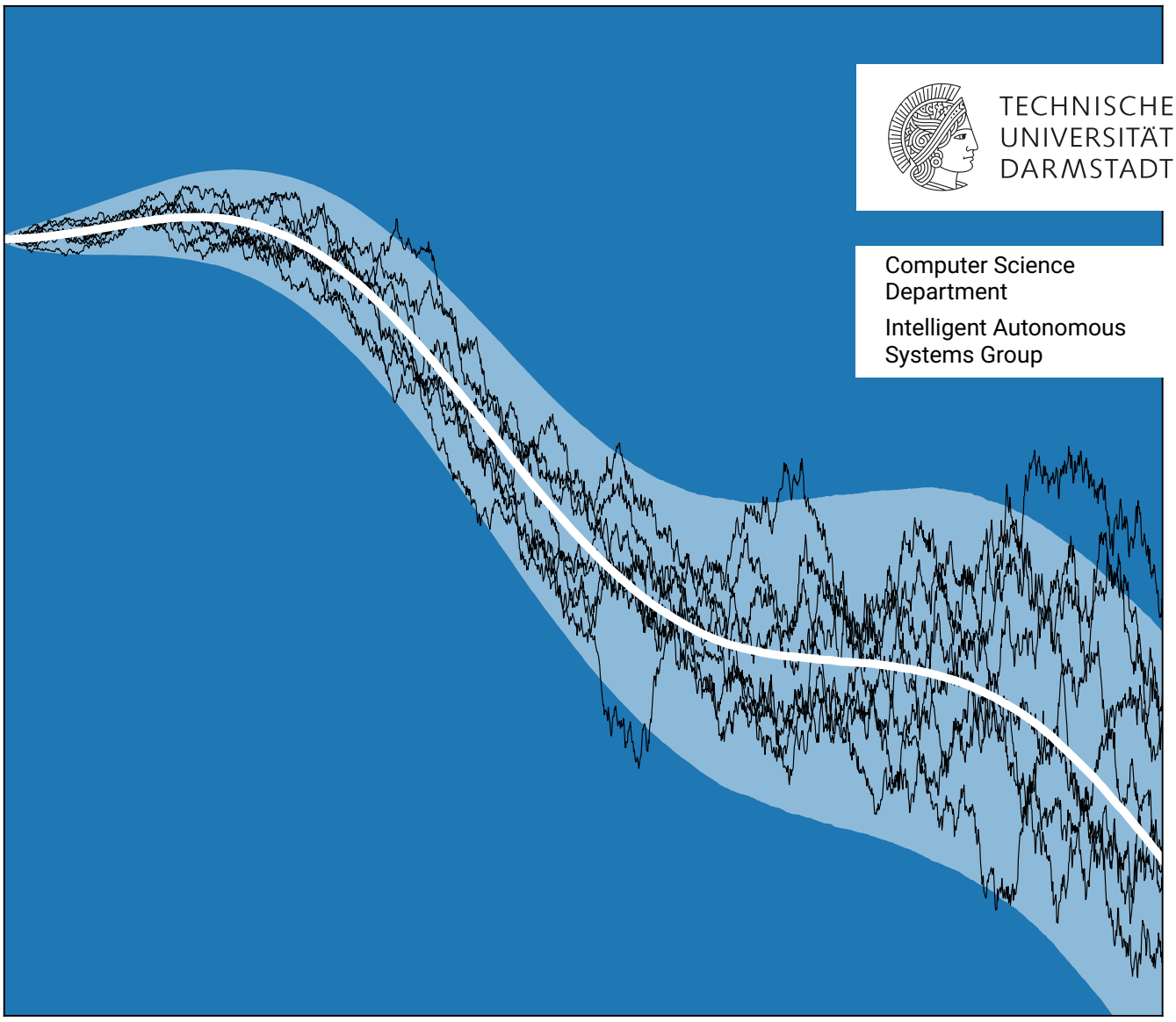
# Deterministic Approximations for Deep State-Space Models

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Andreas Look aus Puschkino, Russische Föderation
Tag der Einreichung: 23.06.2023, Tag der Prüfung: 23.10.2023

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Prof. David Duvenaud, Ph.D.
3. Gutachten: Prof. Melih Kandemir, Ph.D.
Darmstadt, Technische Universität Darmstadt



Deterministic Approximations for Deep State-Space Models

Genehmigte Dissertation von Andreas Look

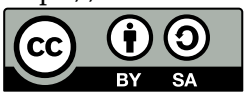
Tag der Einreichung: 23.06.2023
Tag der Prüfung: 23.10.2023

Darmstadt, Technische Universität Darmstadt

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-263529
URL: http://tuprints.ulb.tu-darmstadt.de/26352
Jahr der Veröffentlichung auf TUprints: 2023

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de





## Erklärungen laut Promotionsordnung

### § 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

### § 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

### § 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

### § 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 23.06.2023

A. Look

# Abstract

This thesis focuses on neural network based modeling of stochastic dynamical systems with applications in the context of autonomous driving. We define three goals for the model that must be achieved with low computational cost due to the use of low-compute and energy-constrained chips in autonomous vehicles. First, our model must accurately capture the data uncertainty, which is also referred to as the aleatoric uncertainty. The data uncertainty cannot be reduced by collecting more data since we only have partial information. In essence, we are unable to observe all states, such as the driver's intention. To illustrate this, consider a vehicle approaching a junction with the choice of turning left or right. If the driver does not use an indicator, we cannot determine which direction he will follow. Second, the model must account for interactions between different traffic participants, as traffic is highly interactive. Modeling interactions between traffic participants is vital for accurate traffic forecasting, as the actions of one traffic participant can impact the actions of other traffic participants. For example, imagine a scenario where one vehicle is merging into the lane of another vehicle. Both vehicles need to interact and adjust their speed to accommodate the lane merging. Lastly, as it is impossible to include all traffic scenarios in the training data set, the model needs to account for model uncertainty that arises from the lack of knowledge, which is also known as epistemic uncertainty. Model uncertainty is especially important for traffic scenarios that have not been observed during training. Without accounting for model uncertainty, the model is limited to modeling the intrinsic data uncertainty.

Throughout this thesis, we introduce several advancements to *Deep State-Space Models* (DSSMs) that address the challenges of capturing intrinsic data uncertainty, modeling interactions, and incorporating model uncertainty, all while ensuring low computational cost. DSSMs extend state-space models towards neural transition and emission models. A DSSM describes a partially observable system where each emission is generated by a corresponding latent state. The dynamics of the latent states follow a Markovian structure, where the state at each time point is dependent solely on the previous time point's state. Due to the use of nonlinear neural networks in the transition and emission models, DSSMs

offer high modeling capacity. Moreover, the stochasticity in the transition and emission models allows DSSMs to effectively capture the inherent data uncertainty.

After an introduction and reviewing relevant background material, we focus in the first part of the thesis on fully observed dynamical systems before transitioning to partially observed systems in the subsequent parts. Classical frameworks for simulating stochastic dynamical systems heavily rely on Monte Carlo sampling. As we demonstrate in this thesis, accurate prediction necessitates many particles, which induces a prohibitively high computational cost. To address this issue, we propose an alternative method that is computationally efficient and avoids the need for extensive Monte Carlo sampling. Our method relies on an assumed density approach to approximate the predictive distribution of the model. Specifically, we approximate the model's predictive distribution as a Gaussian at each time step. We estimate its moments by progressive moment matching horizontally in the time direction and vertically through neural network layers. Our proposed method is computationally more efficient than existing numerical integration schemes, as it exploits the layered structure of neural networks. This unimodal approximation lays the foundation for more complex approximations in the later parts. To assess the efficacy of our approach, we explore the application of our method in different domains.

In the second part of this thesis, we focus on partially observable systems and extend our framework towards deterministic uncertainty modeling with interacting agents, where each agent represents a vehicle in an autonomous driving setting. As a graph can capture the relations between different agents, we use a DSSM with graph neural networks in the transition model. Moreover, we extend our deterministic moment matching scheme to accommodate the multimodal nature of traffic forecasting. We demonstrate the applicability of our proposed framework on different autonomous driving datasets.

Finally, we address the challenge of incorporating model uncertainty into DSSMs, which is the uncertainty arising from the lack of knowledge. We achieve this by introducing uncertainty over the neural network weights in the transition model. However, accounting for both data and model uncertainty during inference is computationally expensive, as it requires marginalization over both sources of uncertainty. To address this pain point, we extend our deterministic approximation framework towards uncertainty propagation rules that account for both sources of uncertainty. We provide benchmarks on different domains that demonstrate the applicability of our model as a general-purpose tool.

# Zusammenfassung

In dieser Dissertation liegt der Fokus auf der Modellierung dynamischer Systeme im Kontext des autonomen Fahrens mithilfe neuronaler Netze. Dabei legen wir drei Anforderungen an das Modell fest, die aufgrund der Verwendung von leistungsarmen und energiesparenden Chips in autonomen Fahrzeugen mit geringen Rechenkosten erfüllt werden müssen. Die erste Anforderung besteht darin, dass unser Modell die aleatorische Unsicherheit präzise erfassen muss. Diese Unsicherheit kann nicht durch das Sammeln zusätzlicher Daten reduziert werden. Sie entsteht, da wir nicht in der Lage sind, alle Zustände vollständig zu beobachten, wie zum Beispiel die Absicht des Fahrers. Um dies zu veranschaulichen, betrachten wir ein Fahrzeug, das sich einer Kreuzung nähert und die Wahl hat, nach links oder rechts abzubiegen. Wenn der Fahrer keinen Blinker verwendet, können wir nicht vorhersagen, in welche Richtung er abbiegen wird. Die zweite Anforderung besteht darin, dass das Modell die Interaktionen zwischen verschiedenen Verkehrsteilnehmern berücksichtigen muss. Die Modellierung dieser Interaktionen ist entscheidend für die Verkehrsvorhersage, da die Handlungen eines Verkehrsteilnehmers die Handlungen anderer Verkehrsteilnehmer beeinflussen können. Ein Beispiel hierfür ist eine Situation, in der ein Fahrzeug in die Spur eines anderen Fahrzeugs einfädelt. Beide Fahrzeuge müssen miteinander interagieren und ihre Geschwindigkeit anpassen, um das Einfädeln zu ermöglichen. Schließlich muss unser Modell die epistemische Unsicherheit berücksichtigen, die aus dem Mangel an Wissen resultiert, da unser Trainingsdatensatz nicht alle möglichen Verkehrsszenarien abdecken kann. Die Berücksichtigung der epistemischen Unsicherheit ist besonders wichtig für Verkehrsszenarien, die während des Trainings nicht beobachtet wurden. Ohne diese Unsicherheit einzubeziehen, ist das Modell auf die Modellierung der aleatorischen Unsicherheit beschränkt.

Um diese Herausforderungen zu bewältigen und gleichzeitig einen geringen Rechenaufwand zu gewährleisten, stellen wir verschiedene Erweiterungen für neuronale Zustandsraummodelle vor. Ein Zustandsraummodell beschreibt ein teilweise beobachtetes System, bei dem jede Emission von einem entsprechenden latenten Zustand erzeugt wird. Die Dynamik der latenten Zustände folgt einer Markov-Struktur, bei der der Zustand

zu jedem Zeitpunkt ausschließlich vom Zustand des vorherigen Zeitpunkts abhängt. Die Stochastizität im Zustandsraummodel ermöglicht es uns, die aleatorische Unsicherheit zu modellieren.

Nach einer Einleitung und der Vorstellung relevanter Hintergrundinformationen konzentrieren wir uns im ersten Teil der Dissertation zunächst auf vollständig beobachtbare Systeme, bevor wir zu teilweise beobachtbaren Systemen übergehen. Klassische Methoden zur Simulation stochastischer dynamischer Systeme verwenden oft Monte-Carlo-Simulationen. In dieser Dissertation zeigen wir, dass eine genaue Vorhersage eine hohe Zahl von Partikeln erfordert, was zu hohen Rechenkosten führt. Um dieses Problem zu lösen, schlagen wir eine alternative Methode vor, die rechenintensive Monte-Carlo-Simulationen vermeidet. Unsere neue Methode verwendet eine Gauß-Verteilung zur Approximation der Vorhersageverteilung in jedem Zeitschritt. Die Momente der Gauß-Verteilung im nächsten Zeitschritt werden als eine Funktion der Momente im vorherigen Zeitschritt bestimmt. Dabei werden die Momente horizontal in Zeitrichtung und vertikal durch die Schichten der neuronalen Netze propagiert. Unsere vorgeschlagene Methode ist rechnerisch effizienter als bestehende numerische Integrationsverfahren, da sie die schichtweise Struktur neuronaler Netze ausnutzt. Um die Effektivität unseres Ansatzes zu bewerten, untersuchen wir die Anwendung unserer Methode in verschiedenen Domänen.

Im zweiten Teil der Dissertation konzentrieren wir uns auf teilweise beobachtbare Systeme und erweitern unsere Methode aus dem vorherigen Teil der Dissertation, um dynamische Systeme mit interagierenden Agenten zu modellieren. Hierbei repräsentiert jeder Agent ein Fahrzeug in einer autonomen Fahrumgebung. Da ein Graph Beziehungen zwischen Agenten modellieren kann, verwenden wir neuronale Netze, die auf die Modellierung von Graphen spezialisiert sind. Zusätzlich erweitern wir die unimodale Approximation, die im ersten Teil der Dissertation vorgestellt wurde, zu einer multimodalen Approximation. Wir zeigen die Anwendbarkeit unserer Erweiterungen anhand verschiedener Datensätze im Kontext der Verkehrsvorhersage.

Im letzten Teil der Dissertation führen wir Unsicherheiten in den Gewichten der neuronalen Netze ein, um die epistemische Unsicherheit zu modellieren. Ohne Berücksichtigung der epistemischen Unsicherheit ist die Modellierung auf aleatorische Unsicherheit beschränkt. Die Berücksichtigung sowohl der epistemischen als auch der aleatorischen Unsicherheit ist rechnerisch aufwendig, da über beide Unsicherheitsquellen während der Inferenz marginalisiert werden muss. Wir erweitern unsere Methode aus dem ersten Teil der Dissertation zur Schätzung der Vorhersageverteilung, um den Einsatz von neuronalen Netzen mit stochastischen Gewichten zu ermöglichen. Wir demonstrieren die Anwendbarkeit unserer Erweiterungen anhand von Experimenten in verschiedenen Domänen.

# Contents

**Abstract** **iv**

**1. Introduction** **1**
1.1. Contributions . . . 3
1.2. Outline . . . 4

**2. Background** **6**
2.1. Deep State-Space Models . . . 6
2.2. Transition Kernel . . . 7
2.3. Gaussian Filtering . . . 8
2.4. Graph Neural Networks . . . 9
2.5. Probabilistic Neural Networks . . . 10

**3. An Assumed Density Approximation for Fully Observed Dynamical Systems with Deep Gaussian Transition Models** **12**
3.1. Deep Gaussian Transition Models . . . 13
3.2. Bidimensional Moment Matching . . . 14
3.2.1. Assumed Process Density . . . 15
3.2.2. Computing the Moments of the Update Functions . . . 16
3.2.3. Computing the Cross-Covariance . . . 17
3.2.4. The Bidimensional Moment Matching Algorithm . . . 19
3.3. Moments of Layers and their Derivatives . . . 20
3.3.1. Linear Layer . . . 20
3.3.2. ReLU Activation . . . 21
3.3.3. Dropout . . . 22
3.4. Numerical Properties . . . 24
3.4.1. Cubature . . . 24
3.4.2. Approximation Error. . . . 25
3.4.3. Computational Cost. . . . 26

3.4.4. Multimodal Processes. . . . . 27
3.5. Experiments . . . . . 27
3.5.1. Evaluation Criteria . . . . . 28
3.5.2. Stochastic Recurrent Layers . . . . . 29
3.5.3. Time Series Classification . . . . . 34
3.5.4. Dynamical System Modeling . . . . . 36
3.5.5. High Dimensional Dynamics . . . . . 40
3.6. Summary . . . . . 42

**4. Cheap and Deterministic Inference for Deep State-Space Models of Interacting Dynamical Systems 43**
4.1. Graph Deep State-Space Models . . . . . 46
4.2. Deterministic Approximations for GDSSMs . . . . . 48
4.2.1. Parameter Inference . . . . . 49
4.2.2. Approximating the Predictive Log-Likelihood . . . . . 50
4.2.3. Output Moments of Graph Neural Network Layers . . . . . 52
4.2.4. Sparse Covariance Approximation . . . . . 55
4.3. Experiments . . . . . 58
4.3.1. Evaluation Criteria . . . . . 58
4.3.2. rounD . . . . . 59
4.3.3. NGSIM . . . . . 63
4.3.4. Covariance Approximations . . . . . 65
4.3.5. Out-of-Distribution Testing . . . . . 68
4.4. Summary . . . . . 70

**5. Sampling-Free Probabilistic Deep State-Space Models 71**
5.1. Probabilistic Deep State-Space Models . . . . . 73
5.2. Deterministic Approximations for ProDSSMs . . . . . 74
5.2.1. Extending the Bidimensional Moment Matching Algorithm . . . . . 75
5.2.2. Gaussian Filtering . . . . . 78
5.2.3. Parameter Inference . . . . . 79
5.2.4. Approximating the Predictive Log-Likelihood . . . . . 81
5.2.5. Output Moments of Probabilistic Neural Network Layers . . . . . 83
5.3. Runtime . . . . . 85
5.3.1. Theoretical Runtime . . . . . 85
5.3.2. Measured Runtime . . . . . 86
5.4. Experiments . . . . . 87
5.4.1. Stochastic Recurrent Layers . . . . . 88

5.4.2. Filtering . . . 92
5.4.3. Dynamical System Modeling . . . 94
5.5. Pros and Cons of Different ProDSSM Variants . . . 98
5.6. Summary . . . 99

**6. Conclusion** **100**
6.1. Summary . . . 100
6.2. Outlook . . . 102

**A. Supplementary Material for Chap. 4** **104**
A.1. Proof of Theorem 1 . . . 104
A.2. Training Details . . . 105
A.2.1. rounD . . . 106
A.2.2. NGSIM . . . 106
A.2.3. Out-of-Distribution Testing . . . 106
A.3. Alternative Parameter Inference Methods . . . 106
A.4. Experimental Setup for Out-of-Distribution Testing . . . 107
A.4.1. Dataset Construction . . . 107
A.4.2. Map Processing . . . 107
A.5. Network Architectures . . . 109

**B. Supplementary Material for Chap. 5** **111**
B.1. Similarities between ELBO and Predictive Variational Bayesian Inference . 111
B.2. Moments of a Product of Correlated Normal Variables . . . 112

**List of Acronyms** **115**

**List of Figures** **118**

**List of Tables** **120**

**Curriculum Vitae** **121**

**Publication List** **122**

**Bibliography** **123**

# 1. Introduction

Autonomous driving is a challenging task that can be split into three subproblems. First, the surrounding environment needs to be perceived, which involves, for example, tracking vehicles with onboard cameras and other sensors [1, 2]. Subsequently, it is necessary to reason about the future trajectories of surrounding traffic participants [3, 4, 5, 6, 7, 8]. Finally, autonomous vehicles must decide which action to take while ensuring safety requirements [9, 10, 11]. However, there are also approaches that address these subproblems together. For instance, some methods directly forecast the future behavior of traffic participants based on sensor data [12, 13] or learn a policy from raw data [14] and mitigates modeling the behaviour of surrounding traffic participants.

This thesis focuses on modeling complex dynamical systems using observations from a perception module. Our emphasis lies in modeling the dynamical aspects of the system, and we do not delve into the processing steps involved in handling image or radar data. More concretely, we endeavor to answer the following question:

*How can we efficiently model dynamical systems with interacting agents while accounting for both data and model uncertainty?*

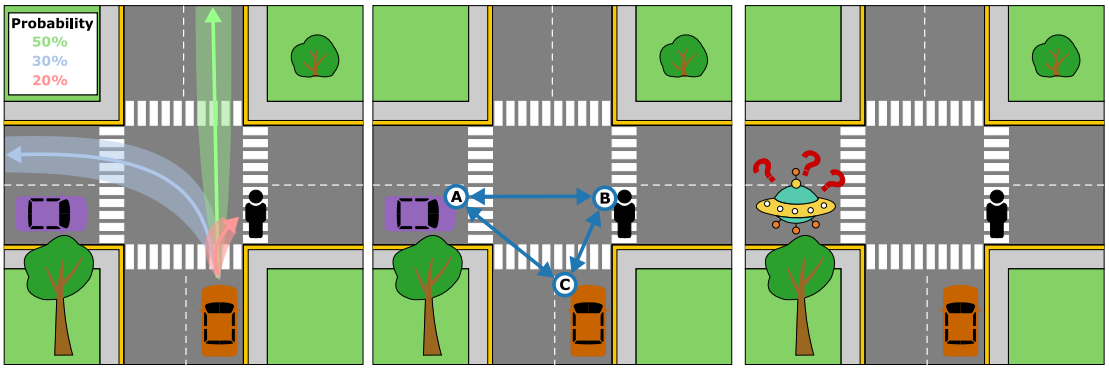
To address this question, we define three goals that the model must achieve, as depicted in Fig. 1.1. It is crucial to accomplish these goals with low cost in order to ensure efficiency. Efficiency plays a vital role in the context of traffic forecasting because autonomous vehicles need to reason in real-time while making computations on low-energy and low-compute chips. Although high-compute chips for vehicles are available, their cost and energy consumption limit their adoption [15]. Therefore, there is a strong demand for efficient algorithms that can operate effectively on low-compute chips [16].

The first goal of the model is to accurately capture the data uncertainty [17], which is also known as aleatoric uncertainty [18, 19]. The data uncertainty arises from the intrinsic variability of the data, which cannot be reduced even if we collect more data. Traffic forecasting is an inherently stochastic task with high data uncertainty as not all states

can be observed, such as the intentions of other traffic participants. Without the ability to observe intentions, we cannot rule out various hypotheses about the future. For instance, consider a vehicle approaching a junction with the possibility to turn left or right. If the driver does not use the indicator, we cannot determine which direction the driver will follow. It is crucial to capture all possible hypotheses, as overlooking even one can lead to potentially catastrophic scenarios.

Second, the model needs to consider interactions between different traffic participants. Traffic is a highly interactive system, and the actions of one traffic participant influence the actions of other traffic participants. For instance, if a driver is already in the process of merging into a lane, it can cause hesitation in another driver who intends to make a lane change into that same lane. Ignoring these interactions will result in an imprecise model.

Third, the model must capture the model uncertainty [17] that arises from the lack of knowledge, also known as epistemic uncertainty [18, 19]. In contrast to the data uncertainty, the model uncertainty can be reduced by collecting more data. However, due to the vast number of possible traffic scenarios, it is impossible to include all of them in the training data. Consequently, the model should handle novel and unseen scenarios by exhibiting higher model uncertainty. This higher model uncertainty prevents the model from making overly confident predictions in unfamiliar situations, enabling cautious and adaptive decision-making in autonomous driving applications [20].




(a) Data uncertainty. (b) Interactions. (c) Model uncertainty.

Figure 1.1.: We define three key goals for probabilistic dynamical system models in the context of autonomous driving: (a) capturing the data uncertainty, (b) modeling interactions, and (c) incorporating model uncertainty. Moreover, we need to achieve these goals with limited computational resources.

## 1.1. Contributions

The contributions of this thesis lead to a model that achieves the three aforementioned goals, i.e., capturing data uncertainty, accounting for interactions, and incorporating model uncertainty. We achieve these three goals with low computational cost. We rely on *Deep State-Space Models* (DSSMs) that provide a principled solution for modeling an unknown dynamical process. In short, DSSMs extend *State-Space Models* (SSMs) towards neural transition and emission models. SSMs are a model class for dynamical systems that assumes that each observation is emitted by a latent variable [21, 22]. The latent variables are coupled via first-order Markovian dynamics, i.e., the state at each time point depends solely on the previous time point's state. The dynamics in latent space are described via the transition model, while the relationship between the latent and observed state is captured by the emission model. By relying on DSSMs, we are able to effectively capture the inherent uncertainty present in the data. These models explicitly account for it through the stochasticity in the transition and emission models. Below is a summary of the contributions of this thesis.

*(i) Assumed Density Approximation*: We propose an algorithm for approximating the predictive distribution of a DSSM, which enables accurate predictions with low computational cost. Our algorithm, called *Bidimensional Moment Matching* (BMM), models the predictive distribution as a Gaussian at each time step. To address the intractable expectation and covariance integrals, we perform moment matching along two dimensions: (i) horizontally across time and (ii) vertically across the layers of the neural network. In order to apply the BMM algorithm, the first two output moments for each neural net layer and the expected Jacobian for the entire network must be computed. Output moments are available in the literature for affine transformation, ReLU activation, and exponential activation. We introduce a novel approximation to the expected Jacobian by assuming decoupled activations across layers. Moreover, to enable the prediction of multimodal distributions, we introduce a *Gaussian Mixture Model* (GMM) over the initial latent state and apply the BMM algorithm to each mixture component individually. By accounting for multiple modes, we can more accurately capture the uncertainty in the data.

*(ii) Interaction Modeling*: As the relations between agents can be expressed as a graph, we propose *Graph Neural Networks* (GNNs) as a building block for DSSMs. A benefit of GNNs over standard feed-forward neural nets is their ability to handle a variable number of agents. To apply the BMM algorithm to DSSMs with GNN layers, we need to be able to calculate the output moments of commonly used GNN layers. We derive moment matching rules for GNN layers, such as the agent-wise affine transformation or mean aggregation.

For scenarios involving many interacting agents, the covariance matrix calculation can become computationally unfeasible, as each pair of agents needs to have its covariance modeled. To tackle this challenge, we introduce several sparse covariance approximations.

*(iii) Weight Uncertainty*: DSSMs can capture only the intrinsic data uncertainty, also known as aleatoric uncertainty. To account for the model uncertainty that arises from limited knowledge, we introduce uncertainty over the neural network weights in the transition model. We adopt an *independent and identically distributed* (iid) Gaussian distribution to model the learnable weight distribution and investigate two distinct weight sampling techniques. The first technique involves resampling the weights at each time step, while the second technique involves sampling the weights only once at the initial time step. We derive an extension of the BMM algorithm for both sampling methods and output moments for neural net layers with weight uncertainty.

## 1.2. Outline

We outline the thesis in Fig. 1.2. After reviewing relevant background material in Chap. 2 we introduce our contributions in three chapters.

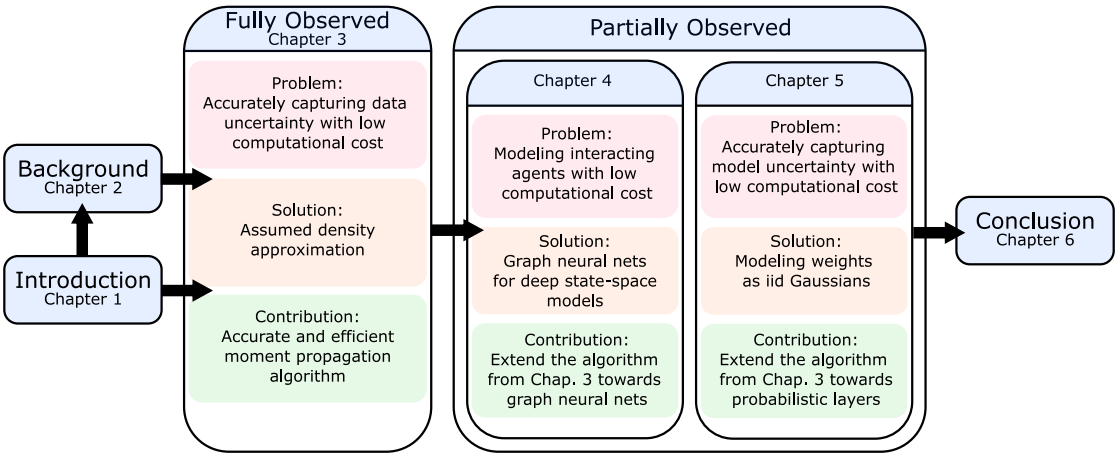



Figure 1.2.: Structure of the thesis. Chap. 2 provides an overview of relevant background material. Chap. 3, 4, and 5 form the main parts. We highlight the problem, the proposed solution, and our key contribution in the main chapters. We conclude the thesis in Chap. 6.

In Chap. 3, we focus on unimodal approximations for fully observed dynamical systems and introduce the BMM algorithm, which enables an accurate approximation of the predictive distribution with low computational cost. We support this chapter with experiments in three different domains: (i) stochastic recurrent layers, (ii) time series classification, and (iii) modeling of fully observed systems. This chapter is based on the article [23] that was published in the *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

In Chap. 4 we introduce GNNs in the transition model in order to account for interactions. We extend the BMM algorithm towards GNNs and multimodal predictions. We conduct experiments on different traffic forecasting datasets. This chapter is based on the article [24] that was published in the *Transactions on Machine Learning Research*.

In Chap. 5 we introduce uncertainty over the weights to account for model uncertainty. We derive moment matching rules for probabilistic layers and extend the BMM algorithm towards uncertainty propagation rules that account for both data and model uncertainty. We conduct experiments in three domains: (i) stochastic recurrent layers, (ii) filtering, and (iii) modeling of partially observed systems. This chapter is based on a preprint [25] that is submitted to the *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

We conclude the thesis and give an outlook in Chap. 6.

# 2. Background

We review relevant background material before we introduce our extensions to DSSMs. In Sec. 2.1, we provide an introduction to DSSMs. Approximations to the transition kernel and Gaussian filtering form the basis of our deterministic approximation of the predictive distribution. The transition kernel describes the distribution of a given point in time conditioned on a past state, while Gaussian filtering allows us to reason about the latent distribution. We review approximation methods for the transition kernel in Sec. 2.2 and Gaussian filtering in Sec. 2.3. We introduce graph neural nets in Sec. 2.4, which we use for interaction modeling. Lastly, we review in Sec. 2.5 probabilistic neural nets, a model class that incorporates model uncertainty.

## 2.1. Deep State-Space Models

*State-Space Models* (SSM) are a model class for dynamical systems [21], [22] that assume that each $D_y$-dimensional observed variable $y_t \in \mathbb{R}^{D_y}$ is emitted by a latent $D_x$-dimensional latent variable $x_t \in \mathbb{R}^{D_x}$. The latent variables are coupled via first-order Markovian dynamics, i.e., the state at time point $x_t$ only depends on the state of the previous time point $x_{t-1}$. Typically the observed state $y_{t-1}$ does not contain all the necessary information to predict the next observed state $y_t$. Consider the case of traffic forecasting in which the observed state $y_t$ contains the position of a vehicle but not its velocity or acceleration data. We can then supply the latent state $x_t$ with the missing information to allow for accurate forecasts about the next time point. Consequently, SSMs are a flexible model class that allows us to make reliable forecasts about complex systems. More formally, the generative model of a SSM can be expressed as

$$x_0 \sim p(x_0), \tag{2.1}$$

$$x_{t+1} \sim p(x_{t+1}|x_t), \tag{2.2}$$

$$y_t \sim p(y_t|x_t). \tag{2.3}$$

Above, $p(x_0)$ is the initial distribution, $p(x_{t+1}|x_t)$ is the transition model, and $p(y_t|x_t)$ is the emission model. The stochasticity in the transition and emission models describes the intrinsic data uncertainty.

A *Deep State-Space Model* (DSSM) is a SSM with neural networks in the transition and emission models. Commonly, these are modeled as input-dependent Gaussians [26, 27]. However, there exists also concurrent work that proposes more expressive distributions [28]. Finally, there exists work that couples state-space models with gated recurrent neural networks [29, 30] whose gating mechanism can help in learning long-term effects.

## 2.2. Transition Kernel

We provide background on how to compute the $t$-step transition kernel, $p(x_t|x_0)$ with $t > 1$, which allows us to propagate the latent state forward in time for general state-space models. It is defined by the following recurrence

$$p(x_t|x_0) = \int p(x_t|x_{t-1})p(x_{t-1}|x_0)dx_{t-1}, \tag{2.4}$$

where $p(x_t|x_{t-1})$ follows Eq. 2.2. Except for simple dynamical systems, such as linear ones [22], there exists no analytical solution, since the distribution $p(x_{t-1}|x_0)$ has to be propagated through the transition model $p(x_t|x_{t-1})$.

Various approximations to the transition kernel have been proposed that can be split into two groups: (i) *Monte Carlo* (MC) based approaches [31, 32, 33] and (ii) deterministic approximations based on *Assumed Densities* (AD) [34]. While MC based approaches can, in the limit of infinitely many samples, approximate arbitrarily complex distributions, they are often slow in practice, and their convergence is difficult to assess. In contrast, deterministic approaches often build on the assumption that the $t$-step transition kernel can be approximated by a Gaussian. This assumption can be justified if the transition model can be locally approximated by a linear Gaussian and the observations are sufficiently densely sampled [35]. The AD approach approximates the $t$-step transition kernel as

$$\begin{aligned} p(x_t|x_0) &\approx \int p(x_t|x_{t-1})\mathcal{N}(\mu_{t-1}, \Sigma_{t-1})dx_{t-1} \\ &\approx \mathcal{N}(\mu_t, \Sigma_t), \end{aligned} \tag{2.5}$$

where the latent state $x_t$ is recursively approximated as a Gaussian with mean $\mu_t \in \mathbb{R}^{D_x}$ and covariance $\Sigma_t \in \mathbb{R}^{D_x \times D_x}$. This simplifies the calculations for solving Eq. 2.4 to

approximating the first two moments. There exist generic numerical integration schemes such as cubature or linearization to approximate the intractable integrals [36, 34, 37]. However, as we will show in this thesis, standard integration schemes scale unfavorably with increasing dimensionality in terms of integration error and time requirements.

## 2.3. Gaussian Filtering

In filtering applications, we are interested in the distribution $p(x_t|y_{1:t})$, where $y_{1:t} = \{y_1, \dots, y_t\}$ denotes the past observations. We denote with the upper index $x$ the moments of $x_t \sim \mathcal{N}(\mu_t^x, \Sigma_t^x)$ and with upper index $y$ the moments of $y_t \sim \mathcal{N}(\mu_t^y, \Sigma_t^y)$. For deep state-space models, the filtering distribution becomes analytically intractable due to the use of nonlinear neural networks in the transition and emission models. To approximate this distribution, we use a general Gaussian Filter [22]. This approximation involves iterating through a prediction and an update step, which we will describe in the following. In concurrent literature [38], the term prior is used to refer to $p(x_t|y_{1:t-1})$. Building upon this line of research, we introduce the term joint prior to describe $p(x_t, y_t|y_{1:t-1})$.

*Prediction:* Approximate the prior $p(x_t|y_{1:t-1})$ with

$$\begin{aligned} p(x_t|y_{1:t-1}) &= \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1} \\ &\approx \int p(x_t|x_{t-1})\mathcal{N}(\mu_{t-1}^x, \Sigma_{t-1}^x)dx_{t-1} \\ &\approx \mathcal{N}(\mu_{t|t-1}^x, \Sigma_{t|t-1}^x), \end{aligned} \tag{2.6}$$

where $p(x_{t+1}|x_t)$ refers to the transition model defined in Eq. 2.2. We arrive at Eq. 2.6 by multiple rounds of moment matching. First, we approximate the filtering distribution $p(x_{t-1}|y_{1:t-1})$ as a Gaussian, and then we approximate the prior $p(x_t|y_{1:t-1})$ as another Gaussian. Here, the index $t|t'$ denotes prior moments, i.e., the moments at time step $t$ conditioned on the observations up to time step $t'$. If $t = t'$, we omit the double index.

*Update*: Approximate the joint prior $p(x_t, y_t|y_{1:t-1})$

$$\begin{aligned} p(x_t, y_t|y_{1:t-1}) &= p(y_t|x_t)p(x_t|y_{1:t-1}) \\ &\approx p(y_t|x_t)\mathcal{N}(\mu_{t|t-1}^x, \Sigma_{t|t-1}^x) \\ &\approx \mathcal{N}\left(\begin{bmatrix}\mu_{t|t-1}^x \\ \mu_{t|t-1}^y\end{bmatrix}, \begin{bmatrix}\Sigma_{t|t-1}^x & \Sigma_{t|t-1}^{xy} \\ \Sigma_{t|t-1}^{yx} & \Sigma_{t|t-1}^y\end{bmatrix}\right), \end{aligned} \tag{2.7}$$

where $\Sigma^{xy}_{t|t-1} \in \mathbb{R}^{D_x \times D_y}$ is the cross-covariance between $x_t$ and $y_t$ and the distribution $p(y_t|x_t)$ is defined in Eq. 2.3. Building a Gaussian approximation to the joint prior (Eq. 2.7) can be performed by similar numerical integration schemes as discussed in Sec. 2.2, i.e., cubature or linearization. Afterwards, we can calculate the filtering distribution $p(x_t|y_{1:t})$ by conditioning the joint prior on the observation $y_t$

$$p(x_t|y_{1:t}) \approx \mathcal{N}(\mu^x_t, \Sigma^x_t), \tag{2.8}$$

where Eq. 2.8 can be obtained from Eq. 2.7 by standard Gaussian conditioning [22]. The resulting distribution has the below moments

$$\mu^x_t = \mu^x_{t|t-1} + K_t(y_t - \mu^y_{t|t-1}), \tag{2.9}$$

$$\Sigma^x_t = \Sigma^x_{t|t-1} - K_t \Sigma^y_{t|t-1} K_t^\top, \tag{2.10}$$

where $K_t \in \mathbb{R}^{D_x \times D_y}$ is the Kalman gain

$$K_t = \Sigma^{xy}_{t|t-1} \left(\Sigma^y_{t|t-1}\right)^{-1}. \tag{2.11}$$

Prior work in the context of DSSM and Gaussian filters [28] encodes observations into a latent space with an invertible neural net and then relies on a linear SSM formulation to be able to solve the filtering equations exactly. To the best of our knowledge, no prior work applies Gaussian filters to general DSSMs.

## 2.4. Graph Neural Networks

*Graph neural networks* (GNNs) have emerged as a powerful method for interaction modeling [39, 40, 41]. Given a set of agents and relational information in the form of a graph, each agent corresponds to one node in the graph that is equipped with a set of features. The relation between the agents is encoded via the edges, and information exchange between the agents takes place by sending messages along the edges. By performing multiple rounds of message-passing, information can flow along the graph. This allows for interactions between non-adjacent agents.

More formally, we define the structure of our GNN as follows. For $M$ agents, a GNN receives as inputs a set of node features $x = \{x^m\}_{m=1}^M$, where $x \in \mathbb{R}^{MD_x}$ and $x^m \in \mathbb{R}^{D_x}$, and a set of edges $\mathcal{E} = \{e^{m,m'}\}_{m,m'=1}^M$ which is part of the context variable $\mathcal{I} \in \mathbb{R}^{D_\mathcal{I}}$. The

edge attribute $e^{m,m'}$ has a binary encoding, where $e^{m,m'} = 1$ if agent $m$ and agent $m'$ are related. The GNN output is an update of the node features, i.e., $z = \text{GNN}(x, \mathcal{I})$ with $z \in \mathbb{R}^{MD_z}$, and consists of the following two steps that may be repeated multiple times:

1. For each agent $m$, receive message $x^{\mathcal{N}_m} \in \mathbb{R}^{D_x}$ by aggregrating information from neighboring agents:

$$x^{\mathcal{N}_m} = \text{AGG}\left(\{x^{m'} | e^{m,m'} = 1\}\right) = \text{AGG}\left(x, \mathcal{I}\right), \tag{2.12}$$

   where $\{x^{m'} | e^{m,m'} = 1\}$ denotes the set of all neighbours of node $m$. The aggregation operation AGG : $\mathbb{R}^{MD_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{D_x}$ is permutation invariant, i.e., it does not change when the ordering of the inputs is swapped and generalizes to a varying number of inputs. A commonly used aggregation operation that we also apply in our work is the mean function.

2. For each agent $m$, update the node information:

$$z^m = \text{UPDATE}(x^m, x^{\mathcal{N}_m}, \mathcal{I}), \tag{2.13}$$

   where UPDATE : $\mathbb{R}^{D_x} \times \mathbb{R}^{D_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{D_z}$ is typically implemented by a neural network.

A simple form of an interacting dynamical system takes the features of each agent at its current position and connects agents within a predefined radius with edges. The GNN operation updates each agent's position and velocity information by taking information of the adjacent traffic participants into account.

## 2.5. Probabilistic Neural Networks

We refer to a neural network as probabilistic if it accounts for uncertainty over the weights. This includes a broader class of models than those in the Bayesian formalism. The classical Bayesian formalism defines a prior $p(w|\phi)$ with hyperparameters $\phi$ over the weights $w \in \mathbb{R}^{D_w}$ and a likelihood $p(\mathcal{D}|w)$ of observing the data $\mathcal{D}$. The posterior $p(w|\mathcal{D})$ is the quantity of interest. During posterior inference, the hyperparameters $\phi$ of the prior are kept constant. As an analytical solution to the posterior is intractable, either *Markov Chain Monte Carlo* (MCMC) [42] or *Variational Inference* (VI) [43] is used. VI introduces an approximate posterior $q(w)$ and maximizes the *Evidence Lower Bound* (ELBO)

$$\mathbb{E}_{q(w)}[\log p(\mathcal{D}|w)] - \text{KL}(q(w)|p(w|\phi)). \tag{2.14}$$

An alternative to the Bayesian formalism consists of first introducing noise over the weights and marginalizing it out in the subsequent step

$$\underset{\phi}{\text{argmax}} \log \int p(\mathcal{D}|w)p(w|\phi)dw, \tag{2.15}$$

which is also known as Type-II *Maximum Likelihood* (ML) or empirical Bayes. This is not a Bayesian method as the prior $p(w|\phi)$ is learned. In contrast, the prior is kept constant in the Bayesian formalism. Notably, [44] learned the prior while inferring the posterior over the neural network weights using VI. Maximizing the marginal likelihood has been used in various machine learning applications, such as evidential deep learning [45, 46, 47], prior networks [48, 49], or PAC-based deep learning [50, 51, 52].

# 3. An Assumed Density Approximation for Fully Observed Dynamical Systems with Deep Gaussian Transition Models

In this chapter, we derive a novel deterministic approximation to the transition kernel for fully observed dynamical systems driven by *Deep Gaussian Transition Models* (DGTMs). Our novel approximation allows for accurate and calibrated predictions with low computational cost. This solution builds the first step towards deterministic modeling of more complex dynamical systems that can be used for traffic forecasting.

A DGTM models the dynamics of an environment with a mean update function governing the deterministic component and a covariance update function governing the instantaneous distortions. In essence, the covariance update function models the inherent data uncertainty. Both update functions are parameterized by deep neural networks. We focus on DGTMs with residual connections, which have a large potential to provide an attractive tool to the machine learning community. These models exhibit strong theoretical links to *Neural Ordinary Differential Equations* (NODEs) [53], *Neural Stochastic Differential Equations* (NSDEs) [54], and *Gaussian Processes* (GPs) [55, 56].

We focus on a central observation: when dynamics is governed by a DGTM, accurate approximation of the transition kernel with *Monte Carlo* (MC) sampling requires a prohibitively large sample set, i.e., computation time. We present a new method that allows for a deterministic approximation of the transition kernel. This method offers well-calibrated prediction uncertainties while requiring much fewer computational resources compared to traditional Monte Carlo sampling techniques. As visualized for a toy case in Fig. 3.1, DGTM predictions with MC sampling require a large sample set to catch up with the calibration level of our deterministic method.

We summarize our contributions as follows:

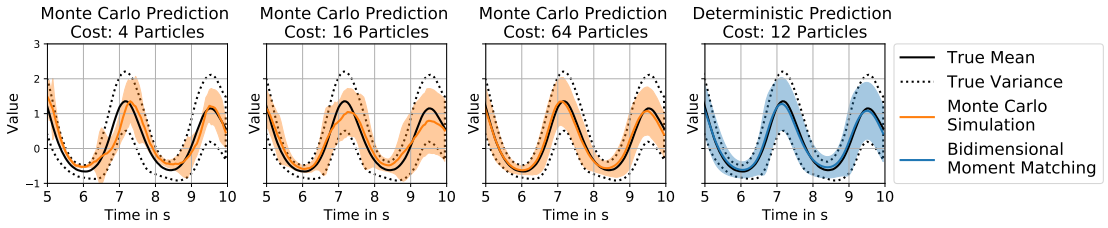



Figure 3.1.: Our deterministic approximation provides well-calibrated uncertainty scores with a computational cost equal to 12 particles. Reaching a comparable level of calibration by Monte Carlo sampling demands at least 64 particles.

*(i)* Performing *Bidimensional Moment Matching (BMM)* to approximate the intractable expectation and covariance integrals: horizontally across time and vertically across the layers of mean and covariance update neural nets.

*(ii)* Using Steins's lemma to simplify the calculation of covariances while matching moments across time.

*(iii)* Approximating the expected Jacobian of a neural net accurately while matching moments across layers.

We investigate the numerical properties of the BMM algorithm in Sec. 3.4 and provide benchmarks against well-established baselines in Sec. 3.5 on various applications.

## 3.1. Deep Gaussian Transition Models

We are concerned with the model family that describes the dynamics of a $D_x-$dimensional stochastic process $x_t$ as a deep Gaussian transition model with the below form

$$x_{t+1} \sim \mathcal{N}(x_{t+1}|x_t + f(x_t)\Delta t, \text{diag}(r(x_t))\Delta t). \tag{3.1}$$

Above, $f : \mathbb{R}^{D_x} \to \mathbb{R}^{D_x}$ is the mean update function that governs the deterministic component of the DGTM. The covariance update function $r : \mathbb{R}^{D_x} \to \mathbb{R}_+^{D_x}$ models the stochasticity of the system. We assume that both $f(x_t)$ and $r(x_t)$ are neural nets with varying numbers of hidden layers and activation functions. We use the shorthand notation $w = \{w_f, w_r\}$ to represent the set of all weights, where $w_f$ corresponds to the weights of the mean update and $w_r$ corresponds to the weights of the covariance update. Further,

$\Delta t \in \mathbb{R}_+$ is the time step size. We assume an evenly spaced discretization, though the time step size $\Delta t$ can be chosen dynamically if desired. The residual structure, i.e., $x_t + f(x_t)\Delta t$, allows for better gradient flow and eases the learning problem [57].

We can interpret the transition model (Eq. 3.1) as a discretized NSDE [54]. This interpretation allows us to access the $t$-step transition kernel $p(x_t|x_0, w)$ by solving the *Fokker-Planck-Kolmogorov* (FPK) equation, which is a potentially high-dimensional partial differential equation with an often intractable solution. As the transition kernel is necessary for likelihood-based parameter inference schemes as well as for uncertainty quantification, various approximations of $p(x_t|x_0, w)$ have been proposed, e.g., MC based approaches [31, 32, 33], methods based on approximate solutions to the FPK equation [58, 59], or deterministic approximations based on an assumed density [34]. Prior work in the context of NSDEs [54, 60] commonly approximates the transition kernel via MC methods. As we find out that sampling noise impairs the predictive calibration (see Fig. 3.1), we derive a novel deterministic approximation of the transition kernel, which efficiently exploits the layered structure of neural networks. To make our writing concise, we exclude the dependence on $w$ in the subsequent text unless it is necessary for defining the loss function. Instead, we use a shorter notation, denoted as $p(x_t|x_0, w) = p(x_t|x_0)$.

## 3.2. Bidimensional Moment Matching

As our approximation of the transition kernel performs moment matching in two directions: (i) time and (ii) layer depth, we term our method *Bidimensional Moment Matching* (BMM). We craft our solution in three steps: (i) approximating the process distribution at every discretization point as a Gaussian, (ii) analytically marginalizing out the transition noise from moment matching update rules, and (iii) approximating the intractable terms in the moment calculations.

Since the transition noise injects randomness to any arbitrarily small time interval, the solution of any DGTM with mean and covariance update networks with at least one hidden layer is analytically intractable. As noted in Sec. 2.2, we may express the $t$-step transition kernel for a given $x_0$ as a series of nested integrals $p(x_t|x_0) = \int p(x_t|x_{t-1})p(x_{t-1}|x_0)dx_{t-1}$. This expression needs to be solved recursively for a given $x_0$ with $p(x_1|x_0)$ given by Eq. 3.1. Prior work [61, 60] evaluates this intractable recurrence relation via MC integration. After sampling multiple trajectories, the dependence on the previous time step can be marginalized out, resembling the transition kernel approximation in the simulated maximum likelihood method [31, 32].

### 3.2.1. Assumed Process Density

As the solution to the nested integrals, which describes the $t$-step transition kernel of a DGTM (Eq. 2.4), is intractable for non-trivial architectures, we approximate the transition kernel at every step $t$ by a Gaussian $p(x_t|x_0) \approx \mathcal{N}(\mu_t, \Sigma_t)$, with mean $\mu_t \in \mathbb{R}^{D_x}$ and covariance $\Sigma_t \in \mathbb{R}^{D_x \times D_x}$. This approximation simplifies the problem to calculating the first two moments of the transition kernel. Plugging the *Assumed Density* (AD) into the recurrence relation for estimation of $p(x_t|x_0)$, as defined in Eq. 2.4, i.e., $p(x_t|x_0) = \int p(x_t|x_{t-1})p(x_{t-1}|x_0)dx_{t-1}$, amounts to approximating the transition kernel at every time point by matching moments progressively in the time direction. We refer to this chain of operations as *Horizontal Moment Matching* (HMM).

Calculating $\mu_t$ and $\Sigma_t$ does not appear to be a simpler problem at first sight than solving Eq. 2.4. However, it is possible to obtain a more pleasant expression by reparametrizing the transition kernel $p(x_t|x_0)$ as

$$\begin{aligned} &\zeta_{t-1} \sim N(\zeta_{t-1}|0, I), \quad x_{t-1} \sim \mathcal{N}(x_{t-1}|\mu_{t-1}, \Sigma_{t-1}), \\ &x_t = x_{t-1} + f(x_{t-1})\Delta t + \sqrt{\text{diag}(r(x_{t-1}))\Delta t}\zeta_{t-1}, \end{aligned} \tag{3.2}$$

where $I$ is the identity matrix with appropriate dimensionality and $\sqrt{\cdot}$ is the Cholesky decomposition. We arrive at the following view of the first moment of $p(x_t|x_0)$ using the law of the unconscious statistician

$$\begin{aligned} \mu_t &= \mathbb{E}[x_{t-1} + f(x_{t-1})\Delta t + \sqrt{\text{diag}(r(x_{t-1}))\Delta t}\zeta_{t-1}] \\ &= \mu_{t-1} + \mathbb{E}[f(x_{t-1})]\Delta t, \end{aligned} \tag{3.3}$$

where $\mathbb{E}[f(x_t)] \in \mathbb{R}^{D_x}$ is the expected value of $f(x_t)$. In order to derive a tractable expression for the variance $\Sigma_t = \mathbb{E}[x_t x_t^\top] - \mathbb{E}[x_t]\mathbb{E}[x_t]^\top$, we first evaluate the second central moment

$$\mathbb{E}[x_t x_t^\top] = \mathbb{E}\big[(x_{t-1}+f(x_{t-1})\Delta t)(x_{t-1}+f(x_{t-1})\Delta t)^\top\big] + \text{diag}\left(\mathbb{E}\big[r(x_{t-1})\big]\right)\Delta t, \tag{3.4}$$

since $\mathbb{E}[\zeta_{t-1}] = 0$ and $\mathbb{E}[\zeta_{t-1}\zeta_{t-1}^\top] = I$. Using the bilinearity of the covariance operator, we obtain the below solution to the second moment of $p(x_t|x_0)$

$$\begin{aligned} \Sigma_t = \Sigma_{t-1} &+ \text{Cov}[f(x_{t-1})]\Delta t^2 + \text{Cov}[f(x_{t-1}), x_{t-1}]\Delta t + \\ &\text{Cov}[f(x_{t-1}), x_{t-1}]^\top \Delta t + \text{diag}\left(\mathbb{E}[r(x_{t-1})]\right)\Delta t, \end{aligned} \tag{3.5}$$

where $\text{Cov}[f(x_t)] \in \mathbb{R}^{D_x \times D_x}$ is the covariance of $f(x_t)$, $\text{Cov}[x_t, f(x_t)] \in \mathbb{R}^{D_x \times D_x}$ is the cross-covariance between the arguments, and $\mathbb{E}[r(x_t)] \in \mathbb{R}^{D_x}$ is the expected value of

$r(x_t)$. Eq. 3.3 and 3.5 have no closed form solution for neural nets and require numerical approximation. Moment matching solutions along similar lines have been developed earlier for SDEs [34, 36], which rely on standard numerical integration schemes. In contrast, we develop in the following sections an integration scheme that efficiently uses the layered structure of neural nets, resulting in a more accurate and faster method.

### 3.2.2. Computing the Moments of the Update Functions

In the following discussion, we focus on approximating the output moments of the mean update $f(x_t)$. We do not delve into the details of the covariance update $r(x_t)$, as its moments can be approximated in the same manner. After applying the HMM scheme, the terms $\mathbb{E}[f(x_t)]$ and $\mathrm{Cov}[f(x_t)]$ amount to the first two moments of a random variable obtained by propagating $x_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ through the neural net

$$f(x_t) = u_L(u_{L-1}(\dots u_2(u_1(x_t))\dots)), \tag{3.6}$$

composed of a chain of $L$ simple functions (layers) $u_l : \mathbb{R}^{D_{x,l-1}} \to \mathbb{R}^{D_{x,l}}$, where $x_t^l \in \mathbb{R}^{D_{x,l}}$ are the features at layer $l$ at time step $t$. These functions are typically an alternation of affine transformations and nonlinear activations. Calculating the moments of $f(x_t)$ is analytically intractable due to the nonlinear activations. We approximate this computation by another round of moment matching, this time by propagating the input noise through the neural net. We define the input as $x_t^0 = x_t$ and the output as $x_t^L = f(x_t)$. To approximate the distributions at each layer, we recursively approximate them as Gaussians

$$p(u_l(x_t^{l-1})) = p(x_t^l) \approx \mathcal{N}(\mathbb{E}[x_t^l], \mathrm{Cov}[x_t^l]) \tag{3.7}$$

with mean $\mathbb{E}[x_t^l] \in \mathbb{R}^{D_{x,l}}$ and covariance $\mathrm{Cov}[x_t^l] \in \mathbb{R}^{D_{x,l} \times D_{x,l}}$. The moments at the last layer $L$ correspond to the expected value of $f(x_t)$, which is denoted as $\mathbb{E}[x_t^L] = \mathbb{E}[f(x_t)]$, and its covariance, denoted as $\mathrm{Cov}[x_t^L] = \mathrm{Cov}[f(x_t)]$. We refer to applying this approximation throughout all neural net layers as *Vertical Moment Matching* (VMM). We provide moments $\mathbb{E}[x_t^l]$ and $\mathrm{Cov}[x_t^l]$ for commonly used layers in Sec. 3.3.

A similar approach has been applied earlier to *Bayesian Neural Nets* (BNN) for random weights in various contexts such as expectation propagation [62, 63], deterministic variational inference [44], and evidential deep learning [47]. To our knowledge, no prior work has applied this approach to propagating input uncertainty through a deterministic network in the dynamics modeling context.

### 3.2.3. Computing the Cross-Covariance

The term $\mathrm{Cov}[x_t, f(x_t)]$ corresponds to the cross-covariance between the input $x_t$, which is a random variable, and its transformation with the mean update $f(x_t)$. Due to the same reasons as the mean and covariance of $f(x_t)$, this cross-covariance term cannot be analytically calculated except for trivial mean update functions. However, cross-covariance is not provided as a direct outcome of VMM. As being neither a symmetric nor a positive semi-definite matrix, an inaccurate approximation of cross-covariance may impair numerical stability. Applying Stein's lemma [64] for the first time in the context of matching moments of a neural net, we obtain a form that is easier to approximate

$$\mathrm{Cov}[x_t, f(x_t)] = \Sigma_t \mathbb{E}[\nabla_{x_t} f(x_t)]^\top, \tag{3.8}$$

where $\mathbb{E}[\nabla_{x_t} f(x_t)] \in \mathbb{R}^{D_x \times D_x}$ is the expected Jacobian. The covariance $\Sigma_t$ is provided from the previous time step, but the expected Jacobian needs to be calculated. Applying the chain rule, the expectation of the derivative of a neural net with respect to a random input reads

$$\mathbb{E}[\nabla_{x_t} f(x_t)] = \mathbb{E}[\nabla_{x_t^{L-1}} u_L(x_t^{L-1}) \dots \nabla_{x_t} u_1(x_t)], \tag{3.9}$$

which is also intractable. We facilitate computation by making the assumption that the mutual information between nonlinear feature maps of different layers is small

$$\int p(x_t^l, x_t^{l'}) \log \left\{ \frac{p(x_t^l, x_t^{l'})}{p(x_t^l) p(x_t^{l'})} \right\} dx_t^l dx_t^{l'} \approx 0, \tag{3.10}$$

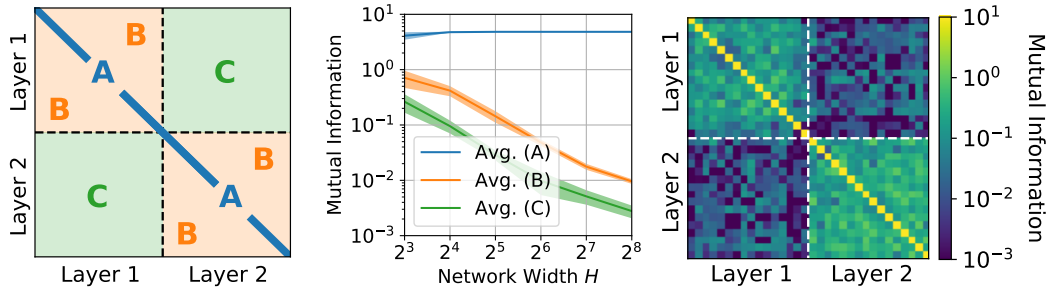
for all pairs $(l, l')$ with $l \neq l'$ and $1 \leq l, l', \leq L$. Applying this assumption of decoupled activations on Eq. 3.9, we get

$$\mathbb{E}[\nabla_{x_t} f(x_t)] \approx \prod_{l=L}^{1} \mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})], \tag{3.11}$$

where $\mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})] \in \mathbb{R}^{D_{x,l} \times D_{x,l-1}}$ is the expected Jacobian of $u_l(x_t^{l-1})$.

We test this assumption empirically by feeding a random input $x \sim N(0, I)$ into a neural net with two fully-connected and equally wide hidden layers of width $H$ with Dropout in between. We provide a detailed discussion on the Dropout layer in the context of DGTMs in Sec. 3.3.3. Using the non-parametric entropy estimation toolbox[1] [65], we can

[1] `https://github.com/gregversteeg/NPEET`




(a) Legend. (b) Result. (c) Example.

Figure 3.2.: Nonlinear activations get statistically independent as the network width increases, supporting our assumption in Eq. 3.11. Imagine a matrix containing mutual information between all pairs of nonlinear activations $x_t^l$ in a two-hidden-layer neural net. Group its entries into blocks as shown in panel (a): the diagonal (A) gives the entropy of an activation, the within-layer off-diagonal block (B) gives the dependence of sibling activations, the cross-layer off-diagonal (C) gives the dependence of activations in different layers. As seen in panel (b), the average mutual information in blocks (B) and (C) decreases sharply with increasing layer width $H$. Solid lines and shaded areas represent average mutual information and its standard deviation over 100 repetitions. We show an example of the matrix with all pairwise mutual information values for a hidden layer width of 16 neurons in panel (c).

efficiently estimate the mutual information between all pairs of hidden layer activations. As depicted in Fig. 3.2b, the mutual information between activations at different layers as well as within a layer shrinks fast when the hidden layers become wider. For a hidden layer width of 16, the mutual information between different layers is by a factor of $\sim$100 smaller than the average entropy in a layer. For a hidden layer width of practical use, such as 64 neurons, the mutual information between different layers is already by a factor of $\sim$1000 smaller than the average entropy in a layer.

Now the problem reduces to taking the expectations of the individual gradient terms. Despite being intractable, these expectations can be efficiently approximated by reusing the outcomes of the VMM step in Eq. 3.7 as follows

$$\mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})] \approx \int \nabla_{x_t^{l-1}} u_l(x_t^{l-1}) \mathcal{N}(\mathbb{E}[x_t^{l-1}], \mathrm{Cov}[x_t^{l-1}]) dx_t^{l-1}. \tag{3.12}$$

We attain a deterministic approximation to Stein's lemma by taking the covariance $\Sigma_t$ from VMM and the expected gradient from Eq. 3.12:

$$\mathrm{Cov}[x_t, f(x_t)] \approx \Sigma_t \left( \prod_{l=L}^{1} \mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})] \right)^\top . \tag{3.13}$$

We refer to applying this approximation throughout all neural net layers as *Backward Vertical Moment Matching* (BVMM). We provide the expected Jacobian $\mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})]$ for commonly used layers in Sec. 3.3.

### 3.2.4. The Bidimensional Moment Matching Algorithm

Given an observed initial value $x_0$, our deterministic method approximates the transition kernel $p(x_t|x_0)$ as a Gaussian $p(x_t|x_0) \approx \mathcal{N}(\mu_t, \Sigma_t)$. The moments are recursively calculated via HMM by applying the moment matching rules (Eq. 3.3 and 3.5) in time direction. We approximate $\mathbb{E}[f(x_t)]$, $\mathrm{Cov}[f(x_t)]$, and $\mathrm{diag}\,(\mathbb{E}[r(x_t)])$ via VMM as defined in Sec. 3.2.2. The cross-covariance $\mathrm{Cov}[x_t, f(x_t)]$ is obtained via Eq. 3.13. We refer to our method as *Bidimensional Moment Matching* (BMM) and provide its pseudocode in Alg. 1. The BMM algorithm uses the *Vertical Moment Matching* (VMM) algorithm as a subroutine for which we provide pseudocode in Alg. 2

**Algorithm 1** Bidimensional Moment Matching (`BMM`)

**Inputs:** $f(x_t)$ ▷ Mean update
$r(x_t)$ ▷ Covariance update
$x_0$ ▷ Initial value
$t$ ▷ Horizon
**Outputs:** Approximate transition kernel $p(x_t|x_0)$
$\mu_0, \Sigma_0 \leftarrow x_0, I\epsilon$ ▷ Initialize, with $\epsilon \in \mathbb{R}_+$ being a small number
**for** time step $t' \in \{0, \cdots, t-1\}$ **do** ▷ Horizontal Moment Matching
  $\mathbb{E}[f(x_{t'})], \mathrm{Cov}[f(x_{t'})], \mathrm{Cov}[x_{t'}, f(x_{t'})] \leftarrow \texttt{VMM}(f, \mu_{t'}, \Sigma_{t'})$ ▷ VMM for mean update
  $\mathbb{E}[r(x_{t'})], \_, \_ \leftarrow \texttt{VMM}(r, \mu_{t'}, \Sigma_{t'})$ ▷ VMM for covariance update
  $\mu_{t'+1} \leftarrow \mu_{t'} + \mathbb{E}[f(x_{t'})]\Delta t$ ▷ Mean at $t'+1$ (Eq. 3.3)
  $\Sigma_{t'+1} \leftarrow \Sigma_{t'} + \mathrm{Cov}[f(x_{t'})]\Delta t^2 + \mathrm{Cov}[x_{t'}, f(x_{t'})]\Delta t + \mathrm{Cov}[f(x_{t'}), x_{t'}]\Delta t + \mathrm{diag}\,(\mathbb{E}[r(x_{t'})])\,\Delta t$ ▷ Covariance at $t'+1$ (Eq. 3.5)
  $p(x_{t'+1}|x_0) \leftarrow \mathcal{N}(\mu_{t'+1}, \Sigma_{t'+1})$ ▷ Transition kernel at $t'+1$
**end for**
**return** $p(x_t|x_0)$

**Algorithm 2** Vertical Moment Matching (VMM)

**Inputs:** $f(x_t)$ ▷ Neural net
$\mu_t$ ▷ Mean
$\Sigma_t$ ▷ Covariance
**Outputs:** Approximate moments $\mathbb{E}[f(x_t)], \mathrm{Cov}[f(x_t)]$ and cross-covariance $\mathrm{Cov}[x_t, f(x_t)]$
$\mathbb{E}[x_t^0], \mathrm{Cov}[x_t^0] \leftarrow \mu_t, \Sigma_t$ ▷ Input distribution
$p(x_t^0) \leftarrow \mathcal{N}(\mathbb{E}[x_t^0], \mathrm{Cov}[x_t^0])$
**for** layer index $l \in \{1, \cdots, L\}$ **do**
*See Sec. 3.3 for expectation, covariance, and Jacobian of different layers*
$\mathbb{E}[x_t^l] \leftarrow \mathbb{E}[u_l(x_t^{l-1})]$ ▷ Expectation at layer $l$ (Eq. 3.7)
$\mathrm{Cov}[x_t^l] \leftarrow \mathrm{Cov}[u_l(x_t^{l-1})]$ ▷ Covariance at layer $l$ (Eq. 3.7)
$J_t^l \leftarrow \mathbb{E}[\nabla_{x_t^{l-1}} u_l(x_t^{l-1})]$ ▷ Jacobian at layer $l$ (Eq. 3.12)
$p(x_t^l) \leftarrow \mathcal{N}(\mathbb{E}[x_t^l], \mathrm{Cov}[x_t^l])$ ▷ Distribution at layer $l$ (Eq. 3.7)
**end for**
$\mathbb{E}[f(x_t)] \leftarrow \mathbb{E}[x_t^L]$ ▷ Mean of $f(x_t)$
$\mathrm{Cov}[f(x_t)] \leftarrow \mathrm{Cov}[x_t^L]$ ▷ Covariance of $f(x_t)$
$\mathrm{Cov}[x_t, f(x_t)] \leftarrow \Sigma_t \left(\prod_{l=L}^{1} J_t^l\right)^\top$ ▷ Cross-covariance (Eq. 3.13)
**return** $\mathbb{E}[f(x_t)], \mathrm{Cov}[f(x_t)], \mathrm{Cov}[x_t, f(x_t)]$

## 3.3. Moments of Layers and their Derivatives

Given the VMM output of the previous layer $x_t^l$ that we approximate as a Gaussian $p(x_t^l) \approx \mathcal{N}(\mathbb{E}[x_t^l], \mathrm{Cov}[x_t^l])$, we show below how the output moments and the expected Jacobian can be calculated for three common layer types: (i) linear layer, (ii) ReLU activation, and (iii) Dropout. Additionally, we provide a discussion about the implications of the Dropout layer and how it can be used to make BMM a tighter approximation.

### 3.3.1. Linear Layer

A linear layer applies an affine transformation

$$x_t^{l+1} = A^{l+1} x_t^l + b^{l+1}, \tag{3.14}$$

where $A^{l+1} \in \mathbb{R}^{D_{x,l+1} \times D_{x,l}}$ and $b^{l+1} \in \mathbb{R}^{D_{x,l+1}}$ correspond to the transformation matrix and bias at layer $l+1$. The output moments are analytically tractable

$$\mathbb{E}[x_t^{l+1}] = A^{l+1} \mathbb{E}[x_t^l] + b^{l+1}, \tag{3.15}$$

$$\mathrm{Cov}[x_t^{l+1}] = A^{l+1} \mathrm{Cov}[x_t^l] (A^{l+1})^\top. \tag{3.16}$$

The expected Jacobian is a constant

$$\mathbb{E}[\nabla_{x_t^l} u_{l+1}(x_t^l)] = A^{l+1}. \tag{3.17}$$

### 3.3.2. ReLU Activation

The output moments of nonlinear activations are analytically not tractable. However, for many types of nonlinearities in widespread use exist tight approximations. For instance, the moments of the ReLU function

$$x_t^{l+1} = \max(0, x_t^l), \tag{3.18}$$

can be approximated as depicted below [44]

$$\mathbb{E}[x_t^{l+1}] \approx \sqrt{\text{diag}(\text{Cov}[x_t^l])}\text{SR}\left(\mathbb{E}[x_t^l]/\sqrt{\text{diag}(\text{Cov}[x_t^l])}\right), \tag{3.19}$$

$$\text{Cov}[x_t^{l+1}] \approx \sqrt{\text{diag}(\text{Cov}[x_t^l])\text{diag}(\text{Cov}[x_t^l])^\top}F(\mathbb{E}[x_t^l], \text{Cov}[x_t^l]), \tag{3.20}$$

where $\text{SR} : \mathbb{R} \to \mathbb{R}_+$ is the elementwise *Soft ReLU* (SR) function, defined as $\text{SR}(x) = \phi(x) + x\Phi(x)$. We denote $\phi : \mathbb{R} \to \mathbb{R}_+$ and $\Phi : \mathbb{R} \to \mathbb{R}_+$ as the *Probability Density Function* (PDF) and *Cumulative Distribution Function* (CDF) of a standard Gaussian variable. The function $F : \mathbb{R}^{D_{x,l}} \times \mathbb{R}^{D_{x,l} \times D_{x,l}} \to \mathbb{R}^{D_{x,l} \times D_{x,l}}$ is defined as

$$F(\mathbb{E}[x_t^l], \text{Cov}[x_t^l]) = \left(A(\mathbb{E}[x_t^l], \text{Cov}[x_t^l]) + \exp -Q(\mathbb{E}[x_t^l], \text{Cov}[x_t^l])\right). \tag{3.21}$$

In order to keep the thesis self-contained, we detail the functions $A : \mathbb{R}^{D_{x,l}} \times \mathbb{R}^{D_{x,l} \times D_{x,l}} \to \mathbb{R}^{D_{x,l} \times D_{x,l}}$ and $Q : \mathbb{R}^{D_{x,l}} \times \mathbb{R}^{D_{x,l} \times D_{x,l}} \to \mathbb{R}^{D_{x,l} \times D_{x,l}}$ below and refer to [44] for the derivation. After introducing the dimensionless vector $\epsilon_t^l = \mathbb{E}[x_t^l]/\sqrt{\text{diag}(\text{Cov}[x_t^l])}$, the function $A(\mathbb{E}[x_t^l], \text{Cov}[x_t^l])$ is estimated as

$$A(\mathbb{E}[x_t^l], \text{Cov}[x_t^l]) = \text{SR}(\epsilon_t^l)\text{SR}(\epsilon_t^l)^\top + \rho_t^l(\text{Cov}[x_t^l])\Phi(\epsilon_t^l)\Phi(\epsilon_t^l)^\top, \tag{3.22}$$

with $\rho_t^l : \mathbb{R}^{D_{x,l} \times D_{x,l}} \to \mathbb{R}^{D_{x,l} \times D_{x,l}}$ defined as

$$\rho_t^l(\text{Cov}[x_t^l]) = \text{Cov}[x_t^l]/\left(\sqrt{\text{diag}(\text{Cov}[x_t^l])}\sqrt{\text{diag}(\text{Cov}[x_t^l])^\top}\right). \tag{3.23}$$

The $i, j$-th element of $Q(\mathbb{E}[x_t^l], \text{Cov}[x_t^l])$ can be estimated as:

$$\begin{aligned} Q(\mathbb{E}[x_t^l], \text{Cov}[x_t^l])_{i,j} = &\frac{\rho_{t,i,j}^l}{2g_{t,i,j}^l(1+\bar{\rho}_{t,i,j})}\left((\epsilon_{t,i}^l)^2 + (\epsilon_{t,j}^l)^2\right) - \\ &\frac{\arcsin(\rho_{t,i,j}^l) - \rho_{t,i,j}^l}{\rho_{t,i,j}g_{t,i,j}^l}\epsilon_{t,i}^l\epsilon_{t,j}^l - \log\left(\frac{g_{t,i,j}^l}{2\pi}\right), \end{aligned} \tag{3.24}$$

with the shorthand notation $g_t^l = \arcsin(\rho_t^l) + \rho_t^l \oslash (1 + \bar{\rho}_t^l)$, and $\bar{\rho}_t^l = \sqrt{1 - \rho_t^l \odot \rho_t^l}$. We denote with $\oslash, \odot$ elementwise division, multiplication.

Since activation functions are applied elementwise, off-diagonal entries of the expected gradient are zero. The diagonal of the Jacobian of the ReLU function is the expected Heaviside step function [44]

$$\text{diag}\left(\mathbb{E}\left[\nabla_{x_t^l} u_{l+1}(x_t^l)\right]\right) \approx \Phi\left(\mathbb{E}[x_t^l] / \sqrt{\text{diag}(\text{Cov}[x_t^l])}\right). \tag{3.25}$$

### 3.3.3. Dropout

Dropout is defined as the elementwise mapping

$$x_t^{l+1} = x_t^l \odot \beta_t^l / q, \tag{3.26}$$

where $\beta_t^l \in \mathbb{R}^{D_{x,l}}$ is a random variable consisting of Bernoulli$(q)$ distributed entries. The moments are available as

$$\mathbb{E}[x_t^{l+1}] = \mathbb{E}[x_t^l], \tag{3.27}$$

$$\text{Cov}[x_t^{l+1}] = \text{Cov}[x_t^l] + \text{diag}\left(\frac{1-q}{q}\left(\text{Cov}[x_t^l] + (\mathbb{E}[x_t^l])(\mathbb{E}[x_t^l])^\top\right)\right). \tag{3.28}$$

We obtain the first moment by using the independence between $x_t^l$ and $\beta_t^l$

$$\mathbb{E}[u_{l+1}(x_t^l)] = \mathbb{E}[x_t^l \odot \beta_t^l / q] = \mathbb{E}[x_t^l]\mathbb{E}[\beta_t^l]/q = \mathbb{E}[x_t^l]. \tag{3.29}$$

We derive diagonal and off-diagonal entries in $\text{Cov}[u_{l+1}(x_t^l)]$ separately. We obtain for $i = j$

$$\begin{aligned}
\text{Cov}[u_{l+1}(x_t^l)]_{i,i} &= \mathbb{E}\left[(u_{l+1}(x_t^l)_i)^2\right] - \mathbb{E}[u_{l+1}(x_t^l)_i]^2 \\
&= \frac{1}{q^2}\mathbb{E}\left[(x_{t,i}^l)^2\right]\mathbb{E}\left[(\beta_{t,i}^l)^2\right] - \mathbb{E}[x_{t,i}^l]^2 \\
&= \mathbb{E}\left[(x_{t,i}^l)^2\right] - \mathbb{E}[x_{t,i}^l]^2 + \frac{1-q}{q}\mathbb{E}\left[(x_{t,i}^l)^2\right] \\
&= \text{Cov}[x_t^l]_{i,i} + \frac{1-q}{q}\left(\text{Cov}[x_t^l] + (\mathbb{E}[x_t^l])(\mathbb{E}[x_t^l])^\top\right)_{i,i}
\end{aligned} \tag{3.30}$$

and for $i \neq j$

$$\begin{aligned}
\mathrm{Cov}[u_{l+1}(x_t^l)]_{i,j} &= \mathbb{E}\left[u_{l+1}(x_t^l)_i u_{l+1}(x_t^l)_j\right] - \mathbb{E}[u_{l+1}(x_t^l)_i]\mathbb{E}[u_{l+1}(x_t^l)_j] \\
&= \frac{1}{q^2}\mathbb{E}\left[x_{t,i}^l x_{t,j}^l \beta_{t,i}^l \beta_{t,j}^l\right] - \mathbb{E}[x_{t,i}^l]\mathbb{E}[x_{t,j}^l] \\
&= \mathbb{E}\left[x_{t,i}^l x_{t,j}^l\right] - \mathbb{E}[x_{t,i}^l]\mathbb{E}[x_{t,j}^l] \\
&= \mathrm{Cov}[x_t^l]_{i,j}. \qquad (3.31)
\end{aligned}$$

The Jacobian of the Dropout layer is the identity matrix

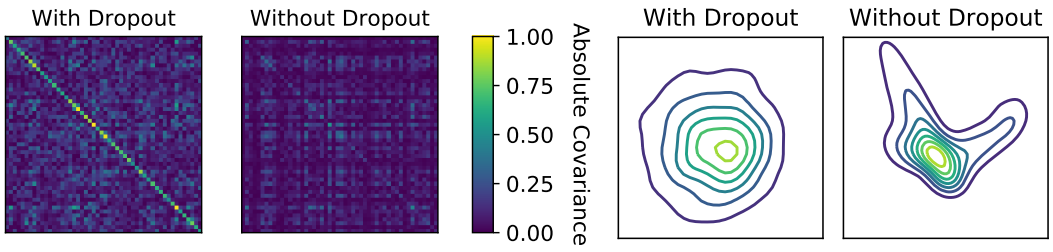
$$\mathbb{E}\left[\nabla_{x_t^l} u_{l+1}(x_t^l)\right] = \mathbb{E}\left[I\beta_t^l/q\right] = I. \qquad (3.32)$$

#### Dropout tightens the Gaussian assumption

As shown in the above equations, Dropout increases the value of the diagonal entries in the covariance matrix relative to its off-diagonal entries. Consequently, Dropout is helpful to reduce the correlation coefficient between different activations in the same layer, making their sum approach the normal distribution due to the *Central Limit Theorem* (CLT).

As Dropout is added, we see in Fig. 3.3a the covariance matrix at a hidden layer to be dominated by its diagonal values, which results in an approximately Gaussian output as shown in Fig. 3.3b. Though Dropout does not strictly guarantee total decorrelation, i.e., negligible off-diagonal covariances compared to diagonal ones, we observe in our experiments this assumption to be a good approximation for neural networks layer widths of practical relevance.

In Sec. 3.2.3, we discussed how an increasing layer width results in a decrease in the mutual information between activations within the same layer, as well as between different layers. This observation is presented in Fig. 3.2b, where we visualize the mutual information between different neurons for varying layer widths. For a layer width of 16, the mutual information within a layer is approximately 10 times smaller than the average entropy in that layer. Furthermore, for neural networks with layers containing more than 64 neurons, the mutual information is approximately 100 times smaller. As shown in Fig. 3.2c the mutual information is dominated by its diagonal values for a layer width of 16.




(a) Intermediate activation.

(b) Output distribution.

Figure 3.3.: Dropout reduces the correlation coefficient between different activations. We pass a multivariate normal distributed random vector through a neural net with three 50-neuron-wide hidden layers with ReLU activation. The off-diagonals of the covariance matrix of the activation map are suppressed when Dropout is used after each ReLU activation, as shown in panel (a) for an intermediate layer and panel (b) for the output layer. Decorrelation of a large number of co-variates makes a normal distribution an accurate approximation of their sum due to the central limit theorem.

## 3.4. Numerical Properties

We investigate the numerical properties of the BMM algorithm in terms of integration error, approximation error of the expected Jacobian, computation cost, and generalization to multiple modes. We compare cubature as an alternative choice to VMM, which is a standard numerical method for approximating the expectation of a smooth function with respect to a normal distribution. Our empirical findings demonstrate that VMM has favorable computational properties over cubature.

### 3.4.1. Cubature

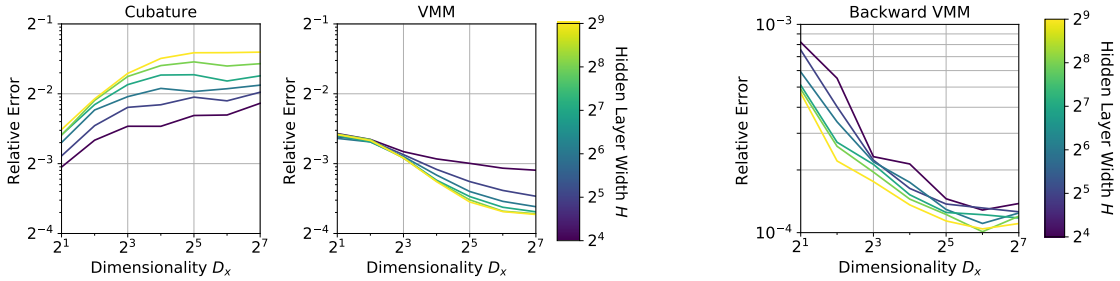
Cubature estimates the expected value of a nonlinear function $f(x)$ with respect to a Gaussian $x \sim \mathcal{N}(\mu, \Sigma)$ as a weighted sum of point mass evaluations

$$\int f(x)\mathcal{N}(\mu, \Sigma)dx \approx \sum_{i=1}^{U} \omega_i f(\mu + \sqrt{\Sigma}\xi_i). \tag{3.33}$$

The coefficients $\omega_i \in \mathbb{R}$ and $\xi_i \in \mathbb{R}^{D_x}$ are predetermined by a heuristic that aims to spread the particles in a maximally information-preserving way. There exist multiple heuristics for choosing $\omega_i$ and $\xi_i$. In this thesis, we use the commonplace heuristic *Unscented Transform* (UT) [66], which evaluates the above expression as a sum of $U = 2D_x + 1$ elements for a $D_x-$dimensional input space.

### 3.4.2. Approximation Error.

In Fig. 3.4a, we compare VMM and cubature in approximating $\mathbb{E}[f(x)]$ for a normally distributed input $x$ and the effect of neural net width and input/output dimensionality on approximation accuracy for a randomly initialized neural net $f(x)$.




(a) Integration error comparison.

(b) Error of expected Jacobian.

Figure 3.4.: Comparison of VMM versus cubature in terms of relative error. We generate a randomly initialized neural net $f(x)$. The dimensionalities of $x$ and $f(x)$ are equal and vary in the horizontal axis. The neural net $f(x)$ has three fully-connected layers of varying widths color-coded according to the heatmap, ReLU activation, and Dropout with rate 0.2. As cubature cannot handle a random $f(x)$ and discontinuous activations, we evaluate it with tanh activation and without Dropout. We aim to approximate the intractable expectation $e = \int f(x)\mathcal{N}(x|\mu, \Sigma)dx$, where $\mu \sim \mathcal{N}(0, I)$ and $\Sigma \sim \mathcal{W}(I, \dim(I))$ with Wishart distribution $\mathcal{W}$. We repeat the experiment 512 times and report the average relative error $||e - \hat{e}||^2/||e||^2$ in panel (a), where $e$ is represented by averaging over $10$ million MC simulations and $\hat{e}$ is approximated via cubature and VMM, respectively. In panel (b), we report the average relative error of the true Jacobian $\mathbb{E}[\nabla_x f(x)]$, which is obtained by the Monte Carlo simulation, and our approximate Jacobian obtained by backward vertical moment matching.

For low dimensionalities, the relative error of VMM is approximately equal to cubature. The approximation error of VMM shrinks with increasing hidden layer width and dimensionality. This is expected since summing a larger number of decorrelated variables makes the assumption of normally distributed intermediate activations more accurate due to the central limit theorem [67]. Similarly, we observe the approximation error of the expected Jacobian by Backward VMM to decrease with increasing input dimensionality and hidden layer width as shown in Fig. 3.4b.

### 3.4.3. Computational Cost.

It requires $\mathcal{O}(STHD_x + STH^2)$ computations to propagate $S$ particles with dimensionality $D_x$ along $T$ time steps when dynamics are governed by a DGTM with hidden layer width $H$. The first term is due to the computational cost of the $H \times D_x$-dimensional affine transformation in the first layer. The second term $\mathcal{O}(STH^2)$ is due to the computational cost of the $H \times H$-dimensional affine transformations in the subsequent hidden layers, which require $\mathcal{O}(H^2)$ computations. Our method BMM approximates the $S \to \infty$ limit, while requiring only $\mathcal{O}(THD_x^2 + TH^2D_x + TH^3)$ computations. The first two terms are due to the computational cost of the $H \times D_x$-dimensional affine transformation in the first layer. The third term is due to the computational cost of the $H \times H$-dimensional affine transformations in the subsequent hidden layers. Replacing VMM with cubature in our framework results in an algorithm requiring $\mathcal{O}(THD_x^2 + TH^2D_x + TD_x^3)$ computations. Cubature requires at least $\mathcal{O}(D_x)$ DGTM evaluations, which causes the additional factor $D_x$ in the first two terms. The third term arises from the Cholesky decomposition of the input, as shown in Eq. 3.33 during point selection. We visualize in Fig. 3.5 the wall clock time of VMM and cubature as a function of dimensionality and hidden layer width. Dimensionality sets a bottleneck for cubature while it barely affects VMM. Contrarily, VMM gets significantly slower as the hidden layer width increases while the computational cost of cubature remains similar. VMM is adaptable to setups requiring high learning capacity by building narrow and deep architectures. However, dimensionality is an external factor that limits the applicability of cubature.

As shown in Fig. 3.5, VMM has a runtime of approximately 10 ms for an input dimensionality and hidden layer width up to 128 for architectures with three hidden layers, which is sufficient for most applications. In case larger neural network architectures are required, the runtime can be reduced by sparse covariance approximations, which we discuss in Chap. 4.

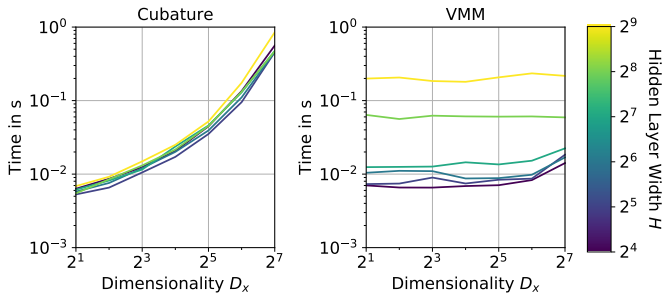


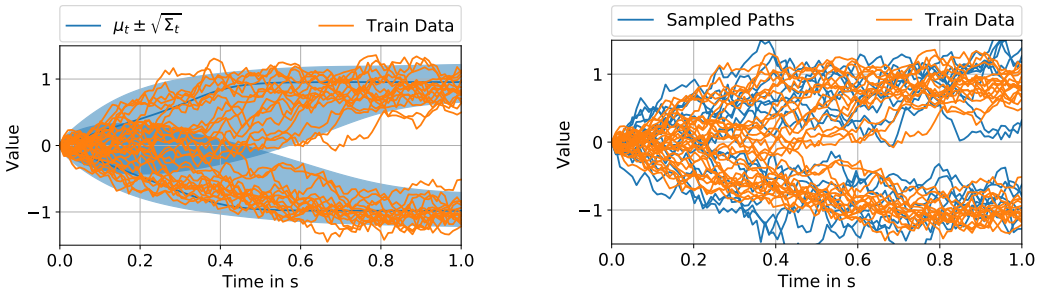

Figure 3.5.: We measure the computation time of the numerical experiment in Sec. 3.4.2 and report its average as a function of input dimensionality.

### 3.4.4. Multimodal Processes.

Deterministic prediction of unimodal densities with DGTMs is an unsolved problem of its own. We restrict our focus in this chapter to unimodal solutions as the first inevitable step towards multimodal solutions. That being said, our method can generalize to multiple modes under mild assumptions. For instance, if training sequences come with the ground-truth knowledge of the modality they belong to, a separate unimodal DGTM can be fit to each mode, and their mixture can be used during prediction. If modality assignments are not known a priori, an initial clustering step can be applied. We visualize prediction results in Fig. 3.6 on the bimodal double-well dynamics after clustering the training data and training a separate DGTM on each mode. Beyond that, we introduce in Chap. 4 another method to approximate multimodal distributions. In short, we model the predictive distribution as a *Gaussian Mixture Model* (GMM), and the distribution of the GMM at the initial time step is modeled by an auxiliary neural net. We then apply the BMM algorithm to each component separately.

## 3.5. Experiments

We demonstrate in four different experiments that DGTMs can be used for a broad range of scenarios. In Sec. 3.5.2, we use DGTMs as a stochastic recurrent layer for neural networks. We demonstrate the performance of DGTMs as a stochastic recurrent layer on eight different UCI datasets. Then we benchmark DGTMs on time series classification tasks on the MNIST and IMDB datasets in Sec. 3.5.3. In Sec. 3.5.4, we use DGTMs




(a) Predictions.

(b) Sampled paths.

Figure 3.6.: Multimodal predictions with a DGTM applied to double-well potential dynamics: $x_{t+1} = x_t + 4x_t(1 - x_t^2)\Delta t + \sqrt{\Delta}\zeta_t$. We train a separate model on each mode and predict with BMM in panel (a) and with DGTM-MC in panel (b).

for dynamical system modeling. We benchmark on three different datasets that enable a comparison to the state-of-the-art methods for learning-based modeling of dynamics. Lastly, we compare in Sec. 3.5.5 our method against Monte Carlo sampling for modeling high dimensional dynamics. We provide details regarding the evaluation in Sec. 3.5.1.

### 3.5.1. Evaluation Criteria

The prediction quality is assessed in terms of *Mean Squared Error* (MSE), *Negative Log-Likelihood* (NLL), *Expectation of Coverage Probability Error* (ECPE), as well as the *Expected Calibration Error* (ECE). Both ECPE and ECE measure the calibration of the uncertainty estimates of our model, i.e., how well the predictive distribution covers the true data distribution. The ECPE is a suitable metric when the target variable is continuous, while the ECE is suitable when the target is discrete, i.e., classification tasks. ECPE measures the absolute difference between true confidence and the empirical coverage probability as [68]

$$ECPE = \frac{1}{J}\sum_{j=1}^{J} |\hat{p}_j - p_j|, \tag{3.34}$$

where $p_j$ and $\hat{p}_j$ is the true frequency and empirical frequency, respectively. Loosely speaking, ECPE is small if $p$ percent of the data lies in the predicted $p$-percent confidence interval. We choose $J = 10$ equally spaced confidence levels between 0 and 1. By taking

the average over all test samples $x_i \in \mathcal{D}_{test}$, which lie in a predicted confidence interval, the empirical frequency is estimated as

$$\hat{p}_j = \frac{\sum_i^{|\mathcal{D}_{test}|} \mathbb{I}\{x_i \leq \hat{F}_i^{-1}(p_j)\}}{|\mathcal{D}_{test}|}. \tag{3.35}$$

In contrast to [68], we consider in our work the case of multivariate predictors, which complicates the estimation of the predicted inverse cumulative distribution function $\hat{F}_i^{-1}(p)$. If $x_i$ is normally distributed, we can analytically estimate $\hat{F}_i^{-1}(p)$ as a function of model outputs $\mu_i$ and $\Sigma_i$. As discussed by [69], we may define the cumulative distribution function as the probability that a sample lies inside the ellipsoid determined by its Mahalanobis distance. The ellipsoidal region is analytically obtained as

$$(x_i - \mu_i)^T \Sigma_i^{-1} (x_i - \mu_i) \leq \chi^2_{D_x}(p) = \hat{F}_i^{-1}(p), \tag{3.36}$$

where $\chi^2_{D_x}$ is the chi-squared distribution with $D_x$ degrees of freedom.

The ECE is commonly used for assessing the calibration quality for classification tasks. It measures the difference between the fraction of predictions that are correct (accuracy) and the mean of the probabilities (confidence) for a probability interval, called a bin. We use $J = 10$ equally spaced bins. As an example, bin number six contains all predictions within the probability range $[0.5, 0.6)$. We use the calculation procedure as proposed in [70]

$$ECE = \sum_{j=1}^{J} \frac{|B_j|}{N} |\text{acc}(B_j) - \text{conf}(B_j)|, \tag{3.37}$$

where $N$ is the number of samples, $B_j$ is the set of predictions whose prediction confidence falls into the $j$-th bin, $\text{acc}(B_j)$ is the average accuracy in the $j$-th bin, and $\text{conf}(B_j)$ is the average confidence in the $j$-th bin.

### 3.5.2. Stochastic Recurrent Layers

DGTMs can be used as a general-purpose layer for neural networks, just as neural ODEs [53]. Similarly, [71] proposed stochastic dynamical models as a layer and optimized the evidence lower bound by taking samples. This idea was further expanded by [54] to encompass continuous stochastic dynamical systems. In contrast to prior work, we overcome the need to take samples and propose an alternative training objective based on maximum likelihood estimation. We demonstrate its application on a regression task.

Given an input $x \in \mathbb{R}^{D_x}$, we aim to predict the target value $y \in \mathbb{R}$. We propagate the input $x = x_0$ through the DGTM for a varying flow time $t$ and obtain as a result $x_t$, from which we predict $y$. The mapping from $x_0$ to $x_t$ can be interpreted as a stochastic recurrent layer. We formulate the probability of observing the target value conditioned on the input as

$$p(y|x_0, w_f, w_r, w_y) = \int p(y|x_t, w_y) p(x_t|x_0, w_f, w_r) dx_t, \tag{3.38}$$

where $p(x_t|x_0, w_f, w_r)$ is determined by the underlying DGTM with parameters $w_f$ and $w_r$. We approximate the solution to the above integral with our proposed BMM scheme. We further assume that the mapping from $x_t$ to $y$ is linear with parameters $w_y$. A linear mapping allows us to analytically marginalize out $x_t$ from the above expression. We learn the parameters $w_f, w_r, w_y$ by *Maximum Likelihood Estimation* (MLE)

$$\underset{w_f, w_r, w_y}{\text{argmax}} \; \mathbb{E}\left[p(y|x_0, w_f, w_r, w_y)\right]. \tag{3.39}$$

We vary the time $t$ between $2$ and $8$ seconds and choose $\Delta t = 0.5$ seconds.

### Datasets

We use eight UCI datasets[1], which have varying input dimensionality $D_x$ and size $N$. These datasets are commonplace used for benchmarking stochastic models such as neural networks [72, 62] as well as Gaussian Processes [56, 73, 74]. We use the experimental setup as defined in [62]. We use 20 random splits, where $90\%$ of the data are used for training, and $10\%$ are used for testing. We use for all datasets a batch size of 32.

### Baselines

We benchmark our method (BMM) against different DGTM variants as well as against commonly used neural regression models. Our DGTM variants have $103D_x + 51$ parameters. We use a neural mean update function with one hidden layer of size 40 with $81D_x + 40$ parameters and a neural covariance update function with one hidden layer of size 10 with $21D_x + 10$ parameters. After propagating an input for a defined time horizon, we use an affine transformation with $D_x + 1$ parameters in order to map the DGTM prediction to the regression target. The dropout rate has been adjusted for each dataset separately.

[1] `https://archive.ics.uci.edu/datasets`

*(i) DGTM-MC* [71]: Monte Carlo sampling-based prediction with DGTM. We use for DGTM-MC $2(2D_x + 1)$ particles, as it equals the number of function evaluations for DGTM-Cubature.

*(ii) Cubature*: A variant of ours using cubature in place of VMM to approximate $\mu_{t+1}$ in Eq. 3.3 and covariance $\Sigma_{t+1}$ in Eq. 3.5.

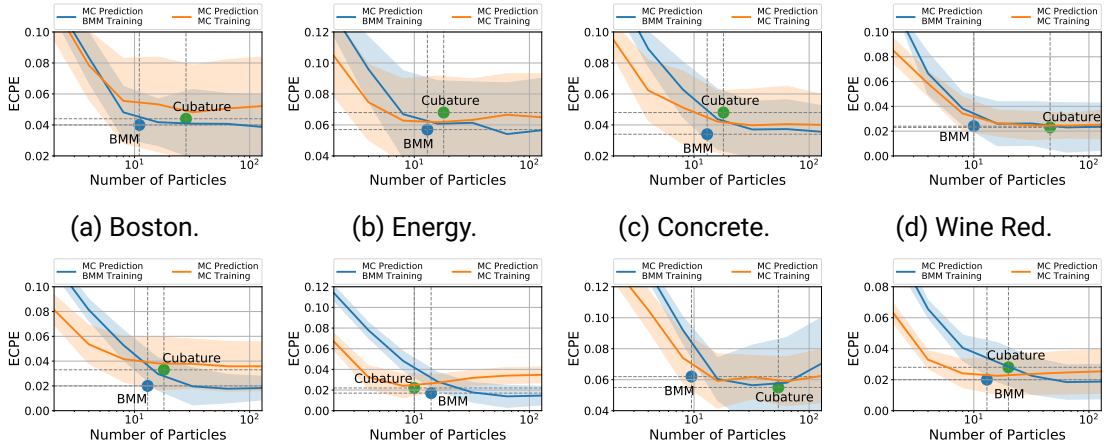
*(iii) DVI* [44]: DVI is an inference technique for Bayesian neural nets. In this model class, one arrives at a probabilistic model by allowing for uncertainty over the weights. The evidence lower bound is approximated in a deterministic manner by applying moment matching rules. DVI has $100D_x + 202$ parameters. This makes the DGTM variants slightly more parameter efficient as $D_x \leq 50$ in all experiments. DVI has the computational cost of $\mathcal{O}(H^3)$, where $H$ is the hidden layer width. For comparison, our method has the computational cost of $\mathcal{O}(TH^3)$ as described in Sec. 3.4.3.

*(iv) Dropout* [72]: This method introduces stochasticity by applying a Bernoulli distributed masking scheme in the affine layers. This method has the least parameters: $50D_x + 101$. Dropout has the computational cost of $\mathcal{O}(SH^2)$, where $S$ corresponds to the number of MC evaluations.

Dropout and DVI are strongly linked, as Dropout can be interpreted as the sampling-based version of DVI [75]. In consequence, given a limited computational budget, Dropout can lead to high gradient variance during training, which can result in deteriorated solutions. Both DVI and Dropout assume a probabilistic model by injecting noise over the neural net weights, which corresponds to the noise due to the lack of knowledge. Our proposed model captures data uncertainty, which corresponds to irreducible uncertainty of the data-generating process, and model uncertainty. Data uncertainty is modeled via the covariance update function and model uncertainty via Dropout. As our model captures both noise sources in contrast to Dropout and DVI, we expect our model to quantify uncertainty more accurately in terms of lower NLL. Looking ahead, in Chap. 5, we will explore more flexible approaches to incorporate model uncertainty beyond fixed Dropout rates. Specifically, we will introduce learnable weight uncertainty as a way to account for model uncertainty. Furthermore, when our method is used as a layer for regression tasks, we can adjust the model capacity by increasing the flow time while keeping the number of weights constant. Contrarily, the model capacity of DVI, as well as of Dropout, is fixed by the architecture. In consequence, DVI and Dropout have the same disadvantages compared to our method from a modeling perspective, i.e., they are less parameter efficient, do not model input noise, and cannot be used out-of-the-box for dynamical stochastic systems.

### Results

We report NLL in Tab. 3.1 and RMSE in Tab. 3.2. Despite the fact that DGTMs are dynamical systems, they can achieve competitive results as a layer. DGTM-BMM shows decreasing RMSE and NLL with increasing flow time since the expressiveness increases. DGTM-BMM has lower NLL than DVI in five datasets and the same NLL in one dataset. We account the increased performance of our model to its higher parameter efficiency and capability of modeling both noise sources, i.e., parameter and input noise. Increasing the integration time leads in some datasets to degraded predictive performance for the MC and cubature DGTM variants. We attribute the improved prediction accuracy of BMM over cubature to its reduced approximation error, as demonstrated in Fig. 3.4a. We observe in Fig. 3.7 the ECPE of our method BMM to decrease or to be on par with DGTM-MC. When using our method only for training and instead performing testing by drawing samples, we observe the same calibration levels as directly testing with our method. This indicates a tight approximation of BMM since the same uncertainty calibration is reached as the infinite particle limit at a lower computational cost.




(a) Boston. (b) Energy. (c) Concrete. (d) Wine Red.

(e) Kin8nm. (f) Power. (g) Naval. (h) Protein.

Figure 3.7.: Cost-benefit analysis of calibration on the regression task for a flow time of 8 seconds. One particle is equal to one MC simulation along a trajectory. BMM is our proposed method, Cubature is our method that uses cubature for VMM, the orange line is training and prediction with MC sampling, and the blue line is training with our method and prediction with MC sampling. We show mean and standard deviation over 10 runs.

Table 3.1.: Negative log-likelihood values of 8 benchmark datasets. We report average and standard error over 20 runs.

| | Boston | Energy | Concrete | Wine Red | Kin8nm | Power | Naval | Protein |
|---|---|---|---|---|---|---|---|---|
| Dropout [72] | 2.46(0.06) | 1.99(0.02) | 3.04(0.02) | 0.93(0.01) | -0.95(0.01) | **2.80(0.01)** | -3.80(0.01) | 2.89(0.00) |
| DVI [44] | 2.41(0.02) | 1.01(0.06) | 3.06(0.01) | **0.90(0.01)** | -1.13(0.00) | **2.80(0.00)** | **-6.29(0.04)** | 2.85(0.00) |
| DGTM-MC [71] | | | | | | | | |
| Integration Time: 2s | 2.62(0.04) | 1.63(0.05) | 3.19(0.03) | 1.01(0.01) | -1.15(0.01) | 2.97(0.01) | -4.01(0.01) | 2.88(0.01) |
| Integration Time: 4s | 2.60(0.05) | 1.33(0.08) | 3.22(0.03) | 1.01(0.01) | -1.12(0.01) | 2.99(0.01) | -4.01(0.02) | 2.89(0.01) |
| Integration Time: 8s | 2.66(0.07) | 0.96(0.06) | 3.24(0.02) | 1.06(0.01) | -1.06(0.01) | 3.01(0.01) | -4.00(0.02) | 2.93(0.01) |
| DGTM-Cubature (Ours, Ablation) | | | | | | | | |
| Integration Time: 2s | 2.74(0.05) | 1.79(0.02) | 3.31(0.02) | 0.98(0.01) | -0.89(0.00) | 2.85(0.01) | -3.87(0.01) | 2.92(0.00) |
| Integration Time: 4s | 2.63(0.05) | 1.45(0.02) | 3.30(0.02) | 0.98(0.01) | -0.98(0.01) | 2.85(0.01) | -4.18(0.04) | 2.89(0.01) |
| Integration Time: 8s | 2.56(0.08) | 1.42(0.02) | 3.28(0.02) | 0.99(0.01) | -0.97(0.01) | 2.86(0.01) | -4.32(0.07) | 2.90(0.00) |
| DGTM-BMM (Ours, Proposed) | | | | | | | | |
| Integration Time: 2s | 2.45(0.02) | 1.18(0.07) | 3.00(0.01) | 0.97(0.01) | -1.14(0.00) | **2.81(0.01)** | -3.92(0.01) | 2.85(0.00) |
| Integration Time: 4s | 2.41(0.02) | 0.82(0.06) | **2.92(0.02)** | 0.96(0.01) | -1.21(0.01) | **2.81(0.01)** | -4.01(0.01) | **2.77(0.01)** |
| Integration Time: 8s | **2.37(0.03)** | **0.70(0.06)** | **2.92(0.02)** | 0.93(0.02) | **-1.22(0.00)** | **2.80(0.01)** | -4.45(0.02) | **2.76(0.01)** |

Table 3.2.: RMSE values of 8 benchmark datasets. We report average and standard error over 20 runs.

| | Boston | Energy | Concrete | Wine Red | Kin8nm | Power | Naval | Protein |
|---|---|---|---|---|---|---|---|---|
| Dropout [72] | **2.97(0.19)** | 1.66(0.04) | 5.23(0.12) | **0.62(0.01)** | 0.10(0.00) | **4.02(0.04)** | **0.01(0.00)** | **4.36(0.01)** |
| DVI [44] | - | - | - | - | - | - | - | - |
| DGTM-MC [71] | | | | | | | | |
| Integration Time: 2s | 3.97(0.14) | 2.68(0.07) | 6.14(0.10) | 0.66(0.01) | **0.08(0.00)** | 4.42(0.03) | **0.01(0.00)** | 4.67(0.01) |
| Integration Time: 4s | 3.65(0.15) | 2.38(0.19) | 6.02(0.08) | 0.66(0.01) | **0.08(0.00)** | 4.45(0.03) | **0.01(0.00)** | 4.66(0.01) |
| Integration Time: 8s | 3.84(0.19) | **0.73(0.08)** | 6.18(0.13) | 0.68(0.01) | 0.09(0.00) | 4.56(0.03) | **0.01(0.00)** | 4.77(0.02) |
| DGTM-Cubature (Ours, Ablation) | | | | | | | | |
| Integration Time: 2s | 4.34(0.16) | 2.08(0.06) | 6.49(0.10) | 0.64(0.00) | 0.10(0.00) | 4.15(0.03) | **0.01(0.00)** | 4.51(0.02) |
| Integration Time: 4s | 3.80(0.15) | 1.37(0.03) | 6.08(0.09) | 0.64(0.00) | 0.09(0.00) | 4.17(0.03) | **0.01(0.00)** | 4.44(0.02) |
| Integration Time: 8s | 3.49(0.15) | 1.12(0.04) | 5.99(0.09) | 0.64(0.00) | 0.09(0.00) | 4.16(0.03) | **0.01(0.00)** | 4.44(0.02) |
| DGTM-BMM (Ours, Proposed) | | | | | | | | |
| Integration Time: 2s | 3.41(0.13) | 2.23(0.09) | 5.47(0.10) | 0.64(0.01) | **0.08(0.00)** | 4.07(0.03) | **0.01(0.00)** | 4.63(0.01) |
| Integration Time: 4s | 3.17(0.14) | 1.40(0.18) | 5.26(0.10) | 0.64(0.00) | **0.08(0.00)** | 4.11(0.04) | **0.01(0.00)** | 4.48(0.00) |
| Integration Time: 8s | 3.26(0.15) | 0.87(0.13) | **5.12(0.09)** | **0.63(0.00)** | **0.08(0.00)** | 4.07(0.03) | **0.01(0.00)** | 4.45(0.00) |

### 3.5.3. Time Series Classification

In time series classification, we aim to predict the label $y \in \{1, 2, \ldots, K\}$ after observing the time series $U = \{u_t\}_{t=0}^{T-1}$ for $T$ time steps. We treat the observations $u_t \in \mathbb{R}^{D_u}$ as inputs to a latent DGTM with an input-dependent transition model $p(x_{t+1}|x_t, u_t, w_f, w_r)$ with parameters $w_f$ and $w_l$. The probability of observing the label $y$ is

$$p(y|U, w_f, w_r, w_y) = \int p(y|x_T, w_y)p(x_T|U, w_f, w_r)dx_T, \tag{3.40}$$

with the marginal distribution

$$p(x_T|U, w_f, w_r) = \int \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t, w_f, w_r)p(x_0)dx_0, \ldots, x_{T-1}. \tag{3.41}$$

Our algorithm BMM allows us to approximate the marginal $p(x_T|U, w_f, w_r, w_y)$. The conditional distribution $p(y|x_T, w_y)$ is modeled as a linear layer with parameters $w_y$ followed by softmax in order to map $x_T$ to the distribution over class labels. Following [44], the output moments of the softmax layer can be closely approximated, which allows us to compute the target distribution $p(y|U, w_f, w_r, w_y)$ via moment matching. The parameters $w_f, w_r, w_y$ are inferred by MLE similarly as in Eq. 3.39.

#### Datasets

We benchmark on two datasets:

*(i) MNIST*: This dataset consists of 60k training and 10k testing images of single digits between 0 and 9. Each image has size $28 \times 28$. We treat the images as time series with length 28 and dimensionality 28. We use a batch size of 10.

*(ii) IMDB*: This dataset consists of 37500 training and 12500 testing sequences. Each sequence corresponds to a movie review and has varying length between 2 and 652 words. We follow the tutorial[1] for selecting hyperparameters (e.g. batch size, word dictionary size) since (i) it is easy to reproduce, and (ii) has already been applied in the existing literature [76], where it was shown to be capable of highlighting differences between models and inference strategies. We generate for each run a word dictionary of size 1000 and omit sequences that are longer than 500 words, which are in total 3 omitted sequences. We use a batch size of 50.

[1] `https://www.kaggle.com/code/arunmohan003/sentiment-analysis-using-lstm-pytorch`

### Baselines

We use the same DGTM variants as in the previous section, i.e., DGTM-MC and DGTM-Cubature. We use mean update neural networks with two hidden layers and covariance update neural networks with one hidden layer. We choose a hidden layer size of 50 and a latent dimensionality of 32. Additionally, we compare against two competitive time series classification models:

*(i) LSTM*: We use a stacked LSTM consisting of two LSTMs with a hidden layer size of 64.

*(ii) Transformer*: Our Transformer architecture follows the original work[1]. We use 2 Transformer encoder layers with hidden size 64 and 2 attention heads.

The DGTM/LSTM/Transformer models have 11k/62k/53k parameters for the MNIST experiment and 40k/130k/120k parameters for the IMDB experiment.

### Results

We present time series classification results in Tab. 3.3. Our model DGTM-BMM achieves the same level of performance as LSTM despite having fewer parameters and not being tailored towards modeling long-term effects, demonstrating the applicability of our model as a general-purpose tool. Our proposed model DGTM-BMM outperforms DGTM-MC and DGTM-Cubature. Furthermore, it is worth noting that for the IMDB dataset, it is not possible to train DGTM-Cubature when using a GPU with 40GB memory due to an *Out of Memory* (OOM) error. We observe the Transformer model to be outperformed by LSTM and DGTM-BMM due to possible overparameterization.

Our experimental results also hold when varying the hyperparameters. When rerunning the IMDB experiment with an increased word dictionary of size 10k, a batch size of 250, and a simple (one layer) architecture with 32 neurons and applying Dropout (0.25), we can increase the performance of all methods by up to 3% without changing the relative ranking between the methods.

[1] `https://pytorch.org/tutorials/beginner/transformer_tutorial.html`

Table 3.3.: Results for different time series classification tasks. We report average and standard error over 10 runs. ECE is reported in percent.

| | MNIST | | | IMDB | | |
|---|---|---|---|---|---|---|
| | ACC | NLL | ECE | ACC | NLL | ECE |
| LSTM | **98.19(0.05)** | **0.06(0.00)** | **2.16(0.07)** | 85.61(0.12) | 0.34(0.00) | **15.17(0.11)** |
| Transformer | 95.87(0.21) | 0.13(0.01) | 5.12(0.20) | 82.33(0.41) | 0.40(0.01) | 17.59(0.28) |
| DGTM-MC [71] | 81.60(0.53) | 0.58(0.01) | 20.00(0.30) | 84.92(0.07) | 0.35(0.00) | 16.72(0.09) |
| DGTM-Cubature (Ours, Ablation) | 95.16(0.81) | 0.19(0.03) | 9.19(1.07) | OOM | OOM | OOM |
| DGTM-BMM (Ours, Proposed) | 98.11(0.16) | **0.06(0.03)** | 2.52(0.09) | **85.87(0.16)** | **0.33(0.00)** | 15.48(0.12) |

### 3.5.4. Dynamical System Modeling

As DGTMs describe dynamical systems, they are a natural choice for dynamical system modeling. We may calculate the joint distribution of a trajectory $X = \{x_t\}_{t=0}^{T}$ governed by the DGTM in Eq. 3.1, which is observed on $T+1$ time points, as

$$p(X|w_f, w_r) = p(x_0, \dots, x_T|w_f, w_r) = p(x_0) \prod_{t=0}^{T-1} \mathcal{N}(x_{t+1}|x_t + f(x_t)\Delta t, \text{diag}(r(x_t))\Delta t). \tag{3.42}$$

Given a set of trajectories, the parameters $w_f, w_r$ can be inferred by MLE as in Eq. 3.39. This training objective requires one-step predictions and no Monte Carlo approximations. We instead propose multi-step training with our method BMM to improve long-term predictions. In contrast to the standard training objective, multi-step training propagates the state forward in time without receiving any feedback, mimicking the behavior during test time. Since we want to use the same objective during training and test time, we opt for directly maximizing the *Predictive Log-Likelihood* (PLL) during training

$$\text{PLL}(X|w_f, w_r) = \text{PLL}(x_0, \dots, x_T|w_f, w_r) = \log p(x_0) + \sum_{t=1}^{T} \log p(x_t|x_0, w_f, w_r), \tag{3.43}$$

where the transition kernel $p(x_t|x_0, w_f, w_r)$ for some $t > 0$ can be efficiently approximated with our algorithm BMM. The observation that using one-step ahead predictions in training is not sufficient to obtain reliable multi-step ahead predictions during testing has also been made in [77] where the authors propose a scheduled sampling strategy to gradually switch from single to multi-step ahead predictions during training. Multi-step ahead training

has also been successfully applied for spatio-temporal forecasting[78] and model-based planning [79]. Given the state $x_0$, the PLL aims at optimizing the average marginal log-likelihood $\log p(x_t|x_0, w_r, w_f)$, while the standard objective aims at optimizing the joint likelihood $\log p(x_1, \ldots, x_T|x_1, w_f, w_r)$. The PLL is most useful for tasks that can be solved by assessing the marginal distributions only. An important application class that falls into this category is in the context of autonomous driving, in which the marginal distributions of traffic participants at a given time horizon are often sufficient to control the car [80]. As a consequence, many papers in the autonomous driving literature report as evaluation metrics the performance of their method at fixed time intervals which match our PLL objective [81, 82, 83]. In contrast, optimizing the joint log-likelihood is preferred for tasks that require sampling realistic-looking trajectories, as it is done in sentence generation [84].

### Datasets

We benchmark on three datasets:

*(i) Lotka-Volterra*: We choose stochastic Lotka-Volterra equations as in [85]

$$x_{t+1} = \begin{bmatrix} x_{t,1} \\ x_{t,2} \end{bmatrix} + \begin{bmatrix} 2x_{t,1} - x_{t,1}x_{t,2} \\ x_{t,1}x_{t,2} - 4x_{t,2} \end{bmatrix} \Delta t + \sqrt{\begin{bmatrix} 0.05 & 0.03 \\ 0.03 & 0.09 \end{bmatrix} \Delta t} \zeta_t.$$

We generate 128 paths with a small step size of $\Delta t = 10^{-5}$ seconds. Afterward, we coarsen the dataset such that 200 equally spaced observations between $0 - 10$ seconds remain. The first 100 observations are used for training, and the remaining 100 observations for testing. We use a batch size of 16 and a prediction horizon of $10$ steps.

*(ii) Beijing Air Quality*: The atmospheric air-quality dataset[1] from Beijing [86] consists of hourly measures over the period 2014-2016 at three different locations. The air quality is characterized by 10 different features at each location. Including the timestamp, we obtain in total 34 features. We follow [87] for designing the experimental setup. The first two years are used for training, and we test on the first 48 hours in the year 2016. We use a batch size of 16 and a prediction horizon of $10$ steps for training.

*(iii) 3-DOF-Robot*: The 3-DOF-Robot dataset[2] [88] consists of multiple trajectories with length 14000, 3 input, and 9 output dimensions, recorded with a sampling rate of 1kHz.

[1] `https://archive.ics.uci.edu/dataset/501/beijing+multi+site+air+quality+data`
[2] `https://owncloud.tuebingen.mpg.de/index.php/s/3THSfyBgFrYykPc?path=%2F`

The dataset was recorded at two different operating modes: (i) 50 recordings of low-frequency oscillations and (ii) 50 recordings of high-frequency. We train on the first 38 trajectories and validate on the next 3 trajectories using the low-frequency recordings. We use as a test set the final 9 low-frequency trajectories (IID) and the final 9 high-frequency recordings (Transfer). We use a batch size of 16 and a prediction horizon of $16$ steps for training.

### Baselines

We use the same DGTM variants as in the previous section, i.e., DGTM-MC and DGTM-Cubature, and train them on multi-step predictions. We use mean update neural networks with two hidden layers and covariance update neural networks with one hidden layer. The hidden layer size is 100 for Beijing Air Quality and 50 for the other datasets. Similarly, as in the previous section, we use LSTM/Transformer models and modify them to predict a deterministic or stochastic output. The stochastic output of both models follows a Gaussian distribution for which we predict the mean and variance at each prediction step. We train LSTM/Transformer models on one-step predictions as we found multi-step training to produce deteriorated results due to high variance. Additionally, we compare against:

*(i) DGTM-One-Step*: A DGTM trained on one-step predictions. The training objective is deterministically tractable (see Eq. 3.42). Testing is done with Monte Carlo rollouts.

*(ii) NODE* [53]: A neural ODE that has no stochastic component. We report only MSE.

*(iii) diff WGP* [87]: A SDE as the predictive mean and covariance of a GP, the state of the art of differential equation modeling with GPs.

The DGTM/NODE/LSTM/Transformer models have 3k/3k/42k/122k parameters for the Lotka-Volterra, 24k/17k/61k/132k parameters for the Beijing Air Quality, and 5k/4k/57k /220k parameters for the 3-DOF-Robot dataset.

### Results

As shown in Tab. 3.4, BMM outperforms all baselines in all datasets with respect to both NLL and MSE, with the only exception of NLL in the 3-DOF-Robot (IID) dataset and MSE in the weather dataset in which it performs second best. BMM proves to make more accurate predictions than both DGTM-MC and diff WGP, probably due to the improved stability of the training process thanks to its deterministic objective. This outcome is despite the fact

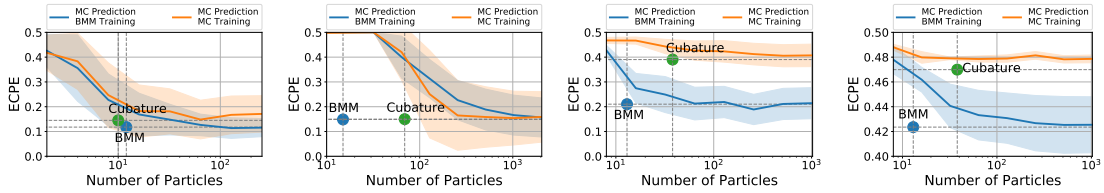
Table 3.4.: Forecasting results for different dynamical system modeling tasks. We report average and standard error over 10 runs.

| | Lotka-Volterra | | Air Quality | | 3-DOF-Robot IID | | 3-DOF-Robot Transfer | |
|---|---|---|---|---|---|---|---|---|
| | MSE | NLL | MSE | NLL | MSE | NLL | MSE | NLL |
| LSTM (Deterministic) | 1.86(0.02) | - | 1.59(0.08) | - | 0.05(0.00) | - | 2.67(0.05) | - |
| Transformer (Deterministic) | 1.94(0.04) | - | **1.41(0.12)** | - | 0.21(0.02) | - | 3.52(0.17) | - |
| LSTM (Stochastic) | 2.41(0.09) | 5.32(0.14) | 2.37(0.04) | 55.79(0.43) | 0.43(0.03) | 9.49(0.23) | 3.03(0.06) | 131(6) |
| Transformer (Stochastic) | 2.19(0.06) | 5.03(0.06) | 2.49(0.05) | 55.43(0.44) | 0.31(0.01) | 10.07(0.41) | 3.62(0.16) | 144(9) |
| NODE [53] | 1.98(0.04) | - | 1.85(0.15) | - | **0.04(0.00)** | - | 3.78(0.12) | - |
| diff WGP [87] | - | - | 2.39(0.04) | 46.92[1](0.75) | - | - | - | - |
| DGTM-MC [71] | 2.07(0.11) | 4.95(0.44) | 1.90(0.07) | 43.84(0.75) | **0.04(0.00)** | 21.69(1.75) | 4.69(0.13) | 1821(12) |
| DGTM-One-Step (Ours, Ablation) | 2.44(0.14) | 5.38(0.27) | 1.75(0.09) | 62.53(0.96) | 0.05(0.00) | 23.40(1.34) | 4.43(0.11) | 1437(25) |
| DGTM-Cubature (Ours, Ablation) | 1.91(0.08) | 4.84(0.19) | **1.45(0.04)** | 37.84(0.39) | **0.04(0.00)** | **5.65(0.33)** | 4.18(0.13) | 352(27) |
| DGTM-BMM (Ours, Proposed) | **1.75(0.03)** | **4.35(0.15)** | **1.44(0.10)** | **34.30(0.50)** | **0.03(0.01)** | 7.27(1.31) | **2.31(0.08)** | **102(12)** |

that diff WGP models the transition noise as a Wishart process, while DGTM models the transition noise as a normal distribution with diagonal covariance. Our results indicate that using a one-step training objective may hinder learning long-term relations, as DGTM-One-Step performs worse than DGTM-Cubature and our method in all datasets. Our model outperforms LSTM/Transformer on stochastic time-series modeling tasks. While we cannot rule out other causes, this might be due to the different objectives when designing the architectures: DGTM are targeted towards stochastic dynamical systems, while LSTM/Transformer architectures aim to capture long-term effects that are more frequent in language modeling tasks than in the data sets used in this experiment. We observe in all datasets that changing the Transformer/LSTM architecture from a deterministic to stochastic output results in higher MSE. A deterministic multi-step training objective for these two methods seems to be a promising future research direction.

Compared to the previous regression task, our method improves stronger on its baselines in terms of uncertainty calibration. As shown in Fig. 3.8, the BMM algorithm reaches a level of uncertainty calibration, which is prohibitively costly for MC sampling. MC sampling requires more than 50 roll-outs in all three applications to match the ECPE, which our deterministic BMM provides. We observe BMM to bring smaller ECPE than cubature in three of the four plots at a comparable or less computational cost.

[1] After correspondence with the authors, we present the correct NLL for diff WGP. In the original paper, an unknown issue caused a shift in the NLL.




(a) Lotka-Volterra. (b) Air Quality. (c) Robot, IID. (d) Robot, Transfer.

Figure 3.8.: Cost-benefit analysis of calibration with different methods. One particle is equal to one MC simulation along a trajectory. BMM is our proposed method, Cubature is our method that uses cubature for VMM, the orange line is training and prediction with MC sampling, and the blue line is training with our method and prediction with MC sampling. We show mean and standard deviation over 10 runs.

### 3.5.5. High Dimensional Dynamics

In this experiment, we compare our inference scheme against Monte Carlo Sampling when varying the dimension $D_x$ and the number of Monte Carlo Samples $S$.

#### Dataset

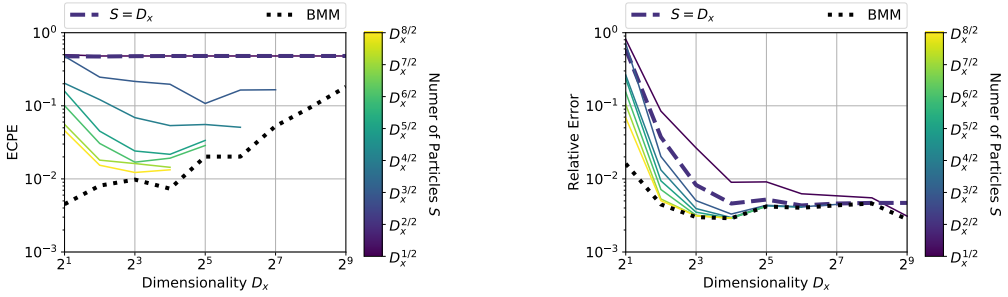
We use a $D_x$-dimensional Ornstein-Uhlenbeck process $x_{t+1} = x_t + (\lambda - x_t)\Delta t + \Gamma\sqrt{\Delta t}\zeta_t$ for data generation. The term $\lambda \in \mathbb{R}^{D_x}$ is sampled from a normal distribution $\lambda \sim \mathcal{N}(0, I)$ and $\Gamma \in \mathbb{R}^{D_x \times D_x}$ is sampled from a Wishart distribution $\Gamma \sim \mathcal{W}(I, \dim(I))$. We vary the dimensionality between $D_x = 2$ and $D_x = 512$. We generate 100 paths for each dimensionality with a length of $T = 10s$ and $\Delta t = 0.1s$.

#### Baselines

We compare our method DGTM-BMM against DGTM-MC using a DGTM with one hidden layer in the mean update neural net and a constant covariance update. The hidden layer size is $D_x$, and the covariance update has dimensionality $D_x \times D_x$. We train the DGTM on one-step predictions, which is a deterministically tractable training objective (see Eq. 3.42). After training, we use BMM as well as MC sampling for testing. This enables a fair comparison between both methods as the same DGTM is used during test time.

### Results

We present our main findings in Fig. 3.9a. First, we observe that, for all methods, the ECPE drops when the number of dimensions increases. The increasing complexity of the problem might explain this; the Ornstein-Uhlenbeck process is described by $O(D_x^2)$ many parameters, while the size of the dataset scales with $O(D_x)$. Next, using a MC sampling strategy with $S = D_x$ particles, which has the same computational complexity as our method (see Sec. 3.4.3), results in uncalibrated predictions regardless of the chosen dimension. Even though Monte Carlo sampling can come close to the ECPE levels of BMM, it never reaches the same ECPE level of BMM, even when using $S = D_x^4$ particles. Furthermore, we frequently encounter an OOM error on a CPU and 32GB memory for DGTM-MC for high dimensions and costly sampling strategies beyond $S = D_x$. This can be seen by the early stopping of solid lines in Fig. 3.9a and 3.9b. Lastly, we observe in Fig. 3.9b that MC sampling can give satisfactory results if the quantity of interest is the mean, i.e., not the covariance. In such cases, the sampling strategy $S = \sqrt{D_x}$ can give an approximation of the true mean with less than 1% of relative error.




(a) ECPE.

(b) Relative error.

Figure 3.9.: Comparison of BMM to MC in terms of ECPE (left) and relative error (right) as a function of the dimensionality of $x$. We calculate the relative error as $\frac{1}{K}\sum_{k=1}^{K}||e_k - \hat{e_k}||^2/||e_k||^2$, where $e_k$ is the true mean at time step $k$ and $\hat{e}_k$ is the predicted mean. The dotted black line is our method BMM, and the solid lines represent MC sampling. The color of the solid lines indicates the number of particles $S$ as a function of the dimensionality $D_x$. The dashed line is the sampling strategy $S = D_x$, which has the same computational complexity as our method $\mathcal{O}(D_x^2)$. We show the mean over 20 runs.

## 3.6. Summary

We proposed a computationally efficient and deterministic approximation of the transition kernel for DGTMs. Our method enables accurate prediction at a moderate computational cost. We presented a general-purpose methodological contribution that is applicable beyond the use cases demonstrated in our experiments.

In the following chapters, we will present several extensions of our method. These include: (i) modeling latent dynamical systems instead of fully observed ones, (ii) modeling multiple modes that allow for capturing complex behavior, (iii) interaction modeling between different agents, (iv) reducing computational costs through low-rank approximations of the covariance, (v) introducing weight uncertainty to account for model uncertainty.

# 4. Cheap and Deterministic Inference for Deep State-Space Models of Interacting Dynamical Systems

Many dynamical systems, such as traffic flow [89, 90], fluid dynamics [91] or human motion [92], involve interactions between agents. *Graph Neural Networks* (GNN) [39] have recently emerged as a powerful tool in these settings since they allow learning the dynamics of interacting systems from data only. Recent research has made great advances in modeling deterministic complex systems by being able to extrapolate from systems with a small number of agents and short time horizons to systems with a high number of agents and long time horizons. These methods have been successfully applied to a number of physical systems covering fluids, rigid solids, and deformable materials [93]. However, for many real-world applications, predicting a single future trajectory for each agent is not enough since the stochasticity in the dynamical system has significant consequences. For instance, in autonomous driving, the driver's intention (e.g. overtaking, turning, lane changing) is a hidden factor that may induce different modes of driving trajectories.

In the deep learning literature, multiple architectures have been successfully applied to partially observable dynamical systems with two prominent classes being (i) recurrent methods that apply a fixed transition model repeatedly at each time step [95, 96], and (ii) history-based methods that aggregate information from the past using either convolutional filters [97, 98] or attention modules [84, 99]. Recurrent methods capture the system's internal state at each time point $t$ in a latent state $x_t$. Predicting the output in this manner respects the causal order of the dynamical system, i.e., the latent state $x_t$ of time point $t$ is needed to compute the latent state $x_{t+1}$ at the next time point $t+1$. For interacting systems, the latent space grows linearly with the number of agents, requiring a high number of MC samples, which can make these methods prohibitively slow. To the best of our knowledge, numerical integration schemes have not been explored for multi-agent dynamical systems. In contrast, probabilistic history-based methods directly predict the distribution over future

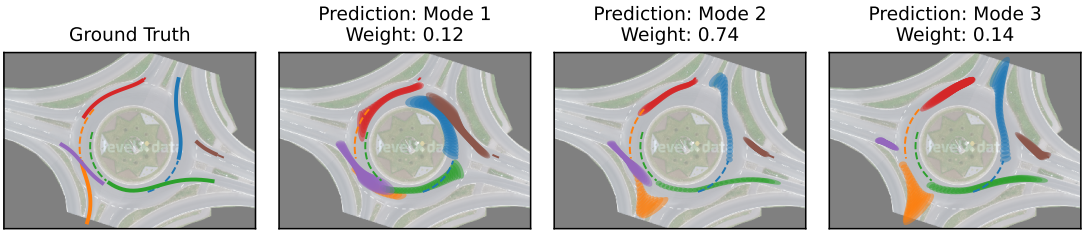



Figure 4.1.: We approximate the predictive distribution of a *Graph Deep State-Space Model* (GDSSM) as a Gaussian mixture distribution via deterministic moment matching rules. Given historical information in the form of observed trajectories (dashed lines), our proposed GDSSM architecture predicts future dynamics while taking interactions between traffic participants into account. We present the true future trajectories (left-most plot) as solid lines. For each mode and traffic participant, we show the predicted 95% confidence interval of our model (three right-most plots). Our model accounts for interactions; for example, the brown vehicle is only entering the roundabout if the blue vehicle is staying in the roundabout (Mode 1). If the blue vehicle is leaving the roundabout, the entering lane for the brown vehicle is blocked by the blue vehicle (Mode 2, Mode 3), and the brown car has to wait. The ground truth data and the map come from the rounD dataset [94].

trajectories, mitigating the sampling overhead. However, this approach makes the learning problem hard as the model needs to learn the future distribution for multiple time steps ahead. Complex models (e.g. large neural networks) are often necessary to account for this. This can prevent their usage in embedded systems with limited memory capacity.

In this chapter, we present a novel approach for modeling stochastic dynamical systems with interacting agents that is able to generate expressive multimodal predictive distributions over future trajectories in a runtime and memory-efficient manner. We model the unknown stochastic dynamical system as a *Deep State-Space Model* (DSSMs) in which the shared dynamics of all agents are modeled in a joint latent space using GNNs. The stochasticity in the transition and emission models of the DSSM captures the intrinsic data uncertainty. We remark that we do not focus on incorporating model uncertainty in this chapter. Instead, we defer the discussion of model uncertainty to Chap. 5. Our model belongs to the family of recurrent neural networks, and we replace the expensive MC operations during training and testing by extending the previously introduced BMM algorithm (see Chap. 3) to the case of DSSMs with interactions and multiple modes. We use GNNs in the

transition model to account for interactions. In addition, to handle multiple modes, we introduce a *Gaussian Mixture Model* (GMM) over the initial latent state. By approximating the predictive distribution in a multimodal manner, we can achieve a more accurate representation of the data uncertainty. We independently apply our moment matching rules for each mixture component to arrive at multimodal predictive distributions over future trajectories. In autonomous driving, the initial latent state is often estimated from historical information and can provide information about the drivers' intentions [3]. Conditioned on the initial latent state, the predictive distribution can often be accurately modeled with a unimodal distribution [12, 83]. Finally, as a wide variety of dense traffic scenarios exist, the high number of agents can result in prohibitively large GMM covariance matrices as their size grows quadratically with the number of traffic participants. We address this problem by proposing structured covariance approximations.

We summarize our contribution as follows:

*(i)* We derive output moments for GNN layers, which makes GNNs applicable to moment matching algorithms. This leads to the first deterministic inference scheme for deep state-space models for interacting systems.

*(ii)* We introduce a GMM distribution over the initial latent states that results in multimodal predictive distributions over future trajectories.

*(iii)* We propose structured approximations to the GMM covariance matrices that can reduce the computational complexity of our approach from $O(M^3)$ to $O(M^2)$, where $M$ is the number of agents.

In our experiments, we benchmark our proposed model on two challenging autonomous driving datasets. Our results demonstrate that our deterministic model has strong empirical performance compared to state-of-the-art alternatives. We visualize the predictive output distribution for a real-world traffic scenario on a roundabout with multiple agents in Fig. 4.1. The future distribution is multimodal, as traffic participants can leave the roundabout from several exits. Our model is capable of predicting multiple modes, which we efficiently approximate as a GMM, and takes interactions into account using GNNs.

To gain further insights into our model and inference scheme, we examine the impact of individual contributions in an ablation study. We also provide an empirical runtime study of our covariance approximations. Our findings indicate that sparse covariance approximations reduce the computational complexity by a factor of up to 100, which makes them favorable for applications with limited computational resources. We conclude our experiments by studying the generalization capabilities of our model on out-of-distribution data, e.g., traffic environments that have not been observed during training.

## 4.1. Graph Deep State-Space Models

We aim to model stochastic dynamical interactions between agents following complex behavioral patterns, such as traffic interactions. We extend deep state-space models to interacting systems by proposing *Graph Deep State-Space Models* (GDSSM), which use graph neural networks in the transition model to capture interactions between agents. We define our probabilistic model in this section and then introduce a novel scheme for efficient and deterministic training and predictions in the subsequent section.

We are interested in modeling the dynamics of $M$ interacting agents with deep state-space models using a coupled latent space. In other words, instead of using a $D_x$-dimensional latent space for each agent, we assume that the agents share a latent space of size $MD_x$. Since (i) the number of agents can vary between scenes, (ii) the transition model should be agnostic to the order of the agents, and (iii) it is challenging to parameterize high-dimensional latent spaces, we use GNNs in the transition model.

More formally, we denote the state of agent $m$ at time step $t$ as $x_t^m \in \mathbb{R}^{D_x}$ and the set of all state variables as $x_t = \{x_t^m\}_{m=1}^M$. The dynamical system follows a DSSM (see Sec. 2.1) with a *Deep Gaussian Transition model* (DGTM)

$$x_0 \sim p(x_0|\mathcal{I}), \tag{4.1}$$

$$x_t \sim \mathcal{N}\left(x_t|x_{t-1} + f(x_{t-1},\mathcal{I}), \operatorname{diag}\left(r(x_{t-1},\mathcal{I})\right)\right), \tag{4.2}$$

$$y_t \sim \mathcal{N}\left(y_t|g(x_t), \operatorname{diag}\left(s(x_t)\right)\right), \tag{4.3}$$

where $\mathcal{I} \in \mathbb{R}^{D_\mathcal{I}}$ is the context variable that encodes auxiliary information, such as historical or relational information. The emission model follows a Gaussian distribution with mean $g : \mathbb{R}^{MD_x} \to \mathbb{R}^{MD_y}$ and variance $s : \mathbb{R}^{MD_x} \to \mathbb{R}_+^{MD_y}$, where $g(x_t)$ and $s(x_t)$ are both neural networks with arbitrary architecture and parameters $w_g$ and $w_s$. The mean update $f : \mathbb{R}^{MD_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{MD_x}$ and the covariance update $r : \mathbb{R}^{MD_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}_+^{MD_x}$ are implemented with the help of graph neural networks

$$f(x_t,\mathcal{I}) = \begin{bmatrix} \tilde{f}(x_t^1, x_t^{\mathcal{N}_1},\mathcal{I}) \\ \vdots \\ \tilde{f}(x_t^M, x_t^{\mathcal{N}_M},\mathcal{I}) \end{bmatrix}, \tag{4.4}$$

$$r(x_t,\mathcal{I}) = \begin{bmatrix} \tilde{r}(x_t^1, x_t^{\mathcal{N}_1},\mathcal{I}) \\ \vdots \\ \tilde{r}(x_t^M, x_t^{\mathcal{N}_M},\mathcal{I}) \end{bmatrix}, \tag{4.5}$$

with parameters $w_f$ and $w_r$. The agent-specific mean update is denoted by $\tilde{f} : \mathbb{R}^{D_x} \times \mathbb{R}^{D_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{D_x}$ and the variance by $\tilde{r} : \mathbb{R}^{D_x} \times \mathbb{R}^{D_x} \times \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}_+^{D_x}$. Both implement the update function as general graph neural networks (see Sec. 2.4). The message to

node $m$ at time step $t$ is denoted as $x_t^{\mathcal{N}_m} \in \mathbb{R}^{D_x}$ and contains aggregated information of the states from all neighboring agents. A deterministic variant of our model, e.g., setting $r(x_t, \mathcal{I}) = 0$, has been successfully used for learning surrogate models for complex physical systems [93]. We further assume that it is sufficient to couple the latent dynamics across the agents (Eq. 4.2) and keep the emission model (Eq. 4.3) independent across agents. Note that our transition model consists of a single aggregation and update step, which is sufficient if the data is densely sampled such that the information flow between agents is fast compared to the evolution of the state dynamics. However, extending the model to multiple message-passing steps per time point is also straightforward by stacking multiple GNN layers in the mean and covariance update function.

Furthermore, we note that although the transition noise factorizes across agents, correlations between agents emerge since the mean and the variance depend not only on the state of the $m$-th agent but also on the states of all neighboring agents. After $a$ aggregation steps, our GNN model accounts for correlations between agent $m$ and agent $m'$ provided they are connected by a path that is at most $a$ steps long. In contrast, methods that only take the state of the $m$-th agent into account do not lead to any correlations, while methods that take the state of all other agents into account lead to a fully correlated covariance matrix after one time step.

To complete the probabilistic description of our model, we further specify the distribution of the initial latent state $x_0 \in \mathbb{R}^{MD_x}$ with a *Gaussian Mixture Model* (GMM)

$$v \sim \mathsf{Cat}([\pi_1(\mathcal{I}), \ldots, \pi_V(\mathcal{I})]), \tag{4.6}$$

$$x_0 \sim \mathcal{N}(x_0 | \mu_{0,v}(\mathcal{I}), \mathsf{diag}(\Sigma_{0,v}(\mathcal{I}))), \tag{4.7}$$

where $\mathsf{Cat}([\pi_1(\mathcal{I}), \ldots, \pi_V(\mathcal{I})])$ is a categorical distribution with $V$ mixture components. Each component is specified by its weight $\pi_v : \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}_+$, mean $\mu_{0,v} : \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{MD_x}$, and diagonal covariance $\Sigma_{0,v} : \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}_+^{MD_x}$. The weights $\pi_{1:V}$ form a standard $V$-simplex. We use a GNN, which we refer to as the embedding model $h : \mathbb{R}^{D_\mathcal{I}} \to \mathbb{R}^{V+2VMD_x}$ with weights $w_h$, to model the initial state distribution

$$h(\mathcal{I}) = \begin{bmatrix} \pi_{1:V}(\mathcal{I}) \\ \mu_{0,1:V}(\mathcal{I}) \\ \Sigma_{0,1:V}(\mathcal{I}) \end{bmatrix}. \tag{4.8}$$

We assume that the context variable $\mathcal{I}$ contains relational information as a set of edges as well as historical information for each agent in the form of an observed trajectory. In a

sense, the embedding model acts hereby as a filter, which learns a distribution over the initial latent state from past observations.

In autonomous driving, the initial latent state can be connected to the drivers' intention [3]. The context information $\mathcal{I}$ is often insufficient to rule out different hypotheses about the future, e.g., does the car behind us want to overtake in the next five seconds or not. Using a mixture model allows us to incorporate different hypotheses into the model in a principled manner, which will ultimately lead to multimodal predictive distributions.

State-space models and graph neural networks have been previously combined for multi-agent trajectory forecasting by [100]. In contrast to our work, the authors (i) use a different model definition by applying recurrent neural networks and a non-Gaussian density in the transition model and (ii) perform Monte Carlo sampling during inference which can lead to slow convergence. The following section shows that our model definition allows for more efficient training by performing deterministic moment matching rules. We compare against Monte Carlo alternatives in our experiments.

## 4.2. Deterministic Approximations for GDSSMs

This section presents our novel inference scheme for GDSSMs that enables efficient and deterministic training and predictions. In Sec. 4.2.1, we first give a short overview of existing inference techniques and compare it to our scheme, which aims at directly maximizing the predictive log-likelihood of future trajectories. The predictive log-likelihood cannot be computed in closed form. We propose an efficient and deterministic approximation to it in Sec. 4.2.2 that relies on the BMM algorithm (see Chap. 3). Our moment matching rules necessitate the computation of output moments and expected Jacobians of graph neural network layers. We present output moments for commonly used layers in Sec. 4.2.3. As our algorithm approximates the output distribution at each time step with a Gaussian mixture distribution over all agents, the resulting covariance matrix for each mixture component can become computationally intractable for a large number of traffic participants. The reason for this is the need to calculate the covariance between all pairs of different traffic participants. We address this pain point in Sec. 4.2.4 by proposing sparse approximations to the covariance.

### 4.2.1. Parameter Inference

For the sake of brevity, we denote the set of all weights as $w = \{w_f, w_r, w_g, w_s, w_h\}$. The training objective of classical inference methods for state-space models can be formalized as depicted below

$$\log p(y_1, \ldots, y_T|\mathcal{I}, w) = \log \int p(x_0|\mathcal{I}, w_h) \prod_{t=1}^{T} p(x_t|x_{t-1}, \mathcal{I}, w_f, w_r) \times p(y_t|x_t, w_g, w_s) dx_0, \ldots, dx_T, \tag{4.9}$$

which aims at directly maximizing the log-likelihood of the data. In the above equation, the term $p(x_t|x_{t-1}, \mathcal{I}, w_f, w_r)$ is defined in Eq. 4.2 and $p(y_t|x_t, w_g, w_s)$ in Eq. 4.3. This quantity can only be computed in closed form if the emission and transition model are linear Gaussians. In our case, the transition model is parameterized by a graph neural network and the emission model by a standard neural network.

Therefore, most existing methods apply either a particle filter [101], variational inference [26, 27], or a combination of both [102, 103, 104] to approximate the log-likelihood. These approaches have in common that they require learning a proposal distribution $q(x_0, \ldots, x_T|y_1, \ldots, y_T)$. In particle filtering, the proposal distribution is recursively defined by the importance function $q(x_t|x_0, \ldots, x_{t-1}, y_1, \ldots, y_t)$ and is optimal in terms of variance by setting it to the true filtering distribution [105]. Variational inference aims to find the best approximation to the true posterior within the chosen variational family by minimizing the *Kullback-Leibler* (KL) divergence between the true and approximate posterior [106], while variational sequential Monte Carlo seeks to minimize the KL divergence on an extended sampling space [104].

However, the proposal distribution is only used as an auxiliary tool during inference and is discarded during test time. For many tasks [81, 82, 83], the quantity of interest is the *Predictive Log-Likelihood* (PLL)

$$\begin{aligned} \text{PLL}(y_1, \ldots, y_T|\mathcal{I}, w) &= \sum_{t=1}^{T} \log p(y_t|\mathcal{I}, w) \\ &= \sum_{t=1}^{T} \log \int p(x_0|\mathcal{I}, w_h) p(x_t|x_0, \mathcal{I}, w_f, w_r) p(y_t|x_t, w_g, w_s) dx_0, x_t, \end{aligned} \tag{4.10}$$

where the transition kernel $p(x_t|x_0, \mathcal{I}, w_f, w_r)$ is defined in Eq. 2.4. Due to the same reasons as discussed in Sec. 3.5.4, we opt for directly optimizing the predictive log-likelihood

$$\underset{w}{\text{argmax}}\, \mathbb{E}\left[\text{PLL}(y_1, \dots, y_T|\mathcal{I}, w)\right], \tag{4.11}$$

with respect to the parameters $w$. In short, the predictive log-likelihood propagates the latent state forward in time without receiving feedback from the observations, mimicking the behavior during test time. In contrast to the predictive log-likelihood objective in the previous chapter (Eq. 3.43), we now address a latent dynamical system and condition the predictive log-likelihood on the context variable $\mathcal{I}$. As a consequence, we need not only to marginalize out all intermediate time steps but also all of the latent states.

For the sake of brevity, we omit the dependence on the weights $w$ in the remainder of this chapter. This omission is motivated by the fact that the weight dependence was solely relevant for defining our loss function.

### 4.2.2. Approximating the Predictive Log-Likelihood

We are interested in the predictive log-likelihood $\text{PLL}(y_1, \dots, y_T|\mathcal{I})$, which describes the predictive log-likelihood of all traffic participants up to time step $T$. In order to calculate it, we need to solve the nested set of integrals given in Eq. 4.10. The BMM algorithm allows us to approximate the distribution $p(x_t|\mathcal{I}) = \int p(x_t|x_0, \mathcal{I})p(x_0|\mathcal{I})dx_0$ in case that the initial state $x_0$ has a Gaussian distribution. However, in our model formulation, the initial latent state $x_0$ follows a GMM to allow for multimodality. In order to account for that, we approximate the marginal latent distribution $p(x_t|\mathcal{I})$ as another GMM with as many components $V$ as the distribution of initial state

$$p(x_t|\mathcal{I}) \approx \sum_{v=1}^{V} \pi_v(\mathcal{I}) p(x_{t,v}|\mathcal{I}). \tag{4.12}$$

To approximate each mixture component $p(x_{t,v}|\mathcal{I})$, the BMM algorithm is applied individually to each component

$$p(x_{t,v}|\mathcal{I}) \approx \mathcal{N}(\mu^x_{t,v}(\mathcal{I}), \Sigma^x_{t,v}(\mathcal{I})), \tag{4.13}$$

where $\mu^x_{t,v}(\mathcal{I}) \in \mathbb{R}^{MD_x}$ and $\Sigma^x_{t,v}(\mathcal{I}) \in \mathbb{R}^{MD_x \times MD_x}$ are the mean and covariance of the $v$-th mixture component at the $t$-th time step in latent space. Assuming a GMM at the initial state allows us to efficiently obtain multimodal predictions while being computationally efficient:

We obtain multimodal distributions by marginalizing over the mixture components, and we compute each component efficiently by applying the BMM algorithm. Finally, we approximate the predictive distribution $p(y_t|\mathcal{I})$ as a GMM

$$\begin{aligned} p(y_t|\mathcal{I}) &= \int p(y_t|g(x_t), \text{diag}(s(x_t)))p(x_t|\mathcal{I})dx_t \\ &\approx \sum_{v=1}^{V} \pi_v(\mathcal{I})\mathcal{N}(\mu_{t,v}^y(\mathcal{I}), \Sigma_{t,v}^y(\mathcal{I})), \end{aligned} \tag{4.14}$$

where $\mu_{t,v}^y(\mathcal{I}) \in \mathbb{R}^{MD_y}$ and $\Sigma_{t,v}^y(\mathcal{I}) \in \mathbb{R}^{MD_y \times MD_y}$ are the mean and covariance of the $v$-th mixture component at the $t$-th time step in the observed space. These two moments are available as

$$\mu_{t,v}^y(\mathcal{I}) = \mathbb{E}[g(x_{t,v})], \tag{4.15}$$

$$\Sigma_{t,v}^y(\mathcal{I}) = \text{Cov}[g(x_{t,v})] + \text{diag}\left(\mathbb{E}[s(x_{t,v})]\right), \tag{4.16}$$

which is a direct outcome of the law of the unconscious statistician. We present the pseudocode for computing $p(y_t|\mathcal{I})$ using our method in Algorithm 3.

**Algorithm 3** Bidimensional Moment Matching in Latent Space (BMMLS)

**Inputs:** $f(x_t, \mathcal{I})$ ▷ Mean update
$r(x_t, \mathcal{I})$ ▷ Covariance update
$g(x_t)$ ▷ Mean emission
$s(x_t)$ ▷ Covariance emission
$h(\mathcal{I})$ ▷ Embedding model
$t$ ▷ Horizon
$\mathcal{I}$ ▷ Context variable
**Outputs:** Approximate predictive distribution $p(y_t|\mathcal{I})$
$\mu_{0,1:V}(\mathcal{I}), \Sigma_{0,1:V}(\mathcal{I}), \pi_v(\mathcal{I}) = h(\mathcal{I})$ ▷ GMM at initial step, Eq. 4.8
**for** mixture component $v \in \{1, \cdots, V\}$ **do**
  Solve with BMM (Alg. 1)
  **for** time step $t' \in \{0, \cdots, t-1\}$ **do** ▷ Horizontal Moment Matching
    $\mu_{t'+1,v}^x \leftarrow \mu_{t'}(\mathcal{I}) + \mathbb{E}[f(x_{t'}, \mathcal{I})]$ ▷ Eq. 3.3
    $\Sigma_{t'+1,v}^x \leftarrow \Sigma_{t'}(\mathcal{I}) + \text{Cov}[f(x_{t'}, \mathcal{I})] + \text{Cov}[x_{t'}, f(x_{t'}, \mathcal{I}),] + \text{Cov}[x_{t'}, f(x_{t'}, \mathcal{I})]^\top + \text{diag}\left(\mathbb{E}[r(x_{t'}, \mathcal{I})]\right)$ ▷ Eq. 3.5
  **end for**
  $\mu_{t,v}^y(\mathcal{I}) \leftarrow \mathbb{E}[g(x_{t,v})]$ ▷ Eq. 4.15
  $\Sigma_{t,v}^y(\mathcal{I}) \leftarrow \text{Cov}[g(x_{t,v})] + \text{diag}\left(\mathbb{E}[s(x_{t,v})]\right)$ ▷ Eq. 4.16
**end for**
**return** $\sum_{v=1}^{V} \pi_v(\mathcal{I})\mathcal{N}(\mu_{t,v}^y(\mathcal{I}), \Sigma_{t,v}^y(\mathcal{I}))$

We note that this approximation becomes exact for locally linear Gaussian transition and emission models, as stated in Thm. 1.

**Theorem 1.** *The predictive distribution $p(y_t|\mathcal{I})$ is analytically computed as*

$$p(y_t|\mathcal{I}) = \sum_{v=1}^{V} \pi_v(\mathcal{I}) \mathcal{N}(\mu_{t,v}^y(\mathcal{I}), \Sigma_{t,v}^y(\mathcal{I})),$$

*for a GDSSM with the below generative model*

$$\begin{aligned}
v &\sim \textit{Cat}([\pi_1(\mathcal{I}), \dots, \pi_V(\mathcal{I})]), \\
x_0 &\sim \mathcal{N}(\mu_{0,v}(\mathcal{I}), \textit{diag}(\Sigma_{0,v}(\mathcal{I}))), \\
x_t &\sim \mathcal{N}\left(x_t | x_{t-1} + f(t, v, \mathcal{I}) x_{t-1}, \textit{diag}\left(r(t, v, \mathcal{I})\right)\right), \\
y_t &\sim \mathcal{N}\left(y_t | g(t, v, \mathcal{I}) x_t, \textit{diag}\left(s(t, v, \mathcal{I})\right)\right),
\end{aligned}$$

*where $f(t, v, \mathcal{I}), r(t, v, \mathcal{I}), g(t, v, \mathcal{I}), s(t, v, \mathcal{I})$ are time $t$, component $v$, and context $\mathcal{I}$ depending matrices with appropriate dimensionality.*

*Proof.* The proof is straightforward as the output moments of the transition and emission model are analytically tractable. We provide details in App. A.1. □

For general transition and emission models, the quality of this scheme depends on two factors: (i) how well the transition and emission model can be approximated in a locally linear fashion and (ii) if the covariance matrix $\Sigma_{t,v}^x$ is small enough over the time horizon $t$. Our experiments show that the approximation works well and further illustrate its behavior on a small toy dataset in Fig. 4.2. Finally, Gaussian mixture models have also been successfully used for filtering problems [107] where moment matching is performed for each component individually in the prediction step.

### 4.2.3. Output Moments of Graph Neural Network Layers

In order to use the BMM framework, we need to be able to calculate the first two output moments as well as the expected Jacobian of graph neural network layers. In the following, we derive the analytic expression of the output moments and expected Jacobian for the common graph neural net layers: (i) node-wise affine transformation and (ii) mean aggregation. Output moments for the standard affine transformation and ReLU activation are provided in Sec. 3.3.

Let $x_t^{l,m} \in \mathbb{R}^{D_{x,l}}$ be the node features at layer $l$ of node $m$ at time step $t$ and $x_t^l = \{x_t^{l,m}\}_{m=1}^M$ the set of all node features with $x_t^l \in \mathbb{R}^{MD_{x,l}}$. For the sake of brevity, we omit the index

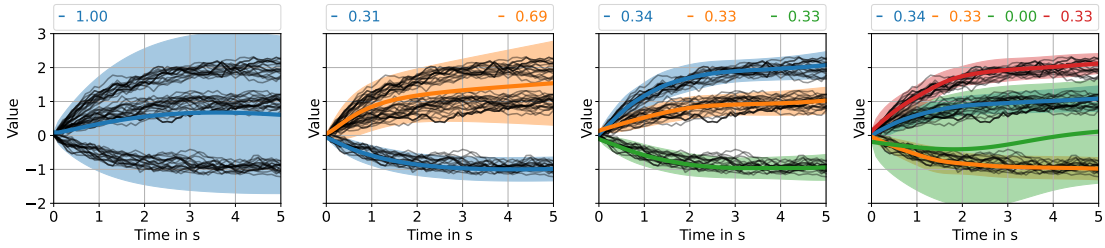



Figure 4.2.: Predictions after training on a one-dimensional, multimodal toy problem. Solid black lines represent the ground truth dynamics with three modes of equal probability. For our model, we report for each predicted mode the mean, 95% confidence interval, and mixture weight. We set the dimension of the latent space to $D_x = 3$ and vary (from left to right) the number of modes $V$ in the model from 1 to 4. For $V < 3$, the number of modes is too small, and we observe mode-covering behavior. For $V \geq 3$, our model can recover the ground truth dynamics. If the number of components is too high ($V = 4$), the mixture weights of redundant components are set to zero.

of the mixture component $v$. In the previous chapter, we used a normal distribution to approximate the distribution at each layer, i.e., $p(x_t^l) \approx \mathcal{N}(\mathbb{E}[x_t^l], \mathrm{Cov}[x_t^l])$. In this notation, the input $x_t^m$ is represented as $x_t^{0,m}$. The update and aggregation functions at layer $l$ are denoted as $u_l : \mathbb{R}^{MD_{x,l-1}} \to \mathbb{R}^{MD_{x,l}}$ and $u_{l,\text{AGG}} : \mathbb{R}^{MD_{x,l}} \times \mathbb{R}^{\mathcal{I}} \to \mathbb{R}^{MD_{x,l}}$, respectively. We denote the mean and covariance of a graph with $M$ nodes at layer $l$ and time step $t$ as

$$\mathbb{E}[x_t^l] = \begin{bmatrix} \mathbb{E}[x_t^{l,1}] \\ \vdots \\ \mathbb{E}[x_t^{l,M}] \end{bmatrix}, \quad (4.17) \qquad \mathrm{Cov}[x_t^l] = \begin{bmatrix} \mathrm{Cov}[x_t^{l,1}] & \dots & \mathrm{Cov}[x_t^{l,1}, x_t^{l,M}] \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}[x_t^{l,M}, x_t^{l,1}] & \dots & \mathrm{Cov}[x_t^{l,M}, x_t^{l,M}] \end{bmatrix}, \quad (4.18)$$

where $\mathbb{E}[x_t^{l,m}] \in \mathbb{R}^{D_{x,l}}$ and $\mathrm{Cov}[x_t^{l,m}, x_t^{l,m'}] \in \mathbb{R}^{D_{x,l} \times D_{x,l}}$. A typical GNN architecture (see Sec. 2.4) consists of alternating aggregation steps, in which information from neighbors is collected, and update steps, in which the features of the node are updated. For the aggregation step, we derive the output moments for the commonly used mean aggregation operation. For the update step, we assume that the neural network is built as a sequence of affine transformations and nonlinearities. The output moments of nonlinear activations are applied independently across agents. As a consequence, their rules do not change when used in the GNN setting, and we can reuse the rules derived in Sec. 3.3.2. Hence it

remains open to derive the output moments for affine transformations, as they are used in GNNs. We note that the mean aggregation and the node-wise affine transformation are special cases of a standard affine layer (see. Sec. 3.3)

### Node-Wise Affine Transformation

The node-wise affine transformation is applied simultaneously to each node $m$ at layer $l$ using a transformation matrix $A^{l+1} \in \mathbb{R}^{D_{x,l+1} \times D_{x,l}}$ and a bias $b^{l+1} \in \mathbb{R}^{D_{x,l+1}}$. The node-wise affine transformation can be understood as a standard affine transformation that acts on the set of all nodes $x_t^l$ as

$$\begin{aligned} x_t^{l+1} &= \begin{bmatrix} A^{l+1} & 0 & \dots & 0 \\ 0 & A^{l+1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A^{l+1} \end{bmatrix} \begin{bmatrix} x_t^{l,1} \\ x_t^{l,2} \\ \vdots \\ x_t^{l,M} \end{bmatrix} + \begin{bmatrix} b^{l+1} \\ b^{l+1} \\ \vdots \\ b^{l+1} \end{bmatrix} \\ &= \underbrace{(I_M \otimes A^{l+1})}_{\hat{A}^{l+1}} x_t^l + \underbrace{(\mathbf{1}_M \otimes b^{l+1})}_{\hat{b}^{l+1}}, \end{aligned} \tag{4.19}$$

where $I_M$ is the identity matrix with shape $M \times M$, $\mathbf{1}_M$ is a vector of ones with shape $M \times 1$, and $\otimes$ is the Kronecker product. The moments of node-wise affine transformation are analytically available as

$$\mathbb{E}[x_t^{l+1}] = \hat{A}^{l+1} \mathbb{E}[x_t^l] + \hat{b}^{l+1}, \tag{4.20}$$

$$\mathrm{Cov}[x_t^{l+1}] = \hat{A}^{l+1} \mathrm{Cov}[x_t^l] (\hat{A}^{l+1})^\top. \tag{4.21}$$

Similarly, as for the standard affine transformation, the expected Jacobian of node-wise affine transformation is analytically available as

$$\mathbb{E}[\nabla_{x_t^l} u_{l+1}(x_t^l)] = \hat{A}^{l+1}. \tag{4.22}$$

### Mean Aggregation

Let the message to node $m$ at layer $l$ and time step $t$ be $x_t^{l,\mathcal{N}_m}$ that we approximate as a Gaussian $x_t^{l,\mathcal{N}_m} \sim \mathcal{N}(\mathbb{E}[x_t^{l,\mathcal{N}_m}], \mathrm{Cov}[x_t^{l,\mathcal{N}_m}])$. A commonly used aggregation operation is

the mean aggregator

$$x_t^{l,\mathcal{N}_m} = \frac{1}{|\mathcal{N}_m|} \sum_{m' \in \mathcal{N}_m} x_t^{l,m'}. \tag{4.23}$$

Let $x_t^{l,\mathcal{N}}$ be the set of all messages, i.e. $x_t^{l,\mathcal{N}} = \{x_t^{l,\mathcal{N}_m}\}_{m=1}^M$, that we also approximate as a Gaussian $x_t^{l,\mathcal{N}} \sim \mathcal{N}(\mathbb{E}[x_t^{l,\mathcal{N}}], \mathrm{Cov}[x_t^{l,\mathcal{N}}])$. The mean aggregation can be equivalently written as a linear transformation

$$x_t^{l,\mathcal{N}} = u_{l,\mathrm{AGG}}(x_t^l, \mathcal{I}) = \underbrace{(\mathcal{A} \otimes I_{D_{x,l}})}_{\hat{\mathcal{A}}^l} x_t^l. \tag{4.24}$$

Above, $\mathcal{A} \in \mathbb{R}^{M \times M}$ denotes the row normalized adjacency matrix, which summarizes the edge information $\mathcal{E}$ in matrix format and $I_{D_{x,l}}$ denotes the identity matrix with dimensionality $D_{x,l} \times D_{x,l}$. The Kronecker product expands the adjacency matrix accordingly to the $D_{x,l}$-dimensional node features. Hence, the mean aggregation corresponds to a linear transformation with a weight matrix consisting of $M \times M$ blocks, where each block is a diagonal matrix of shape $D_{x,l} \times D_{x,l}$. Its moments are analytically available as

$$\mathbb{E}[x_t^{l,\mathcal{N}}, \mathcal{I}] = \hat{\mathcal{A}}^l \mathbb{E}[x_t^l], \tag{4.25}$$

$$\mathrm{Cov}[x_t^{l,\mathcal{N}}, \mathcal{I}] = \hat{\mathcal{A}}^l \mathrm{Cov}[x_t^l] (\hat{\mathcal{A}}^l)^\top. \tag{4.26}$$

The expected Jacobian is available as

$$\mathbb{E}\left[\nabla_{x_t^l} u_{l,\mathrm{AGG}}(x_t^l, \mathcal{I})\right] = \hat{\mathcal{A}}^l. \tag{4.27}$$

### 4.2.4. Sparse Covariance Approximation

For settings with a large number of agents $M$ or with a high-dimensional state $x_t^{l,m}$, the application of the BMM algorithm can become computationally expensive. In the following, we review the computational complexity of the BMM algorithm for GDSSMs. We assume that the GNN model consists of a mean aggregation step followed by multiple node-wise affine transformations and nonlinearities, which is the same architecture that we employ later on in our experiments. The mean aggregation is done for each of the $D_x$ latent states independently, and we denote the maximum hidden layer width as $H$.

Computing the nonlinearities is cheap as the operation acts elementwise, and their effect on the runtime can be neglected during this analysis. The other two operations (see Sec.

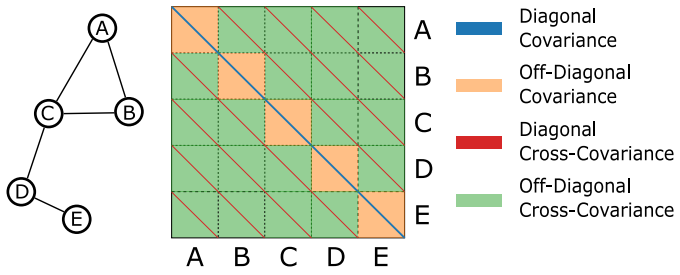



Figure 4.3.: Groupings of the covariance matrix for graph structured data. Consider the example graph with $M = 5$ agents, which is depicted in the left panel. Each agent has dimensionality $D_x$. In the middle panel, we show the covariance matrix between all agents, which consists of $5 \times 5$ blocks, where each block has dimensionality $D_x \times D_x$.

4.2.3) can be described by affine operations for which the weight matrices are heavily structured; the mean aggregation step corresponds to a weight matrix consisting of $M \times M$ diagonal blocks, the node-wise affine transformation to a block-diagonal weight matrix with $M$ blocks of shape $H \times H$. Propagation of the full covariance matrix through a neural network (forward cost) has the computational complexity of $\mathcal{O}(M^2H^3 + M^2H^2D_x + M^2HD_x^2 + M^3D_x^2)$, where the first term is due to the cost of the $H \times H$-dimensional node-wise affine transformations in the hidden layers, the second and third term are due to the cost of the $H \times D_x$-dimensional node-wise affine transformation after the aggregation operation, and the fourth term is due to the aggregation operation.

The computational cost of the expected Jacobian is $\mathcal{O}(MH^3 + MH^2D_x + M^2D_x^2)$, and we give its derivation in the following. Let the expected Jacobian of the aggregation operation be $\mathbb{E}[J_t^{\text{AGG}}]$ and the product of the expected Jacobians of the subsequent neural net layers $\mathbb{E}[J_t^{\text{NET}}] = \prod \mathbb{E}[\nabla_{x_t^l} u_{l+1}(x_t^l)]$. The expected Jacobian of the neural net layers $\mathbb{E}[J_t^{\text{NET}}]$ is block-diagonal with $M$ blocks of shape $D_x \times D_x$ and its computation takes $\mathcal{O}(MH^3 + MH^2D_x)$ time. Multiplying $\mathbb{E}[J_t^{\text{AGG}}]$ with $\mathbb{E}[J_t^{\text{NET}}]$ results in a fully populated matrix where each entry can be computed by a single dot product due to the structure of its factors, and its computation contributes with $\mathcal{O}(M^2D_x^2)$ to the runtime.

Consequently, the total cost is dominated by the forward cost, $\mathcal{O}(M^2H^3 + M^2H^2D_x + M^2HD_x^2 + M^3D_x^2)$, when using the full covariance matrix. For reference, taking a Monte Carlo approach has the computational complexity $\mathcal{O}(SMH^2 + SMHD_x + SM^2D_x)$ where $S$ is the number of Monte Carlo samples. Since the computational cost quickly becomes intractable due to the cubic dependence with respect to the number of agents in the

forward pass, we next propose different sparse approximations to the covariance matrix. Independent of the chosen approximation, the cost of the expected Jacobian remains unchanged, as it does not depend on the covariance matrix.

*(i) Full*: Model the full covariance matrix.
Forward cost: $\mathcal{O}(M^2H^3 + M^2H^2D_x + M^2HD_x^2 + M^3D_x^2)$.
Total cost: $\mathcal{O}(M^2H^3 + M^2H^2D_x + M^2HD_x^2 + M^3D_x^2)$.

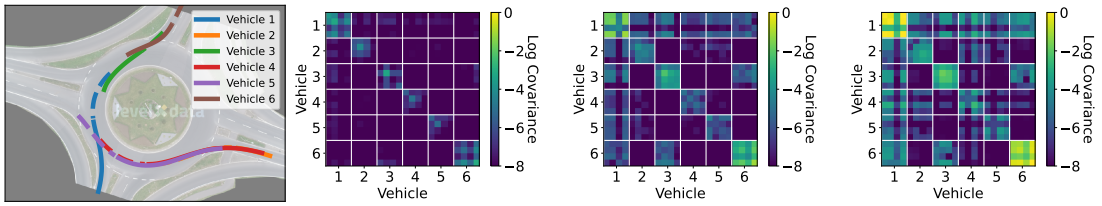
*(ii) Main Diagonal*: Keep the diagonal entries in the covariance blocks, which corresponds to the blue line in Fig. 4.3.
Forward cost: $\mathcal{O}(MH^2 + MHD_x + M^2D_x)$.
Total cost: $\mathcal{O}(MH^3 + MH^2D_x + M^2D_x^2)$.

*(iii) Main Blocks*: Keep the block-diagonal blocks in the covariance matrix, which corresponds to the orange blocks and blue lines in Fig. 4.3.
Forward cost: $\mathcal{O}(MH^3 + MH^2D_x + MHD_x^2 + M^2D_x^2)$.
Total cost: $\mathcal{O}(MH^3 + MH^2D_x + MHD_x^2 + M^2D_x^2)$.

*(iv) All Diagonals*: Structure the covariance in blocks of shape $M \times M$ and keep the diagonal entries in each block, which corresponds to the blue and red lines in Fig. 4.3.
Forward cost: $\mathcal{O}(M^2H^2 + M^2HD_x + M^3D_x)$.
Total cost: $\mathcal{O}(M^2H^2 + M^2HD_x + M^3D_x + MH^3 + MH^2D_x + M^2D_x^2)$.

Note that setting the off-diagonal blocks to zero, as done in Main Blocks and Main Diagonal, corresponds to an independence assumption between the agents and leads to a runtime reduction from $O(M^3)$ to $O(M^2)$.

Finally, it is important to note that the covariance matrix only has the same structure as the graph after the first time step. Agents that are not connected via an edge can still have a non-zero cross-covariance at time step $t$, provided that they are connected by a path that is at most $t$ steps long. In our applications, this leads to non-sparse covariances after a few time steps since the number of agents is small compared to the time horizon. We present the covariance matrix at three different time steps for an exemplary scene in Fig. 4.4. The covariance matrix has an approximately diagonal shape for a short prediction horizon of one second. As the prediction horizon increases, the covariance matrix becomes more complex and is no longer dominated by its diagonal entries. If required, we could further increase the information spread across agents by using an architecture with multiple aggregation steps within the GNN module.




(a) Example scene. (b) Covariance at 1 s. (c) Covariance at 3 s. (d) Covariance at 5 s.

Figure 4.4.: We visualize an example scene from the rounD dataset [94] and the covariance matrix in the latent space at different time steps for the case of a unimodal initial distribution. In the left plot, dashed lines represent the observed history, and solid lines represent the true future trajectories.

## 4.3. Experiments

We provide experiments on two challenging autonomous driving datasets. The first experiment (Sec. 4.3.2) conducts an ablation study, while the second experiment (Sec. 4.3.3) benchmarks our model against state-of-the-art methods. We provide a runtime analysis and a benchmark of our proposed sparse covariance approximations in Sec. 4.3.4. In Sec. 4.3.5, we analyze the generalization capabilities of our model by benchmarking it on novel and unseen traffic environments. We provide details about the evaluation procedure in Sec. 4.3.1.

We provide details about the training in App. A.2, and the architecture for the embedding, transition, and emission models in App. A.5.

### 4.3.1. Evaluation Criteria

As evaluation metrics, we use the *Root Mean Squared Error* (RMSE) and the *Negative Log-Likelihood* (NLL) at fixed time intervals (1s, 2s, 3s, 4s, 5s). We define the RMSE in the multi-agent setting as

$$\mathrm{RMSE}(t) = \sqrt{\frac{1}{N}\sum_{n}^{N}\sum_{m}^{M(n)}\frac{(y_{t,n}^{m} - \mathbb{E}[\hat{y}_{t,n}^{m}])^{T}(y_{t,n}^{m} - \mathbb{E}[\hat{y}_{t,n}^{m}])}{M(n)}}, \tag{4.28}$$

where $N$ is the number of snippets in the dataset, $M(n)$ is the number of agents in the $n$-th snippet, $y_{t,n}^{m}$ is the true location of the $m$-th agent at time point $t$ of the $n$-th snippet,

and $\hat{y}_{t,n}^m$ the corresponding predicted location. We assume that the probabilistic predictor performs model averaging before making its final prediction. This score only evaluates the goodness of the first moment and does not take any information about higher moments into account.

In contrast, the NLL is a strictly proper scoring rule [108] that we define in the multi-agent setting as

$$\mathrm{NLL}(t) = -\frac{1}{N}\sum_{n}^{N}\sum_{m}^{M(n)}\frac{\log p(y_{t,n}^m|\mathcal{I})}{M(n)}. \tag{4.29}$$

For deterministic GDSSMs, we approximate $p(y_{t,n}^m|\mathcal{I})$ directly with the method that we introduced in Sec. 4.2. For GDSSMs that do not follow an assumed density approach, we approximate $p(y_{t,n}^m|\mathcal{I})$ via Monte Carlo integration.

### 4.3.2. rounD

The rounD dataset[1] [94] consists of vehicle trajectories recorded at different roundabouts in Germany. As the roundabouts involve many interactions among vehicles, we expect the predictive distributions to be multimodal and highly complex. We use the recordings from the roundabout in Neuweiler near Aachen for training and testing purposes. The dataset consists of 13,129 tracked objects recorded at 25 Hz in 22 sessions, amounting to a total recording time of 6.6 hours. We remove pedestrians, bicycles, as well as parked vehicles from the dataset, as their influence on the vehicle behavior patterns in the roundabouts is negligible. After dataset curation, we are left with 12,715 tracked objects. We downsample the recordings by a factor of five and construct a dataset consisting of eight-second-long segments with 50 % overlap, resulting in 5405 snippets. We use the first three seconds as the track history, which is part of the context variable $\mathcal{I}$, and the following five seconds as the prediction horizon. The first 18 recording sessions, corresponding to 4,314 snippets, are used for training and validation. The final four recording sessions, corresponding to 1,091 snippets, are used for testing.

We build the connection graph by connecting vehicles with an Euclidean distance of less than 30 meters. The dataset has been recorded with drones, and we keep the original global coordinate system.

[1] `https://www.round-dataset.com`

### Baselines

We compare our method with multiple baseline models. For each baseline, we remove one key assumption of our model. We cite papers that employ similar ideas as appropriate. We did not reimplement these works but performed an ablation study in which we replaced specific components of our model in the interest of a fair comparison. Furthermore, we compare different training strategies as an alternative to maximizing the PLL and analyze the multimodality of our model.

*(i) Model*:

*(i.i)* *GDSSM*: Our model as proposed in Sec. 4.2.

*(i.ii)* *Non-Recurrent GNN* (e.g. [109, 80]): This architecture receives the context variable $\mathcal{I}$, performs one round of message passing, and subsequently outputs one normal distribution for each of the next five seconds without using a recurrent architecture.

*(i.iii)* *No Latent Noise* (e.g. [93]): We remove the noise from the latent dynamics (Eq. 4.2) while keeping the emission model unchanged. The uncertainty can no longer be propagated forward in time as the emission model acts independently for each time point.

*(i.iv)* *Linearity* (e.g. [110]): We remove all nonlinearities from the latent dynamics. We keep the nonlinear neural networks to parameterize the emission model in order to map the dynamics into a latent space in which the system can be linearly approximated.

*(i.v)* *No Interactions* (e.g. [26]): Our model with a diagonal adjacency matrix, i.e., we remove all edges from the graphs. This model neglects interactions between traffic participants.

*(ii) Modes*:

*(ii.i)* $V$ Modes: We vary the number of mixture components $V$ in the GMM prior (Eq. 4.6) in order to test the effect of multimodality.

*(ii.ii)* $\infty$ Modes: This alternative does not follow an assumed density approach, i.e., the marginal latent distribution $p(x_t|\mathcal{I})$ is not approximated as a GMM with a bounded number of modes. Instead, $p(x_t|\mathcal{I})$ is approximated by simulating a large number of trajectories, where each trajectory can follow a different mode [31]. We set the number of particles to 100.

*(iii) Objectives*:

*(iii.i)* *PLL/Det.*: We train the model on the predictive log-likelihood (Eq. 4.10) and use our deterministic approximations for GDSSM as proposed in Sec. 4.2.

*(iii.ii)* *PLL/MC*: We train the model on the predictive log-likelihood (Eq. 4.10) and take an assumed density approach by approximating the predictive distribution (Eq. 4.14) as a GMM. The intractable integrals are solved via *Monte Carlo* (MC) integration. One forward pass through our model amounts to approximately the computational cost of 12 Monte Carlo simulations. For a fair comparison, we use 16 particles during training, which is more costly than training with our proposed moment propagation algorithm, and test with 100 particles. We use the same amount of particles for all MC based methods.

*(iii.iii)* *ELBO/MC* (e.g. [26]): The model is trained by maximizing the *Evidence Lower Bound* (ELBO). We give a description of this loss function in App. A.3. The approximate posterior is a filtering distribution that models the latent state as a normal distribution. We propagate the latent state forward in time direction via Monte Carlo sampling.

*(iii.iv)* *MCO/MC* (e.g. [103]): The model is trained by maximizing the *Monte Carlo Objective* (MCO), which combines particle filters with variational inference. We give a description of this loss function in App. A.3. The proposal distribution is a filtering distribution, and we propagate the particles in time direction via Monte Carlo sampling.

### Results

We provide benchmark results of all methods in Tab. 4.1. First, we compare our deterministic training and testing scheme (GDSSM PLL/Det.) against its Monte Carlo alternative (GDSSM PLL/MC). Though Monte Carlo based training is more costly than training with BMM, the Monte Carlo results are significantly outperformed by our method. Our results indicate that our deterministic approach leads to more effective approximations compared to Monte Carlo sampling despite the approximation error we obtain by our deterministic moment matching scheme. One potential explanation for our finding is that the Monte Carlo approaches suffer under a high variance since the latent space for multi-agent settings grows linearly with the number of agents.

When changing the training objective from the PLL to ELBO/MC or MCO/MC, we observe that the performance is comparable to PLL/MC for the prediction horizon of 1 second. For

longer prediction horizons, the performance of the models that are trained on ELBO/MC or MCO/MC degrades quicker compared to the model trained on PLL. We believe that this behavior can be explained by the mismatch between training and testing objectives. When training on MCO/MC or ELBO/MC, the model obtains feedback from the observations via the proposal distribution, while during testing, it has to produce multi-step ahead predictions without receiving any feedback from the observations.

Next, we study if our method can capture multimodality by increasing the number of components in the GMM prior. It is worth noting that GMMs can approximate arbitrarily complex distributions when the number of components is chosen high enough. In our experiments, we observe that an increase of components significantly decreases the NLL, making the GMM prior a vital ingredient of our method and suggesting that the true predictive distribution is highly multi-modal. If the number of components is chosen too small, our model adjusts its uncertainty predictions accordingly. For example, we observe in Fig. 4.1 the uncertainty of the orange agent increases as the vehicle is close to the exit of the roundabout. The high predictive uncertainty can be explained by two potential future outcomes: the orange vehicle can leave the roundabout or stay inside. Consequently, our model learns to compensate if the number of components is picked too low, which in turn allows us to trade accuracy for computational runtime. When using tailored implementations, one can easily scale up to a larger number of components since their computations can be parallelized without any hurdles. We further note that the RMSE is less affected by the choice of the number of modes and stays within two standard errors. Since the RMSE value measures the error between the true value and the expected value of the predictive distribution, this result suggests that the expected value can already be modeled well by predictive distributions with very few modes (actually, one mode seems to suffice for this).

We proceed to compare our model to a simpler alternative in which the dynamics are assumed to be linear, which allows calculating the moments of the transition model exactly and in closed form [55]. In contrast, our approach GDSSM, approximates the nonlinear dynamics in a local linear way by using deterministic moment matching results. We find that our approach achieves lower RMSE and NLL, which can most likely be attributed to the higher modeling flexibility of our proposed model class.

Our ablation study further shows that discarding latent noise and removing interactions between traffic participants results in higher RMSE and NLL. Compared to modeling the dynamics in a non-recurrent manner, our model achieves a similar RMSE and outperforms its competitor in terms of NLL.

Table 4.1.: rounD results. We report average and standard error over 10 runs.

| | | Non Recur. GNN 1 Mode PLL/Det. | GDSSM ∞ Modes ELBO/MC | GDSSM ∞ Modes MCO/MC | GDSSM No Lat. Noise 1 Mode PLL/Det. | GDSSM Linear 1 Mode PLL/Det. | GDSSM No Interaction 1 Mode PLL/Det. | GDSSM 1 Mode PLL/MC | GDSSM 1 Mode PLL/Det. | GDSSM 2 Modes PLL/Det. | GDSSM 3 Modes PLL/Det. | GDSSM 4 Modes PLL/Det. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RMSE | 1 s | **0.72(0.02)** | 1.53(0.04) | 0.95(0.04) | 1.22(0.08) | 0.88(0.02) | 0.91(0.03) | 0.90(0.02) | 0.79(0.02) | 0.75(0.02) | 0.74(0.03) | **0.73(0.03)** |
| | 2 s | 1.90(0.02) | 3.42(0.09) | 2.43(0.09) | 2.33(0.08) | 2.12(0.03) | 2.10(0.04) | 2.09(0.02) | 1.87(0.03) | **1.84(0.03)** | **1.82(0.03)** | **1.80(0.04)** |
| | 3 s | 3.54(0.03) | 5.99(0.14) | 4.55(0.16) | 3.88(0.07) | 3.88(0.05) | 3.69(0.06) | 3.60(0.04) | **3.36(0.03)** | **3.35(0.03)** | **3.35(0.03)** | **3.33(0.04)** |
| | 4 s | 5.26(0.04) | 9.24(0.26) | 7.62(0.27) | 5.58(0.08) | 6.13(0.09) | 5.39(0.13) | 5.37(0.05) | **5.08(0.04)** | 5.15(0.09) | 5.14(0.08) | **5.06(0.04)** |
| | 5 s | **7.20(0.05)** | 11.59(0.41) | 11.24(0.41) | 7.65(0.10) | 8.83(0.13) | 7.56(0.23) | 7.65(0.06) | **7.24(0.05)** | 7.34(0.12) | 7.35(0.19) | 7.29(0.09) |
| NLL | 1 s | 1.90(0.01) | 2.62(0.04) | 2.79(0.06) | 2.96(0.08) | 1.95(0.08) | 1.67(0.07) | 2.82(0.03) | 1.48(0.05) | 1.34(0.08) | 1.36(0.04) | **1.21(0.05)** |
| | 2 s | 3.25(0.02) | 4.45(0.07) | 4.21(0.03) | 3.85(0.10) | 3.93(0.13) | 3.37(0.08) | 4.11(0.02) | 2.91(0.03) | 2.93(0.07) | 2.93(0.06) | **2.79(0.09)** |
| | 3 s | 4.40(0.02) | 5.64(0.09) | 5.21(0.03) | 5.05(0.13) | 5.11(0.19) | 4.34(0.08) | 4.67(0.09) | **3.87(0.02)** | 4.01(0.07) | **3.77(0.10)** | **3.83(0.08)** |
| | 4 s | 5.13(0.02) | 6.59(0.13) | 6.01(0.03) | 6.17(0.16) | 6.01(0.22) | 4.93(0.09) | 5.01(0.07) | 4.46(0.03) | 4.52(0.11) | **4.21(0.15)** | **4.34(0.18)** |
| | 5 s | 5.71(0.02) | 6.98(0.14) | 6.73(0.04) | 7.24(0.20) | 6.80(0.22) | 5.51(0.10) | 5.41(0.05) | 5.05(0.04) | 4.82(0.24) | 4.59(0.15) | **4.27(0.23)** |

### 4.3.3. NGSIM

The *Next Generation Simulation* (NGSIM) dataset[1] [111] consists of vehicle trajectories recorded at 10 Hz at two different highways, US-101 and I-80, in the United States. The dataset is commonly used for benchmarking traffic forecasting methods and allows us to compare our method against the prior art. We adopt the experimental setup of [112] and use both highway scenarios. As provided in the original publication, we employ a local coordinate system that is centered around an ego-vehicle. We split the scenarios into three 15 minute long time spans resembling mild, moderate, and congested traffic conditions and downsample each trajectory by a factor of two. The test set consists of a fourth of all trajectories randomly sampled from both locations. As in the rounD experiment, we split each trajectory into eight-second-long segments, where the first three seconds are used as the track history and the following five seconds as the prediction horizon.

Similarly, as in [113, 114, 115], we introduce a connection graph based on the lane position of each vehicle. Each vehicle has at most six connections to other vehicles, which are the nearest vehicles in front/behind on the same/left/right lane. The vehicles on the outermost lanes have a maximum of four neighbors, as there are no neighbors to the left or right.

[1] `https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm`

### Baselines

We compare our approach with the following methods:

*(i) Constant Velocity* [116]: This method uses a linear state-space model together with a Kalman filter in order to make predictions. It does not take interactions between agents into account.

*(ii) Convolutional Social* (CS)-LSTM [112]: Interactions between vehicles are modeled by introducing a grid and applying a convolutional layer on top. Dynamics are modeled by a deterministic LSTM, which predicts the mean and the variance of a normal distribution at each time step.

*(iv) Spatio-Temporal* (ST)-LSTM [117]: Interactions are modeled with a spatio-temporal graph structure, i.e., a graph with a position and time depending component. Dynamics are modeled by a deterministic LSTM that predicts the mean and the variance of a normal distribution at each time step. To the best of our knowledge, this is the only GNN based method that also reports NLL.

*(iv) Multiple Futures Prediction* (MFP) [3]: A recurrent model with deterministic transition dynamics. Stochasticity is introduced via the initial state and noisy observations that are fed back into the dynamical model. Interactions are modeled by an attention module.

### Results

We provide benchmark results of our proposed model in Tab. 4.2. Similar to the experiments on the rounD dataset in Sec. 4.3.2, we observe that (i) deterministic training and testing is more efficient than its Monte Carlo based alternative and (ii) increasing the number of modes improves the performance.

Next, we compare our method, using one component in the GMM prior only, with all other unimodal prediction methods. We can observe that our approach outperforms its competitors in terms of NLL. Furthermore, our method gives similar RMSE as other methods and is only outperformed by MFP. However, in contrast to the other methods in the benchmark, MFP reports the best RMSE over 5 Monte Carlo samples, which makes it difficult to draw a fair conclusion.

We subsequently increase the number of modes for our model and for MFP. In terms of NLL, our method achieves superior results when it comes to long-term predictions (3s, 4s, 5s), while MFP performs better for short-term predictions (1s, 2s). Accurate long-term

prediction of the agents in a driving scene is crucial for high-level autonomous driving, as an accurate environment model is a prerequisite for the precise planning of driving controls. For instance, advanced driver-assistance systems usually use time horizons between 3s and 5s for driver warnings and emergency brakes, while autonomous cars aim for a time horizon of 5s or longer in order to ensure safe and comfortable rides [118].

Table 4.2.: NGSIM results. We report average and standard error over 10 runs. For MFP, we report the best RMSE values over 5 Monte Carlo samples.

| | | Const. Vel. 1 Mode | CS-LSTM 1 Mode | ST-LSTM 1 Mode | MFP 1 Mode | MFP 4 Modes | GDSSM 1 Mode PLL/MC | GDSSM 1 Mode PLL/Det. | GDSSM 2 Modes PLL/Det. | GDSSM 3 Modes PLL/Det. | GDSSM 4 Modes PLL/Det. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RMSE | 1 s | 0.75 | 0.61 | 0.51 | **0.54(0.00)** | **0.54(0.00)** | 0.56(0.00) | **0.53(0.01)** | 0.55(0.00) | 0.55(0.01) | 0.57(0.02) |
| | 2 s | 1.81 | 1.27 | 1.21 | **1.16(0.00)** | 1.17(0.00) | 1.27(0.02) | 1.18(0.01) | 1.22(0.01) | 1.23(0.02) | 1.24(0.04) |
| | 3 s | 3.16 | 2.09 | 2.01 | **1.90(0.00)** | 1.91(0.00) | 2.11(0.03) | 1.98(0.02) | 2.02(0.02) | 2.05(0.03) | 2.06(0.04) |
| | 4 s | 4.80 | 3.10 | 3.01 | 2.78(0.00) | **2.75(0.00)** | 3.16(0.04) | 2.99(0.03) | 3.03(0.04) | 3.06(0.05) | 3.05(0.06) |
| | 5 s | 6.70 | 4.37 | 4.31 | 3.83(0.01) | **3.78(0.01)** | 4.47(0.05) | 4.29(0.04) | 4.33(0.07) | 4.33(0.07) | 4.35(0.08) |
| NLL | 1 s | 0.80 | 0.58 | 0.90 | 0.73(0.01) | **-0.65(0.01)** | 0.64(0.04) | 0.19(0.02) | -0.12(0.02) | -0.15(0.03) | -0.16(0.03) |
| | 2 s | 2.30 | 2.14 | 2.41 | 2.33(0.01) | **1.19(0.01)** | 2.10(0.10) | 1.61(0.02) | 1.37(0.02) | 1.35(0.02) | 1.32(0.02) |
| | 3 s | 3.21 | 3.03 | 3.25 | 3.17(0.01) | 2.28(0.01) | 2.92(0.11) | 2.42(0.02) | 2.23(0.02) | 2.23(0.02) | **2.19(0.02)** |
| | 4 s | 3.89 | 3.68 | 3.61 | 3.77(0.01) | 3.06(0.00) | 3.54(0.10) | 3.02(0.02) | 2.88(0.02) | 2.88(0.02) | **2.82(0.02)** |
| | 5 s | 4.44 | 4.22 | 4.36 | 4.26(0.00) | 3.69(0.00) | 4.04(0.10) | 3.50(0.02) | 3.42(0.02) | 3.41(0.02) | **3.36(0.02)** |

### 4.3.4. Covariance Approximations

In this section, we provide a detailed runtime analysis for different covariance approximations and study their impact on the predictive performance.

#### Runtime

We visualize the runtime of different covariance approximations, as well as the runtime of the Monte Carlo alternative in Fig. 4.5 as a function of input dimensionality and number of agents. We use the same NSDE architecture as in our experiments on the rounD and NGSIM dataset. For the Monte Carlo alternative, we visualize the runtime for 16 particles, as we use the same number of particles for training in Sec. 4.3.2 and 4.3.3.

We first confirm that propagation of the full covariance matrix is more costly than any of the proposed approximations. In fact, our sparse covariance approximations can reduce the runtime up to a factor of 100 for systems with a large number of agents and a high input dimensionality. As we derived in Sec. 4.2.4, when using the BMM algorithm with a full covariance matrix or the All Diagonals approximation, the computational cost shows a cubic dependence on the number of agents. In contrast, the Main Diagonal approximation, the Main Blocks approximation, as well as MC based predictions have a quadratic dependence on the number of agents. This makes these two approximations an attractive alternative to MC based predictions, when systems with a high number of agents need to be modeled with a limited computational budget.

For systems with a moderate input dimensionality ($\approx 8$) and number of agents ($\approx 16$), all of our proposed approximations require only up to $5\,\text{ms}$ in order to compute the distribution at the next time point, which corresponds to the cost of 5-10 Monte Carlo simulations.

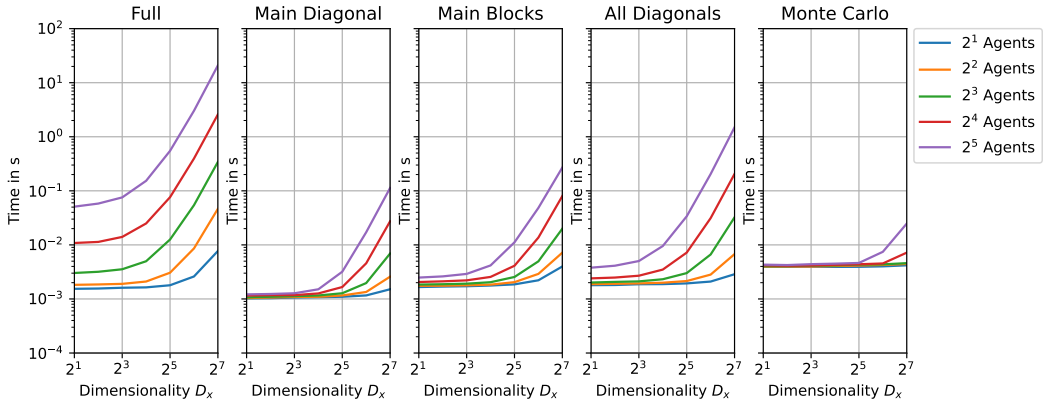



Figure 4.5.: Wallclock time for output moment calculation for the latent dynamics using GNNs with three hidden layers of size 24 for different covariance approximations (from left to the right). For each approximation, we plot the runtime as a function of the input dimensionality $D_x$ and number of agents $M$. The GNNs are initialized at random, and we report the average runtime over 100 repetitions.

### Benchmark

Next, we study the effect of different covariance approximations on the performance. We report the results for the case of a unimodal GDSSM. The results are depicted in Tab. 4.3. We report here our main findings.

First, modeling the full covariance matrix results in the best performance in terms of the lowest RMSE and NLL. Second, the All Diagonals approximation performs the best among the covariance approximations. It achieves comparable RMSE as the full solution but falls slightly behind in NLL when the system is highly interactive (see rounD dataset). In this setting, it outperforms the other two sparse approximations with respect to NLL. This behavior might be explained by the assumptions made in the covariance approximation: it is the only approximation that allows for correlations between agents as its structure only neglects dependencies between latent features.

Third, the differences in performances between the full solution and the different approximations with respect to RMSE lie between one and two standard errors. Modeling the main diagonal only can thus be sufficient for applications with low computational resources and high demand for accuracy by accepting a slight loss in calibration. For these applications, the runtime can be reduced from $O(M^3)$ to $O(M^2)$.

Table 4.3.: Test performance for different covariance approximations on the rounD and NGSIM dataset for the unimodal case. We report average and standard error over 10 runs.

| | | rounD | | | | NGSIM | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Full | Main Diagonal | Main Blocks | All Diagonals | Full | Main Diagonal | Main Blocks | All Diagonals |
| RMSE | 1 s | **0.79(0.02)** | 0.82(0.02) | 0.83(0.04) | **0.78(0.02)** | **0.53(0.01)** | **0.54(0.01)** | 0.55(0.01) | **0.53(0.01)** |
| | 2 s | **1.87(0.02)** | **1.88(0.02)** | **1.89(0.05)** | **1.87(0.02)** | **1.18(0.01)** | **1.18(0.02)** | **1.19(0.02)** | **1.19(0.02)** |
| | 3 s | **3.36(0.03)** | 3.40(0.02) | 3.38(0.06) | **3.34(0.02)** | **1.98(0.02)** | **1.99(0.03)** | 2.05(0.03) | **1.99(0.02)** |
| | 4 s | **5.08(0.04)** | **5.07(0.04)** | 5.09(0.07) | **5.05(0.03)** | **2.99(0.03)** | **2.98(0.04)** | 3.07(0.04) | **2.97(0.03)** |
| | 5 s | **7.24(0.05)** | **7.25(0.06)** | 7.30(0.08) | **7.25(0.05)** | **4.29(0.04)** | 4.34(0.05) | 4.54(0.06) | **4.32(0.04)** |
| NLL | 1 s | **1.48(0.05)** | 1.77(0.06) | 1.79(0.05) | 1.69(0.03) | **0.19(0.02)** | 0.24(0.04) | 0.22(0.03) | **0.18(0.03)** |
| | 2 s | **2.91(0.03)** | 3.34(0.04) | 3.35(0.06) | 3.25(0.02) | **1.61(0.02)** | 1.63(0.03) | 1.64(0.03) | **1.59(0.03)** |
| | 3 s | **3.87(0.02)** | 4.27(0.03) | 4.28(0.05) | 4.18(0.02) | **2.42(0.02)** | **2.44(0.02)** | 2.46(0.03) | **2.43(0.02)** |
| | 4 s | **4.46(0.03)** | 5.00(0.02) | 5.01(0.05) | 4.92(0.02) | **3.02(0.02)** | 3.04(0.02) | 3.06(0.04) | **3.01(0.02)** |
| | 5 s | **5.05(0.04)** | 5.69(0.02) | 5.68(0.06) | 5.64(0.05) | **3.50(0.02)** | 3.59(0.02) | 3.62(0.03) | 3.57(0.02) |

### 4.3.5. Out-of-Distribution Testing

We analyze the generalization capabilities of our model by testing it on out-of-distribution data, e.g., traffic environments that have not been observed during training. For this experiment, we reuse the rounD dataset [94] since it consists of recordings at three different roundabouts. Here, each roundabout corresponds to a separate traffic environment. We select one recording from each roundabout (Kackerstrasse in Aachen, Thiergarten in Alsdorf, and Neuweiler near Aachen) and apply the same data curation steps as in Sec. 4.3.2. In order to generalize between different traffic environments, we change the experimental setup as follows:

*(i) Local coordinate system*: We transform the data into a local coordinate system, which is centered at the ego-vehicle and is oriented to the heading direction of the ego-vehicle.

*(ii) Include map information to the context variable* $\mathcal{I}$: We extract a local map around the ego-vehicle, which spans a rectangle with a length of 74 meters and a width of 44 meters. Afterward, we apply binary masking to the map, which divides the image into drivable and non-drivable areas. Our task is to jointly model all vehicles that lie within this rectangle.

More details on the preprocessing can be found in App. A.4.

#### Results

We analyze the generalization capabilities of our model by comparing the following strategies: (i) training on two traffic environments and testing on a third distinct traffic environment, (ii) directly training the model on the test traffic environment, and (iii) training on all traffic environments. In order to enable a fair comparison, the data for each traffic environment is split into non-overlapping training and test sets.

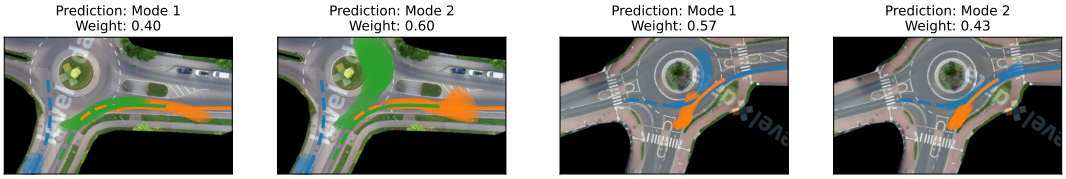
We present the results for different traffic environments in Tab. 4.4. The first column in Tab. 4.4 describes the RMSE and NLL when we train our model on the traffic environments *Thiergarten* (T) and *Neuweiler* (N) and test it on the traffic environment *Kackertstrasse* (K). The second column describes the predictive performance by using scenes from the same traffic environment (K) for training and testing. The third column describes the predictive performance by training on all three traffic environments (KNT) and testing on the traffic environment K. The remaining columns benchmark the generalization capabilities on the traffic environments T and N and are set up in an analogous fashion.

Table 4.4.: Test performance on different traffic environments on the rounD dataset using our method GDSSM (1 mode). We vary the test traffic environment and the training traffic environment. We report average and standard error over 10 runs. Kackertstrasse=K, Thiergarten=T, Neuweiler=N.

| | Test | K | | | T | | | N | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Train | TN | K | KTN | KN | T | KTN | KT | N | KTN |
| RMSE | 1 s | 2.12(0.09) | 1.88(0.05) | **1.55(0.04)** | 1.42(0.04) | 1.49(0.05) | **1.21(0.04)** | 2.98(0.12) | 2.02(0.07) | **1.55(0.04)** |
| | 2 s | 4.56(0.13) | 3.73(0.05) | **3.34(0.09)** | 2.82(0.05) | 2.53(0.09) | **2.27(0.04)** | 5.82(0.19) | 3.38(0.06) | **3.02(0.09)** |
| | 3 s | 7.51(0.24) | 6.05(0.17) | **5.62(0.20)** | 4.42(0.12) | 3.67(0.12) | **3.35(0.07)** | 8.96(0.24) | **4.89(0.13)** | **4.91(0.18)** |
| | 4 s | 10.57(0.34) | 8.65(0.26) | **7.99(0.17)** | 6.75(0.18) | 5.33(0.15) | **5.00(0.16)** | 12.44(0.33) | **6.81(0.14)** | **6.91(0.18)** |
| | 5 s | 13.00(0.31) | 12.02(0.43) | **10.73(0.27)** | 9.97(0.40) | **7.59(0.29)** | **7.41(0.27)** | 14.86(0.42) | **8.77(0.19)** | **8.69(0.31)** |
| NLL | 1 s | 4.11(0.07) | 4.02(0.04) | **3.13(0.06)** | 3.35(0.06) | 3.16(0.06) | **2.88(0.06)** | 4.75(0.07) | 3.84(0.04) | **3.21(0.08)** |
| | 2 s | 5.36(0.10) | 4.84(0.03) | **4.66(0.09)** | 4.22(0.04) | 3.99(0.07) | **3.65(0.04)** | 5.82(0.09) | 4.60(0.05) | **4.42(0.09)** |
| | 3 s | 6.55(0.17) | **5.97(0.07)** | 6.42(0.17) | 5.49(0.10) | 4.98(0.11) | **4.50(0.05)** | 7.32(0.17) | **5.55(0.04)** | 5.68(0.09) |
| | 4 s | 7.71(0.20) | **7.07(0.14)** | 7.57(0.21) | 7.20(0.09) | 6.38(0.19) | **5.89(0.13)** | 8.65(0.18) | **6.79(0.11)** | **6.71(0.08)** |
| | 5 s | 8.63(0.18) | **8.14(0.18)** | **8.08(0.15)** | 8.54(0.12) | 7.45(0.21) | **6.84(0.11)** | 9.11(0.18) | 7.54(0.22) | **7.04(0.14)** |

In all experiments, we observe that the performance increases from (i) using different training environments for training and testing to (ii) using the same environment for training and testing and (iii) using all environments for training. The difference in performance between (i) and (ii) is moderate for the locations K and T demonstrating that our model is capable of generalizing to unseen traffic environments during testing. For location N, the performance difference is increased, which can be explained by studying the locations in more detail: N is a multi-lane roundabout with congested traffic, and the roundabouts K and T are single-lane with moderate traffic. In consequence, the out-of-domain test on the traffic environment N is more challenging. We visualize exemplary predictions of our model GDSSM in Fig. 4.6.

We note that GDSSMs do not consider model uncertainty, which refers to the uncertainty arising from insufficient knowledge. Despite not accounting for this source of uncertainty, we can see in Fig. 4.6 that the predictions made by our GDSSMs model are realistic, as the vehicles remain within their lanes. However, we believe that incorporating model uncertainty into the system can lead to further improvements in predictive performance when dealing with out-of-distribution traffic environments.




(a) Kackertstrasse. (b) Thiergarten.

Figure 4.6.: Predictions of our model GDSSM (2 modes) on out-of-domain traffic environments, i.e. the training and testing traffic environments are distinct. Given the history of each traffic participant (dashed lines), we visualize the predicted 95% confidence interval. Solid lines represent the true future trajectory.

## 4.4. Summary

In this chapter, we proposed GDSSMs in which the latent dynamics of the agents are coupled via GNNs in order to capture interactions among multiple agents. We derived moment matching rules for GNN layers that allow for deterministic inference and introduced a GMM prior over the initial latent states in order to allow for multimodal predictions. Both together lead to an efficient and stable algorithm that is able to produce complex and nonlinear predictive distributions. We confirmed that our novel method shows strong empirical performance on two challenging autonomous driving datasets.

Finally, we proposed sparse approximations to the covariance matrix considering the computational limits of real-world vehicle control units. Depending on the required calibration, our approximations can lead to a significant reduction of the runtime without impeding accuracy.

Up to this point, we focused on two main objectives: accurately capturing data uncertainty and modeling interactions between agents while keeping computational requirements low. In the next chapter, we will address the last goal of accurately capturing model uncertainty. This is crucial as it allows us to account for the uncertainty that arises from the lack of knowledge.

# 5. Sampling-Free Probabilistic Deep State-Space Models

Modeling unknown dynamics from data presents challenges due to the need to consider both the intrinsic uncertainty of the underlying process and the model uncertainty. Data uncertainty, also known as aleatoric uncertainty, is essential to represent the inherent stochasticity of the system [19, 18]. Contrarily, model uncertainty, also known as epistemic uncertainty, is crucial for addressing the uncertainty arising from the lack of knowledge. In the context of autonomous driving, model uncertainty becomes especially important. Due to the vast number of potential traffic scenarios, it is impossible to include all possible scenarios in the training data. When faced with novel and unseen scenarios, the model should exhibit higher model uncertainty compared to scenarios that have already been observed in the training data. This model uncertainty ensures that the model does not make overly confident predictions in unfamiliar situations, allowing for more cautious and adaptive decision-making in autonomous driving applications.

DSSMs provide a principled approach for modeling the data uncertainty in an unknown dynamical process. In the previous chapter, we used neural networks with deterministic weights to model the mean and covariance update functions of the DSSM. While this approach allows for significant flexibility, it is also limited to capturing data uncertainty and does not account for model uncertainty.

Most prior works that take model uncertainty into account make either the simplifying assumption that the transition dynamics are noiseless [119, 120] or that the dynamics are fully observed [18, 121]. Many real-world applications do not satisfy both assumptions and can lead to miscalibrated uncertainties.

There also exists a large body of work for state-space models [122, 123, 124] that use Gaussian Processes to model state transition kernels instead of probabilistic neural networks. While these methods respect both sources of uncertainty, they do not scale well with the size of the latent space. Finally, there is the notable exception of [50] that

aims at learning deep dynamical systems that respect both sources of uncertainty jointly. However, this approach requires to marginalize over the latent states and the neural network weights via plain Monte Carlo, which is infeasible for noisy transition dynamics and contradicts the goal of accurate predictions with low computational cost.

This chapter addresses the open problem of learning unknown dynamics from data that jointly account for data and model uncertainty. We propose an extension of DSSMs that we call *Probabilistic DSSMs* (ProDSSMs). ProDSSMs use neural mean and covariance update functions that account for the uncertainty over the weights of the neural networks. This allows us to capture model uncertainty by attaching uncertainty to the neural net weights while addressing data uncertainty through the deep state-space formulation. To keep things concise, we do not discuss the integration of GNNs into our ProDSSM framework in this chapter. However, our ProDSSM framework is compatible with GNNs, and we consider their integration as a promising avenue for future research. Instead, we focus on feed-forward neural networks and present ProDSSMs as a versatile tool. While ProDSSMs provide us with highly adaptable predictive distributions, the process of inference becomes more challenging due to the combination of uncertainty in both the weights and the latent dynamics.

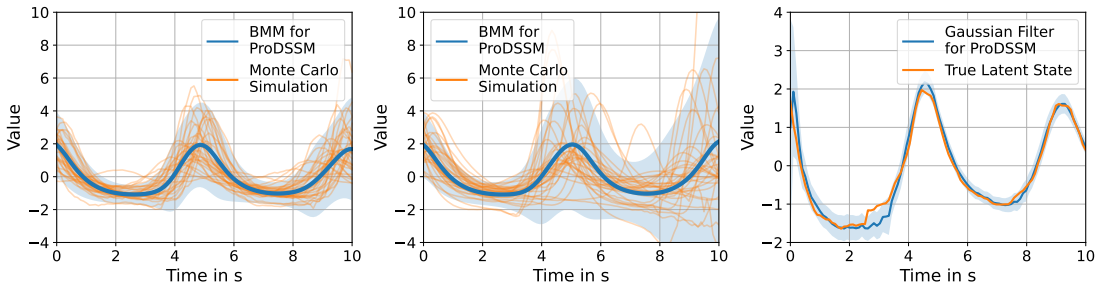
We summarize our contribution as follows:

*(i)* We derive output moments for layers that take weight uncertainty into account, which makes ProDSSMs applicable to moment matching algorithms.

*(ii)* We investigate two distinct weight sampling techniques. The first technique involves resampling the weights at each time step (see Fig. 5.1a), while the second technique involves sampling the weights only once at the initial time step (see Fig. 5.1b). We derive an extension of the BMM algorithm for both sampling methods.

*(iii)* We present an approximation to the general Gaussian filter specifically designed for ProDSSMs (see Fig. 5.1c).

An empirical study complements the chapter (see Sec. 5.4) that begins with an in-depth examination of each individual building block, showcasing their unique strengths. Afterward, we integrate all components and apply our approach to two well-established dynamical modeling benchmark datasets. Our method particularly excels in demanding situations, such as those involving noisy transition dynamics or high-dimensional outputs.




(a) Local Weights. (b) Global Weights. (c) Filtering.

Figure 5.1.: We simulate a dynamical system $p(x_{t+1}|x_t, w_t)$ with uncertainty over the weights $w_t \sim p(w_t)$. We visualize Monte Carlo simulations as orange solid lines. Our deterministic approximation scheme is shown in blue, where the solid line depicts the mean and the shaded area is the 95% confidence interval. We compare two different sampling strategies in panels (a) and (b). In panel (a), we resample at each time step the weights, while in panel (b), the weights are sampled only at the initial time step. In panel (c), we move the dynamical system to a latent space and introduce an emission model $p(y_t|x_t)$. We compare our filtering distribution with the true latent state (orange). The true latent trajectory lies within the 95% confidence interval of the approximate filtering distribution.

## 5.1. Probabilistic Deep State-Space Models

This section presents our *Probabilistic Deep State-Space Model* (ProDSSM) family. Our model can account for model uncertainty by attaching uncertainty to the neural network weights and for data uncertainty by building on the deep state-space formalism. By integrating both sources of uncertainties, our model family promises well-calibrated uncertainties.

Following [125], we consider two variants of propagating the weight uncertainty along a trajectory: the local and global approach. For the local approach, we resample the weights at each time step (see Fig. 5.1a). Contrarily, for the global approach, we sample the weights only once at the initial time step and keep them fixed for all remaining time steps (see Fig. 5.1b).

The transition and emission model of ProDSSMs are defined as follows

$$x_0 \sim p(x_0), \tag{5.1}$$
$$w_0 \sim p(w_0|\phi), \tag{5.2}$$
$$x_{t+1} \sim \mathcal{N}\left(x_t + f(x_t, w_t), \text{diag}(r(x_t, w_t))\right), \tag{5.3}$$
$$w_{t+1} \sim \begin{cases} p(w_{t+1}|\phi), & \text{if Local} \\ \delta(w_{t+1} - w_0), & \text{if Global} \end{cases} \tag{5.4}$$
$$y_t \sim \mathcal{N}\left(g(x_t), \text{diag}\left(s(x_t)\right)\right), \tag{5.5}$$

where $x_t \in \mathbb{R}^{D_x}$ is the latent state, $w_t \in \mathbb{R}^{D_w}$ are the weights, $f : \mathbb{R}^{D_x \times D_w} \to \mathbb{R}^{D_x}$ is the mean update, $r : \mathbb{R}^{D_x \times D_w} \to \mathbb{R}_+^{D_x}$ is the covariance update, $g : \mathbb{R}^{D_x} \to \mathbb{R}^{D_y}$ is the emission mean, and $s : \mathbb{R}^{D_x} \to \mathbb{R}_+^{D_y}$ is the emission variance. We further model the weight distribution $p(w_t|\phi)$ as a Gaussian distribution

$$p(w_t|\phi) = \mathcal{N}(\mu_t^w, \Sigma_t^w), \tag{5.6}$$

with mean $\mu_t^w \in \mathbb{R}^{D_w}$ and covariance $\Sigma_t^w \in \mathbb{R}^{D_w \times D_w}$. Both together define the hyperparameters $\phi = \{\mu_t^w, \Sigma_t^w\}_{t=0}^T$ of our model, where $T$ is the horizon.

In the following, we extend the BMM algorithm towards ProDSSMs and Gaussian filters. Our algorithmic advances are general and can be combined with both weight uncertainties propagation schemes. To make our writing concise, we exclude the dependence on $\phi$ in the subsequent text unless it is necessary for defining the loss function.

## 5.2. Deterministic Approximations for ProDSSMs

ProDSSMs address two types of uncertainty: model uncertainty through weight uncertainty and data uncertainty through the use of the SSM formalism. Joint marginalization over the weights and the latent dynamics presents a significant inference challenge. To this end, we extend the BMM algorithm towards ProDSSMs in Sec. 5.2.1. In contrast to Chap. 4, we do not use an embedding model to infer the initial latent distribution. Instead, we infer the latent distribution with our novel approximation to the Gaussian filter. We introduce our approximation to the Gaussian filter in Sec. 5.2.2, where we also discuss the advantages and disadvantages of using Gaussian filters compared to embedding models. Our algorithmic advances allow for fast and sample-free inference with low compute and lay the basis for our deterministic training objective that we present in Sec. 5.2.3. During

test time, we are interested in computing the predictive log-likelihood. We present our approximation to the predictive log-likelihood and describe the corresponding algorithm in Sec. 5.2.4. Our approximations necessitate the computation of output moments and expected Jacobians of probabilistic neural network layers. We present output moments for commonly used layers in Sec. 5.2.5.

### 5.2.1. Extending the Bidimensional Moment Matching Algorithm

In this section, we extend the BMM algorithm towards ProDSSMs. In contrast to the previous chapters, we need to account for the correlation between the weights and states.

In order to avoid cluttered notation, we introduce the augmented state $z_t = [x_t, w_t]$ that is a concatenation of the latent state $x_t$ and weight $w_t$, with dimensionality $D_z = D_x + D_w$. First, we present a general approach for propagating moments in the time direction. Then, we provide detailed instructions on computing the mean, covariance, and cross-covariance of the update functions.

#### Assumed Process Density

We follow the assumed density approach (see Sec. 2.2) and obtain a Gaussian approximation $p(z_{t+1}|z_0) \approx \mathcal{N}(\mu^z_{t+1}, \Sigma^z_{t+1})$ to the $t$-step transition kernel with mean $\mu^z_t \in \mathbb{R}^{D_z}$ and covariance $\Sigma^z_t \in \mathbb{R}^{D_z \times D_z}$. The mean and the covariance have the structure

$$\mu^z_t = \begin{bmatrix} \mu^x_t \\ \mu^w_t \end{bmatrix}, \tag{5.7}$$

$$\Sigma^z_t = \begin{bmatrix} \Sigma^x_t & \Sigma^{xw}_t \\ \Sigma^{wx}_t & \Sigma^w_t \end{bmatrix}, \tag{5.8}$$

where $\Sigma^x_t \in \mathbb{R}^{D_x \times D_x}$ is the covariance of $x_t$ and $\Sigma^{xw}_t \in \mathbb{R}^{D_x \times D_w}$ is the cross-covariance between $x_t$ and $w_t$.

*(i) Local Weights*: We obtain the update rules for the mean and covariance of the augmented state in the time direction by rewriting them as a function of the weights and latent state (see Sec. 3.2.1 for more details)

$$\mu^z_{t+1} = \begin{bmatrix} \mu^x_t + \mathbb{E}[f(z_t)] \\ \mu^w_{t+1} \end{bmatrix}, \tag{5.9}$$

$$\Sigma^z_{t+1} = \begin{bmatrix} \Sigma^x_{t+1} & 0 \\ 0 & \Sigma^w_{t+1} \end{bmatrix}, \tag{5.10}$$

where the covariance $\Sigma_{t+1}^x$ is a function of prior moments

$$\Sigma_{t+1}^x = \Sigma_t^x + \mathrm{Cov}[f(z_t)] + \mathrm{Cov}[f(z_t), x_t] + \mathrm{Cov}[x_t, f(z_t)] + \mathrm{diag}(\mathbb{E}[r(z_t)]). \tag{5.11}$$

The moments of the weights $\mu_t^w$ and $\Sigma_t^w$ are given, and the cross-covariance $\Sigma_t^{xw}$ is zero due to the resampling step. We are now left with approximating the output moments of $f(z_t)$ and $r(z_t)$, as well as the cross-covariance $\mathrm{Cov}[f(z_t), x_t]$.

*(ii) Global Weights*: Similarly, we can derive the update rules for the case of global weights

$$\mu_{t+1}^z = \begin{bmatrix} \mu_t^x + \mathbb{E}[f(z_t)] \\ \mu_t^w \end{bmatrix} \tag{5.12}$$

$$\Sigma_{t+1}^z = \begin{bmatrix} \Sigma_{t+1}^x & \Sigma_{t+1}^{xw} \\ \Sigma_{t+1}^{wx} & \Sigma_t^w \end{bmatrix}. \tag{5.13}$$

Mean $\mu_t^w$ and covariance $\Sigma_t^w$ are constant. The covariance $\Sigma_{t+1}^x$ can be approximated in a similar manner as described in Eq. 5.11. As for the case of local weights, we are now left with calculating the output moments of $f(z_t)$ and $r(z_t)$, as well as the cross-covariance $\mathrm{Cov}[f(z_t), x_t]$. Additionally, we need also to compute the cross-covariance

$$\Sigma_{t+1}^{xw} = \mathrm{Cov}[x_t + f(z_t), w_t] = \Sigma_t^{xw} + \mathrm{Cov}[f(z_t), w_t]. \tag{5.14}$$

For a standard DSSM architecture, the number of weights exceeds the number of latent dimensions. Since the mean and the covariance over the weights are not updated over time, the computational burden of computing $\Sigma_t^z$ is dominated by the computation of the cross-covariance $\Sigma_t^{xw}$. This covariance becomes zero for the local approach due to the resampling step at each time point. Consequently, the local approach exhibits reduced runtime and memory complexity compared to the global approach. Next, we describe how to approximate all moments and cross-covariance terms.

#### Computing the Moments and Cross-Covariance terms of the Update Functions

In the following, we focus on the approximation of the mean, covariance, and cross-covariance terms for the mean update $f(z_t)$ and omit the discussion on the covariance update $r(z_t)$ as our findings are general and can be applied in a similar manner to $r(z_t)$. Our approximation extends the *Vertical Moment Matching* (VMM) algorithm from Sec. 3.2.2 towards ProDSSMs. Typically, neural networks are a composition of $L$ simple functions (layers) that allows us to write the output as $f(z_t) = u^L(\dots u^1(z_t^0)\dots)$, where $z_t^l \in \mathbb{R}^{D_{z,l}}$ is the augmented state at layer $l$ at time point $t$. We denote the input as $z_t^0 = z_t$.

The state $x_t^{l-1} \in \mathbb{R}^{D_{x,l-1}}$ at layer $l-1$ is updated via the function $u_l : \mathbb{R}^{D_{z,l-1}} \to \mathbb{R}^{D_{x,l}}$ while the weights are not changed

$$z_t^l = \begin{bmatrix} x_t^l \\ w_t \end{bmatrix} = \begin{bmatrix} u_l(x_t^{l-1}, w_t) \\ w_t \end{bmatrix}. \tag{5.15}$$

We omit the layer index for the weights $w_t$, as they are not altered in the intermediate layers. As in Sec. 3.2.2, we approximate the output distribution of each layer recursively

$$p(z_t^l) \approx \mathcal{N}(\mathbb{E}[z_t^l], \mathrm{Cov}[z_t^l]), \tag{5.16}$$

where $\mathbb{E}[z_t^l] \in \mathbb{R}^{D_{z,l}}$ and $\mathrm{Cov}[z_t^l] \in \mathbb{R}^{D_{z,l} \times D_{z,l}}$ are the mean and covariance of $z_t^l$. For global weights, we calculate the full covariance matrix of $\mathrm{Cov}[z_t^l]$ as the cross-covariance between $x_t^l$ and $w_t$ changes after each layer. Contrarily, for local weights, we need only to calculate the covariance of $x_t^l$ and omit the remaining terms. By following this recipe, we can approximate the expected value $\mathbb{E}[f(z_t)] = \mathbb{E}[x_t^L]$, as well as the covariance term $\mathrm{Cov}[f(z_t)] = \mathrm{Cov}[x_t^L]$ and the cross-covariance $\mathrm{Cov}[f(z_t), w_t] = \mathrm{Cov}[x_t^L, w_t]$.

We are now left with approximating the cross-covariance $\mathrm{Cov}[f(z_t), x_t]$. Using Stein's lemma and *Backward Vertical Moment Matching* (BVMM) (see Sec. 3.2.3) the cross-covariance is approximated as

$$\mathrm{Cov}[f(z_t), x_t] \approx \nabla_{z_t^{L-1}} u_L(z_t^{L-1}) \prod_{l=L-1}^{1} \mathbb{E} \begin{bmatrix} \nabla_{z_t^{l-1}} u_l(z_t^{l-1}) \\ \nabla_{z_t^{l-1}} w_t \end{bmatrix} \begin{bmatrix} \Sigma_t^x \\ \Sigma_t^{wx} \end{bmatrix}. \tag{5.17}$$

Here, $\mathbb{E}[\nabla_{z_t^{l-1}} u_l(z_t^{l-1}) \;\; \nabla_{z_t^{l-1}} w_t]^\top \in \mathbb{R}^{D_{z,l} \times D_{z,l-1}}$ represents the expected Jacobian at layer $l$. The expected Jacobian for each layer follows a structured and sparse matrix

$$\mathbb{E} \begin{bmatrix} \nabla_{z_t^{l-1}} u_l(z_t^{l-1}) \\ \nabla_{z_t^{l-1}} w_t \end{bmatrix} = \begin{bmatrix} \mathbb{E}[\nabla_{x_t^{l-1}} u_l(z_t^{l-1})] & \mathbb{E}[\nabla_{w_t} u_l(z_t^{l-1})] \\ 0 & I \end{bmatrix}, \tag{5.18}$$

where $\mathbb{E}[\nabla_{x_t^{l-1}} u_l(z_t^{l-1})] \in \mathbb{R}^{D_{x,l} \times D_{x,l-1}}$ and $\mathbb{E}[\nabla_{w_t} u_l(z_t^{l-1})] \in \mathbb{R}^{D_{x,l} \times D_w}$ are the expected Jacobians of $u_l(z_t^{l-1})$. These Jacobians are calculated by taking the derivative with respect to $x_t^{l-1}$ and $w_t$, respectively. Furthermore, the expected Jacobian $\mathbb{E}[\nabla_{x_t^{l-1}} w_t] \in \mathbb{R}^{D_w \times D_{x,l-1}}$ is zero and $\mathbb{E}[\nabla_{w_t} w_t] \in \mathbb{R}^{D_w \times D_w}$ is the identity matrix $I$. The product of all expected Jacobians in Eq. 5.17 has the same structure as the Jacobian of each layer, which makes calculating the product computationally efficient. For the case of global weights, we must calculate the full product in Eq. 5.17. Contrarily, for the case of local weights, it is sufficient to calculate $\mathbb{E}[\nabla_{x_t^{l-1}} u_l(z_t^{l-1})]$ as $\Sigma_t^{wx}$ is zero.

### 5.2.2. Gaussian Filtering

Our approximation to the filtering distribution, $p(z_t|y_{1:t})$, follows the Gaussian filter (see Sec. 2.3). In contrast to prior work, we extend the filtering step to the augmented state consisting of the latent dynamics and the weights. In standard architectures, the number of latent states is small compared to the number of weights, which makes filtering in our new scenario more demanding. We address this challenge by applying our deterministic moment matching scheme that propagates moments across neural network layers. Additionally, we combine it with our previously derived approximation to the $t$-step transition kernel $p(z_{t+1}|z_0)$ from Sec. 5.2.1. We also verify empirically in Sec. 5.4.2 that standard numerical integration schemes are not well suited for filtering tasks of this type.

The Gaussian filter alternates between the prediction and the update step. In the following, we explain in more detail how our deterministic moment matching scheme can be integrated into both steps. For the prediction step (see Eq. 2.6), we can reuse the assumed density approach that we just derived in order to compute a Gaussian approximation to the distribution $p(z_t|y_{1:t-1})$.

For the update step, we need to find a Gaussian approximation to the joint distribution of the augmented state $z_t$ and observation $y_t$ conditioned on $y_{1:t-1}$ (see Eq. 2.7)

$$p(z_t, y_t|y_{1:t-1}) \approx \mathcal{N}\left(\begin{bmatrix}\mu^z_{t|t-1}\\ \mu^y_{t|t-1}\end{bmatrix}, \begin{bmatrix}\Sigma^z_{t|t-1} & \Sigma^{zy}_{t|t-1}\\ \Sigma^{yz}_{t|t-1} & \Sigma^y_{t|t-1}\end{bmatrix}\right). \tag{5.19}$$

Here, the index $t|t'$ denotes prior moments, i.e., the moments at time step $t$ conditioned on the observations up to time step $t'$. If $t = t'$, we omit the double index. The mean and the covariance of the latent state $z_t$ are known from the prediction step, while their equivalents of the emission $y_t$ are available as

$$\mu^y_{t|t-1} = \mathbb{E}[g(x_t)], \tag{5.20}$$

$$\Sigma^y_{t|t-1} = \mathrm{Cov}[g(x_t)] + \mathrm{diag}(\mathbb{E}[s(x_t)]), \tag{5.21}$$

with $x_t \sim \mathcal{N}(\mu^x_{t|t-1}, \Sigma^x_{t|t-1})$. These moments can be approximated with layerwise moment propagation, as described in the previous section. Finally, we facilitate the computation of the cross-covariance $\Sigma^{yz}_{t|t-1}$ be using Stein's lemma [64]

$$\Sigma^{yz}_{t|t-1} = \mathrm{Cov}[g(x_t), z_t] = \mathbb{E}[\nabla_{x_t} g(x_t)]\Sigma^{xz}_{t|t-1}. \tag{5.22}$$

where the expected Jacobian $\mathbb{E}[\nabla_{x_t} g(x_t)]$ of the mean emission function cannot be computed analytically. As in the previous chapter, we approximate it by BVMM, which reduces the computation to estimate the expected Jacobian per layer.

Once we have calculated the joint distribution, we approximate the conditional as another normal distribution, $p(z_t|y_{1:t}) \approx \mathcal{N}(\mu_t^z, \Sigma_t^z)$, as shown in Eq. 2.11. For the global approach, the Kalman gain $K_t = \Sigma_t^{zy}(\Sigma_t^y)^{-1}$, is dense and the updated covariance matrix $\Sigma_t^z$ of augmented state $z_t$ is also dense. As a consequence, the weights $w_t$ have a non-zero correlation after the update, and the overall variance gets reduced. For the local approach, only the distribution of the states $x_t$ will be updated since the lower block of the gain matrix is zero. The weight distribution, as well as the cross-covariance between the states and weights, is hence not affected by the update step.

In the previous chapter, we relied on an embedding model to infer the latent distribution from past observations. Training a separate neural net to infer the latent state decouples the transition model and the filtering distribution. We believe that using the Gaussian filter instead of the embedding model is advantageous for parameter inference because it allows the gradient signal to be backpropagated from the filtering step to the transition model. We observe the Gaussian filter to result in a better model fit in our experiments in Sec. 5.4.3. However, incorporating map information or other contextual details into our Gaussian filtering algorithm is not a straightforward task.

### 5.2.3. Parameter Inference

We train the ProDSSMs by fitting the hyperparameters $\phi$ to a dataset $\mathcal{D}$. The hyperparameters $\phi$ describe the weight distribution. For the sake of brevity, we introduce the shorthand notation $p(w_{0:T}|\phi) = p(w|\phi)$ to refer to the weights at all time steps with arbitrary horizon $T$. We propose to train the ProDSSM on a Type-II *Maximum A Posteriori* (MAP) objective (see [17] Chap. 5.6)

$$\underset{\phi}{\operatorname{argmax}} \log \int p(\mathcal{D}|w)p(w|\phi)dw + \log p(\phi). \tag{5.23}$$

This objective is also termed as predictive variational Bayesian inference by [52] as it directly minimizes the Kullback-Leibler divergence between the true data generating distribution and the predictive distribution, which we aim to learn. Compared to other learning objectives, Eq. 5.23 provides better predictive performance, is more robust to model misspecification, and provides a beneficial implicit regularization effect for over-parameterized models. We refer to [51, 126, 52] that studies this learning objective for probabilistic neural nets in more detail from a theoretical as well as an empirical point of view.

The typically hard to evaluate likelihood $p(\mathcal{D}|\phi) = \int p(D|w)p(w|\phi)dw$ can be closely approximated with our deterministic moment matching routines. The exact form of the likelihood hereby depends on the task at hand, and we specify in our experiments how the likelihood can be closely approximated for regression problems in Sec. 5.4.1 and for dynamical system modeling in Sec. 5.4.3.

We are now left with defining the hyper-prior $p(\phi)$. Remember, $\phi$ defines the weight distribution that is defined by its two first moments $\mu^w = \mu^w_{0:T}$ and $\Sigma^w = \Sigma^w_{0:T}$. In order to arrive at an analytical objective, we model each entry in $p(\phi)$ independently. We define the hyper-prior of the $i$-th entry of the mean as a standard Normal

$$\log p(\mu^w_i) = \log \mathcal{N}(\mu^w_i|0, I) = -\frac{1}{2}(\mu^w_i)^2 + \text{const.} \tag{5.24}$$

and, assuming that the covariance is diagonal, chose the Gamma distribution for the $(i, i)$-th covariance entry

$$\log p(\Sigma^w_{ii}) = \log \text{Ga}(\Sigma^w_{ii}|\alpha = 1.5, \beta = 0.5) = \frac{1}{2}\log \Sigma^w_{ii} - \frac{1}{2}\Sigma^w_{ii} + \text{const.}, \tag{5.25}$$

where $\alpha$ is the shape parameter and $\beta$ is the rate parameter. We insert the above hyper-prior of the mean and covariance into $\log p(\phi)$ and arrive at

$$\log p(\phi) = \log p(\mu^w) + \log p(\Sigma^w) = \frac{1}{2}\sum_{i=1}^{D_w} \log \Sigma^w_{ii} - (\mu^w_i)^2 - \Sigma^w_{ii} + \text{const.}, \tag{5.26}$$

which leads to a total of $2D_w$ hyperparameters, i.e., one for the mean and one for the variance of each weight.

In contrast, the classical Bayesian formalism keeps the prior $p(w|\phi)$ constant during learning and the posterior $p(w|\mathcal{D})$ is the quantity of interest. As an analytical solution to the posterior is intractable, either *Markov Chain Monte Carlo* (MCMC) [42] or *Variational Inference* (VI) [43] is used. It is interesting to note that the only difference between our formulation and the objective in VI, with a suitable prior choice, is the position of the logarithm in the likelihood $p(\mathcal{D}|\phi)$. Please see App. B.1 for more details. However, we are not aware of any prior work that applies VI in the context of ProDSSMs. Closest to our method is most likely [18] that approximates the posterior over the weights for fully observed stochastic dynamical systems, i.e., without latent states.

### 5.2.4. Approximating the Predictive Log-Likelihood

During test time, we are interested in the predictive log-likelihood at time step $t$ conditioned on the observations $y_{-H:0} = \{y_{-H} \dots, y_0\}$, which we denote as $\text{PLL}(y_t|y_{-H:0}) = \log p(y_t|y_{-H:0})$. The predictive log-likelihood is computed as

$$\begin{aligned}\text{PLL}(y_t|y_{-H:0}) &= \log \int p(y_t|z_t)p(z_t|z_0)p(z_0|y_{-H:0})dz_0, z_t, \\ &= \log \int p(y_t|z_t)p(z_t|y_{-H:0})dz_t. \end{aligned} \tag{5.27}$$

Above, $p(z_0|y_{-H:0})$ is the filtering distribution, $p(z_t|z_0)$ is the $t$-step transition kernel and $p(z_t|y_{-H:0})$ the $t$-step marginal. Prior work on general deep SSMs [27, 29, 26] relies on auxiliary networks in order to approximate the filtering distribution and then uses MC integration in order to compute predictive distribution. Contrarily, we replace the need for auxiliary networks and MC integration with our deterministic moment matching scheme.

Key to computing $\text{PLL}(y_t|y_{-H:0})$ is an accurate approximation of the predictive distribution $p(y_t|y_{-H:0})$. We approximate it by a series of Gaussian approximations:

$$\begin{aligned} p(y_t|y_{-H:0}) &\approx \int p(y_t|z_t)p(z_t|z_0)\mathcal{N}(\mu_0^z, \Sigma_0^z)dz_0, z_t \\ &\approx \int p(y_t|z_t)\mathcal{N}(\mu_{t|0}^z, \Sigma_{t|0}^z)dz_t \\ &\approx \mathcal{N}(\mu_{t|0}^y, \Sigma_{t|0}^y), \end{aligned} \tag{5.28}$$

where the Gaussian $\mathcal{N}(\mu_0^z, \Sigma_0^z)$ approximates the filtering distribution. Its computation is described in Sec. 5.2.2. We obtain the distribution $\mathcal{N}(\mu_{t|0}^z, \Sigma_{t|0}^z)$ as an approximation to the $t$-step marginal $p(z_t|y_{-H:0})$ in Eq. 5.27 by propagating the augmented latent state forward in time as described in Sec. 5.2.1. Finally, we approximate the predictive distribution $p(y_t|y_{-H:0})$ with the Gaussian $\mathcal{N}(\mu_{t|0}^y, \Sigma_{t|0}^y)$ in Eq. 5.28, which can be done by another round of moment matching as also outlined in Eq. 5.20 and 5.21.

We present pseudo-code for approximating the predictive distribution in Alg. 4 that relies on Alg. 5 to approximate the filtering distribution $p(z_0|y_{-H:0}) \approx \mathcal{N}(z_0|\mu_0^z, \Sigma_0^z)$ Both algorithms explicitly do a resampling step for the local weight setting. In practice, it is not necessary, and we omit the calculation. In our algorithmic description, we use for the local approach the same weight distribution $p(w_t) = p(w_0) = \mathcal{N}(\mu_0^w, \Sigma_0^w)$ for every step.

**Algorithm 4** Deterministic Inference (`DetInf`)

**Inputs:** $f(x_t, w_t)$ ▷ Mean update
$r(x_t, w_t)$ ▷ Covariance update
$g(x_t)$ ▷ Mean emission
$s(x_t)$ ▷ Covariance emission
$p(z_{-H})$ ▷ Initial distribution
$y_{-H:0}$ ▷ Observations
**Outputs:** $p(y_T|y_{-H:0}) \approx \mathcal{N}(\mu^y_{T|0}, \Sigma^y_{T|0})$ ▷ Predictive Distribution
$\mu^z_0, \Sigma^z_0 \leftarrow \texttt{DetFilt}(f, r, g, s, p(z_{-H}), y_{-H:0})$
**for** time step $t \in \{0, \cdots, T-1\}$ **do**
$\mu^x_{t+1|0} \leftarrow \mu^x_{t|0} + \mathbb{E}[f(z_t)]$ ▷ Eq. 5.9 and 5.12
$\Sigma^x_{t+1|0} \leftarrow \Sigma^x_{t|0} + \mathrm{Cov}[f(z_t)] + \mathrm{Cov}[f(z_t), x_t] + \mathrm{Cov}[x_t, f(z_t)] + \mathrm{diag}(\mathbb{E}[r(z_t)])$ ▷ Eq. 5.10 and 5.13
**if** Local **then** $\mu^w_{t+1|0}, \Sigma^w_{t+1|0}, \Sigma^{xw}_{t+1|0} \leftarrow \mu^w_{-H}, \Sigma^w_{-H}, 0$ ▷ Eq. 5.9 and 5.10
**if** Global **then** $\mu^w_{t+1|0}, \Sigma^w_{t+1|0}, \Sigma^{xw}_{t+1|0} \leftarrow \mu^w_t, \Sigma^w_t, \Sigma^{xw}_t + \mathrm{Cov}[f(z_t), w_t]$ ▷ Eq. 5.12, 5.13, and 5.14
$p(z_{t+1}|y_{-H:0}) \leftarrow \mathcal{N}(\mu^z_{t+1|0}, \Sigma^z_{t+1|0})$
**end for**
$\mu^y_{T|0} \leftarrow \mathbb{E}[g(x_T)]$ ▷ Eq. 5.20
$\Sigma^y_{T|0} \leftarrow \mathrm{Cov}[g(x_T)] + \mathrm{diag}(\mathbb{E}[s(x_T)])$ ▷ Eq. 5.21
**return** $\mathcal{N}(\mu^y_{T|0}, \Sigma^y_{T|0})$

**Algorithm 5** Deterministic Filtering (`DetFilt`)

**Inputs:** $f(x_t, w_t)$ ▷ Mean update
$r(x_t, w_t)$ ▷ Covariance update
$g(x_t)$ ▷ Mean emission
$s(x_t)$ ▷ Covariance emission
$p(z_0)$ ▷ Initial distribution
$y_{1:T}$ ▷ Observations
**Outputs:** $p(z_T|y_{1:T}) \approx \mathcal{N}(\mu^z_T, \Sigma^z_T)$ ▷ Filtering Distribution
$p(z_0|y_{1:0}) \leftarrow p(z_0)$
**for** time step $t \in \{0, \cdots, T-1\}$ **do**
$\mu^x_{t+1|t} \leftarrow \mu^x_t + \mathbb{E}[f(z_t)]$ ▷ Eq. 5.9 and 5.12
$\Sigma^x_{t+1|t} \leftarrow \Sigma^x_t + \mathrm{Cov}[f(z_t)] + \mathrm{Cov}[f(z_t), x_t] + \mathrm{Cov}[x_t, f(z_t)] + \mathrm{diag}(\mathbb{E}[r(z_t)])$ ▷ Eq. 5.10 and 5.13
**if** Local **then** $\mu^w_{t+1|t}, \Sigma^w_{t+1|t}, \Sigma^{xw}_{t+1|t} \leftarrow \mu^w_0, \Sigma^w_0, 0$ ▷ Eq. 5.9 and 5.10
**if** Global **then** $\mu^w_{t+1|t}, \Sigma^w_{t+1|t}, \Sigma^{xw}_{t+1|t} \leftarrow \mu^w_t, \Sigma^w_t, \Sigma^{xw}_t + \mathrm{Cov}[f(z_t), w_t]$ ▷ Eq. 5.12, 5.13, and 5.14
$\mu^y_{t+1|t} \leftarrow \mathbb{E}[g(x_t)]$ ▷ Eq. 5.21
$\Sigma^y_{t+1|t} \leftarrow \mathrm{Cov}[g(x_t)] + \mathrm{diag}(\mathbb{E}[s(x_t)])$ ▷ Eq. 5.20
$\Sigma^{yz}_{t+1|t} \leftarrow \mathbb{E}[\nabla_{x_{t+1}} g(x_{t+1})]\Sigma^{xz}_{t+1|t}$ ▷ Eq. 5.22
$K_{t+1} \leftarrow \Sigma^{zy}_{t+1|t}(\Sigma^y_{t+1|t})^{-1}$ ▷ Eq. 2.11
$\mu^z_{t+1} \leftarrow \mu^z_{t+1|t} + K_{t+1}(y_{t+1} - \mu^y_{t+1|t})$ ▷ Eq. 2.9
$\Sigma^z_{t+1} \leftarrow \Sigma^z_{t+1|t} - K_{t+1}\Sigma^y_{t+1|t}K^T_{t+1}$ ▷ Eq. 2.10
$p(z_{t+1}|y_{1:t+1}) \leftarrow \mathcal{N}(\mu^z_{t+1}\Sigma^z_{t+1})$
**end for**
**return** $\mathcal{N}(\mu^z_T, \Sigma^z_T)$

### 5.2.5. Output Moments of Probabilistic Neural Network Layers

In order to use our extension of the BMM framework for ProDSSMs, we need to be able to calculate the first two output moments as well as the expected Jacobian of probabilistic layers. We refer to a layer as probabilistic if it accounts for the uncertainty over the weights. Below, we present the output moments for the linear layer and ReLU activation function for the global as well as local approach.

#### Output Moments of the Linear Layer

A linear layer applies an affine transformation

$$x_t^{l+1} = A_t^{l+1} x_t^l + b_t^{l+1}, \tag{5.29}$$

where the transformation matrix $A_t^{l+1} \in \mathbb{R}^{D_{x,l+1} \times D_{x,l}}$ and bias $b_t^{l+1} \in \mathbb{R}^{D_{x,l+1}}$ are both part of weights $(A_t^{l+1}, b_t^{l+1}) \in w_t$. We note that the set of all transformation matrices and biases $\{(A_t^l, b_t^l)\}_{l=1}^L$ define the weights $w_t$. As the cross-covariance matrix $\mathrm{Cov}[x_t^l, w_t]$ is non-zero for global weights, the transformation matrix $A_t^{l+1}$, bias $b_t^{l+1}$, and state $x_t^l$ are assumed to be jointly normally distributed.

The $i$-th output $x_{t,i}^{l+1} \in \mathbb{R}$ of the affine transformation is calculated as

$$x_{t,i}^{l+1} = \sum_{m=1}^{D_{x,l}} a_{t,i,m}^{l+1} x_{t,m}^l + b_{t,i}^{l+1}, \tag{5.30}$$

where $a_{t,i,j}^{l+1}$ is $(i,j)$-th entry in $A_t^{l+1}$ and $b_{t,i}^{l+1}$ is the $i$-th entry in $b_t^{l+1}$. Given the above update rule, we can calculate the output moments of the affine transformation as

$$\mathbb{E}[x_{t,i}^{l+1}] = \sum_{m=1}^{D_{x,l}} \mathbb{E}[a_{t,i,m}^{l+1} x_{t,m}^l] + \mathbb{E}[b_{t,i}^{l+1}], \tag{5.31}$$

$$\begin{aligned}\mathrm{Cov}[x_{t,i}^{l+1}, x_{t,j}^{l+1}] = &\sum_{m,n=1}^{D_{x,l}} \mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^l, a_{t,j,n}^{l+1} x_{t,n}^l] + \sum_{m=1}^{D_{x,l}} \mathrm{Cov}[b_{t,i}^{l+1}, a_{t,j,m}^{l+1} x_{t,m}^l] + \\ &\sum_{m=1}^{D_{x,l}} \mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^l, b_{t,j}^{l+1}] + \mathrm{Cov}[b_{t,i}^{l+1}, b_{t,j}^{l+1}],\end{aligned} \tag{5.32}$$

$$\mathrm{Cov}[x_{t,i}^{l+1}, w_{t,j}] = \sum_{m=1}^{D_{x,l}} \mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^{l}, w_{t,j}] + \mathrm{Cov}[b_{t,i}^{l+1}, w_{t,j}], \tag{5.33}$$

which is a direct result of the linearity of the $\mathrm{Cov}[\cdot, \cdot]$ operator. In order to compute the above moments, we need to calculate the moments of a product of correlated normal variables, $\mathbb{E}[a_{t,i,m}^{l+1} x_{t,m}^{l}]$, $\mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^{l}, a_{t,j,n}^{l+1} x_{t,n}^{l}]$, and $\mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^{l}, w_{t,j}]$.

Below, we provide the recipe for calculating these quantities. We give a detailed derivation in App. B.2. We make use of Isserlis' theorem [127] and obtain the output moments of a product of correlated normal variables are calculated as

$$\mathbb{E}[a_{t,i,m}^{l+1} x_{t,m}^{l}] = \mathrm{Cov}[a_{t,i,m}^{l+1}, x_{t,m}^{l}] + \mathbb{E}[a_{t,i,m}^{l+1}]\mathbb{E}[x_{t,m}^{l}], \tag{5.34}$$

$$\begin{aligned}\mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^{l}, a_{t,j,n}^{l+1} x_{t,n}^{l}] = &\mathrm{Cov}[a_{t,i,m}^{l+1}, a_{t,j,n}^{l+1}]\mathrm{Cov}[x_{t,m}^{l}, x_{t,n}^{l}] + \\ &\mathrm{Cov}[a_{t,i,m}^{l+1}, a_{t,j,n}^{l+1}]\mathbb{E}[x_{t,m}^{l}]\mathbb{E}[x_{t,n}^{l}] + \\ &\mathrm{Cov}[x_{t,m}^{l}, x_{t,n}^{l}]\mathbb{E}[a_{t,i,m}^{l+1}]\mathbb{E}[a_{t,j,n}^{l+1}] + \\ &\mathrm{Cov}[a_{t,i,m}^{l+1}, x_{t,n}^{l}]\mathrm{Cov}[x_{t,m}^{l}, a_{t,j,n}^{l+1}] + \\ &\mathrm{Cov}[a_{t,i,m}^{l+1}, x_{t,n}^{l}]\mathbb{E}[x_{t,m}^{l}]\mathbb{E}[a_{t,j,n}^{l+1}] + \\ &\mathrm{Cov}[x_{t,m}^{l}, a_{t,j,n}^{l+1}]\mathbb{E}[a_{t,i,m}^{l+1}]\mathbb{E}[x_{t,n}^{l}],\end{aligned} \tag{5.35}$$

$$\mathrm{Cov}[a_{t,i,m}^{l+1} x_{t,m}^{l}, w_{t,j}] = \mathrm{Cov}[a_{t,i,m}^{l+1}, w_{t,j}]\mathbb{E}[x_{t,m}^{l}] + \mathrm{Cov}[x_{t,m}^{l}, w_{t,j}]\mathbb{E}[a_{t,i,m}^{l+1}]. \tag{5.36}$$

The above results are exact and hold for both local and global weights as long as $x_t^l$ and $w_t$ follow a normal distribution. For local weights the cross-covariance terms $\mathrm{Cov}[a_{t,i,m}^{l+1}, a_{t,j,n}^{l+1}]$ and $\mathrm{Cov}[x_{t,m}^{l}, a_{t,j,n}^{l+1}]$ are zero. Setting these cross-covariance terms in Eq. 5.35 to zero recovers the result from [44].

The expected Jacobian is analytically available

$$\mathbb{E}[\nabla_{x_t^l} u_{l+1}(z_t^l)] = \mathbb{E}[A^{l+1}], \tag{5.37}$$

$$\mathbb{E}[\nabla_{A_t^{l+1}} u_{l+1}(z_t^l)] = I \otimes \mathbb{E}[x^l]^\top, \tag{5.38}$$

$$\mathbb{E}[\nabla_{b_t^{l+1}} u_{l+1}(z_t^l)] = I, \tag{5.39}$$

where $I$ is the identity matrix with shape $D_{x,l+1} \times D_{x,l+1}$ and $\otimes$ is the Kronecker product. The product $I \otimes \mathbb{E}[x^l]^\top$ results in a matrix with shape $D_{x,l+1} \times (D_{x,l+1} D_{x,l})$. The remaining entries in the Jacobian $\mathbb{E}[\nabla_{w_t} u_{l+1}(z_t^l)]$, i.e., the derivatives with respect to the unused weights, are zero.

#### Output Moments of the ReLU Activation

The ReLU activation function applies element-wise the max-operator to the latent states while the weights stay unaffected

$$x_t^{l+1} = \max(0, x_t^l). \tag{5.40}$$

Mean $\mathbb{E}[x_t^{l+1}]$ and covariance $\mathrm{Cov}[x_t^{l+1}]$ of the state $x_t^{l+1}$ are provided in Sec. 3.3. Using Stein's lemma [64], we calculate the cross-covariance after the ReLU activation as

$$\mathrm{Cov}[x_t^{l+1}, w_t] = \mathbb{E}[\nabla_{x_t^l} u_{l+1}(z_t^l)]\mathrm{Cov}[x_t^l, w_t], \tag{5.41}$$

where $\mathbb{E}[\nabla_{x_t^l} u_{l+1}(z_t^l)]$ is the expected Jacobian of the ReLU activation. The expected Jacobian is equal to the expectation of the Heaviside function, as described in Sec. 3.3. As the ReLU function involves no learnable weights, the Jacobian $\mathbb{E}[\nabla_{w_t} u_{l+1}(z_t^l)]$ is zero.

## 5.3. Runtime

We analyze the theoretical runtime of our algorithm in Sec. 5.3.1 and then measure its wall clock time in Sec. 5.3.2.

### 5.3.1. Theoretical Runtime

We first investigate the runtime for simulating forwards in time and, secondly, the runtime for filtering applications. We further assume that we have a ProDSSM with maximal hidden layer width $H$ and that the dimensions of $D_x$ and $D_y$ are less than or equal to $H$.

Independent of the weight modeling scheme, predicting the next observation $x_{t+1}$ conditioned on the latent state $x_t$ is done by propagating the state through a series of affine transformations and nonlinear activities. The affine transformations scale polynomially with the hidden layer width, whereas the nonlinearities are elementwise operations and can be neglected.

Approximating the first two output moments by MC simulation requires propagating $S$ particles, resulting thus in a total cost of $\mathcal{O}(SH^2)$. Our method approximates the $S \to \infty$ limit.

For global weights, the computational cost of our method is $\mathcal{O}(H^4 + D_w H^2)$ where $D_w$ is the number of weight parameters. The first term, $\mathcal{O}(H^4)$, is due to the computational cost of the covariance $\mathrm{Cov}[A_t^{l+1} x_t^l, A_t^{l+1}, x_t^l] \in \mathbb{R}^{H \times H}$, where the computation of each matrix entry scales with $O(H^2)$ (see Eq. 5.32). The second term, $\mathcal{O}(D_w H^2)$, is due to the cross-covariance $Cov[A_t^{l+1} x_t^l, w_t^l] \in \mathbb{R}^{H \times W}$, where the computation of each entry scales with $O(H)$ (see Eq. 5.33). Similarly, calculating the product of Jacobians in Eq. 5.17 has the computational cost of $\mathcal{O}(D_w H^2)$.

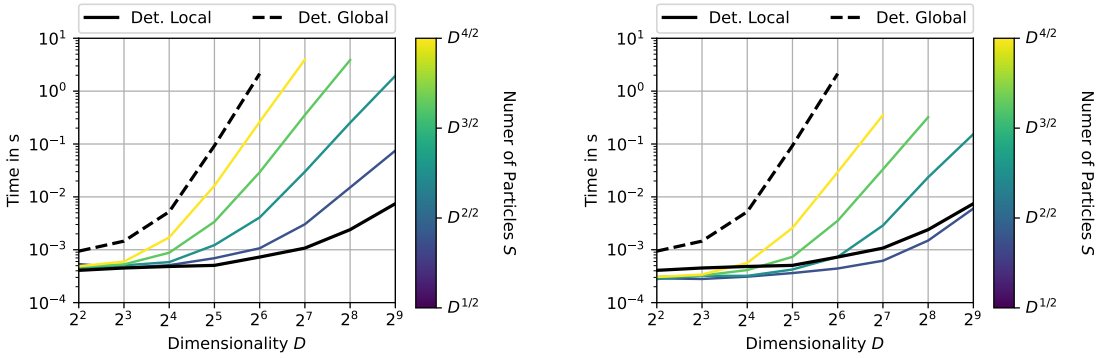
For local weights, the weights and the states are independent. As a result, we can simplify the computation of the first term to $\mathrm{Cov}[A_t^{l+1} x_t^l, A_t^{l+1}, x_t^l] = \mathbb{E}[A_t^l] \mathrm{Cov}[x_t^l, x_t^l] \mathbb{E}[A_t^{l+1}]^\top$ and the second term, $\mathrm{Cov}[A_t^{l+1} x_t^l, w_t^l]$ becomes zero. Furthermore, when calculating the expected Jacobian, we can omit the derivatives with respect to the weights. This leads to a runtime reduction to $O(H^3)$.

Our filtering algorithm necessitates $\mathcal{O}(H^3)$ computations to approximate the output moments of the emission independent of the weight modeling scheme. For global weights approximating the cross-covariance between the emissions and augmented latent state involves $\mathcal{O}(H^3 + D_w H^2)$ computations. Forming the gain matrix involves $\mathcal{O}(H^3 + D_w H^2)$ computations. The first term is caused by inverting the covariance matrix of the emissions, and the second term is caused by multiplying the inverse covariance matrix with the cross-covariance of the augmented latent state (see Eq. 2.11). Lastly, updating the moments of latent state (see Eq. 2.10) involves $\mathcal{O}(H(H + D_w)^2)$ computations which is the most time-consuming step and dominates the total runtime. Similarly, the computational cost of our algorithm for local weights can be derived and has a total cost of $\mathcal{O}(H^3)$.

### 5.3.2. Measured Runtime

We visualize in Fig. 5.2 the wallclock time of our method for approximating the mean and covariance of the observation $y_{t+1}$ conditioned on the mean and covariance of the latent state $x_t$ at the prior time step. Additionally, we visualize the runtime of the MC baseline with different sampling strategies as a function of the dimensionality $D = D_x = D_y = H$. The early stops indicate when we run out of memory. We conduct the experiment on a CPU with 32GB memory. For $S = D$ particles, the MC baseline has the same theoretical runtime as our method for local weights. In practice, we observe our method for local weights to be faster than the MC baseline with $S = D$ when we include the runtime of the weight sampling procedure. When we exclude the runtime of the weight sampling procedure, our method is faster for $D > 64$. Furthermore, our method for global weights

is slower and runs out of memory earlier than all baselines. We leave optimizing the runtime of our method for global weights as a direction for future work.




(a) MC runtime including sampling.

(b) MC runtime measured without sampling.

Figure 5.2.: We visualize the runtime of approximating mean $\mu_{t+1}^y$ and covariance $\Sigma_{t+1}^y$ of the observation $y_{t+1}$ conditioned on the augmented state $z_t$ at the prior time step with mean $\mu_t^z$ and covariance $\Sigma_t^z$. We vary on the x-axis the dimensionality $D$. We use the same dimensionality for the observation $y_t$ and latent state $x_t$, i.e., $D_x = D_y = D$. We use randomly initialized transition and emission models with one hidden layer of width $H = D$. The solid/dashed line represents the runtime of our deterministic approximation for local/global weights. The colored lines represent the runtime of the MC approximation with varying number of particles $S$ as a function of dimensionality $D$. In the left panel, we take into account the runtime of the weight sampling procedure for the MC baseline. In the right panel, we ignore the runtime of the weight sampling procedure.

## 5.4. Experiments

By taking model and data uncertainty into account, ProDSSMs can produce flexible and well-calibrated predictions over a wide range of scenarios. Core to our algorithm is a new moment matching scheme that can be applied for assumed density approximation (see Sec. 5.2.1) and for Gaussian filtering (see Sec. 5.2.2). In our experiments, we first analyze each of these algorithmic advances in isolation before putting everything together.

For this, we first explore in Sec. 5.4.1 our assumed density approximation in the context of stochastic recurrent layers on eight UCI datasets. Then, we study our approximation to the Gaussian Filter in Sec. 5.4.2 on a nonlinear filtering task. We connect both steps and benchmark our full method in Sec. 5.4.3 on two well-established dynamical modeling datasets. Finally, we summarize our empirical findings in Sec. 5.5.

### 5.4.1. Stochastic Recurrent Layers

We first demonstrate the usefulness of our uncertainty propagation scheme as proposed in Sec. 5.2.1 on a regression task with inputs $x \in \mathbb{R}^{D_x}$ and outputs $y \in \mathbb{R}$, similarly to the experiment in Sec. 3.5.2. We interpret the input as the latent state at the initial time step, $x = x_0$. Conditioned on the initial latent state, we can calculate the predictive distribution $p(y|x_0, \phi)$ as

$$p(y|x_0, \phi) = \int p(y|x_T)p(x_T|x_0, w_0)p(w_0|\phi)dw_0, x_T. \tag{5.42}$$

The transition kernel $p(x_T|x_0, w_0)$ is defined by Eq. 5.3, and the emission model $p(y|x_T)$ follows Eq. 5.5. The mapping from $x_0$ to $x_T$ can be interpreted as a deep stochastic layer. As the latent state is given, the filtering step of our algorithm becomes futile.

The dataset $\mathcal{D} = \{(x_0^n, y^n)\}_{n=1}^N$ consists of $N$ input-output tuples. The likelihood term $p(\mathcal{D}|\phi)$ in Eq. 5.23 is given by

$$p(\mathcal{D}|\phi) = \prod_{n=1}^{N} p(y^n|x_0^n, \phi), \tag{5.43}$$

where $p(y^n|x_0^n, \phi)$ follows Eq. 5.42.

#### Datasets

We use the same datasets as in Sec. 3.5.2. In short, we use eight UCI datasets with varying input dimensionality and size. We follow Sec. 3.5.2 for the design of the network architecture. The mean/covariance functions are neural nets with one hidden layer and 40/10 hidden units. The observation function is a single linear layer.

### Baselines

We compare different variants of our method ProDSSM and provide benchmarks against the baselines from Sec. 3.5.2.

*(i) ProDSSM variants*:

- Det. vs MC: We may approximate Eq. 5.42 either via *Monte Carlo* (MC) simulation or by using our *Deterministic* (Det.) method that we introduced in Sec. 5.2.1. We vary the number of particles, i.e., MC simulations, during training and test time.
- Local vs. Global: In the local approach, the weights are resampled at each step. Contrarily, the weights are sampled once at the initial step and then kept constant throughout the remaining steps in the global approach (see. Eq. (5.4)).

*(ii) DGTM*: Our model from Sec. 3.5.2.

*(iii) Dropout* [72]: This method uses a single feed-forward neural net to predict the output, i.e., it does not rely on continuous depth layers. Stochasticity is introduced by applying a Bernoulli distributed masking scheme in all affine layers.

*(iv) DVI* [44]: This method proposes a deterministic inference scheme for Bayesian neural nets. Uncertainty is introduced by allowing for weight uncertainty over the neural net weights. Similarly as Dropout, this method uses a feed-forward neural net.

### Results

We report the *Negative Log-Likelihood* (NLL) in Tab. 5.1, and the *Root Mean Squared Error* (RMSE) in Tab. 5.2.

First, we compare the local and the global weight approach using our deterministic approximation scheme. For five datasets, the differences between both methods are less than one standard error. For the remaining three datasets, the global variant did not converge within the time limit of 72 hours, and it is therefore outperformed by its local alternative. We use a time limit for the training runs in order to limit our carbon footprint. This is motivated by the high computational cost of the deterministic approximation for the global weight setting. The time limit is a multiple of 24 and at least $10\times$ the runtime of the deterministic approximation for the local setting. For training, we use a NVIDIA Tesla V100 with 32GB.

Table 5.1.: Negative log-likelihood for 8 datasets. We report average and standard error over 20 runs. Results marked by * did not converge within 72 hours.

| | Boston | Energy | Concrete | Wine Red | Kin8nm | Power | Naval | Protein |
|---|---|---|---|---|---|---|---|---|
| Dropout | 2.46(0.06) | 1.99(0.02) | 3.04(0.02) | 0.93(0.01) | -0.95(0.01) | **2.80(0.01)** | -3.80(0.01) | 2.89(0.00) |
| DVI | 2.41(0.02) | 1.01(0.06) | 3.06(0.01) | **0.90(0.01)** | -1.13(0.00) | **2.80(0.00)** | **-6.29(0.04)** | 2.85(0.00) |
| DGTM | 2.37(0.03) | 0.70(0.06) | **2.92(0.02)** | 0.93(0.02) | -1.22(0.00) | **2.80(0.01)** | -4.45(0.02) | **2.76(0.01)** |
| ProDSSM: MC, Local | | | | | | | | |
| Train: 8 \|\| Test: 32 | 2.42(0.03) | 0.47(0.03) | 3.02(0.02) | 0.96(0.01) | -1.25(0.00) | 2.85(0.01) | -5.88(0.09) | 2.86(0.01) |
| Train: 8 \|\| Test:128 | 2.41(0.02) | **0.44(0.03)** | 3.01(0.03) | 0.95(0.01) | -1.28(0.00) | 2.83(0.01) | -5.91(0.08) | 2.84(0.01) |
| Train: 32 \|\| Test: 32 | 2.38(0.03) | 0.47(0.06) | 3.06(0.03) | 0.95(0.01) | -1.27(0.01) | 2.82(0.01) | -6.08(0.07) | 2.81(0.01) |
| Train: 32 \|\| Test:128 | 2.37(0.02) | **0.43(0.04)** | 2.99(0.01) | 0.93(0.01) | **-1.29(0.01)** | **2.79(0.01)** | -6.10(0.07) | **2.77(0.01)** |
| Train: 128 \|\| Test: 32 | 2.42(0.04) | **0.45(0.05)** | 3.09(0.04) | 0.96(0.01) | -1.26(0.01) | 2.83(0.01) | -6.15(0.07) | 2.83(0.01) |
| Train: 128 \|\| Test:128 | **2.36(0.03)** | **0.42(0.04)** | 3.00(0.03) | 0.93(0.01) | **-1.30(0.01)** | **2.79(0.01)** | -6.17(0.07) | **2.77(0.01)** |
| ProDSSM: MC, Global | | | | | | | | |
| Train: 8 \|\| Test: 32 | 2.49(0.02) | 0.56(0.03) | 3.08(0.02) | 0.96(0.01) | -1.22(0.01) | 2.85(0.01) | -6.16(0.05) | 2.89(0.01) |
| Train: 8 \|\| Test:128 | 2.46(0.02) | 0.54(0.03) | 3.06(0.01) | 0.94(0.01) | -1.24(0.01) | 2.83(0.01) | -6.19(0.05) | 2.87(0.01) |
| Train: 32 \|\| Test: 32 | 2.50(0.06) | 0.52(0.06) | 3.08(0.02) | 0.96(0.01) | -1.22(0.01) | 2.84(0.01) | -6.18(0.07) | 2.81(0.01) |
| Train: 32 \|\| Test:128 | 2.44(0.05) | 0.50(0.06) | 3.03(0.02) | 0.93(0.01) | -1.25(0.01) | 2.81(0.01) | -6.22(0.07) | **2.77(0.01)** |
| Train: 128 \|\| Test: 32 | 2.44(0.04) | 0.54(0.05) | 3.10(0.04) | 0.97(0.02) | -1.22(0.01) | 2.83(0.01) | **-6.28(0.05)** | 2.82(0.01) |
| Train: 128 \|\| Test:128 | 2.41(0.04) | 0.50(0.05) | 3.03(0.02) | 0.93(0.01) | -1.25(0.01) | **2.80(0.01)** | **-6.30(0.04)** | **2.77(0.01)** |
| ProDSSM: Det., Local | **2.33(0.03)** | **0.43(0.04)** | 3.00(0.03) | 0.92(0.01) | **-1.30(0.00)** | **2.79(0.01)** | -5.52(0.03) | **2.76(0.01)** |
| ProDSSM: Det., Global | **2.34(0.02)** | **0.44(0.03)** | 2.99(0.04) | 0.92(0.00) | -1.27(0.01)* | **2.79(0.01)** | -4.75(0.08)* | 2.82(0.01)* |

Next, we compare the local and global weight approach using an MC approximation and varying the number of particles. We observe lower NLL and RMSE as we increase the number of particles. In order to achieve good predictive performance, a high number of particles is required. The local variant is in five datasets, while the global variant is only in three datasets among the best-performing methods. We conjecture that the difference in performance can be attributed to the higher gradient variance for the global variant, which makes training more difficult.

Using 128 MC samples and focusing on the local weight variant, MC sampling and our deterministic approximation perform en par except for the Naval dataset. There is little uncertainty in the Naval dataset, and the better performance of the MC variant can most likely be attributed to numerical issues. Our deterministic approximation is computationally more efficient, and restricting the MC approach to the same computational budget would result in approximately 12 samples, which is insufficient for good performance.

Table 5.2.: RMSE for 8 datasets. We report average and standard error over 20 runs. Results marked by * did not converge within 72 hours.

| | Boston | Energy | Concrete | Wine Red | Kin8nm | Power | Naval | Protein |
|---|---|---|---|---|---|---|---|---|
| Dropout | **2.97(0.19)** | 1.66(0.04) | 5.23(0.12) | **0.62(0.01)** | 0.10(0.00) | **4.02(0.04)** | **0.01(0.00)** | **4.36(0.01)** |
| DVI | - | - | - | - | - | - | - | - |
| DGTM | 3.26(0.15) | 0.87(0.13) | **5.12(0.09)** | **0.63(0.00)** | 0.08(0.00) | 4.07(0.03) | **0.01(0.00)** | 4.45(0.00) |
| ProDSSM: MC, Local | | | | | | | | |
| Train: 8 \|\| Test: 32 | 3.17(0.14) | 0.43(0.01) | 5.48(0.10) | 0.64(0.00) | **0.07(0.00)** | 4.14(0.03) | **0.01(0.00)** | 4.63(0.02) |
| Train: 8 \|\| Test:128 | **3.14(0.13)** | **0.42(0.03)** | 5.36(0.11) | 0.64(0.00) | **0.07(0.00)** | 4.07(0.03) | **0.01(0.00)** | 4.56(0.02) |
| Train: 32 \|\| Test: 32 | **3.11(0.13)** | **0.41(0.01)** | 5.48(0.11) | **0.63(0.01)** | **0.07(0.00)** | 4.04(0.03) | **0.01(0.00)** | 4.46(0.01) |
| Train: 32 \|\| Test:128 | **3.11(0.13)** | **0.41(0.01)** | 5.43(0.11) | **0.63(0.00)** | **0.07(0.00)** | **4.00(0.03)** | **0.01(0.00)** | 4.39(0.01) |
| Train: 128 \|\| Test: 32 | **3.05(0.12)** | **0.41(0.01)** | **5.21(0.08)** | 0.64(0.01) | **0.07(0.00)** | 4.04(0.03) | **0.01(0.00)** | 4.44(0.02) |
| Train: 128 \|\| Test:128 | **3.04(0.12)** | **0.41(0.01)** | **5.18(0.09)** | **0.63(0.00)** | **0.07(0.00)** | **4.00(0.03)** | **0.01(0.00)** | **4.37(0.02)** |
| ProDSSM: MC, Global | | | | | | | | |
| Train: 8 \|\| Test: 32 | 3.39(0.13) | 0.47(0.01) | 5.66(1.00) | 0.64(0.00) | **0.07(0.00)** | 4.15(0.03) | **0.01(0.00)** | 4.70(0.04) |
| Train: 8 \|\| Test:128 | 3.27(0.12) | 0.46(0.01) | 5.60(0.10) | **0.63(0.00)** | **0.07(0.00)** | 4.09(0.03) | **0.01(0.00)** | 4.62(0.03) |
| Train: 32 \|\| Test: 32 | 3.17(0.14) | 0.45(0.02) | 5.50(0.10) | 0.64(0.00) | **0.07(0.00)** | 4.09(0.03) | **0.01(0.00)** | 4.44(0.01) |
| Train: 32 \|\| Test:128 | **3.15(0.13)** | 0.44(0.02) | 5.44(0.10) | **0.63(0.00)** | **0.07(0.00)** | 4.04(0.03) | **0.01(0.00)** | 4.39(0.01) |
| Train: 128 \|\| Test: 32 | **3.16(0.12)** | 0.46(0.01) | 5.53(0.07) | 0.64(0.00) | **0.07(0.00)** | 4.05(0.03) | **0.01(0.00)** | 4.40(0.01) |
| Train: 128 \|\| Test:128 | **3.14(0.12)** | 0.45(0.01) | 5.46(0.08) | **0.63(0.00)** | **0.07(0.00)** | **4.01(0.03)** | **0.01(0.00)** | **4.36(0.02)** |
| ProDSSM: Det., Local | **2.99(0.13)** | **0.41(0.01)** | 5.24(0.12) | **0.63(0.00)** | **0.07(0.00)** | **3.99(0.03)** | **0.01(0.00)** | **4.35(0.02)** |
| ProDSSM: Det., Global | **3.05(0.11)** | **0.42(0.02)** | 5.24(0.14) | **0.63(0.00)** | **0.07(0.00)*** | **4.01(0.03)** | **0.01(0.00)*** | 4.57(0.03)* |

Lastly, we compare our method against established baselines. ProDSSM with local weights is for five out of eight datasets among the best-performing methods in terms of NLL, thereby outperforming its competitors. We observe ProDSSM to outperform the DGTM baseline that we introduced in Sec. 3.5.2. For the DGTM baseline, we relied on Dropout layers to incorporate model uncertainty. Consequently, the DGTM baseline uses a single parameter, the Dropout rate, to account for model uncertainty. In contrast, for ProDSSM, we model a distribution over each weight with two parameters: mean and variance. The improved predictive performance of ProDSSM, as compared to DGTM, can be attributed to its capability to capture model uncertainty more accurately.

### 5.4.2. Filtering

Next, we benchmark our new moment matching propagation scheme from Sec. 5.2.2 for Gaussian filters on a standard filtering task.

#### Datasets

In order to ensure that this experiment only evaluates the performance with respect to the filtering task, we use a two-step approach for creating data. In the first step, we create probabilistic ground-truth models; in the second step, we apply our newly created models in order to generate data for the filtering task.

*Step 1*: We first train DSSM and our two ProDSSM variants on the kink dataset, which describes a nonlinear dynamical system with varying emission noise $s = \{0.008, 0.08, 0.8\}$. See also Sec. 5.4.3 for more details. In total, we obtain nine trained models with three different emission noise levels and three different model variants.

*Step 2*: For each trained model, we construct a new dataset by simulating trajectories with a (Pro-)DSSM. Each trajectory has length $T = 120$, and we simulate 10 trajectories per dataset. We assess the performance by calculating the NLL and RMSE of observing the true latent state on these nine datasets. The transition and emission models in this experiment are thereby fixed to the ground truth dynamics from Step 1.

#### Baselines

We benchmark our filtering algorithm against two established baselines.

*(i) Unscented Kalman Filter* (UKF): This filter and our method share similarities as they are both instances of the Gaussian filter (see. Sec. 2.3). In contrast to our moment propagation approach, the intractable integrals are solved by using the unscented transform that is a numerical integration scheme [22].

*(ii) Neural Filter* (NF): In DSSM literature [27], it is common practice to train a neural net based filter with the generative model by maximizing ELBO. During training, we fix the transition and emission model to the ground truth from Step 1. We follow [26] for network design and use a recurrent neural net architecture that produces a sample $x_t$ at each time step $t$ as a function of the prior latent state $x_{t-1}$ and the observation $y_t$.

### Results

We report results in Tab. 5.3. We observe for all methods that with increasing emission noise, it becomes more difficult to infer the latent distribution from the observations. For deterministic weights, our method performs on par with UKF, while NF is outperformed for medium and higher noise levels.

When switching to probabilistic weight modeling methods, the UKF has higher RMSE and NLL compared to our deterministic method for middle and high emission noise. Increasing the emission noise makes learning the dynamics more challenging and, as a result, leads to higher weight uncertainties. We can also observe this behavior empirically: For low/middle/high observation noise, the average variance of the weights is $0.10/0.29/0.60$ for local weights and $0.05/0.26/0.49$ for global weights. As a consequence, the integration steps in the Gaussian filter become more difficult for increasing noise levels, and the performance of the UKF method deteriorates. In contrast, our newly introduced moment matching scheme performs well across the complete range of noise levels. This observation aligns with our findings from Chap. 3, where we observed that the accuracy of the BMM algorithm is not harmed by an increasing dimensionality. We can interpret the uncertainty over the weights as an increase in dimensionality.

Table 5.3.: NLL and MSE on a nonlinear filtering dataset. We report average and standard error over 10 runs.

| | | $s = 0.008$ | | $s = 0.08$ | | $s = 0.8$ | |
|---|---|---|---|---|---|---|---|
| | | MSE | NLL | MSE | NLL | MSE | NLL |
| Det. | NF | **0.01(0.00)** | **-0.87(0.08)** | **0.08(0.01)** | 0.25(0.10) | 0.73(0.19) | 1.23(0.11) |
| | UKF | **0.01(0.00)** | **-0.89(0.05)** | **0.07(0.00)** | **0.09(0.05)** | **0.49(0.08)** | **1.03(0.08)** |
| | Ours | **0.01(0.00)** | **-0.88(0.04)** | **0.07(0.01)** | **0.10(0.04)** | **0.46(0.07)** | **1.00(0.08)** |
| Loc. | UKF | **0.02(0.01)** | **-0.85(0.06)** | 0.12(0.02) | 0.64(0.20) | 1.35(0.15) | 3.10(0.46) |
| | Ours | **0.01(0.00)** | **-0.90(0.02)** | **0.07(0.03)** | **0.08(0.03)** | **0.44(0.05)** | **1.00(0.05)** |
| Glob. | UKF | **0.01(0.00)** | **-0.91(0.01)** | 0.27(0.06) | 2.89(0.83) | 1.18(0.25) | 3.56(0.89) |
| | Ours | **0.01(0.00)** | **-0.89(0.02)** | **0.06(0.02)** | **0.12(0.03)** | **0.48(0.04)** | **0.98(0.06)** |

### 5.4.3. Dynamical System Modeling

Our proposed model family, ProDSSM, is a natural choice for dynamical system modeling, where we aim to learn the underlying dynamics from a dataset $\mathcal{D} = \{Y^n\}_{n=1}^N$ consisting of $N$ trajectories. For simplicity, we assume that each trajectory $Y^n = \{y_t^n\}_{t=1}^T$is of length $T$. Using the chain rule, the likelihood term $p(\mathcal{D}|\phi)$ in Eq. (5.23) can be written as

$$p(\mathcal{D}|\phi) = \prod_{n=1}^{N} \prod_{t=1}^{T-1} p(y_{t+1}^n | y_{1:t}^n, \phi), \tag{5.44}$$

where we can approximate the predictive distribution $p(y_{t+1}^n|y_{1:t}^n, \phi)$ in a deterministic way as discussed in Sec. 5.2.4.

#### Datasets

We benchmark our method on two different datasets. The first dataset is a well-established learning task with synthetic nonlinear dynamics, and the second dataset is a challenging real-world dataset.

*(i) Kink* [122]: We construct three datasets with varying degrees of difficulty by varying the emission noise level. The transition model is given by $\mathcal{N}(x_{t+1}|f_{kink}(x_t), 0.05^2)$ where $f_{kink}(x_t) = 0.8 + (x_t + 0.2)[1 - 5/(1 + e^{-2x_t})]$ is the kink function. The emission model is defined as $\mathcal{N}(y_t|x_t, s)$, where we vary $s$ between $\{0.008, 0.08, 0.8\}$. We simulate for each value of $s$ 10 trajectories of length $T = 120$. We follow the experimental protocol as defined in [124] and perform 10 training runs where each run uses data from a single simulated trajectory only. The mean function is realized with a neural net with two hidden layers and 50 hidden units, and the variance is a trainable constant. For MC based ProDSSM variants, we use 64 samples during training. The cost of our deterministic approximation for the local approach is ≈50 samples. We compare the performance of the different methods with respect to model uncertainty, i.e., epistemic uncertainty, by evaluating if the learned transition model $p(x_{t+1}|x_t)$ covers the ground-truth dynamics. In order to calculate NLL and MSE, we place 70 evaluation points on an equally spaced grid between the minimum and maximum latent state of the ground truth time series and approximate for each point $x_t$ the mean $\mathbb{E}[x_t] = \int f(x_t, w_t)p(w_t)dw_t$ and variance $\mathrm{Cov}[x_t] = \int (f(x_t, w_t) - \mathbb{E}[x_t])^2 p(w_t)dw_t$ using 256 Monte Carlo samples.

This dataset is commonly used for benchmarking GP based dynamical models [122, 124]. To the best of our knowledge, it has not been used in the context of DSSMs prior to this work.

*(ii) Mocap*: We follow [119] for preprocessing and designing the experimental setup. We extract the dataset from the CMU motion capture library[1]. It consists of 23 sequences from a single person. We use 16 sequences for training, 3 for validation, and 4 for testing. Each sequence consists of measurements from 50 different sensors. We follow [119] for designing the network architecture and training process, i.e., we use only the first three observations to infer the latent distribution. For MC based ProDSSM variants, we use 32 samples during training and 256 during testing. The cost of our deterministic approximation for the local approach is approximately 24 samples. For numerical comparison, we compute NLL and MSE on the test sequences.

### Baselines

We use the same ProDSSM variants as in our deep stochastic layer experiment (Sec. 5.4.1). Additionally, we compare the Gaussian filter with the embedding model in order to infer the latent state. Furthermore, we compare against well-established baselines from GP and neural net based dynamical modeling literature.

*(i) ProDSSM variants*:

- GF vs. EM: We compare two approaches for modeling the latent state distribution. The first approach uses the *Gaussian Filtering* (GF) algorithm, as proposed in Sec. 5.2.2. The second approach involves a neural network based *Embedding Model* (EM). These embedding models have been introduced in Chap. 4 for traffic forecasting applications.

*(ii) DSSM*: This method is equal to ProDSSM when removing the weight uncertainty. We use our deterministic moment matching algorithm together with Gaussian Filtering to train the DSSM model.

*(iii) VCDT* [122]: This method relies on GPs to model a SSM. The distribution of the latent state is forward propagated via sampling. Training is performed using doubly stochastic variational inference jointly over the GP posterior and the latent states.

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

*(iv) Laplace GP* [124]: A GP based dynamical model that applies stochastic variational inference for the Gaussian process posterior and the Laplace approximation over the latent states.

*(v) ODE2VAE* [119]: The dynamics are modeled as latent neural ordinary differential equations. Stochasticity is introduced by accounting for uncertainty over the weights. An additional neural net is used to approximate the latent distribution of the initial latent state, and the model does not account for transition noise.

*(vi) E-PAC-Bayes-Hybrid* [50]: The dynamics is modeled as a neural stochastic differential equation and accounts for data and model uncertainty. Marginalization over the latent states and the weights is performed using Monte Carlo sampling. This method focuses on integrating prior knowledge, either in the form of physics or by transfer learning across similar tasks, into the dynamics, hence the term *Hybrid*. For the kink dataset, we reimplement this method without using prior knowledge.

### Results

First, we analyze the results on the kink dataset. We visualize the learned mean function of the transition model in Fig. 5.3. The confidence intervals capture the mean of the true transition model. The model uncertainty increases with increasing noise levels. We observe increased uncertainty for $x_t < -3$ and $x_t > 1$. It is important to note that these regions lie outside of the training data. This finding suggests that our model accurately captures model uncertainty, demonstrating its reliability in scenarios beyond the training distribution.

We present the numerical results of this benchmark in Tab. 5.4. For low ($s = 0.008$) and middle emission noise ($s = 0.08$), all of our ProDSSM variants achieve on-par performance with existing GP based dynamical models and outperform ODE2VAE. For high emission noise ($s = 0.8$), our ProDSSM variants perform significantly better than previous approaches. The MC variants achieve for low and middle noise levels the same performance as the deterministic variants. As the noise is low, there is little function uncertainty, and few MC samples are sufficient for accurate approximations of the moments. If the emission noise is high, the marginalization over the latent states and the weights becomes more demanding, and the MC variant is outperformed by its deterministic counterpart. Furthermore, we observe that for high observation noise, the local weight variant of our ProDSSM model achieves lower NLL than the global variant. We cannot report results for DSSM since this model does not account for model uncertainty.

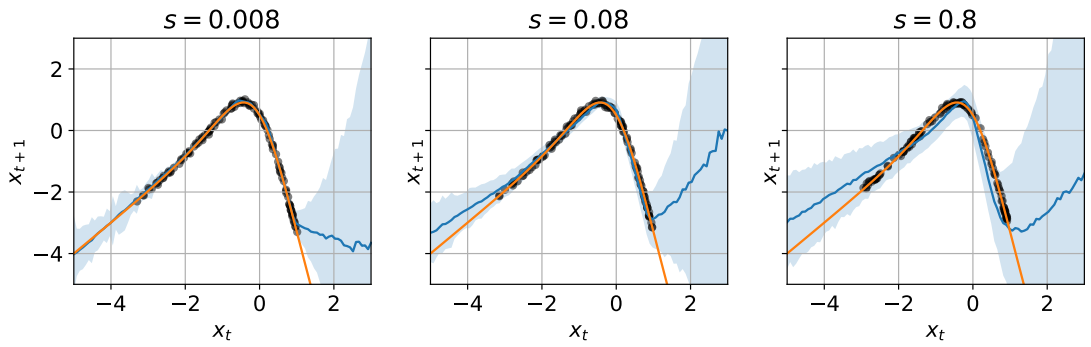



Figure 5.3.: For increasing noise level $s$, we observe increased model uncertainty. We visualize the true mean function $f(x_t)$ as an orange solid line and the latent states as black dots. The blue solid line is the expected value of the learned mean function, and the shaded area represents the 95% confidence interval.

On the Mocap dataset, our best-performing ProDSSM variant, which is the local weight variant with deterministic inference, outperforms all baselines. Notably, it even outperforms E-PAC-Bayes-Hybrid, which uses an additional dataset from another motion-capture task in order to learn the prior distribution over the neural network weights. Furthermore, we observe our best-performing ProDSSM variant to outperform its non-probabilistic counterpart, denoted as DSSM. For a fair comparison, we trained DSSM with our deterministic moment matching algorithm and used the Gaussian filter to infer the distribution over the latent states. This configuration closely resembles our top-performing ProDSSM variant. We further note that the ProDSSM variant with global weights and deterministic inference was not able to converge within the time limit.

When comparing the kink dataset and the Mocap dataset, the differences between the MC and deterministic ProDSSM variants become more pronounced. The Mocap dataset has a higher dimensionality, requiring a larger number of MC samples for accurate approximations.

Furthermore, we observe that in both datasets, replacing the Gaussian filtering algorithm with a neural embedding model to infer the latent state distribution from observations leads to a deterioration in performance.

Table 5.4.: NLL and MSE for different dynamical system modeling tasks. We report average and standard error over 10 runs. Results marked by * did not converge within 120 hours.

| | Kink | | | | | | Mocap | |
|---|---|---|---|---|---|---|---|---|
| | $s = 0.008$ | | $s = 0.08$ | | $s = 0.8$ | | | |
| | MSE | NLL | MSE | NLL | MSE | NLL | MSE | NLL |
| VCDT | - | **-1.53(0.31)** | - | 1.10(0.72) | - | 4.16(1.97) | - | - |
| Laplace GP | - | **-1.35(0.04)** | - | -0.36(0.08) | - | 1.08(0.15) | - | - |
| ODE2VAE | 0.01(0.00) | -0.07(0.46) | **0.02(0.00)** | 0.29(0.43) | **0.22(0.05)** | 4.11(1.42) | 8.09(0.62) | - |
| E-PAC-Bayes-Hybrid | 0.02(0.00) | -0.67(0.07) | 0.05(0.00) | 0.47(0.11) | 0.41(0.07) | 1.13(0.12) | 7.84(0.44) | 253.64(20.01) |
| DSSM | - | - | - | - | - | - | 7.87(0.69) | 64.65(1.27) |
| ProDSSM variants | | | | | | | | |
| GF, MC, Local | **0.00(0.00)** | **-1.46(0.04)** | 0.04(0.01) | **-0.44(0.06)** | 0.28(0.05) | 0.82(0.11) | 10.36(0.67) | 74.74(1.68) |
| GF, MC, Global | **0.00(0.00)** | **-1.50(0.07)** | 0.04(0.01) | **-0.46(0.10)** | 0.31(0.05) | 1.13(0.24) | 10.65(1.25) | 71.42(1.70) |
| GF, Det., Local | **0.00(0.00)** | **-1.50(0.03)** | 0.04(0.01) | **-0.41(0.07)** | **0.22(0.04)** | **0.54(0.07)** | **6.98(0.17)** | **61.99(0.53)** |
| GF, Det., Global | **0.00(0.00)** | **-1.53(0.03)** | 0.03(0.01) | **-0.47(0.07)** | **0.22(0.05)** | 0.72(0.17) | 21.29(0.82)* | 67.34(1.28)* |
| EM, Det., Local | **0.00(0.00)** | -0.47(0.22) | **0.02(0.00)** | -0.11(0.29) | **0.24(0.04)** | 1.26(0.37) | 9.83(1.43) | **61.85(1.27)** |

## 5.5. Pros and Cons of Different ProDSSM Variants

In this section, we discuss the advantages and disadvantages of different ProDSSM variants with respect to their theoretical properties as well as our experimental findings.

First, we compare the local and global variants of our approach. In the local variant, we resample the weight at each time step, while, for the global variant, we keep the weights fixed for the complete trajectory. Independently of the chosen inference scheme, our experiments did not find a clear winner, provided that both variants converged. However, the local variant is mathematically more convenient as it decorrelates subsequent time steps. This property can be exploited for sample-free inference, where it results in a lower computational burden. Our empirical evidence confirms that this variant leads to more feasible solutions, whereas the global alternative is much slower and often did not converge in a reasonable amount of time.

Focusing on the local approach, we can observe that our moment matching inference scheme outperforms its MC counterpart when using the same computational budget.

Disregarding runtime constraints, the MC variant still fails to surpass the performance of its deterministic alternative, indicating that (i) the Gaussian assumption is appropriate and (ii) the approximation error of our propagation scheme is negligible.

Despite the increased computational complexity of the global approach, we believe it warrants further exploration due to its ability to facilitate uncertainty decomposition [18], i.e., allowing for the separation of data and model uncertainty. In contrast, the local approach does not support uncertainty decomposition, as both sources of uncertainty are intertwined at each time step. Moreover, the global approach could be beneficial when transitioning from discrete to continuous dynamical systems. In such cases, it is desirable to achieve a parsimonious solution that works well across various numerical solvers and step sizes.

Lastly, we observed the Gaussian filtering algorithm to outperform the neural embedding model in two different experiments. However, incorporating contextual information, for example, map information, into the Gaussian filtering algorithm is not a straightforward task.

## 5.6. Summary

In this chapter, we proposed ProDSSMs, a general framework for modeling unknown dynamical systems that respect data and model uncertainty. Inference for this model class is hard since we need to propagate the uncertainty over the neural network weights and the latent states along a trajectory. We addressed this challenge by introducing a novel inference scheme that exploits the internal structure of ProDSSMs and enjoys sample-free inference. Our algorithm is general and can be applied to a variety of tasks and account for different weight sampling strategies.

In our experiments, we observed that our deterministic algorithm with local weights achieves better predictive performance in terms of lower NLL and MSE than its sampling-based counterpart under a fixed computational budget. Furthermore, we observed ProDSSMs to outperform their non-probabilistic counterparts in terms of predictive performance.

# 6. Conclusion

This thesis presented several extensions to the DSSM framework, aiming to address the central question that we raised in Sec. 1:

*How can we efficiently model dynamical systems with interacting agents while accounting for both data and model uncertainty?*

Solving this question is crucial as accurately modeling the surrounding environment, i.e., the dynamical system, is an important step towards achieving fully autonomous vehicles. Once the surrounding environment is accurately and reliably modeled, autonomous vehicles can make informed decisions about which actions to take.

In the following, we discuss in Sec. 6.1 how each of our contributions helps to answer the aforementioned question. Additionally, we discuss open problems as well as novel applications beyond autonomous driving in Sec. 6.2.

## 6.1. Summary

We introduced multiple extensions to the DSSM framework that focused on different parts of the aforementioned key question.

In Chap. 3 we addressed the goal of efficiently capturing data uncertainty. We introduced the BMM algorithm that efficiently approximates the predictive distribution of a DGTM as a Gaussian. A DGTM represents the transition model of a DSSM. The BMM algorithm established the foundation for modeling DSSMs and opened the door for more advanced assumed density approximations beyond the simple unimodal Gaussian in the later chapters. We demonstrated that the BMM algorithm achieved excellent predictive performance on various tasks, such as time series classification and dynamical system modeling, compared to its Monte Carlo variant as well as established baselines with low

computational cost. Additionally, we compared the BMM algorithm to standard numerical integration techniques and found it to be more accurate and faster. Notably, unlike standard numerical integration methods, the accuracy of the BMM algorithm improves as the input dimensionality and network complexity increase.

In Chap. 4 our primary focus was modeling interacting agents. If we ignore the interactions between different agents, our forecasts will be inaccurate. We introduced GNNs as a building block for DSSMs and derived moment matching rules for commonly used GNN layers in order to make the BMM algorithm applicable. Beyond that, we enhanced the efficiency of the BMM algorithm by introducing sparse covariance approximations. Furthermore, we improved the accuracy of our model by representing the predictive distribution of a DSSM as a GMM instead of a single Gaussian. By adopting a GMM, we effectively account for the multimodal nature of traffic forecasting. For instance, it enables us to capture different hypotheses about driver intentions, resulting in a more accurate representation of the data uncertainty. We conducted experiments on two real-world traffic forecasting datasets to evaluate the performance of our proposed model and our extensions to the BMM algorithm. The results demonstrated the strong performance of our approach.

In Chap. 5 we focused on the task of incorporating model uncertainty into the DSSM framework. To do so, we introduced weight uncertainty to the transition model of the DSSM. Weight uncertainty allows us to model the uncertainty stemming from the lack of knowledge, known as epistemic uncertainty. By accounting for both data uncertainty and model uncertainty, our model becomes more accurate, resulting in a lower NLL. Additionally, we observed an increase in model uncertainty for inputs that were not part of the training data during the dynamical system modeling experiment on the kink dataset. This increased uncertainty can help to prevent overconfident predictions for novel traffic scenarios that were not encountered during training. To model the distribution over the weights, we used iid Gaussians. We derived moment matching rules for probabilistic layers, enabling efficient inference and making the BMM algorithm applicable to DSSMs with weight uncertainty. A key challenge was accounting for the correlations between the weights and the states when deriving the extended moment matching rules. We found that our algorithm was significantly more accurate than standard numerical integration methods for DSSMs with weight uncertainty. This aligns with the observation from Chap. 3 that the accuracy of the BMM algorithm is not affected by increasing dimensionality, unlike standard numerical integration methods. To evaluate our model and the extensions to the BMM algorithm, we conducted experiments on various tasks, including filtering and dynamical system modeling. The results demonstrated the strong performance of our approach.

In summary, our extensions enable efficient modeling of dynamical systems with interacting agents within the DSSM framework. By considering both data uncertainty and model uncertainty, our model effectively captures the stochastic nature of traffic forecasting, leading to improved predictive performance in terms of lower NLL and RMSE. However, despite these advancements, there are still unresolved problems and questions, which we discuss in the next section.

## 6.2. Outlook

In this section, we discuss open research questions of our proposed framework and present potential solutions. Beyond that, we discuss potential future applications.

*(i) Determining the Number of Modes*: In Chap. 4 we used a neural network to infer the distribution of the initial latent state, assuming a fixed number of modes. However, the appropriate number of modes varies depending on the scenario. For instance, scenarios with a high number of traffic participants require a larger number of modes to effectively model future outcomes compared to scenarios with fewer participants. Therefore, a valuable enhancement to our proposed framework would be to extend the embedding model towards a variable number of modes rather than a fixed number.

*(ii) Contextual Information and the Gaussian Filter*: In Chap. 5 we introduced a novel algorithm to infer the latent state distribution by employing our approximation to the Gaussian filter. Notably, the inference of the latent distribution solely relies on past observations and does not take into account any contextual information. This is in contrast to our approach presented in Chap. 4, where we used a neural network to approximate the latent distribution. This neural network incorporated contextual information, such as the map or interactions between agents. Incorporating contextual information is crucial for generating realistic predictions. Since the embedding model becomes obsolete when using our Gaussian filter, we need to explore alternative methods of including the context information. One potential approach is to include contextual information as an additional input to the transition model.

*(iii) Weight Uncertainty for GNNs*: In Chap. 5 we proposed ProDSSMs that extend DSSMs towards transition models with uncertainty over their weights. We focused on applications with feed-forward neural nets. It remains an open task to extend our ProDSSM framework towards GNNs.

*(iv) Weight Uncertainty beyond the Transition Model*: In Chap. 5 we introduced weight uncertainty in the transition model. However, it is still unclear whether including weight uncertainty in the transition model alone is sufficient or if we also need to consider weight uncertainty in the emission model. Additionally, if we use an embedding model to infer the latent distribution, as we did in Chap. 4, it remains to be explored whether we should incorporate weight uncertainty into the embedding model as well.

*(v) Continuous-Time Systems*: Our proposed BMM algorithm can also be applied to continuous-time systems, such as neural ODEs [53] and SDEs [60]. This extension is particularly useful for applications that involve irregularly sampled observations.

*(vi) Transformers and DSSMs*: Transformers [84] are a popular neural network architecture that has significantly advanced many machine learning applications, including text generation [128] and image generation [129]. An interesting research direction is to extend our BMM algorithm to the transformer architecture, which offers two potential benefits: (i) reduced sample variance during transformer training due to our BMM algorithm and (ii) potentially improved predictive performance for DSSMs by using transformer architectures. In Sec. 3.5.4, we benchmarked DGTMs based on feed-forward neural networks against transformers. However, in this experiment, we did not observe any advantage of using transformers over feed-forward neural networks. It is worth noting that this experiment was performed on a task involving fully observed dynamical systems. For partially observed systems, transformers might provide an advantage through the use of the cross-attention module [84]. For instance, the cross-attention module allows the transition model to pay attention to contextual information such as the map.

*(vii) Applications beyond Traffic Forecasting*: Our proposed BMM algorithm has the potential to be employed as a subroutine in various applications other than traffic forecasting. In the context of variational autoencoders [130], it can replace the sampling step in evidence maximization. Another potential application is using the BMM algorithm for multi-step training of diffusion models [131].

# A. Supplementary Material for Chap. 4

## A.1. Proof of Theorem 1

**Theorem 1.** *The predictive distribution $p(y_t|\mathcal{I})$ is analytically computed as*

$$p(y_t|\mathcal{I}) = \sum_{v=1}^{V} \pi_v(\mathcal{I}) \mathcal{N} \mu_{t,v}^{y}(\mathcal{I}), \Sigma_{t,v}^{y}(\mathcal{I})),$$

*for a GDSSM with the below generative model*

$$\begin{aligned}
v &\sim \textit{Cat}([\pi_1(\mathcal{I}), \dots, \pi_V(\mathcal{I})]), \\
x_0 &\sim \mathcal{N}(\mu_{0,v}(\mathcal{I}), \textit{diag}(\Sigma_{0,v}(\mathcal{I}))), \\
x_t &\sim \mathcal{N}\left(x_t|x_{t-1} + f(t,v,\mathcal{I})x_{t-1}, \textit{diag}\left(r(t,v,\mathcal{I})\right)\right), \\
y_t &\sim \mathcal{N}\left(y_t|g(t,v,\mathcal{I})x_t, \textit{diag}\left(s(t,v,\mathcal{I})\right)\right),
\end{aligned}$$

*where $f(t,v,\mathcal{I}), r(t,v,\mathcal{I}), g(t,v,\mathcal{I}), s(t,v,\mathcal{I})$ are time $t$, component $v$, and context $\mathcal{I}$ depending matrices with appropriate dimensionality.*

*Proof.* The proof is straightforward as the output moment of the transition and emission model are analytically available

$$\begin{aligned}
p(y_t|\mathcal{I}) &= \int p(y_t|v,x_t,\mathcal{I}) p(x_t|v,x_0,\mathcal{I}) p(v,x_0|\mathcal{I}) dv dx_0 dx_t \\
&= \sum_{v=1}^{V} \pi_v(\mathcal{I}) \int p(y_t|v,x_t,\mathcal{I}) p(x_t|v,x_0,\mathcal{I}) \mathcal{N}(\mu_{0,v}(\mathcal{I}), \text{diag}(\Sigma_{0,v}(\mathcal{I}))) dx_0 dx_t \\
&= \sum_{v=1}^{V} \int p(y_t|v,x_t,\mathcal{I}) \mathcal{N}(\mu_{t,v}^{x}(\mathcal{I}), \Sigma_{t,v}^{x}(\mathcal{I})) dx_t
\end{aligned}$$

$$= \sum_{v=1}^{V} \pi_v(\mathcal{I}) \mathcal{N}(\mu_{t,v}^y(\mathcal{I}), \Sigma_{t,v}^y(\mathcal{I})).$$

The moments at time step $t$ of a linear time-depending dynamical system are analytically available as [22]

$$\begin{aligned}
\mu_{t,v}^x(\mathcal{I}) &= \prod_{t'=1}^{t} \left(I + f(t', v, \mathcal{I})\right) \mu_{0,v}^x(\mathcal{I}), \\
\Sigma_{t,v}^x(\mathcal{I}) &= \left[\prod_{t'=1}^{t} (I + f(t', v, \mathcal{I}))\right] \text{diag}(\Sigma_{0,v}^x(\mathcal{I})) \left[\prod_{t'=1}^{t} (I + f(t', v, \mathcal{I}))^\top\right] \\
&\quad + \sum_{t'=1}^{t-1} \left[\prod_{t''=t'+1}^{t} (I + f(t'', v, \mathcal{I}))\right] \text{diag}(r(t', v, \mathcal{I})) \left[\prod_{t''=t'+1}^{t} (I + f(t'', v, \mathcal{I}))^\top\right] \\
&\quad + \text{diag}(r(t, v, \mathcal{I}))
\end{aligned}$$

We obtain the same expression via the BMM algorithm, which is easy to prove by inserting the locally linear system into Eq. 3.3 and 3.5. Finally, mean $\mu_{t,v}^y(\mathcal{I})$ and covariance $\Sigma_{t,v}^y(\mathcal{I})$ of the output at time step $t$ are available as [22]

$$\begin{aligned}
\mu_{t,v}^y(\mathcal{I}) &= \mathbb{E}[g(t, v, \mathcal{I}) x_t] = g(t, v, \mathcal{I}) \mu_{t,v}^x(\mathcal{I}), \\
\Sigma_{t,v}^y(\mathcal{I}) &= \text{Cov}[g(t, v, \mathcal{I}) x_t] + \text{diag}\left(\mathbb{E}[s(t, v, \mathcal{I})]\right) \\
&= g(t, v, \mathcal{I}) \Sigma_{t,v}^x(\mathcal{I}) g(t, v, \mathcal{I})^\top + \text{diag}\left(s(t, v, \mathcal{I})\right).
\end{aligned}$$

□

## A.2. Training Details

We train all models with the ADAM optimizer and stochastic mini-batches. We use a batch size of 4 and a learning rate of 0.0001. In order to accelerate the training of multi-modal GDSSMs we initialize the transition and observation neural nets with the pretrained versions of the uni-modal GDSSMs.

### A.2.1. rounD

We train deterministic models (GDSSM Det. and Non Recur. GNN) for 50k weight updates. For the MC based models (GDSSM MC) we use 100k weight updates. The dataset contains 4,314 training and 1,091 testing snippets. We do not use a separate validation dataset as we observed no overfitting.

### A.2.2. NGSIM

We train all models for 1000k updates on the NGSIM dataset. The dataset contains 5,922k/860k/1,505k train/validation/test snippets. We validate the models on 1k random minibatches from the validation dataset after every 10k weight updates.

### A.2.3. Out-of-Distribution Testing

We train all models for 50k weight updates. The dataset consists of three sub-datasets with 254/ 251/ 137 snippets, where 80% of each sub-dataset are used for training and the remaining 20% for testing. Due to the small size of the sub-datasets we observed overfitting. We address this by using the last 20% of each training sub-dataset for validation. We validate the model after every 1k weight updates.

## A.3. Alternative Parameter Inference Methods

In contrast to Sec. 2.5, we review parameter inference methods for latent dynamical systems, as we used them for our baselines in Sec. 4.3.2. One commonly used inference method in the context of machine learning circumvents maximizing the log-likelihood and instead maximizes the *Evidence Lower Bound* (ELBO)

$$\text{ELBO}(y_1, \dots, y_T|\mathcal{I}) = \mathbb{E}_{q(x_0,\dots,x_T)}\left[\log \frac{p(y_1, \dots, y_T, x_0, \dots, x_T|\mathcal{I})}{q(x_0, \dots, x_T)}\right], \tag{A.1}$$

that involves learning an approximation $q(x_0, \dots, x_T)$ to the intractable smoothing distribution $p(x_0, \dots, x_T|y_1, \dots, y_T, \mathcal{I})$ [26].

A tighter bound to the log-likelihood $\log p(y_1, \dots, y_T|\mathcal{I})$ can be obtained by calculating the importance weighted log-likelihood, which we refer to as the *Monte Carlo Objective* (MCO) [103, 132]

$$\text{MCO}(y_1, \dots, y_T|\mathcal{I}) = \mathbb{E}_{q(x_0,\dots,x_T)}\left[\log \frac{1}{S}\sum_{s=1}^{S} \frac{p(y_1, \dots, y_T, x_{0,s}, \dots, x_{T,s}|\mathcal{I})}{q(x_{0,s} \dots, x_{T,s})}\right], \tag{A.2}$$

where $S$ is the number of Monte Carlo samples and $x_{t,s}$ is the $s$-th sample at time step $t$. For state-space models, recent work combined the MCO with particle filters [102, 104].

## A.4. Experimental Setup for Out-of-Distribution Testing

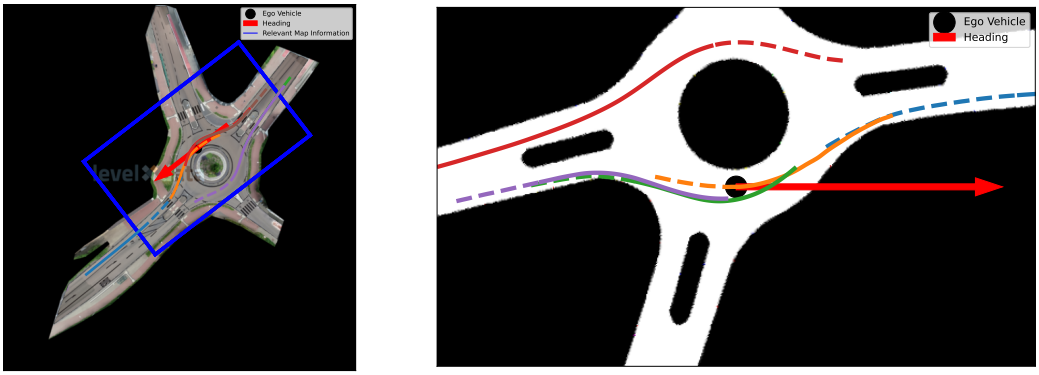
### A.4.1. Dataset Construction

We construct a dataset that consists of three different traffic environments. We select one recording from the roundabout in *Kackertstrasse* (K) in Aachen, one recording from the roundabout in *Thiergarten* (T) in Alsdorf, and one recording from the roundabout in *Neuweiler* (N) near Aachen. We use the same preprocessing procedure as in Sec. 4.3.2. We remove pedestrians, bicycles, and parked vehicles from each traffic environment. There remain 319/264/389 tracked objects over a time span of 0.3/0.3/0.15 hours at the traffic environments K/T/N. We downsample the recordings by a factor of 5 and then construct for each traffic environment a dataset that consists of 8 s long snippets with 50% overlap. The first three seconds are used as the track history, and the following five seconds as the prediction horizon. We obtain 254/251/137 snippets for the traffic environments K/T/N. For each traffic environment, we use the first 80% snippets for training and the remaining 20% snippets for testing.

### A.4.2. Map Processing

The map processing follows existing work in the domain of traffic forecasting [133, 80]. Given an ego-vehicle and its history, we first calculate the heading of the vehicle. The heading is calculated as the average heading direction over the last 0.2 observed seconds. The position and the heading direction jointly define the *Region-of-Interest* (ROI) on the map. The ROI is a rectangle with a length of 74 meters and a width of 44 meters, which is centered at the ego-vehicle and oriented according to the heading of the ego-vehicle.

We further convert the RGB image into a binary road image. We visualize the processed map information in Fig. A.1b



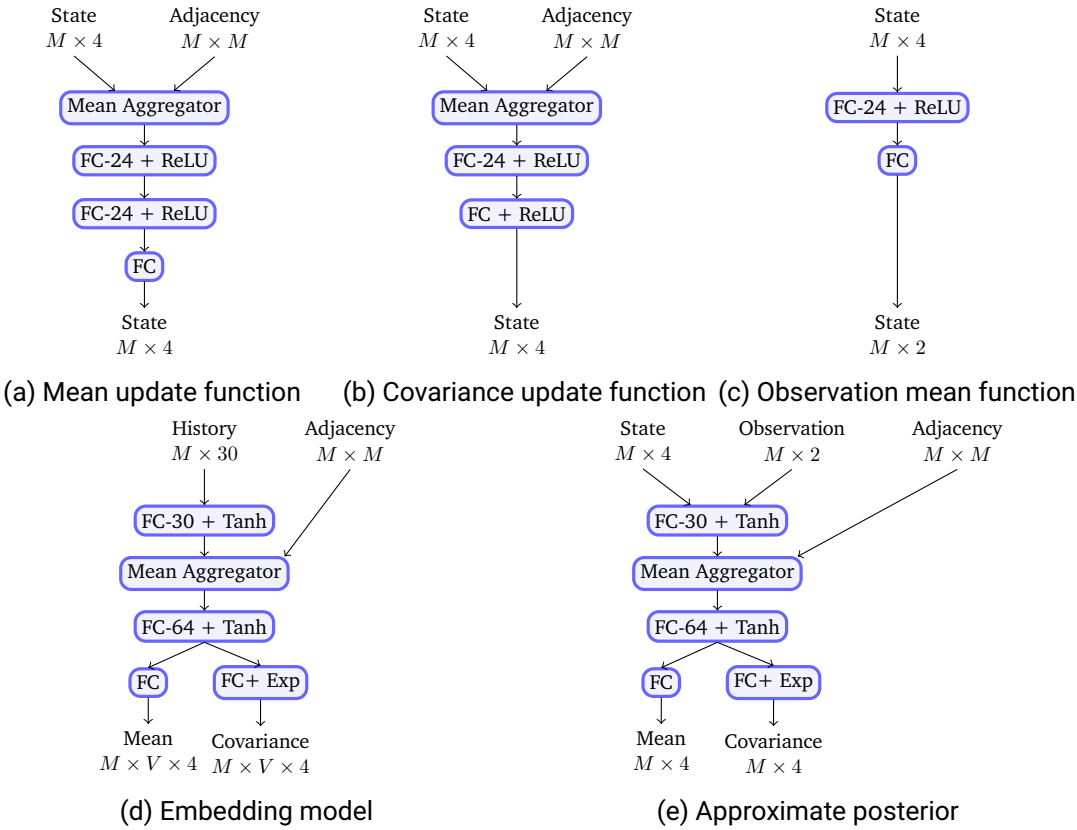

(a) Full map with ego-vehicle.

(b) Rotated, cropped, and masked map information.

Figure A.1.: Processing of map information.

## A.5. Network Architectures

We use neural network architectures without a world model (see Fig. A.2) for the experiments in Sec. 4.3.2, 4.3.3, and 4.3.4.



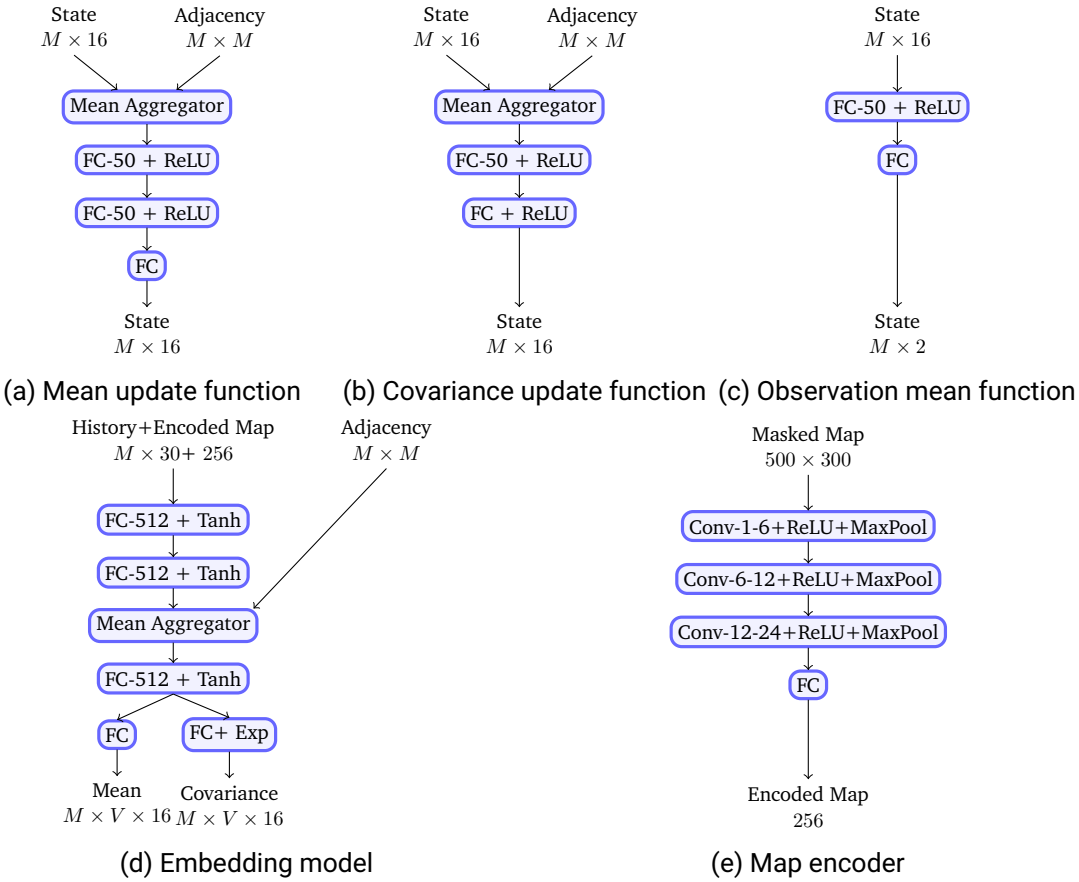

(a) Mean update function (b) Covariance update function (c) Observation mean function

(d) Embedding model (e) Approximate posterior

Figure A.2.: Architectures without map information. For fully connected layers we give the number of output neurons.

The embedding model receives the history of all $M$ agents. This history is 3 seconds long with a time step of $0.2$ seconds and consists of two-dimensional coordinates. The output of the embedding model is a GMM with $V$ mixture components. We use the mean aggregator in the embedding model as well as the mean and covariance update functions. Mean and covariance update functions are neural networks that conduct at each prediction step one round of message passing and then calculate the output. The emission model uses a

neural net for the mean function $g(x_t)$ and a constant vector for $s(x_t)$. The approximate posterior is used in Sec. 4.3.2 for the baselines that are trained on the ELBO or MCO.




(a) Mean update function (b) Covariance update function (c) Observation mean function

(d) Embedding model (e) Map encoder

Figure A.3.: Architectures with map information. For fully connected layers we give the number of output neurons. For convolutional layers we give the number of input channels and output channels.

In the experiments in Sec. 4.3.5, we use an architecture with a world model (see Fig. A.3). It closely follows the architectures in Fig. A.2. We add an additional neural net, which encodes the masked map of size 500x300 into a flattened 256-dimensional vector. This map embedding is used as an additional input to the embedding model.

# B. Supplementary Material for Chap. 5

## B.1. Similarities between ELBO and Predictive Variational Bayesian Inference

The classical Bayesian formalism defines a prior $p(w|\phi)$ with hyperparameters $\phi$ over the weights $w \in \mathbb{R}^{D_w}$ and a likelihood $p(\mathcal{D}|w)$ of observing the data $\mathcal{D}$. The posterior $p(w|\mathcal{D})$ is the quantity of interest. During posterior inference, the hyperparameters $\phi$ of the prior are kept constant. As an analytical solution to the posterior is intractable, either *Markov Chain Monte Carlo* (MCMC) [42] or *Variational Inference* (VI) [43] is used. VI introduces an approximate posterior $q(w)$ and maximizes the *Evidence Lower Bound* (ELBO)

$$\mathbb{E}_{q(w)}[\log p(\mathcal{D}|w)] - \mathrm{KL}(q(w)|p(w|\phi)). \tag{B.1}$$

Commonly, the approximate posterior is modeled as a Gaussian distribution [44]. The KL-divergence between two Gaussians $q = \mathcal{N}(\mu^w, \Sigma^w)$ and $p = \mathcal{N}(\mu_p^w, \Sigma_p^w)$ with dimensionality $D_w$ is available in closed form as

$$\mathrm{KL}(q|p) = \frac{1}{2}\big[\log\frac{\Sigma_p^w}{\Sigma^w} - D + \mathrm{Tr}((\Sigma_p^w)^{-1}\Sigma^w) + (\mu_p^w - \mu^w)^T(\Sigma_p^w)^{-1}(\mu_p^w - \mu^w)\big], \tag{B.2}$$

where $\mu_i$ denotes the $i$-th entry of the mean vector of $q$. We further assume that $q$ is modeled with a diagonal covariance and the $i$-th entry of the diagonal is denoted with $\Sigma_{ii}$. For the case of a standard normal prior $p(w|\phi) = \mathcal{N}(0, I)$, the KL-divergence between the prior and approximate posterior takes the below form

$$\mathrm{KL}(q|p) = \frac{1}{2}\sum_{i=1}^{D_w} -\log\Sigma_{ii}^w + \Sigma_{ii}^w + (\mu_i^w)^2 - const., \tag{B.3}$$

which is equivalent to the negative hyper-prior in Eq. 5.26. We summarize that the ELBO is equivalent to our proposed training objective in Eq. 5.23 up to the position of the logarithm in the likelihood term.

## B.2. Moments of a Product of Correlated Normal Variables

We are interested in calculating the moments of a product of correlated normal variables, $\mathbb{E}[a^{l+1}_{t,i,m}x^l_{t,m}], \mathrm{Cov}[a^{l+1}_{t,i,m}x^l_{t,m}, a^{l+1}_{t,j,n}x^l_{t,n}]$, and $\mathrm{Cov}[a^{l+1}_{t,i,m}x^l_{t,m}, w_{t,j}]$. In the following, we assume that all variables are jointly normally distributed.

The expectation $\mathbb{E}[a^{l+1}_{t,i,m}x^l_{t,m}]$ follows from the definition of the covariance

$$\mathrm{Cov}[a^{l+1}_{t,i,m}, x^l_{t,m}] = \mathbb{E}[a^{l+1}_{t,i,m}x^l_{t,m}] - \mathbb{E}[a^{l+1}_{t,im}]\mathbb{E}[x^l_{t,m}]. \tag{B.4}$$

Contrarily, computing the cross-covariance $\mathrm{Cov}[a^{l+1}_{t,i,m}x^l_{t,m}, a^{l+1}_{t,j,n}x^l_{t,n}]$ is more sophisticated. In order to avoid cluttered notation, we omit in the derivation the time and layer index. The arguments $a_{i,m}, x_m, a_{j,n}, x_n$ are normally distributed $[a_{i,m}, x_m, a_{j,n}, x_n] \sim \mathcal{N}(\mu, \Sigma)$ with mean and covariance

$$\mu = \begin{bmatrix} \mu_{a_{i,m}} \\ \mu_{x_m} \\ \mu_{a_{j,n}} \\ \mu_{x_n} \end{bmatrix}, \tag{B.5}$$

$$\Sigma = \begin{bmatrix} \Sigma_{a_{i,m},a_{i,m}} & \Sigma_{a_{i,m},x_m} & \Sigma_{a_{i,m},a_{j,n}} & \Sigma_{a_{i,m},x_n} \\ \Sigma_{x_m,a_{i,m}} & \Sigma_{x_m,x_m} & \Sigma_{x_m,a_{j,n}} & \Sigma_{x_m,x_n} \\ \Sigma_{a_{j,n},a_{i,m}} & \Sigma_{a_{j,n},x_m} & \Sigma_{a_{j,n},a_{j,n}} & \Sigma_{a_{jn},x_n} \\ \Sigma_{x_n,a_{i,m}} & \Sigma_{x_n,x_m} & \Sigma_{x_n,a_{j,n}} & \Sigma_{x_n,x_n} \end{bmatrix}. \tag{B.6}$$

We first calculate the expectation of the product of four Gaussian random variables

$$\begin{aligned} \mathbb{E}[a_{i,m}x_m a_{j,n}x_n] &= \mathbb{E}[(a_{i,m} - \mu_{a_{i,m}} + \mu_{a_{i,m}})(x_m - \mu_{x_m} + \mu_{x_m}) \\ &\qquad (a_{j,n} - \mu_{a_{j,n}} + \mu_{a_{j,n}})(x_n - \mu_{x_n} + \mu_{x_n})] \\ &= \mathbb{E}[(\bar{a}_{i,m} + \mu_{a_{i,m}})(\bar{x}_m + \mu_{x_m})(\bar{a}_{j,n} + \mu_{a_{j,n}})(\bar{x}_n + \mu_{x_n})], \end{aligned} \tag{B.7}$$

where the accent $\bar{x}$ denotes the centered version of the random variable $x$. We execute the product and arrive at

$$\begin{aligned}\mathbb{E}[a_{i,m}x_m a_{j,n}x_n] = \mu_{a_{i,m}}\mu_{x_m}\mu_{a_{j,n}}\mu_{x_m} + \cancel{\mu_{a_{i,m}}\mu_{x_m}\mu_{a_{j,n}}\mathbb{E}[\bar{x}_n]}+\\ \cancel{\mu_{a_{i,m}}\mu_{x_m}\mu_{x_m}\mathbb{E}[\bar{a}_{j,n}]} + \cancel{\mu_{a_{i,m}}\mu_{a_{j,n}}\mu_{x_m}\mathbb{E}[\bar{x}_m]}+\\ \cancel{\mu_{x_m}\mu_{a_{j,n}}\mu_{x_m}\mathbb{E}[\bar{a}_{i,m}]} + \mu_{a_{i,m}}\mu_{x_m}\mathbb{E}[\bar{a}_{j,n}\bar{x}_n]+\\ \mu_{a_{i,m}}\mu_{a_{j,n}}\mathbb{E}[\bar{x}_m\bar{x}_n] + \mu_{a_{i,m}}\mu_{x_n}\mathbb{E}[\bar{x}_m\bar{a}_{j,n}]+\\ \mu_{x_m}\mu_{a_{j,n}}\mathbb{E}[\bar{a}_{i,m}\bar{x}_n] + \mu_{x_m}\mu_{x_n}\mathbb{E}[\bar{a}_{i,m}\bar{a}_{j,n}]+\\ \mu_{a_{j,n}}\mu_{x_n}\mathbb{E}[\bar{a}_{i,m}\bar{x}_m] + \cancel{\mu_{a_{i,m}}\mathbb{E}[\bar{x}_m\bar{a}_{j,n}\bar{x}_n]}+\\ \cancel{\mu_{x_m}\mathbb{E}[\bar{a}_{i,m}\bar{a}_{j,n}\bar{x}_n]} + \cancel{\mu_{a_{j,n}}\mathbb{E}[\bar{a}_{i,m}\bar{x}_m\bar{x}_n]}+\\ \cancel{\mu_{x_n}\mathbb{E}[\bar{a}_{i,m}\bar{x}_m a_{j,n}]} + \mathbb{E}[\bar{a}_{i,m}\bar{x}_m\bar{a}_{j,n}\bar{x}_n]. \end{aligned} \tag{B.8}$$

Due to Isserlis' theorem [127], any odd central moment of a product of centered Gaussian random variables is zero. In order to calculate $\mathbb{E}[\bar{a}_{i,m}\bar{x}_m\bar{a}_{j,n}\bar{x}_n]$ we make once more use of Isserlis' theorem and arrive at

$$\mathbb{E}[\bar{a}_{i,m}\bar{x}_m\bar{a}_{j,n}\bar{x}_n] = \mathbb{E}[\bar{a}_{i,m}\bar{x}_m]\mathbb{E}[\bar{a}_{j,n}\bar{x}_n] + \mathbb{E}[\bar{a}_{i,m}\bar{a}_{j,n}]\mathbb{E}[\bar{x}_m\bar{x}_n] + \mathbb{E}[\bar{a}_{i,m}\bar{x}_n]\mathbb{E}[\bar{x}_m\bar{a}_{j,n}]. \tag{B.9}$$

Plugging everything together, we arrive at a tractable expression for the expectation of four Gaussian random variables

$$\begin{aligned}\mathbb{E}[a_{i,m}x_m a_{j,n}x_n] =& \mu_{a_{i,m}}\mu_{x_m}\mu_{a_{j,n}}\mu_{x_n} + \Sigma_{a_{i,m},x_m}\Sigma_{a_{j,n},x_n}+\\ &\Sigma_{a_{i,m},x_m}\mu_{a_{j,n}}\mu_{x_n} + \Sigma_{a_{j,n},x_n}\mu_{a_{i,m}}\mu_{x_m}+\\ &\Sigma_{a_{i,m},a_{j,n}}\Sigma_{x_m,x_n} + \Sigma_{a_{i,m},a_{j,n}}\mu_{x_m}\mu_{x_n}+\\ &\Sigma_{x_m,x_n}\mu_{a_{i,m}}\mu_{a_{j,n}} + \Sigma_{a_{i,m},x_n}\Sigma_{x_m,a_{j,n}}+\\ &\Sigma_{a_{i,m},x_n}\mu_{x_m}\mu_{a_{j,n}} + \Sigma_{x_m,a_{j,n}}\mu_{a_{i,m}}\mu_{x_n}. \end{aligned} \tag{B.10}$$

Given the above result, we can calculate the covariance as

$$\begin{aligned}\mathrm{Cov}[a_{i,m}x_m, a_{j,n}x_n] &= \mathbb{E}[a_{i,m}x_m a_{j,n}x_n] - \mathbb{E}[a_{i,m}x_m]\mathbb{E}[a_{j,n}x_n]\\ &= \Sigma_{a_{i,m},a_{j,n}}\Sigma_{x_m,x_n} + \Sigma_{a_{i,m},a_{j,n}}\mu_{x_m}\mu_{x_n}+\\ &\quad \Sigma_{x_m,x_n}\mu_{a_{i,m}}\mu_{a_{j,n}} + \Sigma_{a_{i,m},x_n}\Sigma_{x_m,a_{j,n}}+\\ &\quad \Sigma_{a_{i,m},x_n}\mu_{x_m}\mu_{a_{j,n}} + \Sigma_{x_m,a_{j,n}}\mu_{a_{i,m}}\mu_{x_n}. \end{aligned} \tag{B.11}$$

We obtain the result for $\mathrm{Cov}[a^l_{t,i,m}x^l_{t,m}, w_{t,j}]$ by setting $x_n = 1$ and $a_{j,n} = w_j$ in Eq. B.11

$$\begin{aligned}\mathrm{Cov}[a_{i,m}x_m, w_j] = &\cancel{\Sigma_{a_{i,m},w_j}\Sigma_{x_m,1}} + \Sigma_{a_{i,m},w_j}\mu_{x_m} + \\ &\cancel{\Sigma_{x_m,1}\mu_{a_{i,m}}\mu_{a_j}} + \cancel{\Sigma_{a_{i,m},1}\Sigma_{x_m,w_j}} + \\ &\cancel{\Sigma_{a_{i,m},1}\mu_{x_m}\mu_{w_j}} + \Sigma_{x_m,w_j}\mu_{a_{i,m}} \\ = &\Sigma_{a_{i,m},w_j}\mu_{x_m} + \Sigma_{x_m,w_j}\mu_{a_{i,m}}. \end{aligned} \tag{B.12}$$

# List of Acronyms

| Acronym | Description |
|---|---|
| iid | Independently and Identically Distributed |
| AD | Assumed Density |
| BMM | Bidimensional Moment Matching |
| BNN | Bayesian Neural Network |
| BVMM | Backward Vertical Moment Matching |
| CDF | Cumulative Distribution Function |
| CLT | Central Limit Theorem |
| CS | Convolutional Social |
| Det. | Deterministic |
| DGTM | Deep Gaussian Transition Model |
| DSSM | Deep State-Space Model |
| ECPE | Expectation of Coverage Probability Error |
| ECE | Expected Calibration Error |
| ELBO | Evidence Lower Bound |
| EM | Embedding Model |
| FPK | Fokker-Planck-Kolmogorov |
| GDSSM | Graph Deep State-Space Model |
| GF | Gaussian Filter |

| | |
|---|---|
| GMM | Gaussian Mixture Model |
| GP | Gaussian Process |
| GNN | Graph Neural Network |
| HMM | Horizontal Moment Matching |
| K | Kackertstrasse |
| KL | Kullback-Leibler |
| MCMC | Markov Chain Monte Carlo |
| MC | Monte Carlo |
| MCO | Monte Carlo Objective |
| MFP | Multiple Futures Prediction |
| MLE | Maximum Likelihood Estimation |
| ML | Maximum Likelihood |
| MSE | Mean Squared Error |
| N | Neuweiler |
| NF | Neural Filter |
| NGSIM | Next Generation Simulation |
| NLL | Negative Log-Likelihood |
| NODE | Neural Ordinary Differential Equation |
| NSDE | Neural Stochastic Differential Equation |
| ODE | Ordinary Differential Equation |
| OOM | Out of Memory |
| PDF | Probability Density Function |
| PLL | Predictive Log-Likelihood |
| ProDSSM | Probabilistic Deep State-Space Model |
| RMSE | Root Mean Squared Error |

| | |
|---|---|
| SDE | Stochastic Differential Equation |
| SR | Soft ReLU |
| SSM | State-Space Model |
| ST | Spatio-Temporal |
| T | Thiergarten |
| UKF | Unscented Kalman Filter |
| UT | Unscented Transform |
| VI | Variational Inference |
| VMM | Vertical Moment Matching |

# List of Figures

## List of Figures


1.1. Motivation . . . . . 2
1.2. Thesis structure . . . . . 4

3.1. Assumed density predictions with the BMM algorithm . . . . . 13
3.2. Mutual Information between hidden layers in a neural network . . . . . 18
3.3. Influence of Dropout on the correlation between hidden layers . . . . . 24
3.4. Numerical accuracy of the BMM algorithm. . . . . 25
3.5. Empirical runtime of the BMM algorithm. . . . . 27
3.6. Multimodal predictions with a DGTM . . . . . 28
3.7. Cost-benefit analysis of the calibration for regression tasks with a DGTM . 32
3.8. Cost-benefit analysis of the calibration for dynamical system modeling tasks with a DGTM . . . . . 40
3.9. Dynamical system modeling results on a high dimensional dataset with a DGTM . . . . . 41

4.1. Multimodal predictions for a traffic scenario with a GDSSM . . . . . 44
4.2. Multimodal predictions with a DSSM . . . . . 53
4.3. Structure of the covariance matrix for a GDSSM . . . . . 56
4.4. Covariance between agents as a function of the prediction horizon . . . . . 58

4.5. Empirical runtime of a GDSSM . . . . . 66
4.6. Multimodal predictions with a GDSSM on out-of-domain traffic environments 70

5.1. Predictions and filtering with a ProDSSM . . . . . 73
5.2. Empirical runtime of a ProDSSM . . . . . 87
5.3. Visualization of the kink function . . . . . 97

A.1. Processing of map information . . . . . 108
A.2. GDSSM architecture without a world model . . . . . 109
A.3. GDSSM architecture with a world model . . . . . 110

# List of Tables

## List of Tables


3.1. NLL for UCI regression tasks with a DGTM . . . . . 33

3.2. RMSE for UCI regression tasks with a DGTM . . . . . 33

3.3. Time series classification results with a DGTM . . . . . 36

3.4. Dynamical system modeling results with a DGTM . . . . . 39

4.1. Traffic forecasting results on the rounD dataset with a GDSSM . . . . . 63

4.2. Traffic forecasting results on the NGSIM dataset with a GDSSM . . . . . 65

4.3. Traffic forecasting results on the rounD and NGSIM dataset with a GDSSM for different sparse covariance approximations . . . . . 67

4.4. Traffic forecasting results on out-of-domain traffic environments with a GDSSM . . . . . 69

5.1. NLL for UCI regression tasks with a ProDSSM . . . . . 90

5.2. RMSE for UCI regression tasks with a ProDSSM . . . . . 91

5.3. Filtering results with a ProDSSM . . . . . 93

5.4. Dynamical system modeling results with a ProDSSM . . . . . 98

# Curriculum Vitae

## Education

| | |
|---|---|
| **Ph.D. Candidate** | **TU Darmstadt** |
| Probabilistic Modeling for Dynamical Systems<br>in cooperation with Bosch Center for AI | 08/2019 - Now |
| **Master of Science** | **FAU Erlangen** |
| Power Engineering | 04/2014 -03/2016 |
| **Bachelor of Science** | **FAU Erlangen** |
| Power Engineering | 09/2010 -04/2014 |

## Experience

| | |
|---|---|
| **Research Scientist** | **Bosch Center for AI** |
| Behavior Learning for Autonomous Driving | 01/2023 - Now |
| **Research Assistant** | **University Stuttgart** |
| Predictive Maintenance for Power Plants | 05/2016 - 07/2019 |
| **Student Assistant** | **FAU Erlangen** |
| Automatization of an Engine Test Rig | 05/2014 -09/2015 |
| **Intern** | **MPEI Moscow** |
| Automatization of a Reverse Osmosis Plant | 10/2013 -02/2014 |

# Publication List

1. **Look A.**; Rakitsch B.; Kandemir M.; Peters J. (2023). Sampling-Free Probabilistic Deep State-Space Models, Submitted to Transactions on Pattern Analysis and Machine Intelligence (TPAMI)
2. **Look A.**; Rakitsch B.; Kandemir M.; Peters J. (2023). Cheap and Deterministic Inference for Deep State-Space Models of Interacting Dynamical Systems, Transactions on Machine Learning Research (TMLR).
3. **Look A.**; Kandemir M.; Rakitsch B.; Peters J. (2022). A Deterministic Approximation to Neural SDEs, Transactions on Pattern Analysis and Machine Intelligence (TPAMI).
4. Haussmann M.; Gerwinn S.; **Look A.**; Rakitsch B.; Kandemir M. (2021). Learning Partially Known Stochastic Dynamics with Empirical PAC Bayes, Artificial Intelligence and Statistics (AISTATS).
5. **Look A.**; Doneva S.; Kandemir M.; Gemulla R.; Peters J. (2020). Differentiable Implicit Layers, NeurIPS Workshop on Machine Learning for Engineering.
6. **Look A.**; Kandemir M. (2019). Differential Bayesian Neural Nets, NeurIPS Workshop on Bayesian Deep Learning.
7. **Look A.**; Riedelbauch S. (2019). Dealing with Limited Access to Data: Comparison of Deep Learning Approaches, International Joint Conference on Neural Networks (IJCNN).
8. **Look A.**; Kirschner O.; Riedelbauch S. (2018). Building Robust Classifiers with Generative Adversarial Networks for Detecting Cavitation in Hydraulic Turbines, International Conference on Pattern Recognition Applications and Methods (ICPRAM).
9. **Look A.**; Kirschner O.; Riedelbauch S.; Necker J. (2018). Detection and Level Estimation of Cavitation in Hydraulic Turbines with Convolutional Neural Networks, International Symposium on Cavitation (CAV).

# Bibliography


[1] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-Task Multi-Sensor Fusion for 3D Object Detection", in *CVPR*, 2019.

[2] B. Yang, R. Guo, M. Liang, S. Casas, and R. Urtasun, "RadarNet: Exploiting Radar for Robust Perception of Dynamic Objects", in *ECCV*, 2020.

[3] C. Tang and R. R. Salakhutdinov, "Multiple Futures Prediction", in *NeurIPS*, 2019.

[4] B. Ivanovic and M. Pavone, "The Trajectron: Probabilistic Multi-Agent Trajectory Modeling with Dynamic Spatiotemporal Graphs", in *ICCV*, 2019.

[5] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces", in *CVPR*, 2016.

[6] Y. Hoshen, "VAIN: Attentional Multi-Agent Predictive Modeling", in *NeurIPS*, 2017.

[7] B. Kim, C. Kang, J. Kim, S. Lee, C. Chung, and J. Choi, "Probabilistic Vehicle Trajectory Prediction over Occupancy Grid Map via Recurrent Neural Network", in *ITSC*, 2018.

[8] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "SpAGNN: Spatially-Aware Graph Neural Networks for Relational Behavior Forecasting from Sensor Data", in *ICRA*, 2020.

[9] M. Henaff, A. Canziani, and Y. LeCun, "Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic", in *ICLR*, 2019.

[10] C. Hubmann, J. Schulz, M. Becker, D. Althoff, and C. Stiller, "Automated Driving in Uncertain Environments: Planning With Interaction and Uncertain Maneuver Prediction", *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 1, 2018.

[11] J. Hardy and M. Campbell, "Contingency Planning over Probabilistic Hybrid Obstacle Predictions for Autonomous Road Vehicles", in *IROS*, 2010.

[12] H. Cui *et al.*, "Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks", in *ICRA*, 2019.

[13] N. Rhinehart, R. McAllister, K. M. Kitani, and S. Levine, “PRECOG: Prediction Conditioned On Goals in Visual Multi-Agent Settings”, 2019.

[14] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, “Multimodal End-to-End Autonomous Driving”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, 2022.

[15] L. Liu *et al.*, “Computing Systems for Autonomous Driving: State-of-the-Art and Challenges”, *IEEE Internet of Things Journal*, vol. 8, no. 8, 2020.

[16] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen, and A. Y. Ding, “A Survey on Approximate Edge AI for Energy Efficient Autonomous Driving Services”, *arXiv*, vol. abs/2304.14271, 2023.

[17] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2013.

[18] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, “Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning”, in *ICML*, 2017.

[19] A. Kendall and Y. Gal, “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?”, in *NeurIPS*, 2017.

[20] B. Paden, M. Čáp, S. Yong, D. Yershov, and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles”, *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, 2016.

[21] T. B. Schön, A. Wills, and B. Ninness, “System Identification of Nonlinear State-Space Models”, *Automatica*, vol. 47, no. 1, 2011.

[22] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[23] A. Look, M. Kandemir, B. Rakitsch, and J. Peters, “A Deterministic Approximation to Neural SDEs”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, 2023.

[24] A. Look, B. Rakitsch, M. Kandemir, and J. Peters, “Cheap and Deterministic Inference for Deep State-Space Models of Interacting Dynamical Systems”, *TMLR*, 2023.

[25] A. Look, M. Kandemir, B. Rakitsch, and J. Peters, “Sampling-Free Probabilistic Deep State-Space Models”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023, Under Review.

[26] R. G. Krishnan, U. Shalit, and D. Sontag, “Structured Inference Networks for Nonlinear State Space Models”, in *AAAI*, 2017.

[27] J. Bayer, M. Soelch, A. Mirchev, B. Kayalibay, and P. van der Smagt, “Mind the Gap when Conditioning Amortised Inference in Sequential Latent-Variable Models”, in *ICLR*, 2021.

[28] E. de Bézenac *et al.*, “Normalizing Kalman Filters for Multivariate Time Series Analysis”, in *NeurIPS*, 2020.

[29] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, “A Recurrent Latent Variable Model for Sequential Data”, in *NeurIPS*, 2015.

[30] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther, “Sequential Neural Models with Stochastic Layers”, in *NeurIPS*, 2016.

[31] M. W. Brandt and P. Santa-Clara, “Simulated Likelihood Estimation of Diffusions with an Application to Exchange Rate Dynamics in Incomplete Markets”, *Journal of Financial Economics*, vol. 63, no. 274, 2002.

[32] A. R. Pedersen, “A New Approach to Maximum Likelihood Estimation for Stochastic Differential Equations Based on Discrete Observations”, *Scandinavian Journal of Statistics*, vol. 22, no. 1, 1995.

[33] O. Elerian, S. Chib, and N. Shephard, “Likelihood Inference for Discretely Observed Nonlinear Diffusions”, *Econometrica*, vol. 69, no. 4, 2001.

[34] S. Särkkä, J. Hartikainen, I. S. Mbalawata, and H. Haario, “Posterior Inference on Parameters of Stochastic Differential Equations via Non-Linear Gaussian Filtering and Adaptive MCMC”, *Statistics and Computing*, vol. 25, no. 2, 2015.

[35] S. Eleftheriadis, T. Nicholson, M. Deisenroth, and J. Hensman, “Identification of Gaussian Process State Space Models”, in *NeurIPS*, 2017.

[36] S. Särkkä and J. Sarmavuori, “Gaussian Filtering and Smoothing for Continuous-Discrete Dynamic Systems”, *Signal Processing*, vol. 93, no. 2, 2013.

[37] A. Solin, E. Tamir, and P. Verma, “Scalable Inference in SDEs by Direct Matching of the Fokker-Planck-Kolmogorov Equation”, in *NeurIPS*, 2021.

[38] A. Jazwinski, *Stochastic processes and filtering theory*. Acad. Press, 1970.

[39] P. W. Battaglia *et al.*, “Relational inductive biases, deep learning, and graph networks”, *arXiv*, vol. abs/1806.01261, 2018.

[40] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive Representation Learning on Large Graphs”, in *NeurIPS*, 2017.

[41] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural Message Passing for Quantum Chemistry”, in *ICML*, 2017.

[42] W. J. Maddox, T. Garipov, P. Izmailov, D. Vetrov, and A. G. Wilson, “A Simple Baseline for Bayesian Uncertainty in Deep Learning”, in *NeurIPS*, 2019.

[43] A. Graves, “Practical Variational Inference for Neural Networks”, in *NeurIPS*, 2011.

[44] A. Wu, S. Nowozin, E. Meeds, R. E. Turner, J. M. Hernandez-Lobato, and A. L. Gaunt, “Deterministic Variational Inference for Robust Bayesian Neural Networks”, in *ICLR*, 2019.

[45] A. Amini, W. Schwarting, A. Soleimany, and D. Rus, “Deep Evidential Regression”, in *NeurIPS*, 2020.

[46] M. Sensoy, L. Kaplan, and M. Kandemir, “Evidential Deep Learning to Quantify Classification Uncertainty”, in *NeurIPS*, 2018.

[47] M. Haussmann, S. Gerwinn, and M. Kandemir, “Bayesian Evidential Deep Learning with PAC Regularization”, in *AABI*, 2021.

[48] A. Malinin and M. Gales, “Predictive Uncertainty Estimation via Prior Networks”, in *NeurIPS*, 2018.

[49] A. Malinin, S. Chervontsev, I. Provilkov, and M. J. F. Gales, “Regression Prior Networks”, *arXiv*, vol. abs/2006.11590, 2020.

[50] M. Haussmann, S. Gerwinn, A. Look, B. Rakitsch, and M. Kandemir, “Learning Partially Known Stochastic Dynamics with Empirical PAC Bayes”, in *AISTATS*, 2021.

[51] A. R. Masegosa, “Learning under Model Misspecification: Applications to Variational and Ensemble Methods”, in *NeurIPS*, 2020.

[52] F. Futami, T. Iwata, N. Ueda, I. Sato, and M. Sugiyama, “Predictive variational Bayesian inference as risk-seeking optimization”, in *AISTATS*, 2022.

[53] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural Ordinary Differential Equations”, in *NeurIPS*, 2018.

[54] B. Tzen and M. Raginsky, “Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit”, *arXiv*, vol. abs/1905.09883, 2019.

[55] S. Särkkä and A. Solin, *Applied Stochastic Differential Equations*. Cambridge University Press., 2019.

[56] P. Hegde, M. Heinonen, H. Lähdesmäki, and S. Kaski, “Deep learning with differential Gaussian process flows”, in *AISTATS*, 2019.

[57] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, in *CVPR*, 2016.

[58] A. Hurn and K. Lindsay, “Estimating the Parameters of Stochastic Differential Equations by Monte Carlo Methods”, *Mathematics and Computers in Simulation*, vol. 48, 1999.

[59] B. Jensen and R. Poulsen, “Transition Densities of Diffusion Processes”, *The Journal of Derivatives*, vol. 9, 2002.

[60] X. Li, T. L. Wong, R. T. Q. Chen, and D. Duvenaud, “Scalable Gradients for Stochastic Differential Equations”, in *AISTATS*, 2020.

[61] A. Look and M. Kandemir, “Differential Bayesian Neural Networks”, in *NeurIPS Workshop Bayesian Deep Learning*, 2019.

[62] J. M. Hernandez-Lobato and R. Adams, “Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks”, in *ICML*, 2015.

[63] S. Ghosh, F. Fave, M. Delle, and J. Yedidia, “Assumed Density Filtering Methods for Learning Bayesian Neural Networks”, in *AAAI*, 2016.

[64] J. S. Liu, “Siegel’s formula via Stein’s identities”, *Statistics & Probability Letters*, vol. 21, 1994.

[65] G. V. S. Shuyang Gao and A. Galstyan, “Efficient Estimation of Mutual Information for Strongly Dependent Variables”, in *AISTATS*, 2015.

[66] E. A. Wan and R. V. D. Merwe, “The Unscented Kalman Filter for Nonlinear Estimation”, in *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000.

[67] S. Wang and C. Manning, “Fast dropout training”, in *ICML*, 2013.

[68] P. Cui, W. Hu, and J. Zhu, “Calibrated Reliable Regression using Maximum Mean Discrepancy”, in *NeurIPS*, 2020.

[69] M. Bensimhou, “N-Dimensional cumulative function, and other useful facts about Gaussians and normal densities”, Tech. Rep., 2013. [Online]. Available: `https://upload.wikimedia.org/wikipedia/commons/a/a2/Cumulative_function_n_dimensional_Gaussians_12.2013.pdf`.

[70] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On Calibration of Modern Neural Networks”, in *ICML*, 2017.

[71] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”, in *ICML*, 2014.

[72] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”, in *ICML*, 2016.

[73] H. Salimbeni and M. Deisenroth, “Doubly Stochastic Variational Inference for Deep Gaussian Processes”, in *NeurIPS*, 2017.

[74] J. Lindinger, D. Reeb, C. Lippert, and B. Rakitsch, “Beyond the Mean-Field: Structured Deep Gaussian Processes Improve the Predictive Uncertainties”, in *NeurIPS*, 2020.

[75] D. P. Kingma, T. Salimans, and M. Welling, “Variational Dropout and the Local Reparameterization Trick”, in *NeurIPS*, 2015.

[76] M. Kandemir, A. Akgül, M. Haussmann, and G. Unal, “Evidential Turing Processes”, in *ICLR*, 2022.

[77] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”, in *NeurIPS*, 2015.

[78] S. Pal, L. Ma, Y. Zhang, and M. Coates, “RNN with Particle Flow for Probabilistic Spatio-temporal Forecasting”, in *ICML*, 2021.

[79] D. Hafner *et al.*, “Learning Latent Dynamics for Planning from Pixels”, in *ICML*, 2019.

[80] M. Herman *et al.*, “Pedestrian Behavior Prediction for Automated Driving: Requirements, Metrics, and Relevant Features”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, 2022.

[81] N. Djuric *et al.*, “Uncertainty-aware Short-term Motion Prediction of Traffic Actors for Autonomous Driving”, in *IEEE WACV*, 2020.

[82] A. Jain *et al.*, “Discrete Residual Flow for Probabilistic Pedestrian Behavior Prediction”, in *CoRL*, 2019.

[83] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, “MultiPath: Multiple Probabilistic Anchor Trajectory Hypotheses for Behavior Prediction”, *arXiv*, vol. abs/1910.05449, 2019.

[84] A. Vaswani *et al.*, “Attention Is All You Need”, in *NeurIPS*, 2017.

[85] G. Abbati, P. Wenk, M. Osborne, A. Krause, B. Schölkopf, and S. Bauer, “AReS and MaRS Adversarial and MMD-Minimizing Regression for SDEs”, in *ICML*, 2019.

[86] S. Zhang, G. Bin, D. Anlan, H. Jing, X. Ziping, and C. S. Xi, “Cautionary tales on air-quality improvement in Beijing”, *Proceedings of the Royal Society: Mathematical,Physical and Engineering Sciences*, vol. 473, 2017.

[87] M. Jørgensen, M. P. Deisenroth, and H. Salimbeni, “Stochastic Differential Equations with Variational Wishart Diffusions”, in *ICML*, 2020.

[88] D. Agudelo-España *et al.*, “A Real-Robot Dataset for Assessing Transferability of Learned Dynamics Models”, in *ICRA*, 2020.

[89] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting”, in *ICLR*, 2018.

[90] B. Yu, H. Yin, and Z. Zhu, “Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting”, in *IJCAI*, 2018.

[91] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun, “Lagrangian Fluid Simulation with Continuous Convolutions”, in *ICLR*, 2019.

[92] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, “Structural-RNN: Deep Learning on Spatio-Temporal Graphs”, in *CVPR*, 2016.

[93] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, “Learning to Simulate Complex Physics with Graph Networks”, in *ICML*, 2020.

[94] R. Krajewski, T. Moers, J. Bock, L. Vater, and L. Eckstein, “The rounD Dataset: A Drone Dataset of Road User Trajectories at Roundabouts in Germany”, in *ITSC*, 2020.

[95] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen Netzen”, *Diploma, Technische Universität München*, vol. 91, no. 1, 1991.

[96] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”, *arXiv*, vol. abs/1412.3555, 2014.

[97] S. Bai, J. Z. Kolter, and V. Koltun, “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling”, *arXiv*, vol. abs/1803.01271, 2018.

[98] A. v. d. Oord *et al.*, “WaveNet: A Generative Model for Raw Audio”, *arXiv*, vol. abs/1609.03499, 2016.

[99] S. Li *et al.*, “Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting”, in *NeurIPS*, 2019.

[100] F. Yang, L. Chen, F. Zhou, Y. Gao, and W. Cao, “Relational State-Space Model for Stochastic Multi-Object Systems”, in *ICLR*, 2020.

[101] T. B. Schön *et al.*, “Sequential Monte Carlo Methods for System Identification”, *IFAC*, vol. 48, no. 28, 2015.

[102] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei, “Variational Sequential Monte Carlo”, in *AISTATS*, 2018.

[103] C. J. Maddison *et al.*, “Filtering Variational Objectives”, in *NeurIPS*, 2017.

[104] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood, “Auto-Encoding Sequential Monte Carlo”, in *ICLR*, 2018.

[105] A. Doucet, S. Godsill, and C. Andrieu, “On sequential Monte Carlo sampling methods for Bayesian filtering”, *Statistics and computing*, vol. 10, no. 3, 2000.

[106] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, “Variational Inference: A Review for Statisticians”, *Journal of the American Statistical Association*, vol. 112, no. 518, 2017.

[107] D. Alspach and H. Sorenson, “Nonlinear Bayesian estimation using Gaussian sum approximations”, *IEEE transactions on automatic control*, vol. 17, no. 4, 1972.

[108] T. Gneiting and A. E. Raftery, “Strictly Proper Scoring Rules, Prediction, and Estimation”, *Journal of the American statistical Association*, vol. 102, no. 477, 2007.

[109] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun, “Implicit Latent Variable Model for Scene-Consistent Motion Forecasting”, in *ECCV*, 2020.

[110] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, “Learning Compositional Koopman Operators for Model-Based Control”, in *ICLR*, 2020.

[111] J. Halkias and J. Colyar, “US Highway 101 Dataset”, Tech. Rep., 2007. [Online]. Available: `https://rosap.ntl.bts.gov/view/dot/38724/Print`.

[112] N. Deo and M. Trivedi, “Convolutional Social Pooling for Vehicle Trajectory Prediction”, in *CVPR Workshop*, 2018.

[113] F. Diehl, T. Brunner, M. Le, and A. Knoll, “Graph Neural Networks for Modelling Traffic Participant Interaction”, in *IEEE Intelligent Vehicles Symposium*, 2019.

[114] D. Lenz, F. Diehl, M. Truong-Le, and A. C. Knoll, “Deep Neural Networks for Markovian Interactive Scene Prediction in Highway Scenarios”, in *IEEE Intelligent Vehicles Symposium*, 2017.

[115] T. A. Wheeler and M. J. Kochenderfer, “Factor Graph Scene Distributions for Automotive Safety Analysis”, in *ITSC*, 2016.

[116] J. Mercat, N. Zoghby, G. Sandou, D. Beauvois, and G. Gil, “Inertial Single Vehicle Trajectory Prediction Baselines and Applications with the NGSIM Dataset”, *arXiv*, vol. abs/1908.11472, 2019.

[117] G. Chen, L. Hu, Q. Zhang, Z. Ren, X. Gao, and J. Cheng, “ST-LSTM: Spatio-Temporal Graph Based Long Short-Term Memory Network For Vehicle Trajectory Prediction”, in *IEEE ICIP*, 2020.

[118] A. Philipp and D. Goehring, “Analytic Collision Risk Calculation for Autonomous Vehicle Navigation”, in *ICRA*, 2019.

[119] C. Yildiz, M. Heinonen, and H. Lahdesmaki, “ODE2VAE: Deep generative second order ODEs with Bayesian neural networks”, in *NeurIPS*, 2019.

[120] V. Iakovlev, C. Yildiz, M. Heinonen, and H. Lähdesmäki, “Latent Neural ODEs with Sparse Bayesian Multiple Shooting”, 2023.

[121] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, “Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks”, in *ICLR*, 2017.

[122] A. D. Ialongo, M. Van Der Wilk, J. Hensman, and C. E. Rasmussen, “Overcoming Mean-Field Approximations in Recurrent Gaussian Process Models”, in *ICML*, 2019.

[123] A. Doerr *et al.*, “Probabilistic Recurrent State-Space Models”, in *ICML*, 2018.

[124] J. Lindinger, B. Rakitsch, and C. Lippert, “Laplace Approximated Gaussian Process State-Space Models”, in *UAI*, 2022.

[125] Chua, Kurtland and Calandra, Roberto and McAllister, Rowan and Levine, Sergey, “Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models”, in *NeurIPS*, 2018.

[126] W. R. Morningstar, A. Alemi, and J. V. Dillon, “PACm-Bayes: Narrowing the Empirical Risk Gap in the Misspecified Bayesian Regime”, in *AISTATS*, 2022.

[127] G. C. Wick, “The Evaluation of the Collision Matrix”, *Phys. Rev.*, vol. 80, 2 1950.

[128] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, in *NAACL-HLT*, 2019.

[129] A. Ramesh *et al.*, “Zero-Shot Text-to-Image Generation”, in *ICML*, 2021.

[130] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes”, in *ICLR*, 2014.

[131] J. Ho, A. Jain, and P. Abbeel, “Denoising Diffusion Probabilistic Models”, in *NeurIPS*, 2020.

[132] Y. Burda, R. B. Grosse, and R. Salakhutdinov, “Importance Weighted Autoencoders”, in *ICLR*, 2016.

[133] M. Bansal, A. Krizhevsky, and A. S. Ogale, “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”, *arXiv*, vol. abs/1812.03079, 2018.