is relevant in real-world situations and if the model is able to generate constant variance predictions, when learned from data. Overall, the squeezing effect can be rated as not relevant when using real-world data, which is generally subject to noise. In this case, predicting into the future, the variance usually increases with each time step. With respect to the constant variance case, Figure 5.21 provides an example taken from the sdd:hyang00 dataset, where the $\mathcal{N}$-MDN outputs an unimodal prediction, which maintains almost constant lateral variance. In this example, this is achieved by slowly morphing an almost circular covariance ellipse towards a covariance ellipse with increased longitudinal variance. Here, the increase of the longitudinal variance is a way of coping with uncertainties about the actual speed of the observed trajectory. Observing both, the effect of an increasing longitudinal variance over time while maintaining near constant lateral variance, show the low relevance of the squeezing effect for real-world applications.

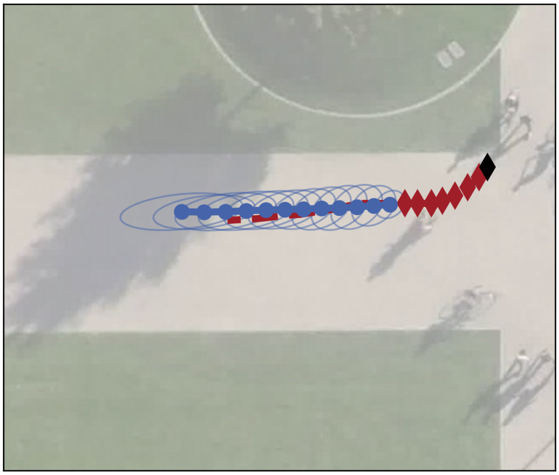


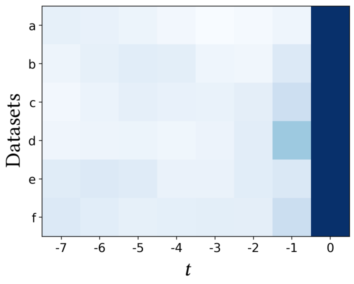
Figure 5.21: Exemplary prediction of a $\mathcal{N}$-MDN maintaining a near constant lateral variance while increasing longitudinal variance at the same time.

5.1.5.5 Implicit Input Attention

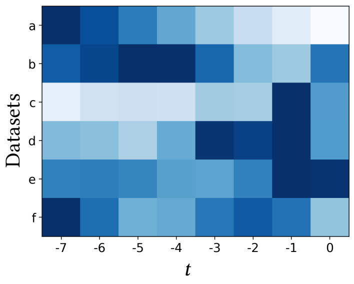
With the prediction generated by an $\mathcal{N}$-MDN being based on just the N_{obs} observed positions, this section investigates the influence of each input on the generated prediction, with respect to their position within the observed sequence. Recall, that an $\mathcal{N}$-MDN prediction is given in terms of a curve weight distribution π, as well as a set of control point mean vectors $\boldsymbol{\mu}$ and covariance matrices $\boldsymbol{\Sigma}$, which yield the predicted mean curve and the region of uncertainty around it. Following this, it is especially interesting to see, if different parts of a given input sequence are considered for generating π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

As there is no attention mechanism explicitly built into the $\mathcal{N}$-MDN architecture, the model's attention to different parts of a given sequence can be calculated using the gradient of each generated output with respect to the inputs. Using *PyTorch*, its *autograd* module can be used for this, which calculates the respective gradients by performing a backward pass through the network given the generated output. Figure 5.22 depicts the resulting gradient-based implicit attention maps for each dataset in the evaluation. For each dataset, gradient magnitudes are averaged for each output, i.e. π, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. As in previous sections, the $\mathcal{N}$-MDN$_d$ variant is considered.

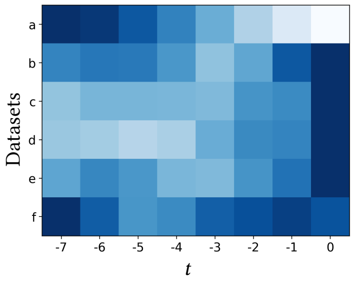
Figure 5.22a, 5.22b and 5.22c depict the input attention on a per dataset basis for each of the model outputs separately. Figure 5.22a reveals that for generating the mean vectors $\boldsymbol{\mu}$, the most important input is given by the last observed position with an additional, but weaker, contribution by the second last element. This observation is in line with the findings given in [Sch20a]. Opposed to that, for determining the weights and covariance matrices, a mix of multiple observations spread across the entire observed sequence is considered. The choice of which observations to rely on varies between datasets. This is most likely due to random effects during training and the model being trained for each dataset individually. Especially in the case of the covariance matrix, it makes sense to incorporate multiple observations from a given sequence, as a noise estimation can be expected to be more accurate using more data samples. Figure 5.22c depicts the input attention for each model output averaged over all datasets and summarizes the aforementioned findings.



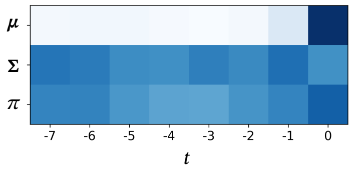
(a) Mean vectors $\boldsymbol{\mu}$



(b) Covariance matrices $\boldsymbol{\Sigma}$



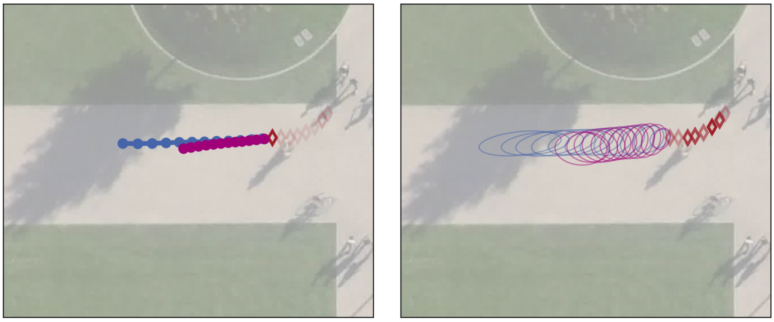
(c) Weighting π



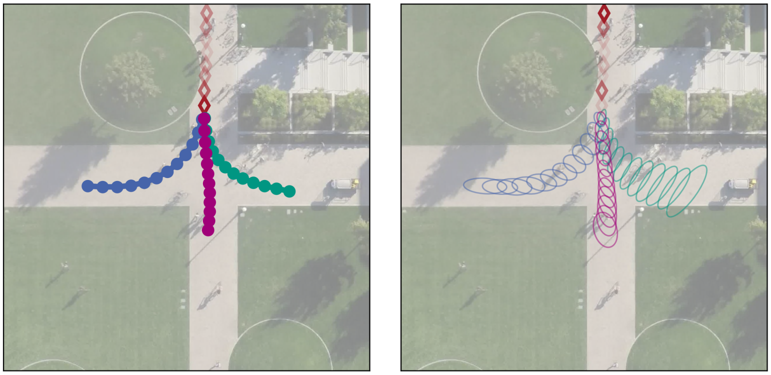
(d) Overall average

Figure 5.22: Heatmap visualization of the models attention to each of the $N_{\text{obs}} = 8$ observations when generating predictions in terms of curve weights π, mean vectors $\boldsymbol{\mu}$ and covariance matrices $\boldsymbol{\Sigma}$. Time steps t are given relative to the last observation at $t = 0$. Inputs with no influence on the output are depicted in white and inputs with the most influence (per row) are given in dark blue. In figures (a) – (c), the datasets biwi:eth, biwi:hotel, crowds:zara01, crowds:zara02, sdd:bookstore03 and sdd:hyang00 are depicted along the y-axis (a – f).

To accompany the heatmap visualizations, Figure 5.23 depicts the influence of each observation on respective mean vectors and covariance matrices for two exemplary input sequences. Both examples support the observation, that for generating the mean vectors, the most recent observations are the most important. Further, the covariance matrices are determined using several observations spread across the observed sequence.



(a)



(b)

Figure 5.23: Visualization of the influence of observations along a given input sequence for two exemplary sequences. More important observations for determining the mean vectors (left) and covariance matrices (right) are depicted with higher color intensity.

5.1.6 Summary

In summary, this section gave a detailed overview of the human trajectory prediction task, commonly used datasets and state-of-the-art prediction models. The latter are most commonly variants of Recurrent Mixture Density Networks, Variational Autoencoders and Generative Adversarial Networks. This overview was followed by an extensive evaluation of the $\mathcal{N}$-Curve model in comparison to these commonly used models, using different performance measures and corresponding baselines. The performance measures include the average and final displacement error for measuring the performance of maximum likelihood predictions, as well as the negative log likelihood for assessing the probabilistic prediction performance. In this evaluation, the $\mathcal{N}$-Curve model shows competitive results, outperforming most other generic probabilistic sequence models in the comparison.

5.2 Human Motion Prediction

The primary goal of this section is the evaluation of the scalability of the $\mathcal{N}$-Curve model to higher-dimensional data. For this, the task of human motion prediction is considered. Note that in the literature, human trajectory prediction (see the previous Section 5.1) is sometimes confused with human motion prediction. To clarify, human trajectory prediction is concerned with human movement along a trajectory through an observed scene based on observed 2- or 3-dimensional locations. Opposed to that, human motion prediction targets the motion of the human body when performing different actions and is based on sequences of human poses.

Thus, in human motion prediction, a prediction model is tasked to generate a sequence of human poses resembling some action performed by an observed subject. The generation is thereby conditioned on a given initial observation of the performed action. Each element in the sequences to process is given by a human pose. Such human poses are commonly represented as a set of 3D joint positions, which can be connected via a skeleton definition. The number of 3D joints describing a human pose varies between datasets. To give an example, in the Human3.6m dataset [Ion13], a human pose is described by 32 3D joints, yielding a 96-dimensional vector.

5.2.1 Datasets

Looking at datasets which provide human pose sequences, the most commonly used ones include the *CMU mocap database*[1], the *Human3.6m dataset* (abbrev.: h3.6m, [Ion13]) and the *NTU RGB+D dataset* [Sha16]. Among these datasets, for the task of human motion prediction the h3.6m dataset is the most widely used. This is due to existence of a standard evaluation protocol, allowing to re-use previous results of different approaches tackling the prediction task. Some details on the h3.6m dataset are given in Table 5.14. Figure 5.24 depicts an example of a pose sequence for the *walking* action taken from the h3.6m dataset.

Table 5.14: Details of the Human3.6m dataset.

The Human3.6m dataset	
Number of subjects	11
Pose representation	3D joint positions and angles (32 joints)
Recorded actions	directions, discussion, eating, seated activities, greeting, taking photo, posing, making purchases, smoking, waiting, walking, sitting on chair, talking on the phone, walking dog, walking together
Sample rate	50Hz

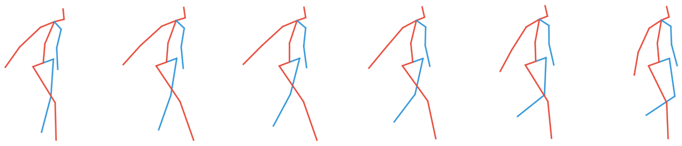


Figure 5.24: Exemplary human motion sequence taken from the Human3.6m dataset. The left arm and leg are depicted in blue. For illustration purposes, the sample rate is reduced to 12.5Hz.

[1] http://mocap.cs.cmu.edu/

5.2.2 Evaluation Protocol

For enabling a repeatable and comparable evaluation, approaches presented in the literature commonly follow the standard evaluation protocol provided in [Fra15] and [Jai16]. According to this protocol, multiple data pre-processing steps are performed prior to training and evaluation. First, the pose representation as provided in the h3.6m dataset is converted into an exponential map representation of each joint using a specific pre-processing of global translation and rotation as specified in [Tay07]. Following the change in representation, the data is standardized by subtraction of the mean and division by the standard deviation along each dimension. Then, dimensions with constant values are dropped from the representation. The resulting pose representation then consists of 17 joints and a global translation component, yielding a 54-dimensional representation. Finally, the sequence sample rate is reduced to 25Hz.

Using the pre-processed data, training is performed on a subset of actions using subjects S1, S6, S7, S8, S9 and S11. The action subset includes *walking, eating, smoking, discussion, directions, greeting, phoning, posing, purchases, sitting, sittingdown, takingphoto, waiting, walkingdog, walkingtogether.* The test dataset then contains actions performed by subject S5, collecting 8 sub-sequences of specific actions using a fixed seed. The considered set of actions in the test dataset is restricted to the representative actions *walking*, *eating*, *smoking* and *discussion*. For prediction, a given model is tasked to predict up to $N_{\text{pred}} = 10$ time steps (400 milliseconds) into the future, given an observation of $N_{\text{obs}} = 50$ time steps (2 seconds) of a given action. The prediction performance is then measured in terms of the mean angle error[1]

$$MAE = \frac{1}{M} \sum_{i=1}^{M} ||\hat{\mathbf{y}}_t^i - \mathbf{y}_t^i||_2, \tag{5.13}$$

[1] Using an euler angle representation, which can be calculated from the exponential map representation

calculated after 80, 160, 320 and 400 milliseconds, using M samples of the same action. With a sample rate of 25Hz, this corresponds to $t = 2$, $t = 4$, $t = 8$ and $t = 10$ time steps. This restriction to short-term prediction[1] is introduced due to the stochasticity of human motion preventing a quantitative evaluation of longer time horizons [Fra15].

5.2.3 Baselines and Comparison Models

For comparison, there are several commonly used simple and neural network-based baselines. Common simple baselines are given by the *Zero-velocity* model [Mar17], which constantly predicts the last observation, and a running average approach of the last n observed poses. The running average approach will be abbreviated as *Run. avg. n*. Regarding neural network-based baselines, the most prevalent models include the *LSTM-3LR* [Fra15], the *ERD* [Fra15] and the *SRNN* [Jai16] models. While the LSTM-3LR is a three-layer LSTM network, the ERD and SRNN models are more tailored towards learning a meaningful representation of a given observation to base their prediction on. The *Encoder-Recurrent-Decoder* model (abbrev.: ERD) is a type of RNN that combines representation learning with learning temporal dynamics. To achieve this, the input to the RNN is encoded into a representation, where learning pose dynamics is easier. The *Structural RNN* (abbrev.: SRNN) on the other hand aims to incorporate semantic knowledge about the data structure into the model architecture. Following the fact, that a sequence of poses can be represented by a (manually designed) spatio-temporal graph, the SRNN provides an approach for transforming such a graph into a feedforward mixture of RNNs.

Beyond these common baselines, recent approaches to human motion prediction are commonly based on either Recurrent Neural Networks (e.g. [Gho17, Gop19]), (sequence-to-sequence) Generative Adversarial Networks (e.g. [Gui18, Kun19]) or Graph Neural Networks (abbrev.: GNN, e.g. [Mao19,

[1] Short-term prediction is defined as predicting less than 560ms into the future.

Li20]). The latter thereby consider the actual configuration of the joints according to the underlying skeleton. For the following quantitative evaluation, representatives for each base architecture are selected.

In the group of Recurrent Neural Networks, besides the baselines presented above, another interesting approach is given by the *QuaterNet* model [Pav18]. As opposed to the other approaches in this comparison, which regress joint rotations using the exponential map representation, the QuaterNet model uses a quaternion-based representation of joint rotations. This change in representation targets the issue of discontinuities, which can occur when using an exponential map representation. Further, joint position errors are considered in the training loss function, trying to incorporate the varying impact of joints on the pose.

Looking at the GAN-based approaches, the *Adversarial Geometry-aware Encoder-Decoder* (abbrev.: AGED, [Gui18]) and *Bidirectional 3D Human Motion Prediction GAN* (abbrev.: BiHMP-GAN [Kun19]) models are considered in the quantitative evaluation. Both of these models rely on a seq2seq RNN which is embedded in an adversarial training approach. The BiHMP-GAN model, on the one hand, incorporates a pose embedding, comparable to the ERD, and uses a bidirectional RNN architecture [Sch97] in its discriminator network. On the other hand, the AGED model exploits the intrinsic geometric structure of 3D rotations during training of the generator, by using a geodesic distance between joint rotations. This is opposed to the common approach of using an euclidean distance between predicted and ground truth joint angles.

Finally, among GNN-based approaches, the *Traj-GCN* [Mao19] and the *Adversarial GCN* (abbrev.: A-GCN, [Cui20]) models are included in the evaluation. Both models are based on *Graph Convolutional Networks* (abbrev.: GCN, [Kip17]) and thus encode spatial dependencies in human poses by treating a pose as a generic graph. The Traj-GCN model proposes to work in trajectory space instead of the traditionally used pose space, in order to encode temporal information. Further, graph connectivity is learned automatically during training. Similar to the second aspect, the A-GCN learns the connection strength between nodes in the graph. Following this, poses are represented as a dynamic graph, where natural connections between joint pairs are exploited explicitly. Beyond that, links between geometrically separated joints can be learned implicitly. Using an adversarial training approach, the A-GCN could also be put into the group of GAN-based approaches, thus blurring the line between the groups. An overview of the presented baseline and comparison models is depicted in Table 5.15.

Table 5.15: Overview of the baseline and comparison models considered in this evaluation.

Model	Type	Year
Zero-velocity [Mar17]	Simple baseline	2017
Run. avg. n [Mar17]	Simple baseline	2017
LSTM-3LR [Fra15]	Neural baseline	2015
ERD [Fra15]	Neural baseline	2015
SRNN [Jai16]	Neural baseline	2016
QuaterNet [Pav18]	RNN	2018
AGED [Gui18]	GAN	2017
BiHMP-GAN [Kun19]	GAN	2019
Traj-GCN [Mao19]	GNN	2019
A-GCN [Cui20]	GNN	2020

5.2.4 $\mathcal{N}$-Curve Model Setup

Following the common approach of processing pose sequences in an exponential map representation, training and prediction in the $\mathcal{N}$-MDN will be based on this representation. With a focus on scalability, the generic version of the model is used as in the human trajectory prediction evaluation (Section 5.1). Therefore, model extensions tailoring the model towards the task of human motion prediction are disregarded. Further, the use of a more domain-specific loss function, i.e. the geodesic loss function proposed in [Gui18], is also not considered. This is due to the fact, that it cannot be easily integrated into the log-likelihood loss function for learning the mean vectors and covariance matrices jointly.

For the evaluation, two variants of the $\mathcal{N}$-MDN generating unimodal predictions are employed. For the first variant, denoted as $\mathcal{N}$-MDN$_a$, the hyperparameters (see also Figure 5.5) are set as $d_{\text{enc}} = 1024$, $n_{\text{curves}} = 1$ and $n_{\text{cpts}} = 4$.

A 4-layer LSTM is used as the sequence encoder. Further, the $\mathcal{N}$-MDN is parameterized to generate diagonal covariance matrices only. This is common practice due to covariance estimation becoming more difficult in higher-dimensional data [Ha18, Raz20]. Mixture Density Networks are especially afflicted by this, where the estimation of higher-dimensional covariance matrices contributes to numerical instabilities [Rup17, Mak19]. Still, with the $\mathcal{N}$-MDN processing full pose representations, it can be expected that dependencies between dimensions are captured implicitly, regardless of the generated $\mathcal{N}$-Curve only providing diagonal covariance matrices. In order to provide more expressive covariance matrices, an additional variant of the $\mathcal{N}$-MDN is evaluated. This variant is denoted as $\mathcal{N}$-MDN$_b$ and generates $\mathcal{N}$-Curves with sparse covariance matrices, which model inter-joint correlations and the correlations between the dimensions of the global translation. With each joint and the global translation being represented by a 3-dimensional (sub-)vector within the pose representation, resulting covariance matrices consist of 18 $3x3$ block matrices. To prevent numerical instabilities, $\mathcal{N}$-MDN$_b$ is realized as an ensemble of $\mathcal{N}$-MDNs, where each network models the 3 dimensions of the global translation or a single joint, respectively. The outputs of each network in the ensemble are then combined into the targeted 54-dimensional $\mathcal{N}$-Curve. In this case, all joints are now modeled independently. Each $\mathcal{N}$-MDN in the ensemble is parameterized with $d_{\text{enc}} = 128$, $n_{\text{curves}} = 1$ and $n_{\text{cpts}} = 4$, using a 1-layer LSTM as sequence encoder.

5.2.5 Quantitative Results

This section provides the quantitative results of the $\mathcal{N}$-MDN variants and the comparison models on the test dataset according to the standard protocol. The results for the simple and neural baselines are taken from [Mar17]. For the comparison models, the results are gathered from their respective papers. Thereby, only the overall best performing model variant, if there are any, is considered. The joint angle errors are reported in Tables 5.16 and 5.17. It should be noted, that the error standard deviation is commonly not reported in the literature, thus the standard deviation is left out for all models in comparison.

Table 5.16: Mean angle error (lower is better) for short-term human motion prediction on the Human3.6m dataset for the representative actions *walking* and *eating*. Commonly used simple and neural baselines are provided at the top and recent domain-specific models in the middle.

	walking				eating			
milliseconds	80	160	320	400	80	160	320	400
Zero-velocity	0.39	0.68	0.99	1.15	0.27	0.48	0.73	0.86
Run. avg. 2	0.48	0.74	1.02	1.17	0.32	0.52	0.74	0.87
LSTM-3LR	0.77	1.00	1.29	1.47	0.89	1.09	1.35	1.46
ERD	0.93	1.18	1.59	1.78	1.27	1.45	1.66	1.80
SRNN	0.81	0.94	1.16	1.30	0.97	1.14	1.35	1.46
QuaterNet	0.21	0.34	0.56	0.62	0.20	0.35	0.58	0.70
AGED	0.22	0.36	0.55	0.67	0.17	0.28	0.51	0.64
BiHMP-GAN	0.33	0.52	0.63	0.67	0.20	0.33	0.54	0.70
Traj-GCN	0.18	0.31	0.49	0.56	0.16	0.29	0.50	0.62
A-GCN	0.16	0.29	0.46	0.57	0.16	0.27	0.49	0.64
$\mathcal{N}$-MDN$_a$	0.25	0.41	0.64	0.73	0.22	0.36	0.61	0.75
$\mathcal{N}$-MDN$_b$	0.21	0.35	0.60	0.72	0.20	0.35	0.59	0.72

Table 5.17: Mean angle error (lower is better) for short-term human motion prediction on the Human3.6m dataset for the representative actions *smoking* and *discussion*. Commonly used simple and neural baselines are provided at the top and recent domain-specific models in the middle.

	smoking				discussion			
milliseconds	80	160	320	400	80	160	320	400
Zero-velocity	0.26	0.48	0.97	0.95	0.31	0.67	0.94	1.04
Run. avg. 2	0.30	0.52	0.99	0.97	0.41	0.74	0.99	1.09
LSTM-3LR	1.34	1.65	2.04	2.16	1.88	2.12	2.25	2.23
ERD	1.66	1.95	2.35	2.42	2.27	2.47	2.68	2.76
SRNN	1.45	1.68	1.94	2.08	1.22	1.49	1.83	1.93
QuaterNet	0.25	0.47	0.93	0.90	0.26	0.60	0.85	0.93
AGED	0.27	0.43	0.82	0.84	0.27	0.56	0.76	0.83
BiHMP-GAN	0.26	0.50	0.91	0.86	0.33	0.65	0.91	1.00
Traj-GCN	0.22	0.41	0.86	0.80	0.20	0.51	0.77	0.85
A-GCN	0.20	0.38	0.79	0.82	0.19	0.45	0.72	0.81
$\mathcal{N}$-MDN$_a$	0.26	0.49	0.95	0.93	0.30	0.65	0.94	1.04
$\mathcal{N}$-MDN$_b$	0.26	0.48	0.91	0.91	0.33	0.71	0.99	1.04

Looking at the results, the $\mathcal{N}$-MDN variants generally outperform the simple, yet strong, baselines in this task. The neural baselines, which are themselves more generic models, similar to the $\mathcal{N}$-MDN, are outperformed by a large margin. Expectedly, being a generic model, the $\mathcal{N}$-MDN falls a little bit behind in comparison with the domain-specific models.

Comparing both variants of the $\mathcal{N}$-MDN, the results are very similar. Variant a performs slightly better on the *discussion* action, whereas variant b performs slightly better on the other actions. Differences between the predictions generated by both variants are further detailed in the qualitative results Section 5.2.6.

In summary, the $\mathcal{N}$-Curve models performs quite well on the given task, despite being a more generic probabilistic sequence model. As such, the model is not specifically built to capture the underlying tree-like structure of the data, nor does it employ a specialized loss function. An additional culprit contributing to less accurate predictions may be given by the smoothing behavior of the model, which is examined in more detail in Section 5.2.6. Finally, the quantitative evaluation shows that the model scales well to modeling higher-dimensional data.

5.2.6 Qualitative Results

For the qualitative evaluation, exemplary predictions generated by the $\mathcal{N}$-MDN variants are examined. Thereby, differences between both variants and some insight into the behavior of the model is provided. Following this, Figures 5.25 – 5.28 depict exemplary predictions for all four actions in the test dataset for both model variants.

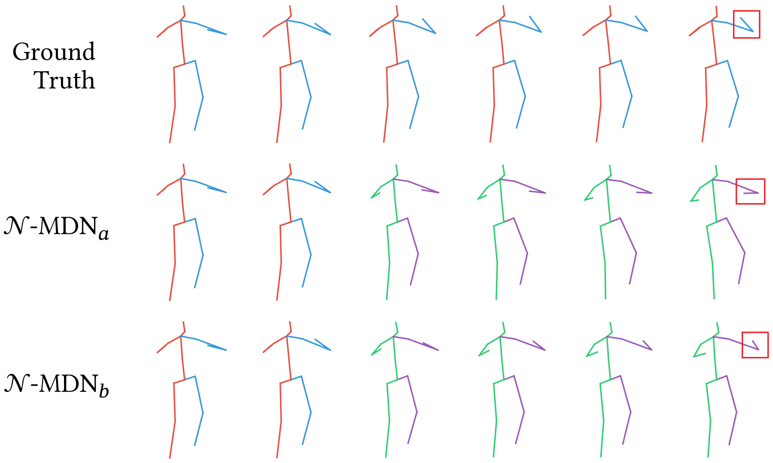


Figure 5.25: Qualitative comparison of predictions generated by both variants of the $\mathcal{N}$-MDN on the *discussion* action. $\mathcal{N}$-MDN$_a$ generates diagonal covariance matrices. $\mathcal{N}$-MDN$_b$ generates sparse covariance matrices, which model inter-joint correlations. For illustration purposes, the sample rate is reduced to 12.5Hz. For each prediction, the last 2 observed poses are depicted together with an prediction of 320 milliseconds (4 time steps) into the future. The full ground truth sequence of poses is depicted in the first row. The left arm and leg are depicted in blue (ground truth) or purple (prediction), respectively. Regions of interest are highlighted.

Looking at the *discussion* action depicted in Figure 5.25, a noticeable difference between both $\mathcal{N}$-MDN variants can be observed looking at the movement of the left arm. While $\mathcal{N}$-MDN$_a$ predicts a downward movement, $\mathcal{N}$-MDN$_b$ generates a more accurate prediction. Apart from that, both variants generate the same wrong movement for the right arm, indicating that the actual motion deviates from the average motion considering similar cases.

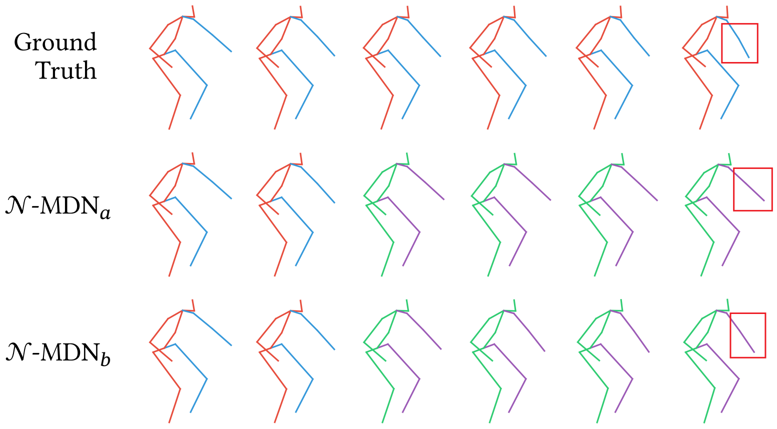


Figure 5.26: Qualitative comparison of predictions generated by both variants of the $\mathcal{N}$-MDN on the *eating* action. $\mathcal{N}$-MDN$_a$ generates diagonal covariance matrices. $\mathcal{N}$-MDN$_b$ generates sparse covariance matrices, which model inter-joint correlations. For illustration purposes, the sample rate is reduced to 12.5Hz. For each prediction, the last 2 observed poses are depicted together with an prediction of 320 milliseconds (4 time steps) into the future. The full ground truth sequence of poses is depicted in the first row. The left arm and leg are depicted in blue (ground truth) or purple (prediction), respectively. Regions of interest are highlighted.

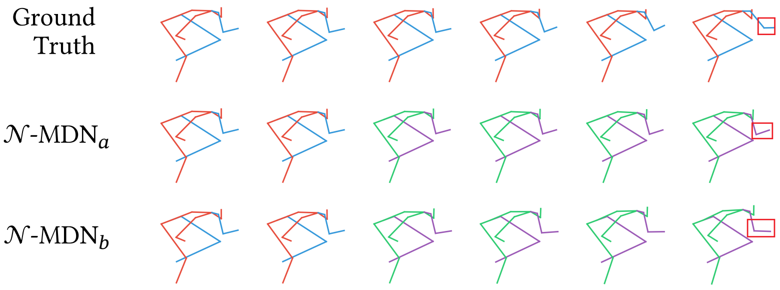


Figure 5.27: Qualitative comparison of predictions generated by both variants of the $\mathcal{N}$-MDN on the *smoking* action. $\mathcal{N}$-MDN$_a$ generates diagonal covariance matrices. $\mathcal{N}$-MDN$_b$ generates sparse covariance matrices, which model inter-joint correlations. For illustration purposes, the sample rate is reduced to 12.5Hz. For each prediction, the last 2 observed poses are depicted together with an prediction of 320 milliseconds (4 time steps) into the future. The full ground truth sequence of poses is depicted in the first row. The left arm and leg are depicted in blue (ground truth) or purple (prediction), respectively. Regions of interest are highlighted.

The actions *eating* (Figure 5.26) and *smoking* (Figure 5.27) both show, apart from a few joints, a static pose throughout the sequence. As such, only subtle movements can be observed looking at the left arm. With respect to the predictions generated by both $\mathcal{N}$-MDN variants, these movements are seemingly averaged out in some way and thus not captured by the model. This smoothing effect is more visible when looking at single dimensions of the pose representations, as depicted in Figure 5.31 towards the end of this section.

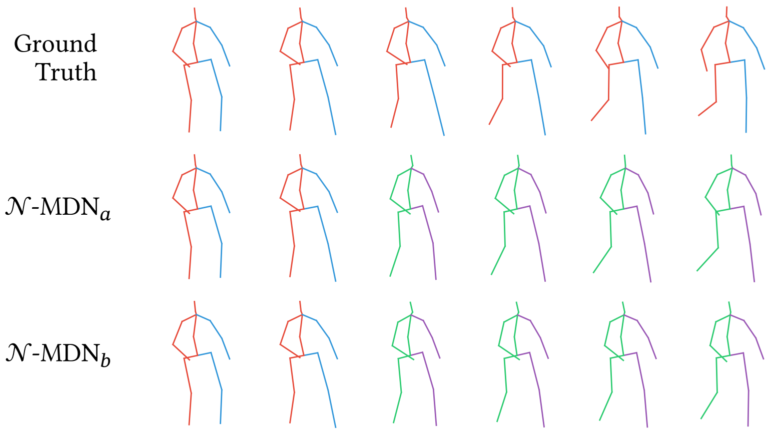
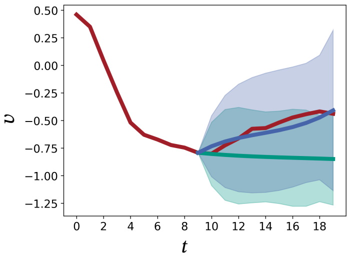


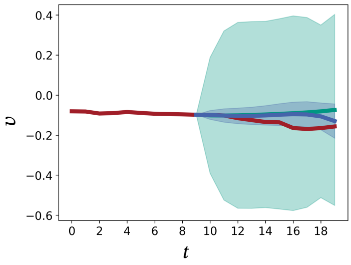
Figure 5.28: Qualitative comparison of predictions generated by both variants of the $\mathcal{N}$-MDN on the *walking* action. $\mathcal{N}$-MDN$_a$ generates diagonal covariance matrices. $\mathcal{N}$-MDN$_b$ generates sparse covariance matrices, which model inter-joint correlations. For illustration purposes, the sample rate is reduced to 12.5Hz. For each prediction, the last 2 observed poses are depicted together with an prediction of 320 milliseconds (4 time steps) into the future. The full ground truth sequence of poses is depicted in the first row. The left arm and leg are depicted in blue (ground truth) or purple (prediction), respectively.

The action, which yields the most accurate prediction, is given by the *walking* action depicted in Figure 5.28. This is most likely due to this action consisting of more obvious motion of the entire body. Further, the *walking* action is more periodic than for example the *discussion* action. As such, it is more predictable and thus easier to model using a statistical model. In the given example, the observed subject slowly turns to the right. This is also correctly captured by both $\mathcal{N}$-MDN variants. Besides that, it can be seen that both variants capture the general trend in motion, but the predicted motion is not as nuanced and pronounced as the actual motion. This is, again, most likely due to the smoothing property of the model.

In order to gain more insight into the predictions generated by the $\mathcal{N}$-MDN variants, selected dimensions of the pose representation are depicted in the following. In this case, the mean prediction with corresponding standard deviation is provided. The standard deviation can be obtained via marginalization from the covariance matrix at each predicted time step.



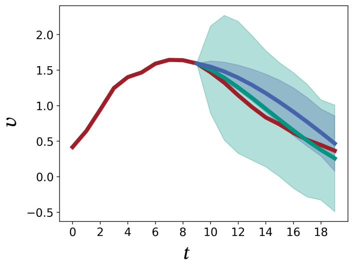
(a) *discussion*: Third dimension of the representation of the left wrist.



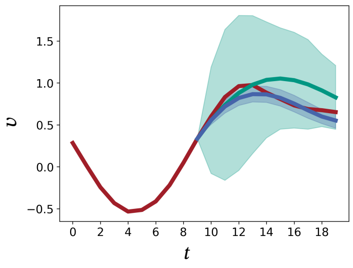
(b) *smoking*: First dimension of the representation of the left wrist.

Figure 5.29: Visualization of selected pose representation dimensions for the purpose of illustrating differences and similarities between $\mathcal{N}$-MDN variants. The green curve depicts $\mathcal{N}$-MDN$_a$ and the blue curve depicts $\mathcal{N}$-MDN$_b$. For both curves, the σ region around the curve is indicated by a shaded region. The ground truth is depicted in red. Time steps are depicted along the x axis and the unit-less value v of the selected dimension is given on the y axis.

As mentioned before, there is a noticeable difference in the predicted motion of the left arm for the *discussion* action, when comparing both $\mathcal{N}$-MDN variants (see Figure 5.25). This can be seen looking at the third dimension of the representation of the left wrist (see Figure 5.29a). While the $\mathcal{N}$-MDN$_b$ variant (blue) follows the ground truth, the $\mathcal{N}$-MDN$_a$ variant (green) falsely predicts an almost constant value. With respect to the subtle arm movements in the *smoking* action (see Figure 5.27), it can be seen, that both model variants predict almost constant values for the left wrist, whereas the ground truth slightly deviates from the constant prediction (see Figure 5.29b).



(a) *discussion*: First dimension of the global translation.

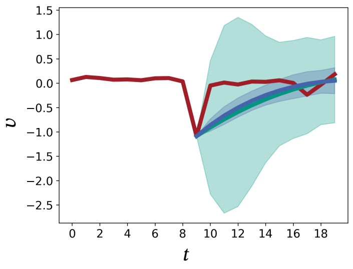


(b) *walking*: First dimension of the representation of the right knee.

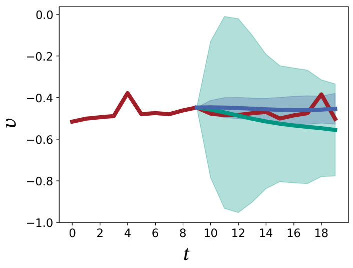
Figure 5.30: Visualization of selected pose representation dimensions for the purpose of illustrating the capability of the $\mathcal{N}$-MDN capturing general trends in human motion. The green curve depicts $\mathcal{N}$-MDN$_a$ and the blue curve depicts $\mathcal{N}$-MDN$_b$. For both curves, the σ region around the curve is indicated by a shaded region. The ground truth is depicted in red. Time steps are depicted along the x axis and the unit-less value v of the selected dimension is given on the y axis.

Although the $\mathcal{N}$-MDN is not quite well-suited for capturing subtle motions in a sequence of poses, it is well capable of capturing the general motion of an observed subject. This can be seen in Figure 5.30. Here, exemplary pose representation dimensions taken from the *discussion* and *walking* examples are illustrated. In both cases, the $\mathcal{N}$-MDN variants generate $\mathcal{N}$-Curves following the correct trend with respect to the ground truth.

Finally, the innate *smoothing* feature of the $\mathcal{N}$-Curve model is quite noticeable looking at the predictions generated by the $\mathcal{N}$-MDN variants. By generating a compact representation, the $\mathcal{N}$-Curve model generally averages out small variations in the data and thus primarily captures trends in the data. The model thereby copes with small variations by varying the variance of the control points accordingly. This smoothing effect is depicted in Figure 5.31.



(a) *eating*: First dimension of the representation of the hip.



(b) *smoking*: Third dimension of the global translation.

Figure 5.31: Visualization of selected pose representation dimensions for the purpose of illustrating the smoothing property of the $\mathcal{N}$-Curve model. The green curve depicts $\mathcal{N}$-MDN$_a$ and the blue curve depicts $\mathcal{N}$-MDN$_b$. For both curves, the σ region around the curve is indicated by a shaded region. The ground truth is depicted in red. Time steps are depicted along the x axis and the unit-less value v of the selected dimension is given on the y axis.

On a final note, the $\mathcal{N}$-MDN$_a$ variant generally generates higher variances than the $\mathcal{N}$-MDN$_b$ variant. This may be due to $\mathcal{N}$-MDN$_a$ having to cope with larger variations in the data, as it processes full 54-dimensional pose representations, whereas networks within the $\mathcal{N}$-MDN$_b$ ensemble only need to deal with 3-dimensional data.

5.2.7 Summary

In this section, the scalability of the $\mathcal{N}$-Curve model in terms of data dimensionality was evaluated. For this, the task of human motion prediction, where sequences of high-dimensional pose representations have to be modeled, was considered. The results show, that the $\mathcal{N}$-Curve model is well-capable of representing higher-dimensional data by increasing the dimensionality of the stochastic control points accordingly. While the $\mathcal{N}$-Curve model outperforms common baselines on the task, it falls a little bit behind in comparison with recent domain-specific models. However, this was expected, as the $\mathcal{N}$-Curve model is a generic model, while the domain-specific models incorporate additional information about the data, such as the arrangement of joints by using graph networks.

6 Summary

Throughout this thesis, an approach for modeling stochastic processes with bounded index sets, the $\mathcal{N}$-Curve model, based on a probabilistic extension of Bézier curves ($\mathcal{N}$-Curves) has been presented. Thereby, a stochastic process is defined by Gaussian mixture distributions, which evolve along a mixture of $\mathcal{N}$-Curves. By basing the $\mathcal{N}$-Curve model on Bézier curves, a compact representation of a stochastic process can be achieved. Together with its proposed implementation based on Mixture Density Networks, the model provides a fully regression-based approach to probabilistic sequence modeling, which does not rely on Monte Carlo techniques during inference, thus reaching set goals. By using parametric curves and optimizing in function space rather than the d-dimensional space of sequence values, the proposed model is able to generate smooth continuous predictions in a single inference step. Thereby, learning a probability distribution over parametric curves is in line with Gaussian processes, which the underlying $\mathcal{N}$-Curves provide a special case for. Different properties of the model were examined by conducting several toy examples on synthetically generated data.

The model has been evaluated extensively on the task of human trajectory prediction, targeting the overall performance of the model in an application context, which proved the viability and capabilities of the model. Looking at the evaluation results, the $\mathcal{N}$-Curve model outperforms other generic probabilistic sequence models on different error measures capturing unimodal and multi-modal prediction performance. These models are commonly used as a basis for more sophisticated, domain-specific models. Further, difficulties in measuring multi-modal prediction performance were discussed. In the scope of this discussion, a small experiment was conducted, in which the application of the Wasserstein metric as a performance measure was proposed. In addition to this broader evaluation, the model's scalability to higher-dimensional data has been shown by applying it to a human motion prediction task. While the $\mathcal{N}$-Curve model outperformed common simple and neural network-based baselines, being a more generic model, it generated slightly less accurate predictions in comparison to recent domain-specific models. Beyond the scalability assessment, difficulties in covariance estimation in higher dimensions and the smoothing property of the $\mathcal{N}$-Curve model were discussed.

Finally, extending on the concept of $\mathcal{N}$-Curves, a conceptual extension to the model, which is capable of modeling infinite stochastic processes, has been presented. For this extension, denoted as the meta-time $\mathcal{N}$-Curve model, a proof of concept on synthetically generated data has been provided, showing the overall viability of the approach in specific cases.

7 Thoughts on Future Research

This final chapter provides an overview of possible directions for future research building on the findings of this thesis.

7.1 Tackle Practical Limitations

First of all, revealed practical limitations of the $\mathcal{N}$-Curve model could be tackled. Thereby, the most relevant limitations can be given by covariance estimation in higher dimensions and the assumed stochastic independence of $\mathcal{N}$-Curve control points.

Covariance Estimation: As indicated in Section 5.2, estimating covariance matrices in high-dimensional data oftentimes leads to numerical instabilities during model training. This is mainly due to the increasing number of correlations that have to be estimated and the necessary condition of covariance matrices to be positive definite. As a result, oftentimes only diagonal covariance matrices are employed. A first step towards tackling this problem was taken by targeting sparse covariance matrices, in which only a few dimensions were correlated. Beyond that, it might be interesting to investigate more advanced approaches to covariance estimation (e.g. [Zho11, Che19]) and their applicability to training Mixture Density Networks.

Stochastic Dependencies: For deriving a closed-form loss function for the (recurrent) $\mathcal{N}$-MDN implementation, independence of $\mathcal{N}$-Curve control points was assumed (see Section 4.1). This independence can be sub-optimal when using the $\mathcal{N}$-Curve model as a generative model (see Section 3.1.3), as Bézier curves sampled from an $\mathcal{N}$-Curve not necessarily have a shape similar to the mean curve. This can be obstructive when $\mathcal{N}$-Curves, estimated from some dataset, should be used to enrich the dataset with more, synthetically generated, sequences similar to those present in the dataset. Following this, it could be interesting to examine, how stochastic dependencies between control points can be incorporated into the model.

7.2 Model Extensions

Apart from these practical challenges, several model extensions could be approached, targeting different parts of the presented model.

Interpolation of Arbitrary Distributions: In the current formulation, the $\mathcal{N}$-Curve model interpolates Gaussian control points in order to obtain a sequence of Gaussian curve points. Following this, the question arises if it would be possible to interpolate control points following arbitrary probability distributions, in order to obtain a probabilistic curve with curve points then following a combined arbitrary probability distribution. To achieve this, the operation of combining multiple control points would need to be extended to a more abstract or general concept, which allows the transformation of given probability distributions. Moving towards this goal, a possible relevant approach might be given by Normalizing Flows (see Section 2.2.3), which can be used to transform simple probability distributions into more complex distribution by applying a chain of invertible mappings.

The Meta-time $\mathcal{N}$-Curve model: In the scope of this thesis, the meta-time $\mathcal{N}$-Curve model (see Section 3.2) was introduced as a conceptual extension to the $\mathcal{N}$-Curve model, lifting some less application-relevant limitations of the original model. However, the toy examples provided in Section 4.2 suggest the viability of the meta-time $\mathcal{N}$-Curve model, especially for modeling long sequences or specifically structured sequential data. Following this, it would be interesting to further explore the capabilities of this model. Looking at the timeline mapping functions introduced in the model definition, it could be especially interesting to examine the possibilities granted by employing learned mapping functions.

Alternative Formulation for Handling Multi-Modality: Currently, the $\mathcal{N}$-Curve model uses a mixture distribution approach for modeling multi-modal stochastic processes. A downside of such an approach is given by the potential blurring of modes when estimating the mixture parameters, as the loss is calculated in terms of a linear combination of all K mixture components. Although mode collapse is mitigated by using Bézier curves as a basis, less well-defined modes can still be a result of using a mixture distribution approach. Thus, in order to achieve more clearly separated modes, more emphasize could be put on the component selection by introducing a notion of attention [Vas17, Dai19] into the model. In this case, an attention mechanism could be used to decide which of the K available $\mathcal{N}$-Curves to select or combine for a given input.

$\mathcal{N}$-Curve Gaussian Processes: Finally, it could be interesting to elaborate more on the properties and potential advantages and disadvantages of the class of Gaussian process kernels induces by an $\mathcal{N}$-Curve in comparison with other kernels. Additionally, it could be examined if and to what extend the $\mathcal{N}$-Curve model and its implementation would benefit from the incorporation of concepts taken from Gaussian processes.

Bibliography

[Ala16] Alahi, Alexandre; Goel, Kratarth; Ramanathan, Vignesh; Robicquet, Alexandre; Fei-Fei, Li and Savarese, Silvio: "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971 (cit. on pp. 90, 91).

[Ami19] Amirian, Javad; Hayet, Jean-Bernard and Pettré, Julien: "Social ways: Learning multi-modal distributions of pedestrian trajectories with gans". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019 (cit. on p. 90).

[Ami20] Amirian, Javad; Zhang, Bingqing; Castro, Francisco Valente; Baldelomar, Juan Jose; Hayet, Jean-Bernard and Pettre, Julien: "OpenTraj: Assessing Prediction Complexity in Human Trajectories Datasets". In: *Proceedings of the Asian Conference on Computer Vision*. 2020 (cit. on p. 92).

[And10] Anderson, Carolyn J.: "Central Limit Theorem". In: *The Corsini Encyclopedia of Psychology*. John Wiley Sons, Ltd, 2010, pp. 1–2. doi: https://doi.org/10.1002/9780470479216.corpsy0160. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470479216.corpsy0160. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470479216.corpsy0160 (cit. on p. 22).

[Arj17] Arjovsky, Martin; Chintala, Soumith and Bottou, Léon: "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223 (cit. on p. 16).

[Bai18] Bai, Shaojie; Kolter, J Zico and Koltun, Vladlen: "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018) (cit. on pp. 7, 9, 90).

[Bar95] Bartels, Richard H; Beatty, John C and Barsky, Brian A: An introduction to splines for use in computer graphics and geometric modeling. Morgan Kaufmann, 1995 (cit. on p. 41).

[Bec18] Becker, S.; Hug, R.; Hübner, W. and Arens, M.: "RED: A simple but effective Baseline Predictor for the TrajNet Benchmark". In: *European Conference on Computer Vision (ECCV) Workshops.* Springer International Publishing, 2018. doi: 10.1007/978-3-030-11015-4_13 (cit. on pp. 9, 90).

[Ben11] Benfold, Ben and Reid, Ian: "Stable multi-target tracking in real-time surveillance video". In: *CVPR 2011.* IEEE. 2011, pp. 3457–3464 (cit. on p. 87).

[Ben15] Bengio, Samy; Vinyals, Oriol; Jaitly, Navdeep and Shazeer, Noam: "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1.* NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 1171–1179 (cit. on p. 8).

[Bha18] Bhattacharyya, Apratim; Schiele, Bernt and Fritz, Mario: "Accurate and diverse sampling of sequences based on a "best of many" sample objective". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 8485–8493 (cit. on pp. 90, 96, 99).

[Bis06] Bishop, Christopher M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006 (cit. on p. 13).

[Bis94] Bishop, Christopher M: "Mixture density networks". In: (1994) (cit. on pp. 13, 52, 55).

[Bis95] Bishop, Christopher M: Neural networks for pattern recognition. Oxford university press, 1995 (cit. on p. 10).

[Blu15] Blundell, Charles; Cornebise, Julien; Kavukcuoglu, Koray and Wierstra, Daan: "Weight uncertainty in neural network". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1613–1622 (cit. on p. 10).

[Boc19] Bock, Julian; Krajewski, Robert; Moers, Tobias; Runde, Steffen; Vater, Lennart and Eckstein, Lutz: "The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections". In: 2019 (cit. on p. 87).

[Bon15] Bonneel, Nicolas; Rabin, Julien; Peyré, Gabriel and Pfister, Hanspeter: "Sliced and radon wasserstein barycenters of measures". In: *Journal of Mathematical Imaging and Vision* 51.1 (2015), pp. 22–45 (cit. on p. 118).

[Bow16] Bowman, Samuel R.; Vilnis, Luke; Vinyals, Oriol; Dai, Andrew; Jozefowicz, Rafal and Bengio, Samy: "Generating Sentences from a Continuous Space". In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 10–21. doi: 10.18653/v1/K16-1002. url: https://aclanthology.org/K16-1002 (cit. on p. 14).

[Box15] Box, George EP; Jenkins, Gwilym M; Reinsel, Gregory C and Ljung, Greta M: Time series analysis: forecasting and control. John Wiley & Sons, 2015 (cit. on p. 7).

[Bra19] Brando, Axel; Rodriguez, Jose A; Vitria, Jordi and Rubio Muñoz, Alberto: "Modelling heterogeneous distributions with an Uncountable Mixture of Asymmetric Laplacians". In: *Advances in neural information processing systems* 32 (2019), pp. 8838–8848 (cit. on p. 13).

[Bri17] Britz, Denny; Goldie, Anna; Luong, Minh-Thang and Le, Quoc: "Massive exploration of neural machine translation architectures". In: *arXiv preprint arXiv:1703.03906* (2017) (cit. on p. 8).

[Cae20] Caesar, Holger; Bankiti, Varun; Lang, Alex H; Vora, Sourabh; Liong, Venice Erin; Xu, Qiang; Krishnan, Anush; Pan, Yu; Baldan, Giancarlo and Beijbom, Oscar: “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020, pp. 11621–11631 (cit. on p. 87).

[Cam15] Campbell, Kieran and Yau, Christopher: “Bayesian Gaussian Process Latent Variable Models for pseudotime inference in single-cell RNA-seq data”. In: *bioRxiv* (2015). doi: https://doi.org/10.1101/026872 (cit. on p. 12).

[Cha18] Chavdarova, Tatjana; Baqué, Pierre; Bouquet, Stéphane; Maksai, Andrii; Jose, Cijo; Bagautdinov, Timur; Lettry, Louis; Fua, Pascal; Van Gool, Luc and Fleuret, François: “Wildtrack: A multi-camera hd dataset for dense unscripted pedestrian detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 5030–5039 (cit. on p. 87).

[Che19] Cheng, Yu; Diakonikolas, Ilias; Ge, Rong and Woodruff, David P: “Faster algorithms for high-dimensional robust covariance estimation”. In: *Conference on Learning Theory.* PMLR. 2019, pp. 727–757 (cit. on p. 151).

[Cho14] Cho, Kyunghyun; Merriënboer, Bart van; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger and Bengio, Yoshua: “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Association for Computational Linguistics, 2014, pp. 1724–1734 (cit. on p. 7).

[Chu14] Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun and Bengio, Yoshua: “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014) (cit. on p. 8).

[Chu15] Chung, Junyoung; Kastner, Kyle; Dinh, Laurent; Goel, Kratarth; Courville, Aaron C and Bengio, Yoshua: “A Recurrent Latent Variable Model for Sequential Data”. In: *Advances in Neural Information Processing Systems*. Ed. by Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M. and Garnett, R. Vol. 28. Curran Associates, Inc., 2015. url: https://proceedings.neurips.cc/paper/2015/file/b618c3210e934362ac261db280128c22-Paper.pdf (cit. on p. 15).

[Cle15] Clevert, Djork-Arné; Unterthiner, Thomas and Hochreiter, Sepp: “Fast and accurate deep network learning by exponential linear units (elus)”. In: *arXiv preprint arXiv:1511.07289* (2015) (cit. on p. 52).

[Con04] Conrad, Keith: “Probability distributions and maximum entropy”. In: *Entropy* 6.452 (2004), p. 10 (cit. on p. 22).

[Cui20] Cui, Qiongjie; Sun, Huaijiang and Yang, Fei: “Learning dynamic relationships for 3d human motion prediction”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 6519–6527 (cit. on pp. 136, 137).

[Dai19] Dai, Zihang; Yang, Zhilin; Yang, Yiming; Carbonell, Jaime; Le, Quoc and Salakhutdinov, Ruslan: “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 2978–2988. doi: 10.18653/v1/P19-1285 (cit. on pp. 9, 153).

[Dam13] Damianou, Andreas and Lawrence, Neil: “Deep gaussian processes”. In: *Artificial Intelligence and Statistics*. 2013, pp. 207–215 (cit. on p. 12).

[Dem77] Dempster, Arthur P; Laird, Nan M and Rubin, Donald B: “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22 (cit. on p. 54).

[Dou09] Doucet, Arnaud and Johansen, Adam M: “A tutorial on particle filtering and smoothing: Fifteen years later”. In: *Handbook of nonlinear filtering* 12.656-704 (2009), p. 3 (cit. on p. 69).

[Dug01] Dugas, Charles; Bengio, Yoshua; Bélisle, François; Nadeau, Claude and Garcia, René: “Incorporating second-order functional knowledge for better option pricing”. In: *Advances in neural information processing systems* (2001), pp. 472–478 (cit. on p. 53).

[Ell09] Ellis, David; Sommerlade, Eric and Reid, Ian: “Modelling pedestrian trajectory patterns with gaussian processes”. In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. IEEE. 2009, pp. 1229–1234 (cit. on p. 12).

[End03] Endres, Dominik Maria and Schindelin, Johannes E: “A new metric for probability distributions”. In: *IEEE Transactions on Information theory* 49.7 (2003), pp. 1858–1860 (cit. on p. 118).

[Far02] Farin, Gerald E and Farin, Gerald: Curves and surfaces for CAGD: a practical guide. Morgan Kaufmann, 2002 (cit. on p. 20).

[Far08] Farouki, Rida T: Pythagorean—hodograph Curves. Springer, 2008 (cit. on p. 28).

[Fer09] Ferryman, James and Shahrokni, Ali: “Pets2009: Dataset and challenge”. In: *2009 Twelfth IEEE international workshop on performance evaluation of tracking and surveillance*. IEEE. 2009, pp. 1–6 (cit. on p. 87).

[Fit14] Fitter, Hetal N; Pandey, Akash B; Patel, Divyang D and Mistry, Jitendra M: “A review on approaches for handling Bézier curves in CAD for manufacturing”. In: *Procedia Engineering* 97 (2014), pp. 1155–1166 (cit. on p. 28).

[Fla21] Flamary, Rémi et al.: “POT: Python Optimal Transport”. In: *Journal of Machine Learning Research* 22.78 (2021), pp. 1–8. url: http://jmlr.org/papers/v22/20-451.html (cit. on p. 123).

[For17] Fortunato, Meire; Blundell, Charles and Vinyals, Oriol: "Bayesian recurrent neural networks". In: *12th Women in Machine Learning Workshop (WiML 2017)* (2017) (cit. on p. 11).

[Fra15] Fragkiadaki, Katerina; Levine, Sergey; Felsen, Panna and Malik, Jitendra: "Recurrent network models for human dynamics". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4346–4354 (cit. on pp. 134, 135, 137).

[Gal16] Gal, Yarin and Ghahramani, Zoubin: "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. 2016, pp. 1050–1059 (cit. on pp. 11, 81).

[Gal17] Gal, Yarin; Hron, Jiri and Kendall, Alex: "Concrete dropout". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3581–3590 (cit. on p. 81).

[Gei13] Geiger, Andreas; Lenz, Philip; Stiller, Christoph and Urtasun, Raquel: "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237 (cit. on p. 87).

[Gho17] Ghosh, Partha; Song, Jie; Aksan, Emre and Hilliges, Otmar: "Learning human motion models for long-term predictions". In: *2017 International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 458–466 (cit. on p. 135).

[Giu21] Giuliari, Francesco; Hasan, Irtiza; Cristani, Marco and Galasso, Fabio: "Transformer networks for trajectory forecasting". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 10335–10342 (cit. on pp. 90, 99).

[Glo10] Glorot, Xavier and Bengio, Yoshua: "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (cit. on p. 53).

[Glo11] Glorot, Xavier; Bordes, Antoine and Bengio, Yoshua: “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings. 2011, pp. 315–323 (cit. on p. 53).

[Goo14] Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron and Bengio, Yoshua: “Generative Adversarial Nets”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2.* NIPS’14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680 (cit. on pp. 15, 90).

[Goo16] Goodfellow, Ian; Bengio, Yoshua and Courville, Aaron: Deep learning. MIT press, 2016 (cit. on p. 8).

[Gop19] Gopalakrishnan, Anand; Mali, Ankur; Kifer, Dan; Giles, Lee and Ororbia, Alexander G: “A neural temporal model for human motion prediction”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2019, pp. 12116–12125 (cit. on p. 135).

[Gör19] Görtler, Jochen; Kehlbeck, Rebecca and Deussen, Oliver: “A visual exploration of Gaussian processes”. In: *Distill* 4.4 (2019), e17 (cit. on p. 37).

[Gra13] Graves, Alex: “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013) (cit. on pp. 13, 90).

[Gui17] Guillaumes, Axel Brando: “Mixture density networks for distribution and uncertainty estimation”. Universitat Politècnica de Catalunya. Facultat d’Informàtica de Barcelona, 2017 (cit. on p. 52).

[Gui18] Gui, Liang-Yan; Wang, Yu-Xiong; Liang, Xiaodan and Moura, José MF: “Adversarial geometry-aware human motion prediction”. In: *Proceedings of the European Conference on Computer Vision (ECCV).* 2018, pp. 786–803 (cit. on pp. 135–137).

[Gul17] Gulrajani, Ishaan; Ahmed, Faruk; Arjovsky, Martin; Dumoulin, Vincent and Courville, Aaron: “Improved Training of Wasserstein GANs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems.* NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 5769–5779 (cit. on p. 16).

[Gup18] Gupta, Agrim; Johnson, Justin; Fei-Fei, Li; Savarese, Silvio and Alahi, Alexandre: “Social gan: Socially acceptable trajectories with generative adversarial networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 2255–2264 (cit. on pp. 16, 90, 91, 99, 101).

[Ha18] Ha, David and Schmidhuber, Jürgen: “Recurrent World Models Facilitate Policy Evolution”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada.* Ed. by Bengio, Samy; Wallach, Hanna M.; Larochelle, Hugo; Grauman, Kristen; Cesa-Bianchi, Nicolò and Garnett, Roman. 2018, pp. 2455–2467 (cit. on p. 138).

[Haa18] Haarbach, Adrian; Birdal, Tolga and Ilic, Slobodan: “Survey of higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation”. In: *2018 International Conference on 3D Vision (3DV).* IEEE. 2018, pp. 381–389 (cit. on p. 28).

[Ham20] Ham, Hyungrok; Jun, Tae Joon and Kim, Daeyoung: “Unbalanced gans: Pre-training the generator of generative adversarial network using variational autoencoder”. In: *arXiv preprint arXiv:2002.02112* (2020) (cit. on p. 16).

[Har17] Hargens, O.: “Detektion ungewöhnlichen Verhaltens auf unterschiedlichen Zeitskalen mittels LSTM-Netzwerken”. Bachelor’s thesis. Karlsruhe Institute of Technology (KIT), 2017 (cit. on p. 1).

[Has09] Hastie, Trevor; Tibshirani, Robert and Friedman, Jerome: The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media, 2009 (cit. on p. 105).

[Has19] Hasan, Irtiza; Setti, Francesco; Tsesmelis, Theodore; Belagiannis, Vasileios; Amin, Sikandar; Del Bue, Alessio; Cristani, Marco and Galasso, Fabio: "Forecasting people trajectories and head poses by jointly reasoning on tracklets and vislets". In: *IEEE transactions on pattern analysis and machine intelligence* (2019) (cit. on p. 95).

[He16] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing and Sun, Jian: "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778 (cit. on p. 57).

[Hel95] Helbing, Dirk and Molnar, Peter: "Social force model for pedestrian dynamics". In: *Physical review E* 51.5 (1995), p. 4282 (cit. on p. 85).

[Her15] Hernández-Lobato, José Miguel and Adams, Ryan: "Probabilistic backpropagation for scalable learning of bayesian neural networks". In: *International Conference on Machine Learning.* 2015, pp. 1861–1869 (cit. on p. 11).

[Hin94] Hinton, Geoffrey E and Zemel, Richard S: "Autoencoders, minimum description length, and Helmholtz free energy". In: *Advances in neural information processing systems* 6 (1994), pp. 3–10 (cit. on p. 14).

[Hoc97] Hochreiter, Sepp and Schmidhuber, Jürgen: "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 7).

[Hol90] Holschneider, Matthias; Kronland-Martinet, Richard; Morlet, Jean and Tchamitchian, Ph: "A real-time algorithm for signal analysis with the help of the wavelet transform". In: *Wavelets.* Springer, 1990, pp. 286–297 (cit. on p. 8).

[Hua17] Huang, Gao; Li, Yixuan; Pleiss, Geoff; Liu, Zhuang; Hopcroft, John E. and Weinberger, Kilian Q.: "Snapshot Ensembles: Train 1, Get M for Free". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cit. on p. 81).

[Hua18] Huang, Chin-Wei; Krueger, David; Lacoste, Alexandre and Courville, Aaron: "Neural autoregressive flows". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2078–2087 (cit. on p. 14).

[Hua19] Huang, Yingfan; Bi, Huikun; Li, Zhaoxin; Mao, Tianlu and Wang, Zhaoqi: "Stgat: Modeling spatial-temporal interactions for human trajectory prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6272–6281 (cit. on p. 91).

[Hub20] Huber, Marco F: "Bayesian Perceptron: Towards fully Bayesian Neural Networks". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 3179–3186 (cit. on p. 11).

[Hug17] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "On the reliability of LSTM-MDL models for pedestrian trajectory prediction". In: *Representations, Analysis and Recognition of Shape and Motion from Imaging Data (RFMI)*. Savoie, France, 2017. doi: 10.1007/978-3-030-19816-9_2 (cit. on p. 57).

[Hug18] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "Particle-based Pedestrian Path Prediction using LSTM-MDL Models". In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2684–2691. doi: 10.1109/ITSC.2018.8569478 (cit. on pp. 13, 69, 90, 99, 100).

[Hug20a] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "A complementary trajectory prediction benchmark". In: *ECCV Workshop on Benchmarking Trajectory Forecasting Models (BTFM)*. 2020 (cit. on p. 4).

[Hug20b] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "A Short Note on Analyzing Sequence Complexity in Trajectory Prediction Benchmarks". In: *Workshop on Long-term Human Motion Prediction (LHMP).* 2020 (cit. on p. 4).

[Hug21] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "Quantifying the Complexity of Standard Benchmarking Datasets for Long-Term Human Trajectory Prediction". In: *IEEE Access* 9 (2021), pp. 77693–77704. doi: 10.1109/ACCESS.2021.3082904 (cit. on pp. 4, 85, 92, 110).

[Ion13] Ionescu, Catalin; Papava, Dragos; Olaru, Vlad and Sminchisescu, Cristian: "Human3. 6m: Large scale datasets and predictive methods for 3d human sensing in natural environments". In: *IEEE transactions on pattern analysis and machine intelligence* 36.7 (2013), pp. 1325–1339 (cit. on pp. 132, 133).

[Iso17] Iso, Hayate; Wakamiya, Shoko and Aramaki, Eiji: "Density estimation for geolocation via convolutional mixture density network". In: *arXiv preprint arXiv:1705.02750* (2017) (cit. on p. 53).

[Iva19] Ivanovic, Boris and Pavone, Marco: "The trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2019, pp. 2375–2384 (cit. on p. 96).

[Izd20] Izdebski, Łukasz; Kopiecki, Ryszard and Sawicki, Dariusz: "Bézier Curve as a Generalization of the Easing Function in Computer Animation". In: *Computer Graphics International Conference.* Springer. 2020, pp. 382–393 (cit. on p. 28).

[Jai16] Jain, Ashesh; Zamir, Amir R; Savarese, Silvio and Saxena, Ashutosh: "Structural-rnn: Deep learning on spatio-temporal graphs". In: *Proceedings of the ieee conference on computer vision and pattern recognition.* 2016, pp. 5308–5317 (cit. on pp. 134, 135, 137).

[Jef88] Jeffreys, H. and Jeffreys, B.S.: “Lagrange’s Interpolation Formula, §9.011”. In: *Methods of Mathematical Physics 3rd Edition.* Cambridge: Cambridge University Press, 1988, p. 260 (cit. on p. 28).

[Jer13] Jerri, Abdul J: The Gibbs phenomenon in Fourier analysis, splines and wavelet approximations. Vol. 446. Springer Science & Business Media, 2013 (cit. on p. 78).

[Jol09] Jolly, KG; Kumar, R Sreerama and Vijayakumar, R: “A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits”. In: *Robotics and Autonomous Systems* 57.1 (2009), pp. 23–33 (cit. on p. 28).

[Joz15] Jozefowicz, Rafal; Zaremba, Wojciech and Sutskever, Ilya: “An empirical exploration of recurrent network architectures”. In: *International conference on machine learning*. PMLR. 2015, pp. 2342–2350 (cit. on p. 8).

[Kan39] Kantorovich, Leonid V: “The mathematical method of production planning and organization”. In: *Management Science* 6.4 (1939), pp. 363–422 (cit. on p. 118).

[Kar19] Karita, Shigeki; Chen, Nanxin; Hayashi, Tomoki; Hori, Takaaki; Inaguma, Hirofumi; Jiang, Ziyan; Someki, Masao; Soplin, Nelson Enrique Yalta; Yamamoto, Ryuichi; Wang, Xiaofei et al.: “A comparative study on transformer vs rnn in speech applications”. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).* IEEE. 2019, pp. 449–456 (cit. on p. 9).

[Ken17] Kendall, Alex and Gal, Yarin: “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in neural information processing systems.* 2017, pp. 5574–5584 (cit. on p. 81).

[Kin14] Kingma, Diederik P. and Welling, Max: “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* 2014. url: http://arxiv.org/abs/1312.6114 (cit. on pp. 14, 90).

[Kin15] Kingma, Diederik P. and Ba, Jimmy: “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.* 2015. url: http://arxiv.org/abs/1412.6980 (cit. on pp. 57, 104).

[Kin16] Kingma, Durk P; Salimans, Tim; Jozefowicz, Rafal; Chen, Xi; Sutskever, Ilya and Welling, Max: “Improved variational inference with inverse autoregressive flow”. In: *Advances in neural information processing systems* 29 (2016), pp. 4743–4751 (cit. on p. 14).

[Kip17] Kipf, Thomas N. and Welling, Max: “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017. url: https://openreview.net/forum?id=SJU4ayYgl (cit. on p. 136).

[Kit12] Kitani, Kris M; Ziebart, Brian D; Bagnell, James Andrew and Hebert, Martial: “Activity forecasting”. In: *European Conference on Computer Vision.* Springer. 2012, pp. 201–214 (cit. on p. 89).

[Kol17] Kolouri, Soheil; Park, Se Rim; Thorpe, Matthew; Slepcev, Dejan and Rohde, Gustavo K: “Optimal mass transport: Signal processing and machine-learning applications”. In: *IEEE signal processing magazine* 34.4 (2017), pp. 43–59 (cit. on p. 118).

[Kol19] Kolouri, Soheil; Nadjahi, Kimia; Simsekli, Umut; Badeau, Roland and Rohde, Gustavo: “Generalized Sliced Wasserstein Distances”. In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 261–272 (cit. on p. 118).

[Kos19] Kosaraju, Vineet; Sadeghian, Amir; Martín-Martín, Roberto; Reid, Ian; Rezatofighi, S Hamid and Savarese, Silvio: "Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks". In: *NeurIPS*. 2019 (cit. on p. 91).

[Kot21] Kothari, Parth; Kreiss, Sven and Alahi, Alexandre: "Human trajectory forecasting in crowds: A deep learning perspective". In: *IEEE Transactions on Intelligent Transportation Systems* (2021) (cit. on p. 85).

[Kul51] Kullback, Solomon and Leibler, Richard A: "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86 (cit. on p. 118).

[Kun19] Kundu, Jogendra Nath; Gor, Maharshi and Babu, R. Venkatesh: "BiHMP-GAN: Bidirectional 3D Human Motion Prediction GAN". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.* AAAI Press, 2019, pp. 8553–8560. doi: 10.1609/aaai.v33i01.33018553. url: https://doi.org/10.1609/aaai.v33i01.33018553 (cit. on pp. 135–137).

[Lak17] Lakshminarayanan, Balaji; Pritzel, Alexander and Blundell, Charles: "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6402–6413 (cit. on p. 81).

[LeC95] LeCun, Yann; Bengio, Yoshua et al.: "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995 (cit. on p. 8).

[Lee17] Lee, Namhoon; Choi, Wongun; Vernaza, Paul; Choy, Christopher B; Torr, Philip HS and Chandraker, Manmohan: "Desire: Distant future prediction in dynamic scenes with interacting agents". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 336–345 (cit. on pp. 90, 97).

[Lee18] Lee, Jaehoon; Bahri, Yasaman; Novak, Roman; Schoenholz, Samuel S.; Pennington, Jeffrey and Sohl-Dickstein, Jascha: "Deep Neural Networks as Gaussian Processes". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, 2018 (cit. on p. 11).

[Ler07] Lerner, Alon; Chrysanthou, Yiorgos and Lischinski, Dani: "Crowds by example". In: *Computer graphics forum.* Vol. 26. 3. Wiley Online Library. 2007, pp. 655–664 (cit. on p. 87).

[Li20] Li, Maosen; Chen, Siheng; Zhao, Yangheng; Zhang, Ya; Wang, Yanfeng and Tian, Qi: "Dynamic multiscale graph neural networks for 3d skeleton based human motion prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2020, pp. 214–223 (cit. on p. 136).

[Lor13] Lorentz, George G: Bernstein polynomials. American Mathematical Soc., 2013 (cit. on p. 20).

[Mac03] MacKay, David JC and Mac Kay, David JC: Information theory, inference and learning algorithms. Cambridge university press, 2003 (cit. on p. 34).

[Maj09] Majecka, Barbara: "Statistical models of pedestrian behaviour in the forum". In: *Master's thesis, School of Informatics, University of Edinburgh* (2009) (cit. on p. 87).

[Mak19] Makansi, Osama; Ilg, Eddy; Cicek, Ozgun and Brox, Thomas: "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2019, pp. 7144–7153 (cit. on pp. 13, 55, 111, 138).

[Mal18] Malinin, Andrey and Gales, Mark: "Predictive uncertainty estimation via prior networks". In: *arXiv preprint arXiv:1802.10501* (2018) (cit. on p. 81).

[Man20] Mangalam, Karttikeya; Girase, Harshayu; Agarwal, Shreyas; Lee, Kuan-Hui; Adeli, Ehsan; Malik, Jitendra and Gaidon, Adrien: "It is not the journey but the destination: Endpoint conditioned trajectory prediction". In: *European Conference on Computer Vision.* Springer. 2020, pp. 759–776 (cit. on p. 89).

[Mao19] Mao, Wei; Liu, Miaomiao; Salzmann, Mathieu and Li, Hongdong: "Learning trajectory dependencies for human motion prediction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 2019, pp. 9489–9497 (cit. on pp. 135–137).

[Mar17] Martinez, Julieta; Black, Michael J and Romero, Javier: "On human motion prediction using recurrent neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017, pp. 2891–2900 (cit. on pp. 135, 137, 138).

[Mat16] Mattos, César Lincoln C.; Dai, Zhenwen; Damianou, Andreas C.; Forth, Jeremy; Barreto, Guilherme A. and Lawrence, Neil D.: "Recurrent Gaussian Processes". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.* Ed. by Bengio, Yoshua and LeCun, Yann. 2016 (cit. on p. 12).

[Met17] Metz, Luke; Poole, Ben; Pfau, David and Sohl-Dickstein, Jascha: "Unrolled Generative Adversarial Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017 (cit. on pp. 16, 111).

[Mir14] Mirza, Mehdi and Osindero, Simon: "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014) (cit. on p. 16).

[Moh20] Mohamed, Abduallah; Qian, Kun; Elhoseiny, Mohamed and Claudel, Christian: "Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 14424–14432 (cit. on pp. 88, 91).

[Mon21] Monti, Alessio; Bertugli, Alessia; Calderara, Simone and Cucchiara, Rita: "Dag-net: Double attentive graph neural network for trajectory forecasting". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 2551–2558 (cit. on p. 90).

[Mur91] Murtagh, Fionn: "Multilayer perceptrons for classification and regression". In: *Neurocomputing* 2.5 (1991), pp. 183–197. doi: https://doi.org/10.1016/0925-2312(91)90023-5. url: https://www.sciencedirect.com/science/article/pii/0925231291900235 (cit. on p. 7).

[Nea92] Neal, Radford M: Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Tech. rep. Citeseer, 1992 (cit. on p. 10).

[Nea96] Neal, Radford M.: Bayesian Learning for Neural Networks. Berlin, Heidelberg: Springer-Verlag, 1996 (cit. on p. 11).

[Ng04] Ng, Andrew Y: "Feature selection, L 1 vs. L 2 regularization, and rotational invariance". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 78 (cit. on p. 68).

[Oh11] Oh, Sangmin; Hoogs, Anthony; Perera, Amitha; Cuntoor, Naresh; Chen, Chia-Chih; Lee, Jong Taek; Mukherjee, Saurajit; Aggarwal, JK; Lee, Hyungtae; Davis, Larry et al.: "A large-scale benchmark dataset for event recognition in surveillance video". In: *CVPR 2011*. IEEE. 2011, pp. 3153–3160 (cit. on p. 87).

[Oor16] Oord, Aäron van den; Dieleman, Sander; Zen, Heiga; Simonyan, Karen; Vinyals, Oriol; Graves, Alex; Kalchbrenner, Nal; Senior, Andrew and Kavukcuoglu, Koray: "WaveNet: A

Generative Model for Raw Audio”. In: *9th ISCA Speech Synthesis Workshop*. 2016, pp. 125–125 (cit. on pp. 8, 90).

[Paj18] Pajouheshgar, Ehsan and Lampert, Christoph H: “Back to square one: probabilistic trajectory forecasting without bells and whistles”. In: *Modeling and decision-making in the spatiotemporal domain (NeurIPS 2018 Workshop)* (2018) (cit. on pp. 97, 117).

[Pas13] Pascanu, Razvan; Mikolov, Tomas and Bengio, Yoshua: “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318 (cit. on p. 7).

[Pas19] Paszke, Adam; Gross, Sam; Massa, Francisco; Lerer, Adam; Bradbury, James; Chanan, Gregory; Killeen, Trevor; Lin, Zeming; Gimelshein, Natalia; Antiga, Luca et al.: “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 8026–8037 (cit. on p. 58).

[Pav18] Pavllo, Dario; Grangier, David and Auli, Michael: “QuaterNet: A Quaternion-based Recurrent Model for Human Motion”. In: *British Machine Vision Conference 2018, BMVC 2018, Newcastle, UK, September 3-6, 2018*. BMVA Press, 2018, p. 299. url: http://bmvc2018.org/contents/papers/0675.pdf (cit. on pp. 136, 137).

[Pel09] Pellegrini, Stefano; Ess, Andreas; Schindler, Konrad and Van Gool, Luc: “You’ll never walk alone: Modeling social behavior for multi-target tracking”. In: *2009 IEEE 12th International Conference on Computer Vision*. IEEE. 2009, pp. 261–268 (cit. on p. 87).

[Pet08] Petersen, Kaare Brandt and Pedersen, Michael Syskind: “The matrix cookbook”. In: *Technical University of Denmark* 7.15 (2008), p. 510 (cit. on p. 23).

[Pra02] Prautzsch, Hartmut; Boehm, Wolfgang and Paluszny, Marco: Bézier and B-spline techniques. Springer Science & Business Media, 2002 (cit. on p. 20).

[Pre07] Press, William H; William, H; Teukolsky, Saul A; Saul, A; Vetterling, William T and Flannery, Brian P: Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press, 2007 (cit. on p. 57).

[Ras06] Rasmussen, Carl Edward and Williams, Christopher K. I.: Gaussian processes for machine learning. Adaptive computation and machine learning. MIT Press, 2006 (cit. on pp. 6, 11, 34, 37).

[Ras17a] Rasouli, Amir; Kotseruba, Iuliia and Tsotsos, John K: "Agreeing to cross: How drivers and pedestrians communicate". In: *IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 264–269 (cit. on p. 87).

[Ras17b] Rasouli, Amir; Kotseruba, Iuliia and Tsotsos, John K: "Are they going to cross? A benchmark dataset and baseline for pedestrian crosswalk behavior". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 206–213 (cit. on p. 87).

[Ras99] Rasmussen, Carl Edward et al.: "The infinite Gaussian mixture model." In: *NIPS*. Vol. 12. 1999, pp. 554–560 (cit. on p. 68).

[Raz20] Razavi, Seyedeh Fatemeh and Hosseini, Reshad: "FRMDN: Flow-based Recurrent Mixture Density Network". In: *arXiv preprint arXiv:2008.02144* (2020) (cit. on p. 138).

[Reb21] Rebaza, Jorge: A first course in applied mathematics. John Wiley & Sons, 2021 (cit. on p. 41).

[Rez15] Rezende, Danilo and Mohamed, Shakir: "Variational inference with normalizing flows". In: *International conference on machine learning*. PMLR. 2015, pp. 1530–1538 (cit. on p. 14).

[Rob16] Robicquet, Alexandre; Sadeghian, Amir; Alahi, Alexandre and Savarese, Silvio: "Learning social etiquette: Human trajectory understanding in crowded scenes". In: *European conference on computer vision*. Springer. 2016, pp. 549–565 (cit. on p. 87).

[Ros58] Rosenblatt, Frank: “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 11).

[Rud16] Ruder, Sebastian: “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016) (cit. on p. 57).

[Rud20a] Rudenko, Andrey; Kucner, Tomasz P; Swaminathan, Chittaranjan S; Chadalavada, Ravi T; Arras, Kai O and Lilienthal, Achim J: “THÖR: Human-Robot Navigation Data Collection and Accurate Motion Trajectories Dataset”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 676–682 (cit. on p. 87).

[Rud20b] Rudenko, Andrey; Palmieri, Luigi; Herman, Michael; Kitani, Kris M; Gavrila, Dariu M and Arras, Kai O: “Human motion trajectory prediction: A survey”. In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935 (cit. on pp. 7, 85, 89).

[Rum86] Rumelhart, David E; Hinton, Geoffrey E and Williams, Ronald J: “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 7).

[Rup17] Rupprecht, Christian; Laina, Iro; DiPietro, Robert; Baust, Maximilian; Tombari, Federico; Navab, Nassir and Hager, Gregory D: “Learning in an uncertain world: Representing ambiguity through multiple hypotheses”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3591–3600 (cit. on p. 138).

[Sad19] Sadeghian, Amir; Kosaraju, Vineet; Sadeghian, Ali; Hirose, Noriaki; Rezatofighi, Hamid and Savarese, Silvio: “Sophie: An attentive gan for predicting paths compliant to social and physical constraints”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1349–1358 (cit. on p. 90).

[Sal20] Salzmann, Tim; Ivanovic, Boris; Chakravarty, Punarjay and Pavone, Marco: “Trajectron++: Dynamically-Feasible Trajectory Forecasting with Heterogeneous Data”. In: *Computer Vision – ECCV 2020*. Cham: Springer International Publishing, 2020, pp. 683–700 (cit. on p. 88).

[Sär13] Särkkä, Simo: Bayesian filtering and smoothing. 3. Cambridge University Press, 2013 (cit. on p. 7).

[Sch13] Schneider, Nicolas and Gavrila, Dariu M: “Pedestrian path prediction with recursive bayesian filters: A comparative study”. In: *German Conference on Pattern Recognition*. Springer. 2013, pp. 174–183 (cit. on pp. 85, 87).

[Sch20a] Schöller, Christoph; Aravantinos, Vincent; Lay, Florian and Knoll, Alois: “What the constant velocity model can teach us about pedestrian motion prediction”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1696–1703 (cit. on pp. 97, 129).

[Sch20b] Schütt, Kristof T; Tkatchenko, Alexandre and Müller, Klaus-Robert: “Learning representations of molecules and materials with atomistic neural networks”. In: *Machine Learning Meets Quantum Physics*. Springer, 2020, pp. 215–230 (cit. on p. 53).

[Sch97] Schuster, Mike and Paliwal, Kuldip K: “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681 (cit. on p. 136).

[Sco18] Scott, David W.: “Kernel Density Estimation”. In: *Wiley StatsRef: Statistics Reference Online*. American Cancer Society, 2018, pp. 1–7. doi: https://doi.org/10.1002/9781118445112.stat07186.pub2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat07186.pub2. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat07186.pub2 (cit. on p. 105).

[Sha16] Shahroudy, Amir; Liu, Jun; Ng, Tian-Tsong and Wang, Gang: “Ntu rgb+ d: A large scale dataset for 3d human activity analysis”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1010–1019 (cit. on p. 133).

[Sil86] Silverman, B. W: Density estimation for statistics and data analysis. London, UK: Chapman & Hall, 1986 (cit. on p. 123).

[Soh15] Sohn, Kihyuk; Lee, Honglak and Yan, Xinchen: "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems* 28 (2015), pp. 3483–3491 (cit. on p. 14).

[Sri14] Srivastava, Nitish; Hinton, Geoffrey; Krizhevsky, Alex; Sutskever, Ilya and Salakhutdinov, Ruslan: "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 81).

[Sto89] Stone, Maureen C and DeRose, Tony D: "A geometric characterization of parametric cubic curves". In: *ACM Transactions on Graphics (TOG)* 8.3 (1989), pp. 147–163 (cit. on p. 28).

[Sut14] Sutskever, Ilya; Vinyals, Oriol and Le, Quoc V.: "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112 (cit. on pp. 7, 90).

[Sza21] Szandała, Tomasz: "Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks". In: *Bio-inspired Neurocomputing*. Springer, 2021, pp. 203–224 (cit. on p. 53).

[Tay07] Taylor, Graham W; Hinton, Geoffrey E and Roweis, Sam T: "Modeling human motion using binary latent variables". In: *Advances in neural information processing systems*. 2007, pp. 1345–1352 (cit. on p. 134).

[Tha19] Tharwat, Alaa; Elhoseny, Mohamed; Hassanien, Aboul Ella; Gabel, Thomas and Kumar, Arun: "Intelligent Bézier curve-based path planning model using Chaotic Particle Swarm Optimization algorithm". In: *Cluster Computing* 22.2 (2019), pp. 4745–4766 (cit. on p. 28).

[Van11] Van Den Berg, Jur; Guy, Stephen J; Lin, Ming and Manocha, Dinesh: “Reciprocal n-body collision avoidance”. In: *Robotics research*. Springer, 2011, pp. 3–19 (cit. on p. 85).

[Vas17] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz and Polosukhin, Illia: “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S. and Garnett, R. Vol. 30. Curran Associates, Inc., 2017. url: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 7, 90, 153).

[Vem18] Vemula, Anirudh; Muelling, Katharina and Oh, Jean: “Social attention: Modeling attention in human crowds”. In: *2018 IEEE international Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4601–4607 (cit. on pp. 90, 91).

[Wan21] Wang, Yongqiang; Shi, Yangyang; Zhang, Frank; Wu, Chunyang; Chan, Julian; Yeh, Ching-Feng and Xiao, Alex: “Transformer in action: a comparative study of transformer-based acoustic models for large scale speech recognition applications”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 6778–6782 (cit. on p. 9).

[War79] Waring, Edward: “Vii. problems concerning interpolations”. In: *Philosophical transactions of the royal society of London* 69 (1779), pp. 59–67 (cit. on p. 28).

[Wer90] Werbos, Paul J: “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560 (cit. on p. 11).

[Wil97] Williams, Christopher KI: “Computing with infinite networks”. In: *Advances in neural information processing systems* (1997), pp. 295–301 (cit. on p. 11).

[Yan19] Yang, Dongfang; Li, Linhui; Redmill, Keith and Özgüner, Ümit: "Top-view trajectories: A pedestrian dataset of vehicle-crowd interaction from controlled experiments and crowded campus". In: *2019 IEEE Intelligent Vehicles Symposium (IV).* IEEE. 2019, pp. 899–904 (cit. on p. 87).

[Yi15] Yi, Shuai; Li, Hongsheng and Wang, Xiaogang: "Understanding pedestrian behaviors from stationary crowd groups". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2015, pp. 3488–3496 (cit. on p. 87).

[Yu17] Yu, Lantao; Zhang, Weinan; Wang, Jun and Yu, Yong: "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Proceedings of the AAAI conference on artificial intelligence.* Vol. 31. 1. 2017 (cit. on p. 16).

[Yu20] Yu, Cunjun; Ma, Xiao; Ren, Jiawei; Zhao, Haiyu and Yi, Shuai: "Spatio-Temporal Graph Transformer Networks for Pedestrian Trajectory Prediction". In: *European Conference on Computer Vision.* Springer. 2020, pp. 507–523 (cit. on p. 90).

[Yua21] Yuan, Ye; Weng, Xinshuo; Ou, Yanglan and Kitani, Kris: AgentFormer: Agent-Aware Transformers for Socio-Temporal Multi-Agent Forecasting. 2021 (cit. on p. 90).

[Zey19] Zeyer, Albert; Bahar, Parnia; Irie, Kazuki; Schlüter, Ralf and Ney, Hermann: "A comparison of Transformer and LSTM encoder decoder models for ASR". In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).* IEEE. 2019, pp. 8–15 (cit. on p. 9).

[Zho11] Zhou, Shuheng; Rütimann, Philipp; Xu, Min and Bühlmann, Peter: "High-dimensional covariance estimation based on Gaussian graphical models". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2975–3026 (cit. on p. 151).

Own publications

[0] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: “On the reliability of LSTM-MDL models for pedestrian trajectory prediction”. In: *Representations, Analysis and Recognition of Shape and Motion from Imaging Data (RFMI).* Savoie, France, 2017. doi: 10.1007/978-3-030-19816-9_2.

[0] Hug, R.; Hübner, W. and Arens, M.: “Interactive concepts for shaping generative models of spatial behavior”. In: *2017 IEEE 4th International Conference on Soft Computing Machine Intelligence (ISCMI).* 2017, pp. 35–41. doi: 10.1109/ISCMI.2017.8279594.

[0] Hug, R.; Hübner, W. and Arens, M.: “Supporting generative models of spatial behavior by user interaction”. In: *25th European Symposium on Artificial Neural Networks, ESANN 2017, Bruges, Belgium, April 26-28, 2017.* 2017. url: http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2017-91.pdf.

[0] Becker, S.; Hug, R.; Hübner, W. and Arens, M.: “RED: A simple but effective Baseline Predictor for the TrajNet Benchmark”. In: *European Conference on Computer Vision (ECCV) Workshops.* Springer International Publishing, 2018. doi: 10.1007/978-3-030-11015-4_13.

[0] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: “Particle-based Pedestrian Path Prediction using LSTM-MDL Models”. In: *21st International Conference on Intelligent Transportation Systems (ITSC).* 2018, pp. 2684–2691. doi: 10.1109/ITSC.2018.8569478.

[0] Becker, S.; Hug, R.; Hübner, W. and Arens, M.: “An RNN-based IMM Filter Surrogate”. In: *Scandinavian Conference on Image Analysis (SCIA).* Vol. 11482. Springer International Publishing, 2019, pp. 387–398. doi: 10.1007/978-3-030-20205-7_32.

[0] Bauckhage, C.; Hübner, W.; Hug, R. and Paß, G.: “12 Tiefe neuronale Netze”. In: *Handbuch der Künstlichen Intelligenz*. Berlin, Boston: De Gruyter, Dec. 2020, pp. 509–570. doi: https://doi.org/10.1515/9783110659948-012. url: https://www.degruyter.com/view/book/9783110659948/10.1515/9783110659948-012.xml.

[0] Bauckhage, C.; Hübner, W.; Hug, R.; Paß, G. and Rüping, S.: “11 Grundlagen des Maschinellen Lernens”. In: *Handbuch der Künstlichen Intelligenz*. Berlin, Boston: De Gruyter, Dec. 2020, pp. 429–508. doi: https://doi.org/10.1515/9783110659948-011. url: https://www.degruyter.com/view/book/9783110659948/10.1515/9783110659948-011.xml.

[0] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: “A complementary trajectory prediction benchmark”. In: *ECCV Workshop on Benchmarking Trajectory Forecasting Models (BTFM)*. 2020.

[0] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: “A Short Note on Analyzing Sequence Complexity in Trajectory Prediction Benchmarks”. In: *Workshop on Long-term Human Motion Prediction (LHMP)*. 2020.

[0] Hug, R.; Hübner, W. and Arens, M.: “Introducing Probabilistic Bézier Curves for N-Step Sequence Prediction”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.06 (Apr. 2020), pp. 10162–10169. doi: 10.1609/aaai.v34i06.6576. url: https://ojs.aaai.org/index.php/AAAI/article/view/6576.

[0] Becker, S.; Hug, R.; Hübner, W.; Arens, M. and Morris, B. T.: “Generating Synthetic Training Data for Deep Learning-Based UAV Trajectory Prediction”. In: *2nd International Conference on Robotics, Computer Vision and Intelligent Systems* (2021). to appear.

[0] Becker, S.; Hug, R.; Hübner, W.; Arens, M. and Morris, B. T.: “Handling Missing Observations with an RNN-based Prediction-Update Cycle”. In: *The 19th International Conference on Computer Analysis of Images and Patterns* (2021). to appear.

[0] Hug, R.; Becker, S.; Hübner, W. and Arens, M.: "Quantifying the Complexity of Standard Benchmarking Datasets for Long-Term Human Trajectory Prediction". In: *IEEE Access* 9 (2021), pp. 77693–77704. doi: 10.1109/ACCESS.2021.3082904.

Supervised student theses

[0] Hargens, O.: “Detektion ungewöhnlichen Verhaltens auf unterschiedlichen Zeitskalen mittels LSTM-Netzwerken”. Bachelor’s thesis. Karlsruhe Institute of Technology (KIT), 2017.

[0] Höß, A.: Pedestrian trajectory prediction using LSTM networks. Master’s thesis. 2018.

[0] Fehrenbach, J.: Interactive Object Segmentation for Intelligent Annotation Tools. Master’s thesis. 2020.

List of Figures

List of Tables

Acronyms

ADE	Average Displacement Error
BNN	Bayesian Neural Network
BRNN	Bayesian Recurrent Neural Network
CNN	Convolutional Neural Network
CVAE	Conditional Variational Autoencoder
DGP	Deep Gaussian Process
ELBO	Evidence Lower Bound
ELU	Exponential Linear Unit
FDE	Final Displacement Error
GAN	Generative Adversarial Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GP	Gaussian Process
GRU	Gated Recurrent Unit

LSTM	Long Short-Term Memory
MAE	Mean Angle Error
MDN	Mixture Density Network
NLL	Negative Log-Likelihood
$\mathcal{N}$-MDN	$\mathcal{N}$-Curve Mixture Density Network
ORCA	Optimal Reciprocal Collision Avoidance
RGP	Recurrent Gaussian Process
R-MDN	Recurrent Mixture Density Network
RNN	Recurrent Neural Network
seq2seq	Sequence-to-Sequence
SMC	Sequential Monte Carlo
TCN	Temporal Convolutional Network
TF	Transformer Network
VAE	Variational Autoencoder
VRNN	Variational Recurrent Neural Network