Missing Data Handling in Amortized Bayesian Inference via Invertible Neural Networks

Zijian Wang

Geboren am 30. April 2000 in Shanghai, China

August 17, 2022

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Jan Hasenauer

Zweitgutachter: Prof. Dr. Alexander Effland

Institut für Angewandte Mathematik

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Abstract

In our data-driven world, sophisticated methods are needed to infer the underlying parameters of mathematical models arising in all branches of science. Amongst the major advances in the field of deep generative modeling, the BayesFlow method was developed to perform amortized likelihood-free Bayesian inference. An invertible neural network is trained on artificial datasets to learn the probabilistic parameter-data relationship, thus rendering the posterior distribution of interest accessible. In this thesis, we evaluate and extend this method so that missing data, which commonly pose a problem e.g. for clinical researchers, can be handled. We propose three approaches to encode missing data, integrate them into the BayesFlow workflow and compare their utility in recovering the posterior distribution conditioned on the available data. We find that an approach in which the data matrix is augmented with a binary row to indicate presence or absence of data and all missing values are replaced by a fill-in constant performs the most robustly.

Zusammenfassung

In unserer datengesteuerten Welt werden komplexe Methoden für die Parameterschätzung in mathematischen Modellen benötigt, die in vielerlei Wissenschaftszweigen zum Einsatz kommen. Zu den wichtigsten Entwicklungen auf dem Gebiet des Deep Generative Modelings gehört die BayesFlow-Methode, welche eine amortisierte Bayessche Inferenz in Fällen erlaubt, in denen die explizite Auswertung der Likelihood-Funktion nicht möglich ist. Hierfür wird ein invertierbares neuronales Netzwerk mit simulierten Datensätzen trainiert, um die probabilistische Parameter-Daten-Beziehung zu erlernen und somit die A-Posteriori-Verteilung zugänglich zu machen. In der vorliegenden Arbeit evaluieren und erweitern wir diese Methode derart, dass auch fehlende Daten berücksichtigt werden können, die z.B. in klinischen Studien ein häufiges Problem darstellen. Wir schlagen drei Ansätze zur Kodierung fehlender Daten vor, integrieren sie in die BayesFlow-Methode und vergleichen ihren Nutzen für die Schätzung der auf die verfügbaren Daten bedingten A-Posteriori-Verteilung. Wir kommen zu der Erkenntnis, dass ein Ansatz, bei dem die Datenmatrix um eine binäre Zeile ergänzt wird, die das Vorhandensein bzw. Fehlen von Daten angibt, und bei dem alle fehlenden Werte durch eine Konstante ersetzt werden, am robustesten funktioniert.

Acknowledgements

First and foremost, I would like to thank Professor Dr. Jan Hasenauer and Dr. Yannik Schälte for the great supervision of my Bachelor thesis. It was very fulfilling to be integrated into their research group and work on a research-related topic at the interface of parameter estimation and deep learning. Special thanks to Yannik again for the regular online meetings and for his careful proofreading, which helped me a lot to improve my work.

Furthermore, I would like to thank Professor Dr. Ullrich Köthe and Dr. Stefan Radev from the collaborating lab from Heidelberg University as well as Marvin Schmitt from the University of Stuttgart, who took their time to answer my questions concerning the BayesFlow method. I would also like to thank my friend and fellow student Paul Müller for the valuable exchange during our studies and proofreading my thesis.

Last but not least, I am very grateful to my family and my girlfriend Celine Alexy for their incredible support and patience throughout the course of my Bachelor studies. It would not have been possible to reach this stage of my life without them.

Contents

1 Introduction

Mathematical models are used to describe data generating processes in a variety of applications ranging from life and physical sciences to economics. While these models become more complex, parameter estimation turns out increasingly challenging. In the context of Bayesian inference, a major difficulty arises when evaluating the likelihood function becomes too expensive or causes numerical instability, making it necessary to employ simulation-based likelihood-free methods. This is for instance the case when the data generating process is modeled by a stochastic differential equation [Picchini, 2014], a Markov jump process [Owen et al., 2014] or a complex agent-based model used e.g. in systems biology to describe multi-scale, multi-cellular systems [Hasenauer et al., 2015; Schälte and Hasenauer, 2020].

Standard likelihood-free methods such as approximate Bayesian computation (ABC) work on the level of individual datasets [Liepe et al., 2010]. When dealing with numerous datasets from the same model, this can lead to prohibitively high computational cost as the entire estimation procedure needs to be redone for every dataset. As an alternative to such case-based methods, novel methods in deep generative modeling focus on learning a generative model of posterior distributions conditional on arbitrary input datasets [Ruthotto and Haber, 2021; Huang et al., 2018]. After upfront model training, these methods then amortize over observed datasets.

A successful representative of this class of methods is called BayesFlow [Radev et al., 2022, 2021; Schmitt et al., 2021]. Given a model of interest, a conditional invertible neural network (cINN) is trained on simulated data to learn a reversible normalizing flow from the parameter space to a latent space using conditioning information from the data. The trained cINN enables efficient posterior sampling by generating samples from the simple latent distribution and transforming them back to the parameter space conditional on any observed dataset. Unlike ABC methods, which operate with an approximation error $\varepsilon > 0$, the BayesFlow method is theoretically exact. Further, BayesFlow incorporates an optional summary network that is trained jointly with the cINN to transform raw datasets into learned fixed-size summary statistics in an information-preserving manner. This in particular makes it possible to handle variable-size data.

The BayesFlow method was developed for and extensively tested on models producing complete datasets. However, in many real-world problems, e.g. clinical trials, researchers are confronted with missing data, which can substantially complicate trustworthy statistical inference. In this thesis, we investigate how the BayesFlow method can be extended to reliably estimate posterior distributions conditioned on the available data in an amortized fashion.

In Section 2, we start with the mathematical statistical foundation of the BayesFlow method, including the derivation of the loss function used for neural network training as well as the proof of correctness under the assumption of perfect convergence. Then, we provide a thorough introduction to the cINN architecture in Section 2.3 and discuss some implementation details in Section 2.4. In Section 3, we use two simple models to validate the generally good performance of BayesFlow on complete datasets and outline some of our experiences regarding the choice of summary networks and prior distributions.

In Section 4, we specify and introduce notations for the type of data missingness that we want to consider in this thesis. Then, we propose three intuitive approaches to encode missing data and present the according learning algorithm to show how these approaches can be integrated into the BayesFlow workflow. In Section 5, we analyze the utility of the proposed approaches for the estimation of posterior distributions conditioned on the available data. While all three approaches perform similarly well on simple models, we find that the approach of augmenting the data matrix with a binary row to indicate presence or absence of data and replacing all missing values by a fill-in constant performs the most robustly when the model complexity is increased.

2 The BayesFlow Method

In this section, we introduce the goal of amortized likelihood-free Bayesian inference and show how the BayesFlow method accomplishes this via the learning of normalizing flows with cINNs. We derive and give interpretations of the loss function that is used for neural network training and explain how invertibility is achieved by the architecture of cINNs. At last, we showcase the learning algorithm and point out some important implementation details.

2.1 Notation and Problem Specification

In the setting of Bayesian inference, we are confronted with a mathematical model $\mathcal{M}(\boldsymbol{\theta})$ whose underlying parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{\theta}}}$ are unknown and shall be inferred from a set of observed data $\boldsymbol{x}_{1:N} = \{\boldsymbol{x}_i\}_{i=1}^N$. Each of the N observations or data points is a vector $\boldsymbol{x}_i \in \mathbb{R}^{n_{\boldsymbol{x}}}$.

Bayesian inference now aims at assessing the posterior distribution, which is given by Bayes' theorem [Lesaffre and Lawson, 2012]:

$$p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}) = \frac{p(\boldsymbol{\theta}, \boldsymbol{x}_{1:N})}{p(\boldsymbol{x}_{1:N})} = \frac{p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}}$$

The posterior integrates all currently available information about the parameters $\boldsymbol{\theta}$, namely the prior knowledge $p(\boldsymbol{\theta})$ from e.g. databases or collaboration partners as well as the knowledge gained from the observed data $\boldsymbol{x}_{1:N}$ and encoded in the likelihood $p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})$. Further, we call $p(\boldsymbol{\theta}, \boldsymbol{x}_{1:N}) = p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$ the Bayesian joint distribution and $p(\boldsymbol{x}_{1:N}) = \int p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}$ the evidence. The latter is just a normalizing constant that does not affect the actual shape of the posterior distribution. Thus, Bayes' theorem is also often stated as:

$$p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}) \propto p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$$

We assume that we have control over the forward model or data generating process $\boldsymbol{\theta} \to \boldsymbol{x}_{1:N}$ in the following sense: Given any parameter $\boldsymbol{\theta}$, we may generate arbitrarily many datasets $\boldsymbol{x}_{1:N}$ by running a simulation program that evaluates a deterministic function g at the parameter values $\boldsymbol{\theta}$ and some random variable $\boldsymbol{\xi}$ representing stochasticity:

$$\boldsymbol{x}_{1:N} \sim p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta}) \quad \Longleftrightarrow \quad \boldsymbol{x}_{1:N} = g(\boldsymbol{\theta}, \boldsymbol{\xi}) \text{ with } \boldsymbol{\xi} \sim p(\boldsymbol{\xi}) \tag{1}$$

This is commonly referred to as sampling from the likelihood. Note that we do not require the stronger ability to evaluate the likelihood function $p(\boldsymbol{x}_{1:N} \,|\, \boldsymbol{\theta})$ for any pair $(\boldsymbol{\theta}, \boldsymbol{x}_{1:N})$, as can be typically done for deterministic models with parametric noise model, e.g. ordinary differential equation (ODE) models [Fröhlich et al., 2019]. Since the likelihood and thus right-hand side of Bayes' formula is assumed to be intractable, the true posterior must be approximated.

Amortizing Bayesian inference means, as opposed to standard case-based approaches, to find a global estimator of the posterior distribution given any dataset $\boldsymbol{x}_{1:N}$. Specifically, we seek a method that is able to approximate the true posterior in the following ways:

- Provide a point estimate of the posterior probability $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ for any $\boldsymbol{\theta}$ and $\boldsymbol{x}_{1:N}$
- Generate representative samples $\boldsymbol{\theta}^{(l)} \sim p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ for any given dataset $\boldsymbol{x}_{1:N}$

Especially the latter is of great importance for visualizing high-dimensional, complexly shaped posterior distributions as well as computing their statistics, such as the maximum a posteriori estimate, posterior moments and quantiles.

2.2 Learning the Posterior via Normalizing Flows

2.2.1 Objective Function for Fixed Dataset Size

We first consider the case where we are given a dataset $\boldsymbol{x}_{1:N}$ of constant size N. For simplicity, we write $\boldsymbol{x} = \boldsymbol{x}_{1:N}$.

Our overall goal is to construct a tractable distribution $p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$ which is parametrized by $\boldsymbol{\phi}$ (yet to be specified) and approximates the true posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$. The main idea consists in the reparametrization of $p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$ in terms of an invertible function $f_{\boldsymbol{\phi}} \colon \mathbb{R}^{n_{\boldsymbol{\theta}}} \to \mathbb{R}^{n_{\boldsymbol{\theta}}}$, $\boldsymbol{\theta} \mapsto \boldsymbol{z}$ that performs a normalizing flow between the posterior distribution of the parameters of interest $\boldsymbol{\theta}$ and some multivariate normal latent variable $\boldsymbol{z}$:

$$\boldsymbol{\theta} \sim p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x}) \quad \Longleftrightarrow \quad \boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x}) \text{ with } \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}_{n_{\boldsymbol{\theta}}}(\boldsymbol{z} \mid \boldsymbol{0}, \boldsymbol{I}_{n_{\boldsymbol{\theta}}}) \tag{2}$$

For a given $\boldsymbol{x}$, we understand $f_{\boldsymbol{\phi}}$ and its inverse $f_{\boldsymbol{\phi}}^{-1}$ as functions of $\boldsymbol{\theta}$ and $\boldsymbol{z}$, respectively. We, however, employ the notations $f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})$ and $f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x})$ to emphasize that these functions are also conditioned on the data $\boldsymbol{x}$.

We implement $f_{\boldsymbol{\phi}}$ as a conditional invertible neural network (will be introduced in Section 2.3) with learnable parameters $\boldsymbol{\phi}$. The challenge has thus shifted to finding network parameters $\boldsymbol{\phi}$ for which the distribution $p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \boldsymbol{x})$ as defined in (2) provides a good approximation of the true posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

In order to measure the quality of approximation, we make use of the Kullback-Leibler (KL) divergence [Taboga, 2021]. This concept originates from the field of information theory and is widely used in the context of machine learning as a distance measure between a true or target distribution P and some approximate distribution Q.

Definition 1. *Let P, Q be two probability distributions with probability density functions p and q with respect to the Lebesgue measure on $\mathbb{R}^{n_{\boldsymbol{\theta}}}$ and assume that P is absolutely continuous with respect to Q. Then, the Kullback-Leibler divergence of Q from P is defined as:*

$$\mathbb{KL}(P \,||\, Q) \coloneqq \int p(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \, d\boldsymbol{\theta}$$

Remark. *The absolute continuity assumption $P \ll Q$ ensures that the integrand is well-defined (up to Lebesgue null sets) according to the convention $0 \cdot \log \frac{0}{q} = 0$ for all $q \geq 0$. For simplicity, we omit the domain of integration when it is the entire space (in this case $\mathbb{R}^{n_{\boldsymbol{\theta}}}$).*

Some of the most important mathematical properties of the KL divergence are summarized in the following lemma.

Lemma 2. *Under the same assumptions as in Definition 1, it holds:*

- $\mathbb{KL}(P \,||\, Q) \geq 0$
- $\mathbb{KL}(P \,||\, Q) = 0$ *if and only if $p = q$ almost everywhere*

Proof. We adapt the proof of a similar statement for discrete variables from Taboga [2021] to the continuous case. Observe that $\mu(d\boldsymbol{\theta}) \coloneqq p(\boldsymbol{\theta}) \, d\boldsymbol{\theta}$ defines a probability measure on $\Omega \coloneqq \{\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{\theta}}} \colon p(\boldsymbol{\theta}) > 0\}$, which has the same null sets as the Lebesgue measure restricted onto Ω.

We have the following chain of (in-)equalities:

$$\begin{aligned} \mathbb{KL}(P \,||\, Q) &= \int_{\Omega} \log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} \, \mu(d\boldsymbol{\theta}) = \int_{\Omega} -\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta})} \, \mu(d\boldsymbol{\theta}) \geq -\log \left(\int_{\Omega} \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta})} \, \mu(d\boldsymbol{\theta}) \right) \\ &= -\log \left(\int_{\Omega} q(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \right) \geq -\log \left(\int_{\mathbb{R}^{n_{\boldsymbol{\theta}}}} q(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \right) = -\log 1 = 0 \end{aligned}$$

The first estimate follows from Jensen's inequality applied to the convex function $t \mapsto -\log t$, and the second one follows by enlarging the domain of integration. Since $t \mapsto -\log t$ is strictly convex, equality can only occur if there exists some constant $C \in \mathbb{R}$ such that $\frac{q}{p} = C$ μ-almost surely or, equivalently, almost everywhere (with respect to Lebesgue measure) on Ω.

Hence, if $\mathbb{KL}(P \,||\, Q) = 0$, we must have $0 = \int_\Omega -\log C \, \mu(d\boldsymbol{\theta}) = -\log C$ and thus $C = 1$, i.e. $p = q$ almost everywhere on Ω. Since $p = 0$ outside of Ω and p, q are normalized densities, we immediately obtain that $q = 0$ almost everywhere outside Ω and thus $p = q$ almost everywhere on $\mathbb{R}^{n_{\boldsymbol{\theta}}}$. On the other hand, it is trivial to check that $p = q$ almost everywhere on $\mathbb{R}^{n_{\boldsymbol{\theta}}}$ implies $\mathbb{KL}(P||Q) = 0$. □

Owing to this lemma, it seems natural to set $P = p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ and $Q = p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ and to minimize their average KL divergence, with the expectation taken over all possible datasets $\boldsymbol{x} \sim p(\boldsymbol{x}) = \int p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}$. Hence, we consider the following optimization problem in terms of the neural network parameters $\boldsymbol{\phi}$:

$$\widehat{\boldsymbol{\phi}} = \operatorname*{argmin}_{\boldsymbol{\phi}} \mathbb{E}_{p(\boldsymbol{x})}[\mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}))] \tag{3}$$

$$= \operatorname*{argmin}_{\boldsymbol{\phi}} \int p(\boldsymbol{x})\, \mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}))\, d\boldsymbol{x} \tag{4}$$

Let us now assume that the cINN is sufficiently deep and that the domain of $\boldsymbol{\phi}$ allows the loss as stated in (4) to attain the value 0 in its global minimum. Then, we can prove the correctness of the learned posterior.

Proposition 3. *Assume that the global minimizer $\widehat{\boldsymbol{\phi}}$ of (4) satisfies*

$$\int p(\boldsymbol{x})\, \mathbb{KL}\big(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})\big)\, d\boldsymbol{x} = 0,$$

then it follows $p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) = p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ for almost every dataset $\boldsymbol{x}$ with $p(\boldsymbol{x}) > 0$. In other words, the learned posterior approximates the true posterior perfectly.

Proof. Since $p(\boldsymbol{x}) \geq 0$ and $\mathbb{KL}\big(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})\big) \geq 0$ for all $\boldsymbol{x}$, the integrand is non-negative. Thus, the integral being 0 implies $p(\boldsymbol{x})\, \mathbb{KL}\big(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})\big) = 0$ for almost every $\boldsymbol{x}$. If such $\boldsymbol{x}$ further satisfies $p(\boldsymbol{x}) > 0$, we can deduce $\mathbb{KL}\big(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})\big) = 0$, i.e. the densities $p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ and $p_{\widehat{\boldsymbol{\phi}}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ coincide almost everywhere. □

In the following, we reformulate the objective so that its components can be computed directly. By definition of the KL divergence, we have:

$$\begin{aligned}
\widehat{\boldsymbol{\phi}} &= \operatorname*{argmin}_{\boldsymbol{\phi}} \int p(\boldsymbol{x})\, \mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}))\, d\boldsymbol{x} \\
&= \operatorname*{argmin}_{\boldsymbol{\phi}} \iint p(\boldsymbol{x})\, p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \log \frac{p(\boldsymbol{\theta} \,|\, \boldsymbol{x})}{p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})}\, d\boldsymbol{\theta}\, d\boldsymbol{x} \\
&= \operatorname*{argmin}_{\boldsymbol{\phi}} \iint p(\boldsymbol{\theta}, \boldsymbol{x}) (\log p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}))\, d\boldsymbol{\theta}\, d\boldsymbol{x} \qquad (5) \\
&= \operatorname*{argmin}_{\boldsymbol{\phi}} \left[- \iint p(\boldsymbol{\theta}, \boldsymbol{x}) \log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})\, d\boldsymbol{\theta}\, d\boldsymbol{x} \right] \qquad (6)
\end{aligned}$$

Note that formulation (6) allows another interesting interpretation of the objective: We seek to maximize the approximate log-posterior probability $\log p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ over parameters $\boldsymbol{\theta}$ and data $\boldsymbol{x}$ coming from the Bayesian joint distribution $p(\boldsymbol{\theta}, \boldsymbol{x})$.

Next, we apply the change of variables formula for probability densities to the transformation $\boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \boldsymbol{x})$. This leads to the relationship

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \boldsymbol{x}) = p_{\boldsymbol{z}}(f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})) \left|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; \boldsymbol{x})\right|$$

where $\boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; \boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{\theta}} f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})$ denotes the Jacobian of $f_{\boldsymbol{\phi}}$ with respect to $\boldsymbol{\theta}$. Inserting this into equation (6) and writing out the Gaussian density $p_{\boldsymbol{z}}(\boldsymbol{z}) = (2\pi)^{-\frac{n_{\boldsymbol{\theta}}}{2}} \exp\!\big(-\frac{1}{2}\|\boldsymbol{z}\|_2^2\big)$ of the latent variable yield:

$$\widehat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[- \iint p(\boldsymbol{\theta}, \boldsymbol{x}) \big(\log p_{\boldsymbol{z}}(f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x})) + \log \left|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; \boldsymbol{x})\right| \big) \, d\boldsymbol{\theta} \, d\boldsymbol{x} \right] \tag{7}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \iint p(\boldsymbol{\theta}, \boldsymbol{x}) \left(\frac{n_{\boldsymbol{\theta}}}{2} \log(2\pi) + \frac{1}{2} \left\| f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x}) \right\|_2^2 - \log \left|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; \boldsymbol{x})\right| \right) d\boldsymbol{\theta} \, d\boldsymbol{x} \tag{8}$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \iint p(\boldsymbol{\theta}, \boldsymbol{x}) \left(\frac{1}{2} \left\| f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \boldsymbol{x}) \right\|_2^2 - \log \left|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; \boldsymbol{x})\right| \right) d\boldsymbol{\theta} \, d\boldsymbol{x} \tag{9}$$

In practice, calculating these high-dimensional integrals is infeasible. Instead, they are approximated by Monte Carlo estimates. For this purpose, we generate samples $\{(\boldsymbol{\theta}^{(m)}, \boldsymbol{x}^{(m)})\}_{m=1}^M$ from the joint distribution $p(\boldsymbol{\theta}, \boldsymbol{x}) = p(\boldsymbol{\theta})\, p(\boldsymbol{x} \,|\, \boldsymbol{\theta})$ by first sampling $\boldsymbol{\theta}^{(m)} \sim p(\boldsymbol{\theta})$ from the prior and then using the simulation program (1) to realize a dataset $\boldsymbol{x}^{(m)} \sim p(\boldsymbol{x} \,|\, \boldsymbol{\theta}^{(m)})$ from the likelihood. Thus, our objective becomes:

$$\widehat{\boldsymbol{\phi}} \approx \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left(\frac{1}{2} \left\| f_{\boldsymbol{\phi}}\big(\boldsymbol{\theta}^{(m)}; \boldsymbol{x}^{(m)}\big) \right\|_2^2 - \log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}\big(\boldsymbol{\theta}^{(m)}; \boldsymbol{x}^{(m)}\big) \right| \right) \tag{10}$$

2.2.2 Objective Function for Variable Dataset Sizes

In real-world problems, it is highly relevant to deal with varying dataset sizes as different replicates of an experiment or a clinical study, e.g. on tumor growth [Stein et al., 2008], are often observed for different durations. Since each layer of the cINN can only take fixed-size input (cf. Section 2.3), there is need to generalize the method from the previous subsection, in particular the objective (10), so that datasets of varying sizes $N \sim \mathcal{U}(N_{\min}, N_{\max})$ can be handled.

The core idea is to train a summary network $h_{\boldsymbol{\psi}}$ with learnable parameters $\boldsymbol{\psi}$ that transforms the raw data $\boldsymbol{x}_{1:N}$ into a fixed-size vector $h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$. It represents the learned summary statistics from which we want to infer the posterior distribution:

$$\boldsymbol{\theta} \sim p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) \quad \Longleftrightarrow \quad \boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) \text{ with } \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) = \mathcal{N}_{n_{\boldsymbol{\theta}}}(\boldsymbol{z} \,|\, \boldsymbol{0}, \boldsymbol{I}_{n_{\boldsymbol{\theta}}})$$

If we train the summary network jointly with the cINN, the objective function can be formulated, in an analogous way to (3), as follows:

$$\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}} = \underset{\boldsymbol{\phi}, \boldsymbol{\psi}}{\operatorname{argmin}} \, \mathbb{E}_{p(N, \boldsymbol{x}_{1:N})} [\mathbb{KL}(p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}) \,||\, p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})))] \tag{11}$$

Assume that the architecture of both the invertible and the summary network allows for perfect convergence, i.e. the loss (11) becomes 0 for optimal network parameters $\widehat{\boldsymbol{\phi}}$ and $\widehat{\boldsymbol{\psi}}$. Then, the approximate posterior $p_{\widehat{\boldsymbol{\phi}}}\big(\boldsymbol{\theta} \,|\, h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{1:N})\big)$ conditioned on the learned summary statistic recovers the exact posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ for almost all data $\boldsymbol{x}_{1:N}$ (cf. proof of Proposition 3). This would in particular imply that the fixed-size summary statistic $h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{1:N})$ is sufficient, i.e. as good as the original data $\boldsymbol{x}_{1:N}$ to perform Bayesian inference [Deans, 2002]. There is not always a sufficient statistic, but formulation (11) is still well-justified since theoretical work has established a tight connection between minimizing the KL divergence and the summary statistic being maximally informative as measured by the concept of mutual information [Deans, 2002; Wolf and George, 2001].

By the same computation as in Section 2.2.1 and the transformation formula

$$p_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})) = p_{\boldsymbol{z}}(f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}))) \left|\det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}(\boldsymbol{\theta}; h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}))\right|, \tag{12}$$

we eventually derive the Monte Carlo estimate of (11):

$$\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}} \approx \underset{\boldsymbol{\phi},\boldsymbol{\psi}}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left(\frac{1}{2} \left\| f_{\boldsymbol{\phi}}\Big(\boldsymbol{\theta}^{(m)}; h_{\boldsymbol{\psi}}\big(\boldsymbol{x}^{(m)}_{1:N^{(m)}}\big)\Big) \right\|_2^2 - \log \left| \det \boldsymbol{J}_{f_{\boldsymbol{\phi}}}\Big(\boldsymbol{\theta}^{(m)}; h_{\boldsymbol{\psi}}\big(\boldsymbol{x}^{(m)}_{1:N^{(m)}}\big)\Big) \right| \right) \tag{13}$$

The right-hand side determines the loss function $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi})$ that is minimized during the training. Thanks to the cINN architecture, the determinant of the Jacobian matrix $\boldsymbol{J}_{f_{\boldsymbol{\phi}}}$ can be computed efficiently (cf. Section 2.3).

2.2.3 Brief Review and Practical Issues

In this subsection, we shortly review the most important findings so far. The BayesFlow method incorporates the major idea of normalizing flows that is frequently used in recent deep generative modeling frameworks [Rezende and Mohamed, 2015; Kingma and Dhariwal, 2018]. Concretely, an invertible neural network is trained to learn a reversible mapping from the parameter space to the latent space conditional on the data. In this way, the possibly complexly shaped posterior is represented as a pushforward transformation of the much simpler normal distribution of the latent variable. This enables inference of the target distribution as follows:

- Given any parameter vector $\boldsymbol{\theta}$, we can numerically compute a point estimate of the desired posterior probability $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ via the change of variables formula (12).
- By sampling from the latent distribution $\boldsymbol{z}^{(l)} \sim \mathcal{N}_{n_{\boldsymbol{\theta}}}(\boldsymbol{z} \,|\, \boldsymbol{0}, \boldsymbol{I}_{n_{\boldsymbol{\theta}}})$ and evaluating the learned mapping in its inverse direction $\boldsymbol{\theta}^{(l)} = f^{-1}_{\widehat{\boldsymbol{\phi}}}\big(\boldsymbol{z}^{(l)}; h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{1:N})\big)$, we can generate an approximate posterior sample $\boldsymbol{\theta}^{(l)} \sim p_{\widehat{\boldsymbol{\phi}}}\big(\boldsymbol{\theta} \,|\, h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{1:N})\big) \approx p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$.

We have seen that under perfect convergence, the BayesFlow method guarantees perfect global approximation of the posterior distribution given any dataset. For this to hold true, the cINN has to map the posterior correctly onto the Gaussian latent space, and the summary network may not induce any loss of information. Intuitively, this is enforced by minimizing the average KL divergence between the true posterior and the approximate posterior conditioned on the summary statistics as well as by replacing the density of the latent variable by the prescribed Gaussian density (cf. transition (7)-(8)).

In practice, we can usually not achieve perfect convergence in the above sense and it may require some fine-tuning of the network hyperparameters to improve the quality of estimation. In addition, recall that perfect convergence refers to the exact loss (11), whereas in practice, its Monte Carlo estimate is used. To account for this, we can increase the batch size M to reduce the mean squared error of Monte Carlo integration, but this will of course lead to a higher computational cost.

2.3 Invertible Architecture

Invertible neural networks (INNs) are a class of models that are capable of learning a non-linear bijective mapping from inputs to outputs, where both the forward and inverse mapping as well as their Jacobians can be computed efficiently [Ardizzone et al., 2019]. A conditional version of INNs, the cINN, is the centerpiece of the BayesFlow method as it is responsible for the task of learning a normalizing flow between the parameter space of interest and the latent space, thus rendering the target posterior distribution tractable. In this section, we want to understand how INNs are composed and how the conditioning on data is realized.

The basic component of an INN is the affine coupling layer (ACL) [Dinh et al., 2017; Radev et al., 2022]. Each ACL incorporates four internal functions $s_1, t_1 \colon \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ and $s_2, t_2 \colon \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$. In the forward direction, the input vector $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2) \in \mathbb{R}^D$ is split into two halves $\boldsymbol{u}_1 \in \mathbb{R}^{d_1}$ and $\boldsymbol{u}_2 \in \mathbb{R}^{d_2}$ such that $d_1 = \left\lfloor \frac{D}{2} \right\rfloor$ and $d_2 = D - d_1$. They are then passed through a sequence of operations to obtain the output vector $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$:

$$\boldsymbol{v}_1 = \boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2)) + t_1(\boldsymbol{u}_2), \qquad \boldsymbol{v}_2 = \boldsymbol{u}_2 \odot \exp(s_2(\boldsymbol{v}_1)) + t_2(\boldsymbol{v}_1)$$

Here, $\odot$ denotes the element-wise multiplication and $\exp(\cdot)$ the element-wise exponential function. The following lemma ensures that the Jacobian matrix of the transformation $\boldsymbol{u} \mapsto \boldsymbol{v}$ has a tractable determinant, which in particular does not depend on the gradients of the four internal functions.

Lemma 4. *Consider the functions*

$$\Phi_1 \colon \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}, \quad (\boldsymbol{u}_1, \boldsymbol{u}_2) \mapsto (\boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2)) + t_1(\boldsymbol{u}_2),\, \boldsymbol{u}_2) = (\boldsymbol{v}_1, \boldsymbol{u}_2)$$
$$\Phi_2 \colon \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}, \quad (\boldsymbol{v}_1, \boldsymbol{u}_2) \mapsto (\boldsymbol{v}_1,\, \boldsymbol{u}_2 \odot \exp(s_2(\boldsymbol{v}_1)) + t_2(\boldsymbol{v}_1)) = (\boldsymbol{v}_1, \boldsymbol{v}_2)$$

and their concatenation $\Phi = \Phi_2 \circ \Phi_1 \colon \boldsymbol{u} \mapsto \boldsymbol{v}$*. Then, the determinant of the Jacobian* $\boldsymbol{J}_\Phi = \frac{\partial \Phi}{\partial \boldsymbol{u}}$ *is given by the product:*

$$\det \boldsymbol{J}_\Phi = \prod \exp(s_1(\boldsymbol{u}_2)) \cdot \prod \exp(s_2(\boldsymbol{v}_1)) = \exp\left(\sum s_1(\boldsymbol{u}_2) + \sum s_2(\boldsymbol{v}_1)\right)$$

Here, the operators $\prod$ *and* $\sum$ *mean the product and sum of all entries of a vector, respectively.*

Proof. By the chain rule and multiplicativity of the determinant, it suffices to show that:

$$\det \boldsymbol{J}_{\Phi_1} = \prod \exp(s_1(\boldsymbol{u}_2)), \qquad \det \boldsymbol{J}_{\Phi_2} = \prod \exp(s_2(\boldsymbol{v}_1))$$

But this follows from the computations

$$\boldsymbol{J}_{\Phi_1} = \begin{pmatrix} \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_1} & \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_2} \\ \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_1} & \frac{\partial \boldsymbol{u}_2}{\partial \boldsymbol{u}_2} \end{pmatrix} = \begin{pmatrix} \operatorname{diag}(\exp(s_1(\boldsymbol{u}_2))) & \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_2} \\ \boldsymbol{0} & \boldsymbol{I}_{d_2} \end{pmatrix},$$
$$\boldsymbol{J}_{\Phi_2} = \begin{pmatrix} \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{v}_1} & \frac{\partial \boldsymbol{v}_1}{\partial \boldsymbol{u}_2} \\ \frac{\partial \boldsymbol{v}_2}{\partial \boldsymbol{v}_1} & \frac{\partial \boldsymbol{v}_2}{\partial \boldsymbol{u}_2} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_{d_1} & \boldsymbol{0} \\ \frac{\partial \boldsymbol{v}_2}{\partial \boldsymbol{v}_1} & \operatorname{diag}(\exp(s_2(\boldsymbol{v}_1))) \end{pmatrix}$$

and the fact that the determinant of a triangular matrix is given by the product of its diagonal entries. □

Another crucial property of the non-linear transformation $\boldsymbol{u} \mapsto \boldsymbol{v}$ is that it is bijective. Indeed, we can easily check that its inverse is given by:

$$\boldsymbol{u}_2 = (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1)) \odot \exp(-s_2(\boldsymbol{v}_1)), \qquad \boldsymbol{u}_1 = (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2)) \odot \exp(-s_1(\boldsymbol{u}_2))$$

Note that the internal functions $s_1(\cdot)$, $s_2(\cdot)$, $t_1(\cdot)$, $t_2(\cdot)$ are only called in their forward direction and therefore need not be invertible themselves. In BayesFlow, they are implemented via fully connected neural networks with exponential linear units [Clevert et al., 2016].

The ACL represents a basic invertible layer that allows for the following learning technique: Use only the forward pass $\boldsymbol{u} \mapsto \boldsymbol{v}$ to optimize the network parameters. Once the training is finished, the invertible architecture will give us the inverse mapping $\boldsymbol{v} \mapsto \boldsymbol{u}$ for free.

In practice, multiple ACLs are stacked to create an invertible chain with sufficient expressivity. The input to the first ACL is the parameter vector $\boldsymbol{\theta}$, and the output of the final ACL is the latent variable $\boldsymbol{z}$. To make the learned transformation depend on the data $\boldsymbol{x}_{1:N}$, we additionally feed the summary vector $\widetilde{\boldsymbol{x}} = h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N})$ into the internal networks of each ACL (hence the name **conditional** INN). Importantly, the internal networks can only deal with fixed input shapes, so a summary network is really needed to reduce variable-size raw data to fixed-size summary statistics. In this way, the forward pass through a single ACL becomes:

$$\boldsymbol{v}_1 = \boldsymbol{u}_1 \odot \exp(s_1(\boldsymbol{u}_2; \widetilde{\boldsymbol{x}})) + t_1(\boldsymbol{u}_2; \widetilde{\boldsymbol{x}}), \qquad \boldsymbol{v}_2 = \boldsymbol{u}_2 \odot \exp(s_2(\boldsymbol{v}_1; \widetilde{\boldsymbol{x}})) + t_2(\boldsymbol{v}_1; \widetilde{\boldsymbol{x}})$$

and the inverse pass through a single ACL becomes:

$$\boldsymbol{u}_2 = (\boldsymbol{v}_2 - t_2(\boldsymbol{v}_1; \widetilde{\boldsymbol{x}})) \odot \exp(-s_2(\boldsymbol{v}_1; \widetilde{\boldsymbol{x}})), \qquad \boldsymbol{u}_1 = (\boldsymbol{v}_1 - t_1(\boldsymbol{u}_2; \widetilde{\boldsymbol{x}})) \odot \exp(-s_1(\boldsymbol{u}_2; \widetilde{\boldsymbol{x}}))$$

The entire chain of ACLs then represents the cINN $f_{\boldsymbol{\phi}}$, which can learn the desired normalizing flow from the parameter space $\boldsymbol{\theta}$ to the latent space $\boldsymbol{z}$ using conditioning information from the data. The complete forward and inverse pass through the cINN are given by $\boldsymbol{z} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta}; \widetilde{\boldsymbol{x}})$ and $\boldsymbol{\theta} = f_{\boldsymbol{\phi}}^{-1}(\boldsymbol{z}; \widetilde{\boldsymbol{x}})$, respectively.

Notice that the determinant of $\boldsymbol{J}_{f_{\boldsymbol{\phi}}}$ remains tractable as a product of tractable determinants as discussed in Lemma 4. This is important for the efficient evaluation of the loss function (13).

2.4 Algorithm and Implementation

In the following, we present the original BayesFlow algorithm to illustrate the workflow with this method (Algorithm 1 and Figure 1). It is directly taken from Radev et al. [2022] with a slight adaptation of notation. The network training makes use of an online learning approach. A crucial feature is therefore that the neural network will never see a training dataset twice so that overfitting is nearly impossible.

It is important to notice that unlike suggested by the loss formulation in (13), the number N of observations is not sampled separately for every dataset, but is kept fixed throughout a training batch. Thus, the true value of the loss function is only approximated when averaging over many iterations. But since the neural network parameters are updated after each iteration, this might affect the convergence properties of the optimization algorithm. On the other hand, there is the clear advantage of being able to perform vectorized operations on the entire batch and thus to speed up step 8 of the algorithm.

For this thesis, we work with the current implementation of the BayesFlow method provided at `https://github.com/stefanradev93/BayesFlow` (accessed on May 06, 2022). It uses the programming language Python and the TensorFlow library for creating the machine learning models. The stochastic gradient descend method for optimizing the network parameters is the Adam optimizer with an initial learning rate of 0.001 and an exponential decay rate of 0.95. Gradient calculation is done via backpropagation of error.

For most of our numerical experiments, we performed 300 training epochs of 1000 iterations using a batch size of 128 and stored the network parameters after the final epoch. Exceptions will be reported when discussing the respective model. We did not carry out any exhaustive search for optimal training hyperparameters nor parallelized the batch simulation. Each neural network was trained on 1 CPU (AMD EPYC 7443 2.85 GHz) with 48 cores and 1 TB RAM from the "unicorn" cluster (Hasenauer lab, University of Bonn, Germany). All code and simulation scripts can be found at `https://github.com/zijianwang2000/BA_MissingData`, and a snapshot is available on Zenodo under the DOI `https://doi.org/10.5281/zenodo.7004878`.

Algorithm 1 Amortized Bayesian inference with the BayesFlow method

Training phase (*online learning with batch size* M):
repeat
  Sample number of observations: $N \sim \mathcal{U}(N_{\min}, N_{\max})$.
  for $m = 1, \ldots, M$ **do**
    Sample model parameters from the prior: $\boldsymbol{\theta}^{(m)} \sim p(\boldsymbol{\theta})$.
    Sample a stochastic instance: $\boldsymbol{\xi}^{(m)} \sim p(\boldsymbol{\xi})$.
    Run the simulation (1) to generate a synthetic dataset: $\boldsymbol{x}_{1:N}^{(m)} = g(\boldsymbol{\theta}^{(m)}, \boldsymbol{\xi}^{(m)})$.
    Pass the dataset $\boldsymbol{x}_{1:N}^{(m)}$ through the summary network: $\widetilde{\boldsymbol{x}}^{(m)} = h_{\boldsymbol{\psi}}(\boldsymbol{x}_{1:N}^{(m)})$.
    Pass $(\boldsymbol{\theta}^{(m)}, \widetilde{\boldsymbol{x}}^{(m)})$ through the inference network in forward direction: $\boldsymbol{z}^{(m)} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(m)}; \widetilde{\boldsymbol{x}}^{(m)})$.
  end for
  Compute the loss $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi})$ according to (13) from the training batch $\left\{(\boldsymbol{\theta}^{(m)}, \widetilde{\boldsymbol{x}}^{(m)}, \boldsymbol{z}^{(m)})\right\}_{m=1}^{M}$.
  Update neural network parameters $\boldsymbol{\phi}, \boldsymbol{\psi}$ via backpropagation.
until convergence to $\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}}$

Inference phase (*given observed or test data* $\boldsymbol{x}_{1:N}^{\mathrm{o}}$):
Pass the observed dataset through the summary network: $\widetilde{\boldsymbol{x}}^{\mathrm{o}} = h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{1:N}^{\mathrm{o}})$.
for $l = 1, \ldots, L$ **do**
  Sample a latent variable instance: $\boldsymbol{z}^{(l)} \sim \mathcal{N}_{n_{\boldsymbol{\theta}}}(\boldsymbol{z} \,|\, \boldsymbol{0}, \boldsymbol{I}_{n_{\boldsymbol{\theta}}})$.
  Pass $(\boldsymbol{z}^{(l)}, \widetilde{\boldsymbol{x}}^{\mathrm{o}})$ through the inference network in inverse direction: $\boldsymbol{\theta}^{(l)} = f_{\widehat{\boldsymbol{\phi}}}^{-1}(\boldsymbol{z}^{(l)}; \widetilde{\boldsymbol{x}}^{\mathrm{o}})$.
end for
Return $\left\{\boldsymbol{\theta}^{(l)}\right\}_{l=1}^{L}$ as a sample from $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}^{\mathrm{o}})$.

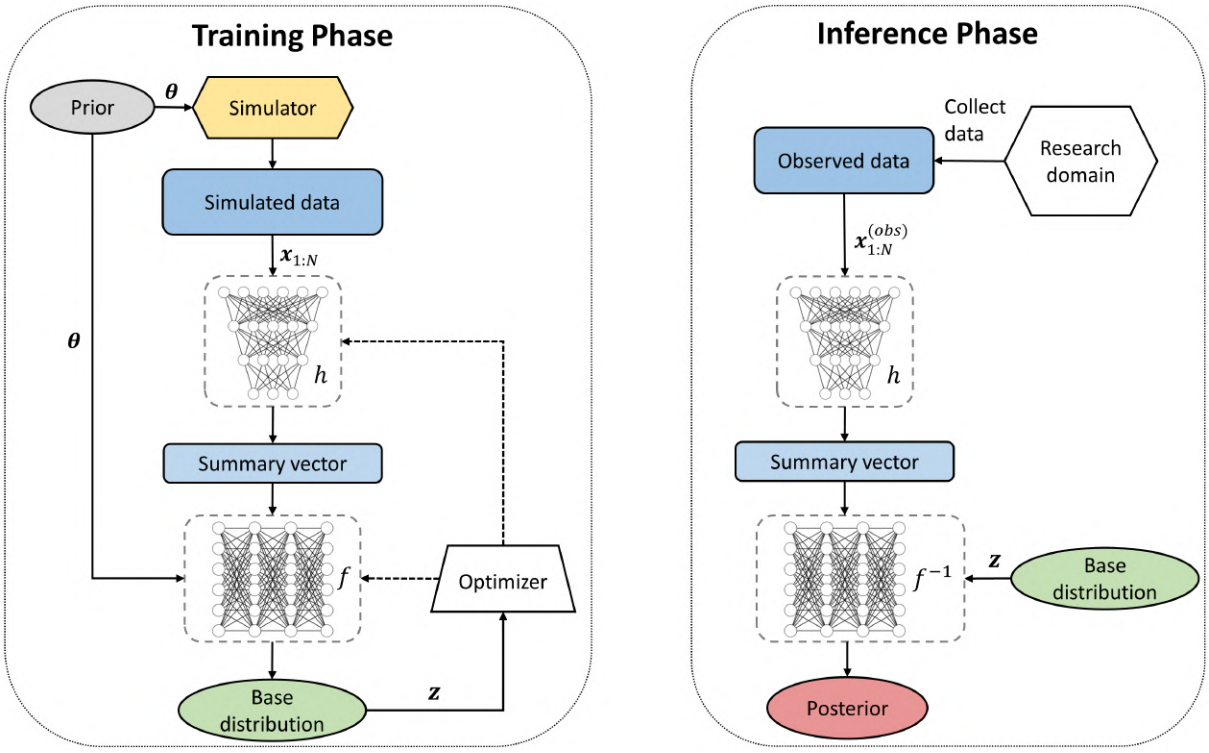


Figure 1: Visualization of the workflow with the BayesFlow method (`https://bayesflow.readthedocs.io/en/latest`, accessed on April 09, 2022)

3 Performance on Complete Data

In this section, we use two simple forward models to validate the overall good performance of the BayesFlow method in the case of complete data. The first one is a two-dimensional normal distribution for which a closed-form solution of the true posterior is available, and we seek to confirm that the method provides correct results in such a trivial case. The second one is the conversion reaction model. It is a simple ODE model that we want to get familiar with, as its time series structure will be of interest when it comes to the handling of missing data later in this thesis. Further, we illustrate some of our experiences with the choice of summary networks and prior distributions. But before discussing the specific models, we will briefly introduce the employed performance validation techniques.

3.1 Performance Validation Methods

The evaluation of Bayesian sampling methods is in general not an easy task. In the following, we provide an overview of the methods that are used to validate and compare the performance of the trained deep generative models in this thesis.

- **Visual fit**: In all our test models, the parameter vector of interest is two-dimensional, so it is natural to plot the samples generated by the BayesFlow method against the level sets of the true posterior to visually assess the quality of approximation. Besides, the majority of our test models belongs to the class of ODE models for which the likelihood function and posterior probability can be evaluated with reasonable numerical effort [Fröhlich et al., 2019]. In addition, we employ numerical quadrature to compute the marginal densities of the true posterior and compare them with the marginalized BayesFlow samples.

- **Posterior predictive check**: For time series models, we plot the trajectories generated by the approximate posterior samples against the trajectory and test dataset generated by the ground truth parameters. In this way, we can visually check how well the re-simulated trajectories fit the given data.

- **KL loss**: We report the average loss of the final epoch. Note that the loss function (13) for optimizing the network parameters is only the Monte-Carlo estimate of the average Kullback-Leibler divergence up to a model-specific constant given by (cf. the transitions (5)-(6) and (8)-(9)):

$$\iint p(\boldsymbol{\theta}, \boldsymbol{x}_{1:N}) \Big(\log p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N}) + \frac{n_{\boldsymbol{\theta}}}{2} \log(2\pi) \Big) \, d\boldsymbol{\theta} \, d\boldsymbol{x}_{1:N}$$

  For simple forward models, we reconstruct this constant via Monte-Carlo integration and report the loss after the constant has been added. In this case, the value of the reported KL loss measures the expected deviation of the learned posterior from the true posterior. For more complex forward models, computing the correction constant faithfully would be too costly, so we only report the uncorrected KL loss (which will of course be mentioned). In this case, only a comparison of the losses achieved by different neural networks on the same forward model can be informative.

- **Validation metrics**: The normalized root mean squared error NRMSE and the coefficient of determination R^2 are defined as follows for a sample of true parameters $\{\nu^{(m)}\}_{m=1}^M$ and a sample of estimated parameters $\{\widehat{\nu}^{(m)}\}_{m=1}^M$:

$$\mathrm{NRMSE} \coloneqq \sqrt{\sum_{m=1}^{M} \frac{\left(\nu^{(m)} - \widehat{\nu}^{(m)}\right)^2}{\nu_{\max} - \nu_{\min}}}, \qquad R^2 \coloneqq 1 - \frac{\sum_{m=1}^{M} \left(\nu^{(m)} - \widehat{\nu}^{(m)}\right)^2}{\sum_{m=1}^{M} \left(\nu^{(m)} - \bar{\nu}\right)^2}$$

  Here, $\nu_{\max}$, $\nu_{\min}$ and $\bar{\nu}$ denote the maximum, minimum and mean of the true parameters, respectively. NRMSE measures how accurately the true parameter values are recovered by the estimates and R^2 measures the proportion of variation in the sample of true parameters

that is explained by the sample of estimated parameters. Perfect recovery is achieved when NRMSE $= 0$ and $R^2 = 1$.

Ideally, we would like to compare the means of true and approximate posteriors across a large number of test datasets. But computing the mean of the true posterior can become quite demanding even if point evaluations of the posterior probability are possible. Thus, apart from the first and simplest model, we will always compare the ground truth parameters (data generating parameters) with the means of the approximate posterior samples. In this case, low NRMSE and high R^2 values are still reasonable indicators of good performance. But it is important to keep in mind that especially for uninformative datasets (e.g. few observations or high noise level), the true posteriors are not necessarily centered at the ground truth parameters so we cannot expect the validation metrics to take perfect values even if the posteriors are approximated perfectly.

- **Simulation-based calibration**: Simulation-based calibration (SBC) exploits the insight that the Bayesian joint distribution is self-consistent [Talts et al., 2018]. Concretely, it can be shown that averaging the exact posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}')$ over data $\boldsymbol{x}'$ coming from the joint distribution $p(\boldsymbol{\theta}', \boldsymbol{x}') = p(\boldsymbol{\theta}')\, p(\boldsymbol{x}' \,|\, \boldsymbol{\theta}')$ will recover the prior distribution (for a proof, see Appendix B in Radev et al. [2022]):

$$p(\boldsymbol{\theta}) = \iint p(\boldsymbol{\theta} \,|\, \boldsymbol{x}')\, p(\boldsymbol{\theta}', \boldsymbol{x}')\, d\boldsymbol{\theta}'\, d\boldsymbol{x}'$$

If the exact posterior is replaced by an inadequate approximation, the above equality will be violated. Such violations can be detected by computing rank statistics which compare a sample of data generating prior parameters with the according approximate posterior samples and inspecting the resulting histogram for uniformity (SBC plot). While uniform SBC plots indicate good approximation, different types of deviations from the uniformity allow interpretations such as over-/underfitting or a systematic bias in the approximate posteriors. For illustrative examples of such deviations and their interpretations, we refer to Talts et al. [2018].

3.2 Multivariate Normal Distribution

In this section, we employ the above introduced techniques to validate the performance of the BayesFlow method on our simplest model, a two-dimensional normal distribution. Concretely, we consider the following forward model:

$$\begin{aligned} \boldsymbol{\mu}^{(m)} &\sim \mathcal{N}_2(\boldsymbol{\mu} \,|\, \mathbf{0}, \boldsymbol{I}_2) && \text{(Prior distribution)} \\ \boldsymbol{x}^{(m)} &\sim \mathcal{N}_2(\boldsymbol{x} \,|\, \boldsymbol{\mu}^{(m)}, \boldsymbol{\Sigma}) && \text{(Sampling from the likelihood)} \end{aligned}$$

$\boldsymbol{\Sigma}$ denotes a known covariance matrix and is set to be $\begin{pmatrix} 0.5 & -0.35 \\ -0.35 & 1 \end{pmatrix}$ in our experiment.

To give a Bayesian interpretation of our problem: We are interested in the posterior probability $p(\boldsymbol{\mu} \,|\, \boldsymbol{x})$ of the mean vector $\boldsymbol{\mu}$ when observing a dataset $\boldsymbol{x} = \boldsymbol{x}_{1:N}$ of size $N = 1$ sampled from the likelihood, and our prior belief is that $\boldsymbol{\mu}$ comes from a 2D standard normal distribution.

This toy model admits a closed-form solution. Standard computations in multivariate statistics show that the ground truth posterior distribution is given by $p(\boldsymbol{\mu} \,|\, \boldsymbol{x}) = \mathcal{N}_2(\boldsymbol{\mu} \,|\, \boldsymbol{B}\boldsymbol{x}, \boldsymbol{B}\boldsymbol{\Sigma})$ where $\boldsymbol{B} := (\boldsymbol{I}_2 + \boldsymbol{\Sigma})^{-1}$.

In our experiment, we choose a cINN with 3 affine coupling layers, each consisting of $[32, 32, 32]$ units. No summary network is needed since the dataset size $N = 1$ is constant and low, and also the dimension $n_{\boldsymbol{x}} = 2$ of the observed vector is low. As this is a very simple model, we train the cINN for only 75 epochs of 1000 iterations. The KL loss is successfully driven down to 0.000, as opposed to an initial value of above 3.

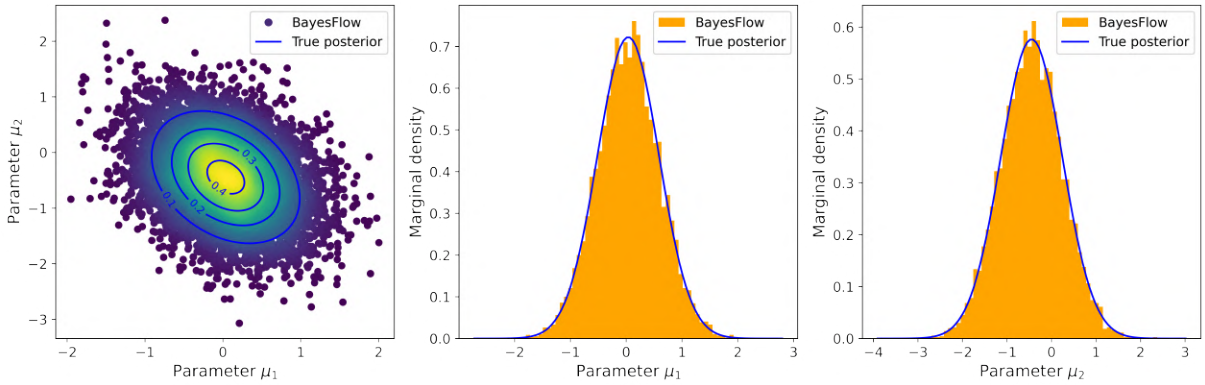
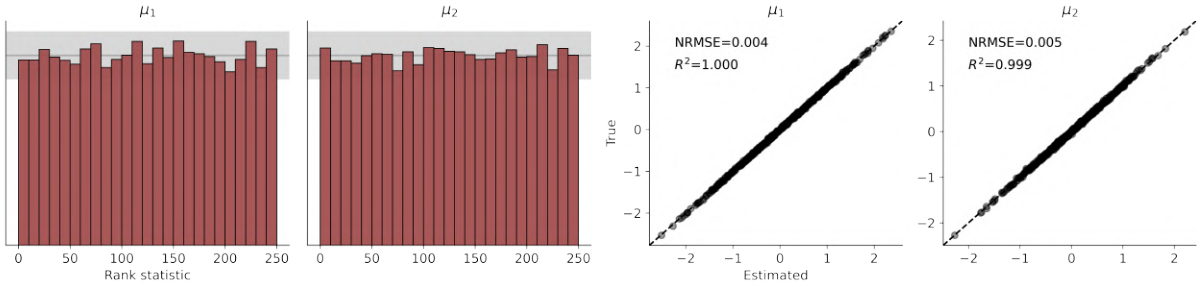


Figure 2: 2D normal distribution model: Visual fit displayed for one example dataset



(a) SBC

(b) Validation metrics

Figure 3: Performance validation for the 2D normal distribution model

To assess the quality of approximation, we inspect the visual fit for multiple test datasets and observe that the BayesFlow samples recover the true posterior distribution very well (Figure 2). In addition, the SBC plot exhibits uniformity and the metrics NRMSE and R^2 (comparing the means of true and approximate posteriors across 500 test datasets) attain nearly optimal values (Figure 3). We thus conclude that BayesFlow indeed provides almost perfect approximation for this simple model.

3.3 Conversion Reaction Model

Let us consider the simple conversion process

$$X_1 \underset{c_2}{\overset{c_1}{\rightleftharpoons}} X_2$$

with rate parameters $c_1, c_2 > 0$. If we denote the concentrations of the involved chemical species by x_1 and x_2, then their dynamics are described by the following reaction rate equation:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -c_1 x_1 + c_2 x_2 \\ c_1 x_1 - c_2 x_2 \end{pmatrix}$$

By specifying the initial value $(x_1(0), x_2(0)) = (1, 0)$, this linear system of ODEs has the unique analytic solution:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \frac{1}{c_1 + c_2} \left[\begin{pmatrix} c_2 \\ c_1 \end{pmatrix} + \begin{pmatrix} c_1 \\ -c_1 \end{pmatrix} e^{-(c_1+c_2)t} \right] \quad \text{for } t \geq 0$$

We assume that the second state is measured up to additive normal noise with known standard deviation $\sigma = 0.015$, i.e. the observation at time t is given by:

$$y_t = x_2(t) + \varepsilon_t \;\text{ with } \varepsilon_t \sim \mathcal{N}(0,\, 0.015^2) \text{ independently in } t$$

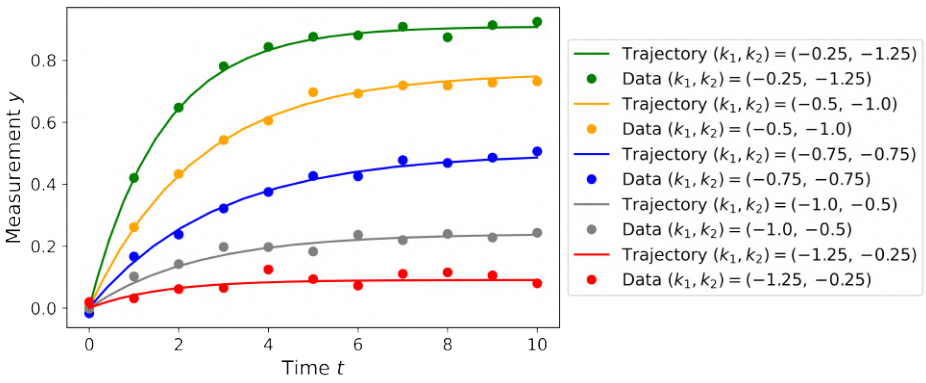


Figure 4: Conversion reaction model with $N = 11$ observations: Simulation of trajectories and noisy data for different parameters (k_1, k_2) from the 2σ-interval of their respective prior distribution

We choose normal priors for the log-scale parameters $k_j = \log_{10}(c_j)$:

$$k_1, k_2 \sim \mathcal{N}(-0.75,\, 0.25^2) \text{ i.i.d.}$$

The prior distribution is broad enough to allow sufficiently different dynamics, but also reasonably narrow so that it is unlikely to sample model parameters leading to very flat trajectories, in which case the measurement noise would be too dominant (Figure 4). Our goal is to infer the posterior distribution $p(k_1, k_2 \,|\, y_{0:10})$ given 11 equidistant data points at $t = 0, 1, \ldots, 10$.

3.3.1 Performance Without Summary Network

In this subsection, we demonstrate that BayesFlow performs very well on the conversion reaction model. We train a cINN with 4 affine coupling layers, each consisting of $[64, 64, 64]$ units. The simulated datasets of constant size $N = 11$ are directly fed into the invertible network, without being preprocessed by any summary network. Within 300 training epochs, the KL loss is driven from an initial value of above 10 down to 0.000.

Across various test datasets, we verify that the true posterior distribution is approximated by the BayesFlow samples almost perfectly. Furthermore, we find that the trajectories generated by the BayesFlow samples precisely fit the observed data. These observations are displayed for one example dataset (Figure 5 and 6).

The SBC plot indicates neither a systematic bias nor over-/underfitting of the learned posterior. The validation metrics confirm that the ground truth parameters are recovered reasonably well (Figure 7). Recall that due to the relatively small dataset size and thus non-negligible measurement noise, we cannot expect NRMSE and R^2 to attain the values 0 and 1, respectively, even if the posterior distributions are approximated perfectly.

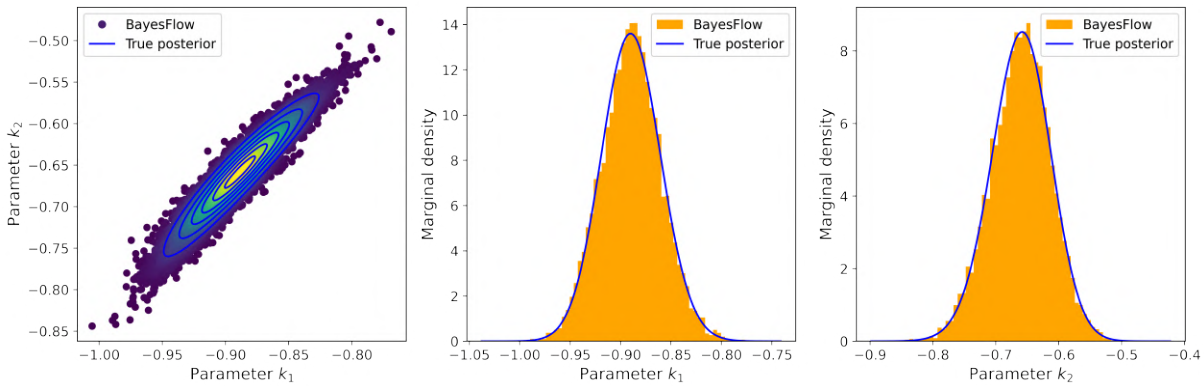


Figure 5: Visual fit for a test dataset generated by ground truth parameters $(k_1, k_2) = (-0.9, -0.7)$

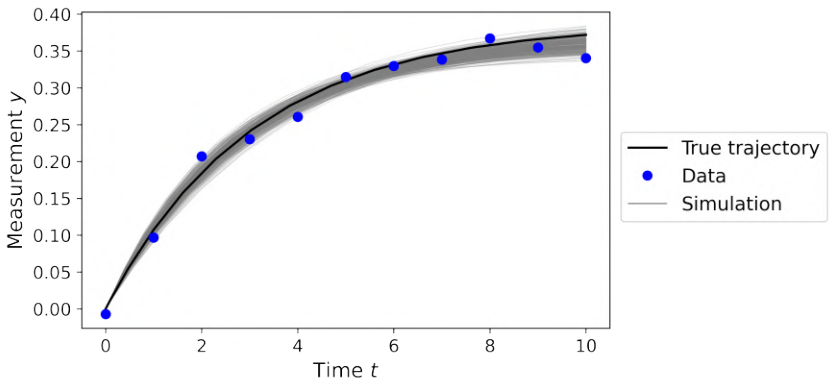
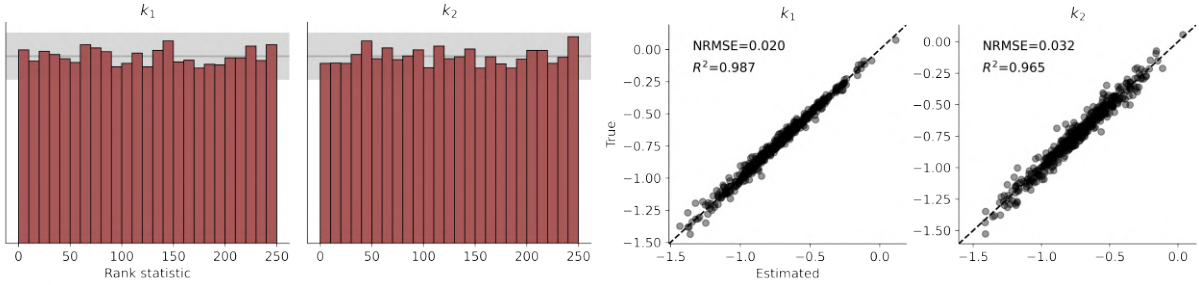


Figure 6: Posterior predictive check using 300 BayesFlow samples



(a) SBC

(b) Validation metrics

Figure 7: Performance validation for the conversion reaction model (no summary network)

3.3.2 Hyperparameter Tuning for Summary Networks

We have seen the good performance of the above BayesFlow workflow that involves no summary statistics learning. From a technical point of view, this was only possible since the considered model produces datasets of constant length. However, in real-world applications, we often find ourselves confronted with variable-size data, e.g. when different replicates of an experiment are monitored for different durations, or when missing data play a role. In these cases, the training of a summary network which maps raw datasets to fixed-size vectors is inevitable.

In fact, the BayesFlow developers recommend the use of a summary network whenever dealing with time series data, even in the case that all datasets share the same length. The intuition behind this is clear: We want to profit from the preprocessing or even dimensionality reduction achieved by a summary network that is suited to capture the time series structure of the data. In this way, the expressive power of the cINN is used more efficiently since it can really focus on the task of learning the desired normalizing flow.

A natural candidate for such a summary network would be Long Short-Term Memory (LSTM) networks [Gers et al., 1999]. They are the state-of-the-art representative of the class of recurrent neural networks, can deal with variable-size data and have proven notably successful not only in the fields of speech recognition [Oruh et al., 2022] and time series forecasting [Sangiorgio and Dercole, 2020], but also as summary networks in the BayesFlow framework [Radev et al., 2022].

In the following, we use the conversion reaction model to investigate the tradeoff between the above-mentioned advantages and the possible disadvantages of increased training time and information loss induced by the summary network. We do not aim to do a detailed benchmark, but rather hope to derive a rule of thumb on how to choose an appropriate LSTM network for the purposes of this thesis, i.e. practically feasible in terms of training time and fair enough to compare different approaches to handle missing data.

Concretely, we train a cINN jointly with a single-layer LSTM network with $n \in \{4, 8, 16\}$ hidden

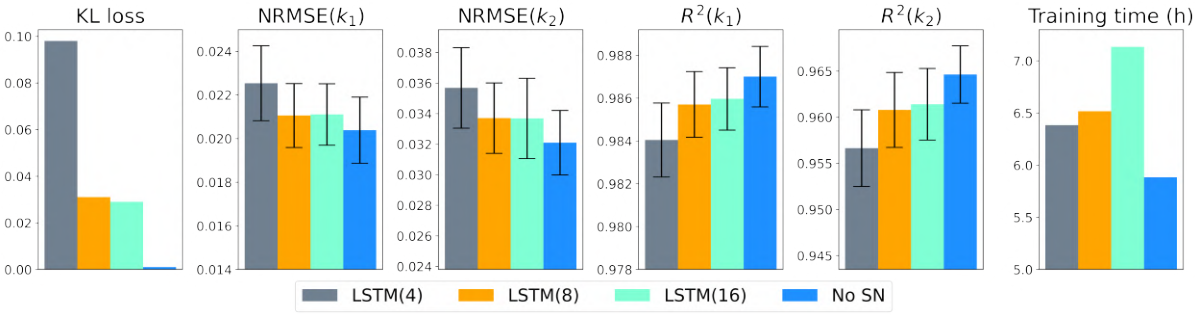


Figure 8: Comparison of KL loss, validation metrics and training time for the tested summary networks. For NRMSE and R^2, bootstrapped means (± 1 standard error) are calculated based on 500 test datasets.

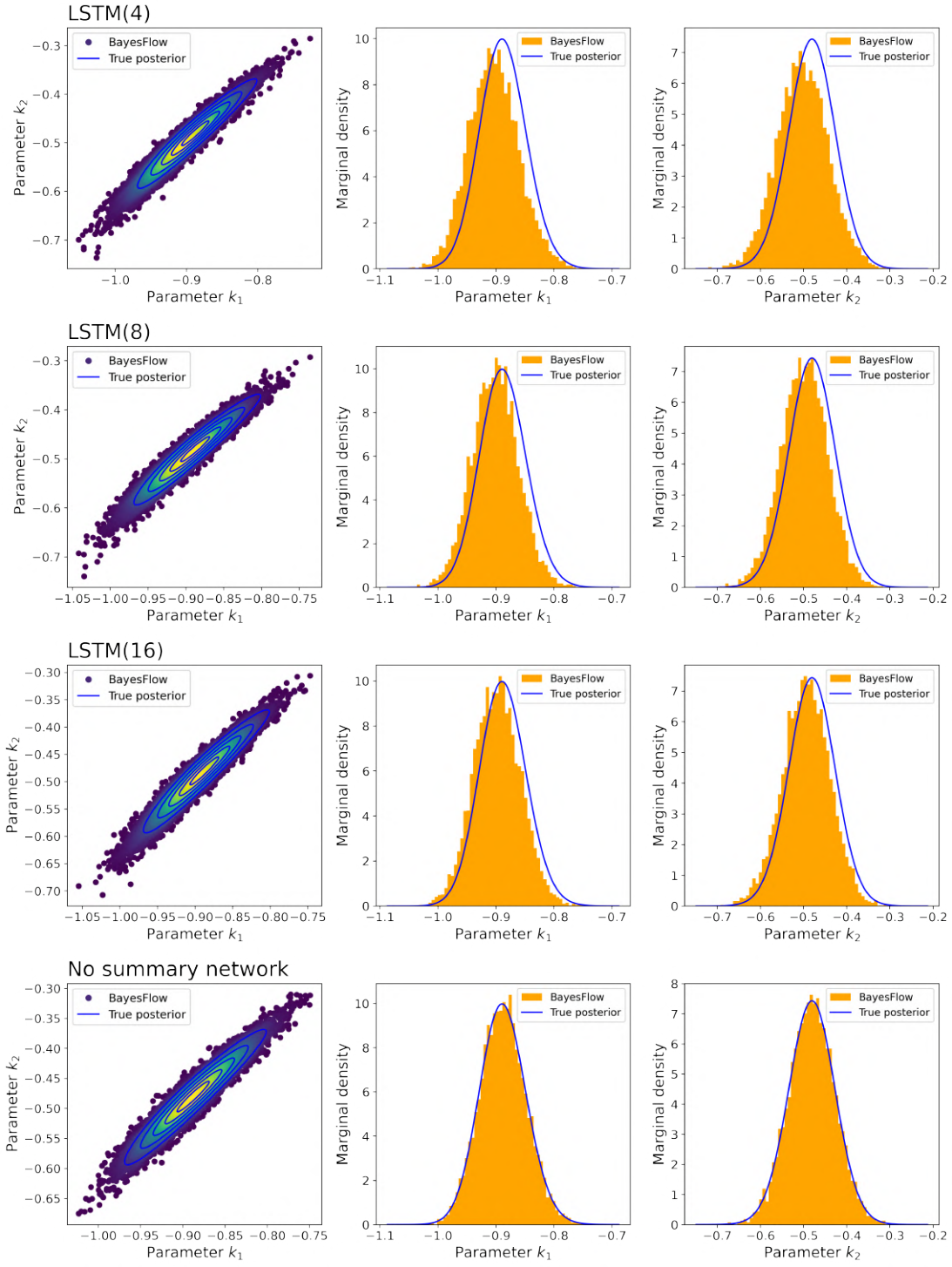


Figure 9: Comparison of visual fit for the tested summary networks based on one example dataset

units, which is abbreviated with LSTM(n). The cINN hyperparameters remain the same as in the previous subsection. While the SBC plots and data fitting look similarly fine and hence are not displayed here, the validation metrics do detect a systematic improvement as the number of hidden units increases (Figure 8). Especially the improvement from LSTM(4) to LSTM(8) is very clear, whereas the improvement from LSTM(8) to LSTM(16) is only slight. And it appears that for this simple ODE model, the workflow using no summary network even outperforms the ones with a LSTM network. These observations are confirmed on test datasets by plotting the respective BayesFlow samples against the true posterior (Figure 9). Last but not least, one should keep in mind that training LSTM networks becomes increasingly expensive with a larger number of hidden units.

To sum it up, hyperparameter tuning for the LSTM summary network remains a difficult but crucial task, since this has a significant influence on both the accuracy and feasibility of the BayesFlow workflow. Although there is by far not enough experimental data to give a universal recommendation, we will, for the rest of this thesis, stick to the following simple rule of thumb:

Consider a time series model which generates datasets of the shape $(N_{\max}, D)$, i.e. each dataset consists of at most $N_{\max}$ D-dimensional observations. Then, we choose a LSTM network with $n = 2^k$ hidden units, where k is the smallest integer such that $2^k \geq N_{\max} \cdot D$.

The underlying heuristic is the following: On the one hand, the lower bound on n should ensure that the LSTM network is expressive enough to capture all the relevant information in a given dataset. On the other hand, the minimizing condition should prevent the LSTM network from being unnecessarily complex and its training from becoming inefficient. The proposed rule of thumb indeed matches our empirical observations from the conversion reaction model (where $N_{\max} = 11$, $D = 1$ and the effect of the LSTM network with $n = 16 \geq 11 \cdot 1$ hidden units seems reasonably good and "saturated"), and it will also prove reasonable for all other models in this thesis.

3.3.3 Difficulty of Learning Hard Boundaries

In many application examples, we have a priori knowledge about lower and upper bounds for the parameters of interest. In these cases, it seems reasonable to assume uniform priors. We will illustrate in this subsection that the BayesFlow method experiences some difficulty in learning the resulting hard-bounded posteriors.

Let us again consider the conversion reaction model introduced at the beginning of Section 3.3. However, we now choose uniform instead of normal priors for the log-scale rate parameters:

$$k_1, k_2 \sim \mathcal{U}(-1.5,\, 0) \text{ i.i.d.}$$

According to Bayes' theorem, the posterior distribution $p(k_1, k_2 \,|\, y_{0:10})$ must then have zero mass outside the square $[-1.5,\, 0]^2$.

We train a cINN with 5 affine coupling layers on this model. We deliberately added one ACL compared to the cINN for the model with normal priors, since we want to account for the more challenging task of learning a normalizing flow between the smooth latent distribution and a hard-bounded target distribution.

Our key observations are as follows: If the high density region of the true posterior is located far away from the boundary of $[-1.5,\, 0]^2$, then the learned posterior fits the true posterior very well (cf. Figure 5). However, if the true posterior is peaked near the boundary, the approximation is rather poor, with a significant portion of the BayesFlow samples lying outside the boundary (Figure 10).

We understand this as a warning when working with uniform priors. Especially when the number of observations is low, i.e. the observed dataset is rather uninformative, it is likely to encounter relatively broad posteriors which have a high density near the boundary of the support of the

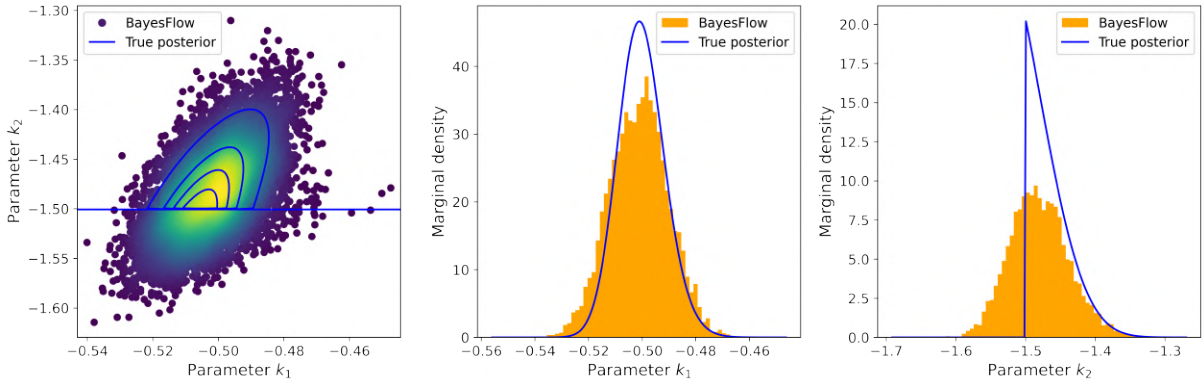


Figure 10: For an example dataset simulated with ground truth parameters $(k_1, k_2) = (-0.5, -1.45)$, the learned posterior clearly exceeds the hard boundary of the true posterior.

uniform prior. In these cases, we need to be aware that the learned posterior might not be able to approximate the sudden decrease of density at the boundary very well. A possible solution is to reparametrize the model parameters via a suitable logistic function and thus to transform hard-bounded uniform priors into smooth priors. This would however also change the inference problem as the posterior is not invariant under reparametrization. Most of the models in this thesis will use normal priors.

4 Missing Data Handling

The term **missing data** (or **missing values**) describes incomplete datasets in which no numerical value is stored for one or multiple observations. In many research areas such as biomedical sciences, missing data can occur and reduce the power and reliability of statistical inference, even if the data collection process is well-designed. For instance, probands of a clinical study might not show up at some of the regular screenings. Further, we could think of the COVID-19 pandemic, where infection numbers are not reported on special holidays or due to the overload of health departments, and data acquisition might vary in different regions.

4.1 Problem Specification

We maintain the overall setting and notation from Section 2.1, but will assume, from now on, that we are dealing with a time series model which produces datasets $\boldsymbol{x}_{1:N}$ of fixed length N. Especially, the observations $\boldsymbol{x}_i \in \mathbb{R}^{n_{\boldsymbol{x}}}$ of such a dataset should be understood as measurements assigned to a strictly increasing sequence of known time points $0 \leq t_1 < \cdots < t_N$.

In the case of missing data, an underlying complete dataset $\boldsymbol{x}_{1:N}$ is only partially known. Let

$$\mathcal{S} \coloneqq \{1 \leq i \leq N \colon \text{No value is available for } \boldsymbol{x}_i\}, \quad \mathcal{T} \coloneqq \{1, \ldots, N\} \setminus \mathcal{S}$$

be the index sets associated to the missing and available data points, respectively. For short, we will refer to them as the set of missing or available indices.

At this point, we need to differentiate between two distributions of interest:

- The posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{\mathcal{T}})$ conditioned on the available data $\boldsymbol{x}_{\mathcal{T}} \coloneqq \{\boldsymbol{x}_i\}_{i \in \mathcal{T}}$
- The posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{1:N})$ conditioned on the complete original dataset $\boldsymbol{x}_{1:N}$

It should be clear that the second problem is intrinsically more challenging as it involves the faithful reconstruction of missing values, which can turn out quite difficult for more complex dynamics. We will provide a brief overview of potential approaches to tackle this problem in Section 5.4.

For the time being, we want to focus on the first problem, the estimation of $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{\mathcal{T}})$. Within this formulation, assuming N to be fixed induces no loss of generality since any dataset $\boldsymbol{x}_{1:N'}$ with fewer time steps $N' < N$ can be simply treated as a special type of missing data with $\mathcal{S} = \{N' + 1, \ldots, N\}$ (cf. Section 5.1.4).

4.2 Encoding Missing Values

In order to estimate the posterior $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{\mathcal{T}})$ conditioned on the available data, we need to find a way to encode missing values so that the neural network can learn to detect and ignore them. To this end, we propose and investigate the following three approaches:

- A1: Augment the data matrix $\left[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\right]$ by a binary row, where 0 indicates absence and 1 indicates presence of a data point. Further, insert a constant value $C \in \mathbb{R}$ for all missing data points. This approach will be referred to as "Augment by 0/1", and the fill-in constant C will be specified if relevant.
- A2: Simply insert a constant value $C \in \mathbb{R}$ for all missing data points. This approach will be called "Insert C".
- A3: Augment the data matrix containing only the available observations by a row of time points associated to these observations. This approach will be called "Time labels".

Example. *The data matrix*

$$\begin{bmatrix} 0.9 & \text{N/A} & 3.1 & 4.0 & \text{N/A} \\ 5.1 & \text{N/A} & 3.0 & 2.2 & \text{N/A} \end{bmatrix}$$

consisting of three available 2D observations at the time points $t_1 = 0.0$, $t_3 = 1.0$ *and* $t_4 = 1.5$ *and two missing observations at* $t_2 = 0.5$ *and* $t_5 = 2.0$ *is processed as follows:*

$$\begin{aligned}
\text{Augment by 0/1} \\ \text{(fill-in value } -1\text{)} \quad &\Longrightarrow \quad \begin{bmatrix} 0.9 & \text{N/A} & 3.1 & 4.0 & \text{N/A} \\ 5.1 & \text{N/A} & 3.0 & 2.2 & \text{N/A} \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} 0.9 & -1.0 & 3.1 & 4.0 & -1.0 \\ 5.1 & -1.0 & 3.0 & 2.2 & -1.0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \\
\text{Insert } -1 \quad &\Longrightarrow \quad \begin{bmatrix} 0.9 & -1.0 & 3.1 & 4.0 & -1.0 \\ 5.1 & -1.0 & 3.0 & 2.2 & -1.0 \end{bmatrix} \\
\text{Time labels} \quad &\Longrightarrow \quad \begin{bmatrix} 0.9 & \text{N/A} & 3.1 & 4.0 & \text{N/A} \\ 5.1 & \text{N/A} & 3.0 & 2.2 & \text{N/A} \\ 0.0 & 0.5 & 1.0 & 1.5 & 2.0 \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} 0.9 & 3.1 & 4.0 \\ 5.1 & 3.0 & 2.2 \\ 0.0 & 1.0 & 1.5 \end{bmatrix}
\end{aligned}$$

4.3 Learning Algorithm for Missing Data Handling

Let $N_{\varnothing}^{\max} < N$ be the maximum admissible number of missing data points. In what follows, we propose a learning algorithm that combines the BayesFlow method exemplarily with the technique "Augment by 0/1" to perform amortized Bayesian inference on datasets with up to $N_{\varnothing}^{\max}$ missing data points.

Algorithm 2 Amortized Bayesian inference for incomplete data using the BayesFlow method and "Augment by 0/1" technique

Training phase (*online learning using datasets with artificially induced missing values*):
repeat
  for $m = 1, \ldots, M$ **do**
    Sample model parameters from the prior: $\boldsymbol{\theta}^{(m)} \sim p(\boldsymbol{\theta})$.
    Sample a stochastic instance: $\boldsymbol{\xi}^{(m)} \sim p(\boldsymbol{\xi})$.
    Run the simulation (1) to generate a synthetic complete dataset: $\boldsymbol{x}_{1:N}^{(m)} = g(\boldsymbol{\theta}^{(m)}, \boldsymbol{\xi}^{(m)})$.
    Sample the number of missing data points: $N_{\varnothing}^{(m)} \sim \mathcal{U}(0, N_{\varnothing}^{\max})$.
    Sample a set $\mathcal{S}^{(m)}$ of $N_{\varnothing}^{(m)}$ missing indices uniformly from $\{1, \ldots, N\}$ without replacement.
    Use the index set $\mathcal{S}^{(m)}$ to perform the augmentation by 0/1 and to replace all missing data points by the fill-in constant C: $\boldsymbol{x}_{1:N}^{(m)} \mapsto \boldsymbol{x}_{\text{aug}}^{(m)}$.
    Pass the augmented dataset $\boldsymbol{x}_{\text{aug}}^{(m)}$ through the summary network: $\widetilde{\boldsymbol{x}}^{(m)} = h_{\boldsymbol{\psi}}(\boldsymbol{x}_{\text{aug}}^{(m)})$.
    Pass $(\boldsymbol{\theta}^{(m)}, \widetilde{\boldsymbol{x}}^{(m)})$ through the inference network in forward direction: $\boldsymbol{z}^{(m)} = f_{\boldsymbol{\phi}}(\boldsymbol{\theta}^{(m)}; \widetilde{\boldsymbol{x}}^{(m)})$.
  end for
  Compute the loss $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi})$ according to (13) from the training batch $\left\{(\boldsymbol{\theta}^{(m)}, \widetilde{\boldsymbol{x}}^{(m)}, \boldsymbol{z}^{(m)})\right\}_{m=1}^{M}$.
  Update neural network parameters $\boldsymbol{\phi}, \boldsymbol{\psi}$ via backpropagation.
until convergence to $\widehat{\boldsymbol{\phi}}, \widehat{\boldsymbol{\psi}}$

Inference phase (*given an incomplete observed or test dataset* $\boldsymbol{x}_{\mathcal{T}}^{\text{o}}$):
Gather the set $\mathcal{S}$ of missing indices from $\boldsymbol{x}_{\mathcal{T}}^{\text{o}}$.
Use $\mathcal{S}$ to perform the augmentation by 0/1 and to insert the constant C for all missing data points: $\boldsymbol{x}_{\mathcal{T}}^{\text{o}} \mapsto \boldsymbol{x}_{\text{aug}}^{\text{o}}$.
Pass the augmented dataset through the summary network: $\widetilde{\boldsymbol{x}}^{\text{o}} = h_{\widehat{\boldsymbol{\psi}}}(\boldsymbol{x}_{\text{aug}}^{\text{o}})$.
for $l = 1, \ldots, L$ **do**
  Sample a latent variable instance: $\boldsymbol{z}^{(l)} \sim \mathcal{N}_{n_{\boldsymbol{\theta}}}(\boldsymbol{z} \,|\, \boldsymbol{0}, \boldsymbol{I}_{n_{\boldsymbol{\theta}}})$.
  Pass $(\boldsymbol{z}^{(l)}, \widetilde{\boldsymbol{x}}^{\text{o}})$ through the inference network in inverse direction: $\boldsymbol{\theta}^{(l)} = f_{\widehat{\boldsymbol{\phi}}}^{-1}(\boldsymbol{z}^{(l)}; \widetilde{\boldsymbol{x}}^{\text{o}})$.
end for
Return $\left\{\boldsymbol{\theta}^{(l)}\right\}_{l=1}^{L}$ as a sample from $p(\boldsymbol{\theta} \,|\, \boldsymbol{x}_{\mathcal{T}}^{\text{o}})$.

The neural network is trained on artificial datasets with a varying number of data points missing completely at random. However, if a certain systematics behind the missingness is known for a real-world problem, the learning algorithm might profit from modifying steps 7 and 8 to reflect this systematic behavior.

Algorithm 2 can be trivially adapted to the technique "Insert C". To integrate the technique "Time labels", one should notice that unlike the two other approaches, the missing data points are simply removed here and not replaced by some constant. This leads to time-labeled data of different length, depending on the number of available data points. But as discussed in Section 2.4, we need the datasets in a training batch to have the same length to enable vectorization. Thus, we have to sample the number $N_\varnothing$ of missing data points before entering the for-loop in step 3 and keep it fixed for the entire batch.

5 Evaluation of Missing Data Handling Approaches

In this section, we use models of different complexity to gradually determine which of the three proposed approaches to encode missing values performs the most robustly.

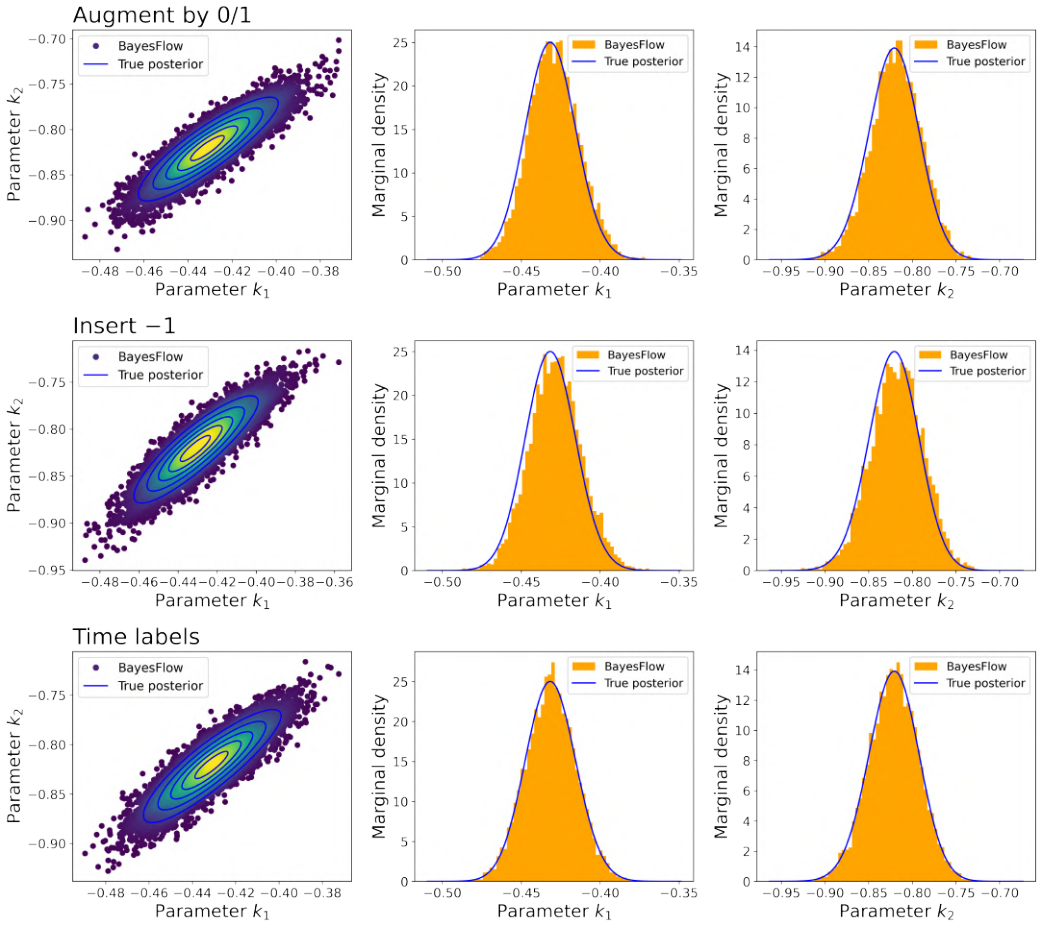
5.1 Conversion Reaction Model

5.1.1 A First Comparison of the Techniques to Encode Missing Values

We consider the conversion reaction model from Section 3.3 and allow at most $N_{\varnothing}^{\max} = 6$ of the overall $N = 11$ one-dimensional observations to be absent.

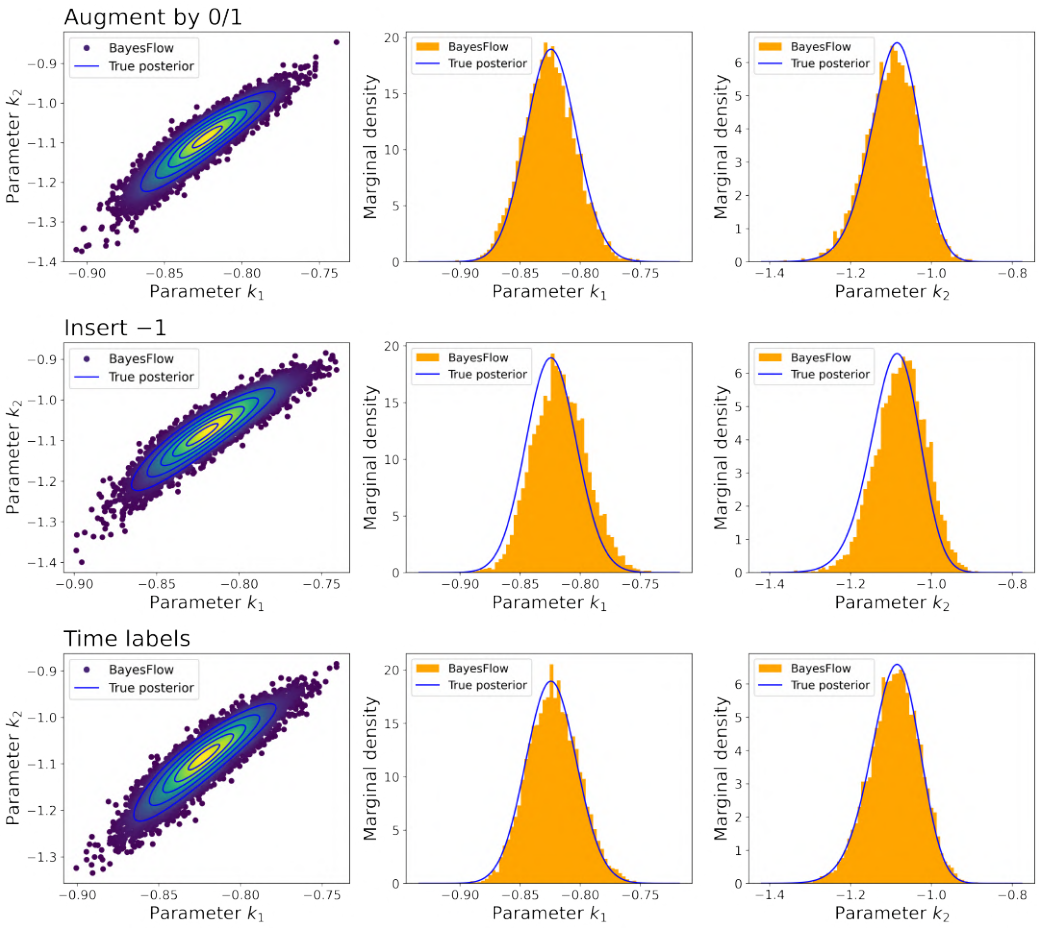
For each of the three approaches "Augment by 0/1 with fill-in constant -1", "Insert -1" and "Time labels", we train a cINN with 5 affine coupling layers of $[64, 64, 64]$ units together with a LSTM(32) network. Here, we deem $C = -1$ suitable as a fill-in value because it is far away from what can be simulated by the model (positive trajectories plus some small measurement noise). The number of hidden units in the LSTM network is determined according to our rule of thumb as the next greater power of two after $22 = 11 \cdot 2$, as the approach "Augment by 0/1" outputs arrays of the shape $(11, 2)$.

Based on test datasets with differently many missing values, we find that all three approaches lead to similar and reasonably good approximations of the desired posterior distribution given the available data (Figure 11). In particular, the visual fit remains relatively good when more than half of the observations are missing (Figure 11c).

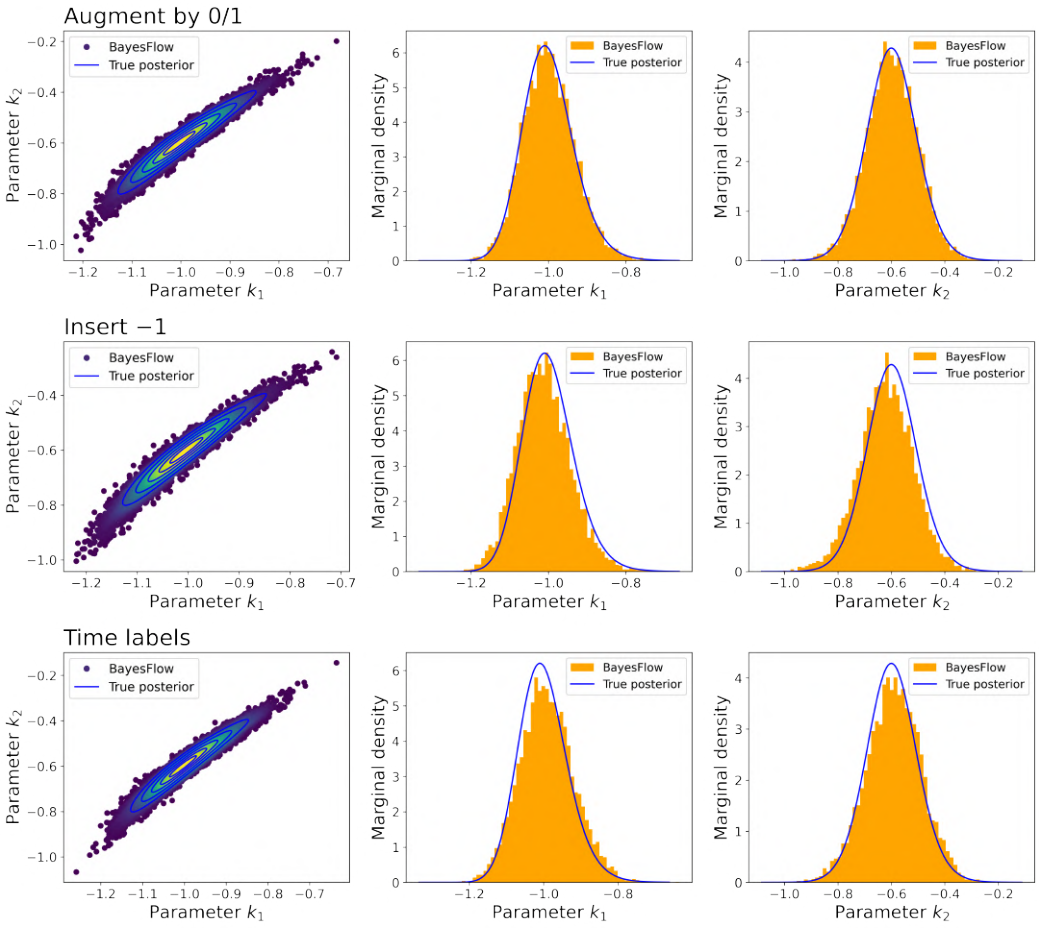


(a) $N_{\varnothing} = 0$ missing observations out of $N = 11$

Figure 11: Conversion reaction model: Comparison of visual fit for the three tested approaches to encode missing values, based on datasets with different ground truth parameters and a different number $N_{\varnothing}$ of missing observations



(b) $N_\varnothing = 3$ missing observations out of $N = 11$



(c) $N_\varnothing = 6$ missing observations out of $N = 11$

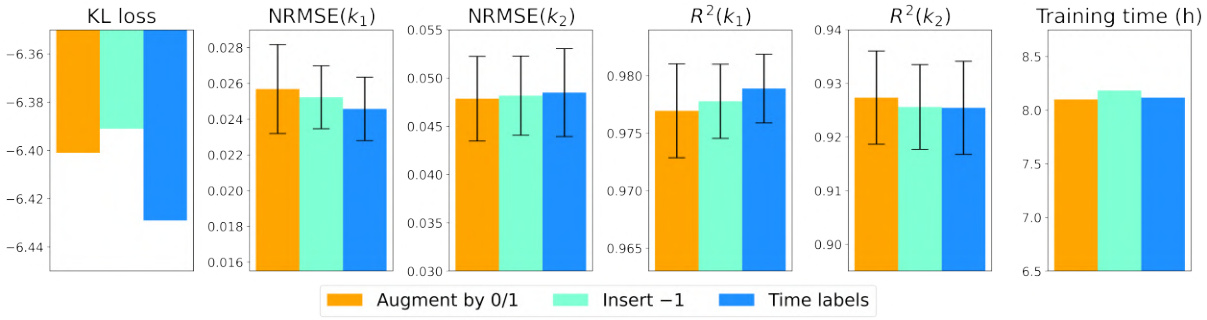


Figure 12: Comparison of KL loss, validation metrics and training time for the three tested approaches to encode missing values. The KL loss is reported without reconstructing the model-specific correction constant. For NRMSE and R^2, bootstrapped means (± 1 standard error) are calculated based on 500 test datasets with a varying number of missing values.

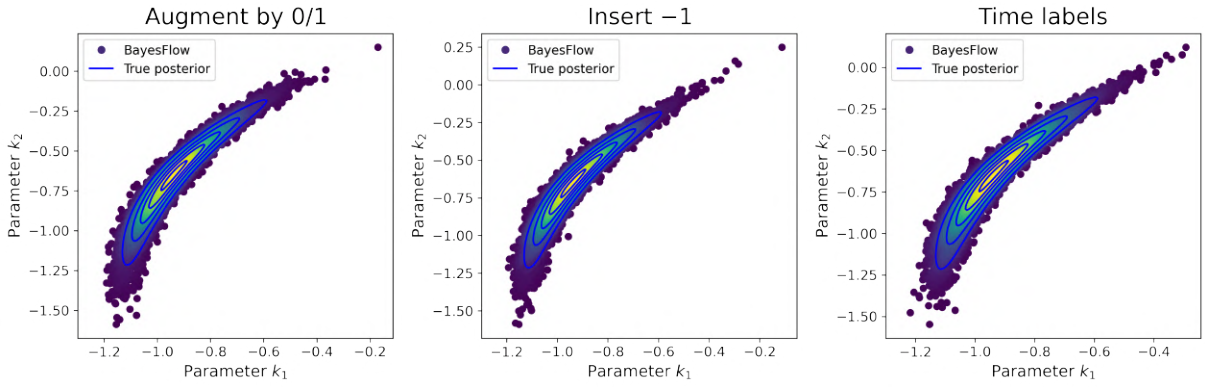
The slightly better or worse behavior on the above example datasets cannot be confirmed in a systematic way, neither by the SBC plots which exhibit uniformity for all three approaches nor by the validation metrics that suggest similarly good performance across different ground truth parameters and degrees of missingness (Figure 12). Besides, the training time is nearly identical because the cost of artificially inducing and encoding missing values is comparable for all three approaches.

5.1.2 Similarly Good Performance on Uninformative Data

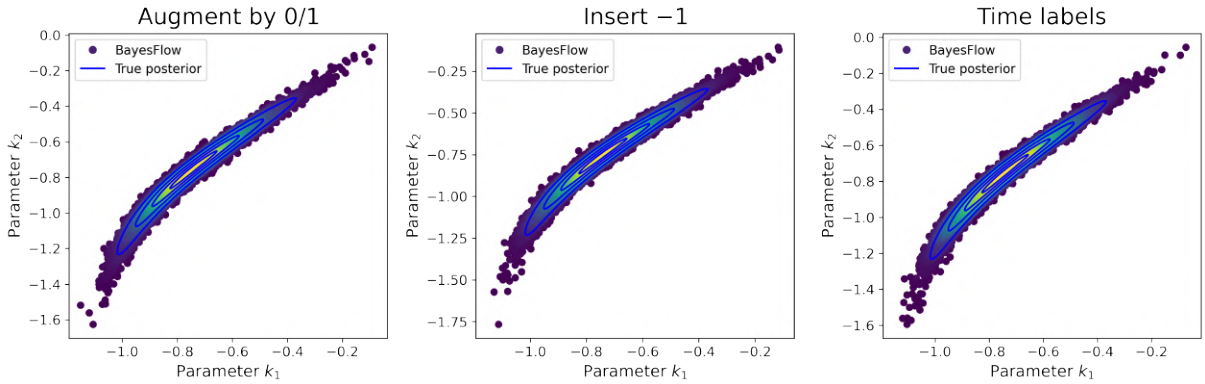
In this subsection, we investigate how the proposed approaches to encode missing data perform when already the complete dataset contains very few observations. To this end, the conversion reaction model is reduced as follows: We assume complete datasets to have $N = 3$ observations at the time points $t_1 = 0$, $t_2 = 5$ and $t_3 = 10$. Up to $N_\varnothing^{\max} = 2$ observations may be absent.

We train a 5-layer cINN with a LSTM(8) network for each of the approaches "Augment by 0/1 with fill-in constant -1", "Insert -1" and "Time labels". Based on test datasets with differently many missing values, we verify that all three approaches lead to nearly perfect approximations, even if the true posterior can be complexly shaped and clearly non-Gaussian (Figures 13a and 13b).

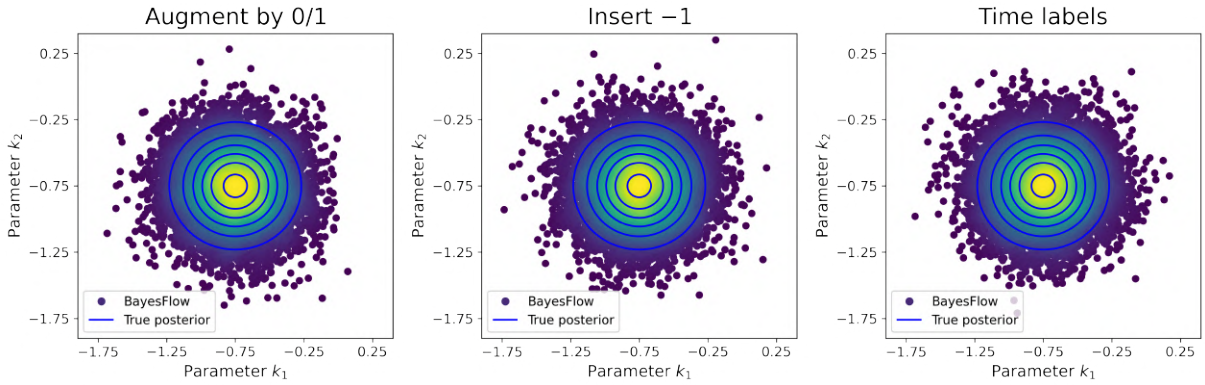
In addition, in the special case that the observations at the time points t_2 and t_3 are missing, we are dealing with a completely uninformative dataset since the only available observation at $t_1 = 0$ is the fixed initial value up to some measurement noise and therefore provides no extra information about the rate parameters. Indeed, all three approaches return the normal prior as the correct posterior (Figure 13c).



(a) $N_\varnothing = 1$ missing observation at t_3



(b) $N_\varnothing = 1$ missing observation at t_2



(c) $N_\varnothing = 2$ missing observations at t_2 and t_3

Figure 13: Reduced conversion reaction model: Comparison of visual fit for the three tested approaches to encode missing values, based on datasets with different ground truth parameters and a different number $N_\varnothing$ of missing observations

5.1.3 Increased Robustness through the Augmentation with 0/1

This subsection serves as a further comparison of the techniques "Augment by 0/1 with fill-in constant C" and "Insert C". Concretely, we explore how the fill-in value C is interpreted by the network in both cases and whether the augmentation with the binary row to indicate the absence or presence of data points brings any benefit.

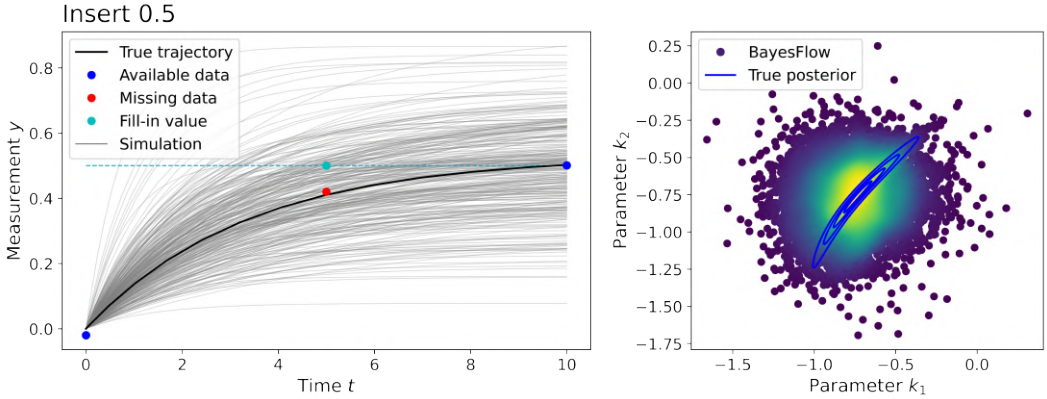
Recall that for the conversion reaction model, it was easy to choose an unambiguous fill-in value $C = -1$ due to the positivity of the simulated trajectories. In this case, we have seen that both approaches, with or without augmentation, estimate the posterior distribution conditioned on the available data correctly. This means they have acquired the ability to interpret the value -1 in the data as missing values and to ignore them for the inference.

But what if no easy choice of the fill-in constant is possible? This may occur for example when the data generating process is modeled by a Gaussian random walk or a sub-/supermartingale and thus produces trajectories that are unbounded or even escape to $\pm$ infinity. Then, there is simply no value C that cannot be simulated.

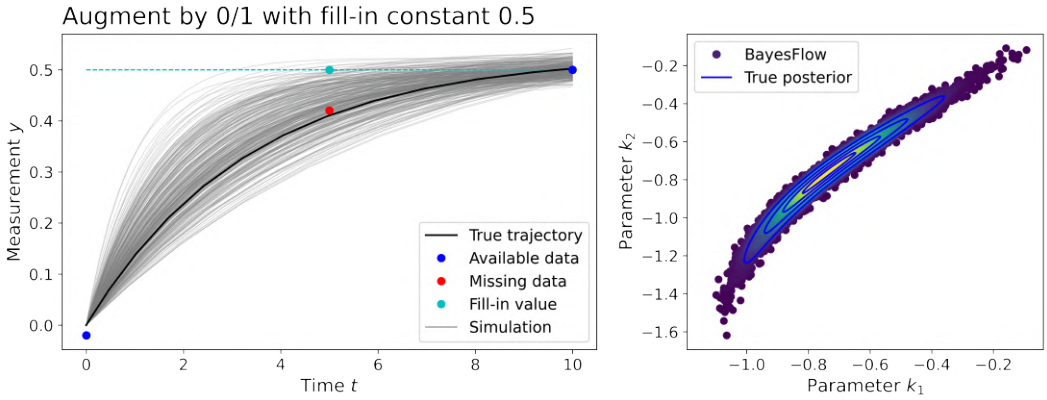
In order to test the robustness of our proposed approaches in such a scenario, let us return to the reduced conversion reaction model from Section 5.1.2 and try out the ambiguous fill-in value $C = 0.5$. This is a plausible data value for a variety of ground truth parameters, e.g. for $(k_1, k_2) = (-0.7, -0.9)$, where the simulated trajectory takes the value $x_2(t_2 = 5) \approx 0.493$, or for $(k_1, k_2) = (-0.8, -0.85)$, where the simulated trajectory takes the value $x_2(t_3 = 10) \approx 0.502$.

For each of the two approaches "Augment by 0/1 with fill-in constant 0.5" and "Insert 0.5", we train a 5-layer cINN jointly with a LSTM(8) network. Our key observations are as follows: The approach "Insert 0.5" interprets values in a small neighborhood of 0.5 always as missing data, which is of course incorrect. On the contrary, the approach "Augment by 0/1 with fill-in constant 0.5" is able to decide correctly whether a value of 0.5 represents a signal or missing value in the data, depending on whether it was augmented by a one or zero. We illustrate this in detail for example datasets belonging to ground truth parameters $(k_1, k_2) = (-0.8, -0.85)$ (Figure 14).

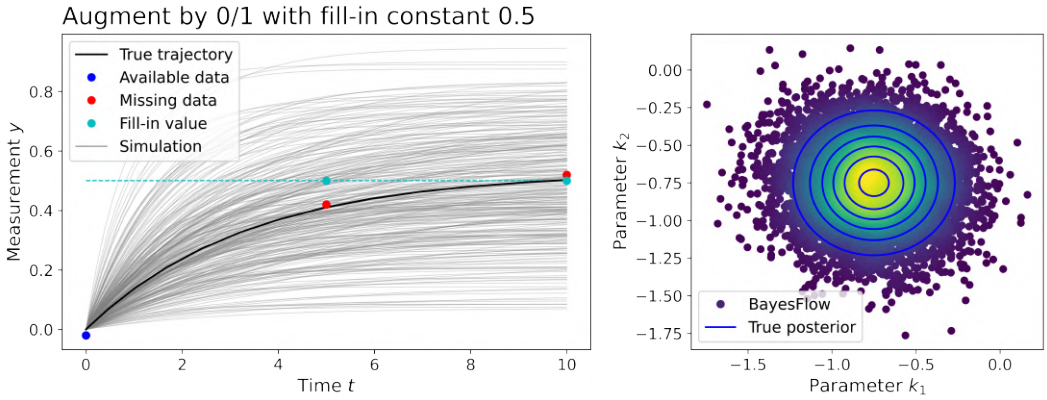
This valuable insight into the interpretation of the fill-in value C by the respective approaches allows us to conclude that the neural network can definitely profit from the augmentation with the binary row. This additional piece of information on the absence or presence of data points is especially crucial to ensure the correct detection of missing data when the model of interest permits no obvious choice of an unambiguous fill-in value. Since the training time is about the same for "Insert C" and "Augment by 0/1 with fill-in constant C", we recommend to always use the latter and more robust approach.



(a) The approach "Insert 0.5" sees the data $\begin{bmatrix} -0.02 & 0.5 & 0.501 \end{bmatrix}$ in which only the second observation is missing. However, the network misinterprets the signal 0.501 as another missing value. Hence, the estimated posterior is wrong, and the third available data point is not fitted by the re-simulated trajectories.



(b) The approach "Augment by 0/1 with fill-in constant 0.5" sees the data $\begin{bmatrix} -0.02 & 0.5 & 0.5 \\ 1 & 0 & 1 \end{bmatrix}$. Based on the augmentation, the network correctly identifies the value 0.5 in the second observation as a missing value and in the third observation as a signal. Consequently, the estimated posterior is correct, and the re-simulated trajectories fit the third data point, but not the second one.



(c) The approach "Augment by 0/1 with fill-in constant 0.5" sees the data $\begin{bmatrix} -0.02 & 0.5 & 0.5 \\ 1 & 0 & 0 \end{bmatrix}$. Based on the augmentation, the network correctly interprets the value 0.5 in the second and third observation as missing value, although 0.5 would be a plausible data value for the third observation. The normal prior is returned as the correct posterior, and the re-simulated trajectories fit neither the second nor the third data point.

Figure 14: Reduced conversion reaction model: Increased robustness through the augmentation with 0/1 when employing an ambiguous fill-in value $C = 0.5$

5.1.4 Variable Dataset Size as a Special Type of Missing Data

In this short paragraph, we come back to the comment from Section 4.1 that datasets of varying length $N' \leq N$ can be easily embedded into a model with fixed dataset size N and missing data. Let us consider the original conversion reaction model with $N = 11$ observations. Assume we want to infer the posterior distribution given a shorter dataset $y_{0:5}$ containing only $N' = 6$ observations at the time points $t = 0, 1, \ldots, 5$. Seeing this as a dataset in which the last 5 of the overall 11 observations are missing, we can use either workflow from Section 5.1.1, e.g. the one based on the technique "Augment by 0/1", to estimate the desired posterior. We show for one example dataset that this idea works in practice (Figure 15).

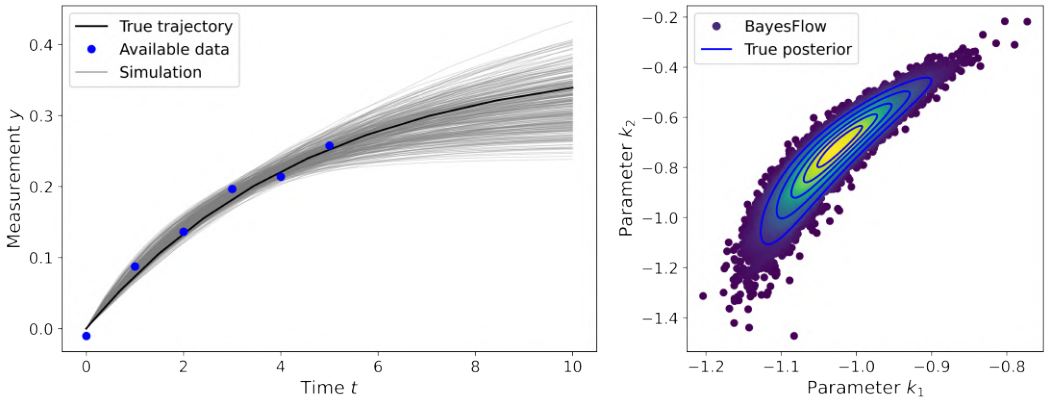


Figure 15: The shorter dataset is treated as a dataset with missing values in the last observations. The posterior is approximated correctly, and the re-simulated trajectories fit the available data points at the beginning and then start to spread.

5.2 Oscillation Model

Biochemical systems exhibiting oscillations are widely studied e.g. in the context of metabolism [Olsen et al., 2003] and cell cycle [Ingolia and Murray, 2004]. From a mathematical computational perspective, dynamic models that produce oscillatory data are in general hard to fit, as the landscape of the cost function to be minimized can be highly irregular and have multiple local minima [Pitt and Banga, 2019].

In this section, we consider the sine curve

$$x(t) = \sin(2\pi a t) + b$$

with frequency parameter a and shift parameter b. We assume that at time t, the value of the sine curve can be observed up to some additive normal noise:

$$y_t = x(t) + \varepsilon_t \text{ with } \varepsilon_t \sim \mathcal{N}(0,\, 0.05^2) \text{ independently in } t$$

A complete dataset contains $N = 41$ observations at $t_k = \frac{1}{4}k$ for $k = 0, 1, \ldots, 40$. In the case of missing data, we allow up to $N_{\varnothing}^{\max} = 21$ observations to be absent.

The following prior distributions are used for the parameters of interest:

$$a \sim \mathcal{U}(0.1,\, 1), \qquad b \sim \mathcal{N}(0,\, 0.25^2)$$

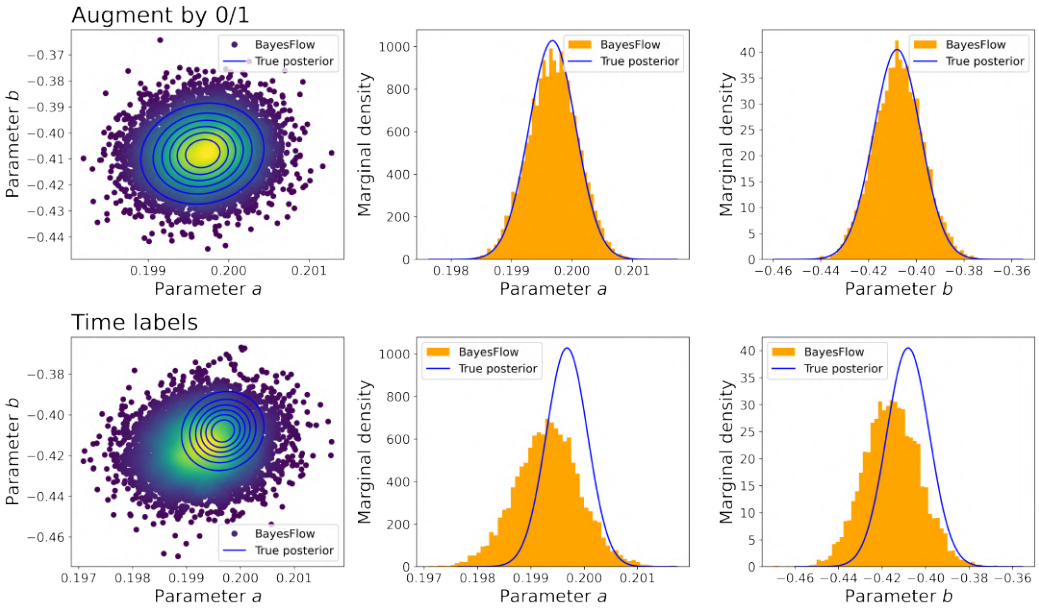
Note that we choose an uniform prior for a not only out of convenience. It is also justified as we expect very strong posterior contraction in the frequency parameter (small changes in the frequency lead to severe changes of the trajectory over the entire observation period) and this diminishes the probability of encountering true posteriors with a high density at the boundary of the support [0.1, 1] (cf. Section 3.3.3). Moreover, we tested extensively that because of the relatively high number of observations, the true posterior is practically always unimodal when the missing indices are sampled completely at random as proposed in Algorithm 2.

5.2.1 Poor Convergence of the "Time labels" Approach

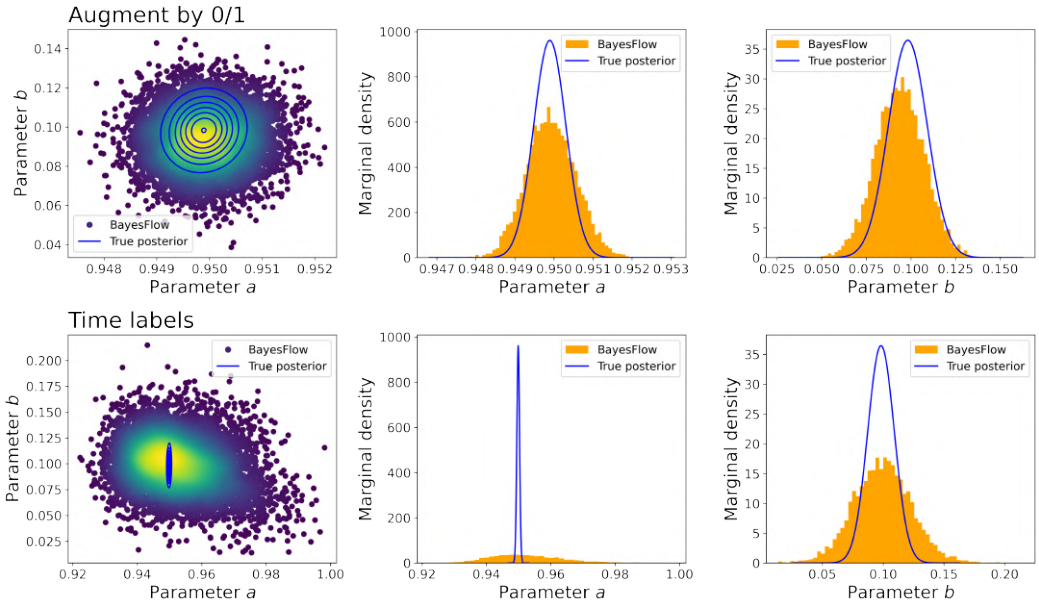
In the following, we use the oscillation model to compare the performance of the two remaining missing data handling approaches, namely "Augment by 0/1" and "Time labels". For each of them, a cINN with 5 affine coupling layers of $[64, 64, 64]$ units is trained with a LSTM(128) summary network. The fill-in value -5 is used for the first approach.

Across test datasets generated by different ground truth parameters and exhibiting differently many missing values, we consistently observe the approach "Augment by 0/1" to approximate the true (unimodal) posterior significantly better than the approach "Time labels" (Figure 16). Especially for higher frequency parameters, which we deem more challenging because only few observations per period are available, the posterior contraction suffers severely when using the approach "Time labels", while it remains reasonably good when using the approach "Augment by 0/1" (Figure 16b). These visual preferences are clearly captured by the final loss as well as the validation metrics (Figure 17).

What draws one's attention is that the final loss achieved by "Augment by 0/1" is much lower than the one achieved by "Time labels" - it seems that the networks did not converge well in the latter case. A closer look into the running loss of 21 consecutive iterations from the final epoch gives us a hint of why this happened (Figure 18).



(a) Ground truth $(a, b) = (0.2, -0.4)$; $N_{\varnothing} = 15$ missing observations out of $N = 41$



(b) Ground truth $(a, b) = (0.95, 0.1)$; $N_{\varnothing} = 20$ missing observations out of $N = 41$

Figure 16: Oscillation model: Comparison of visual fit for the tested missing data handling approaches, based on datasets with different ground truth parameters and a high degree of missingness

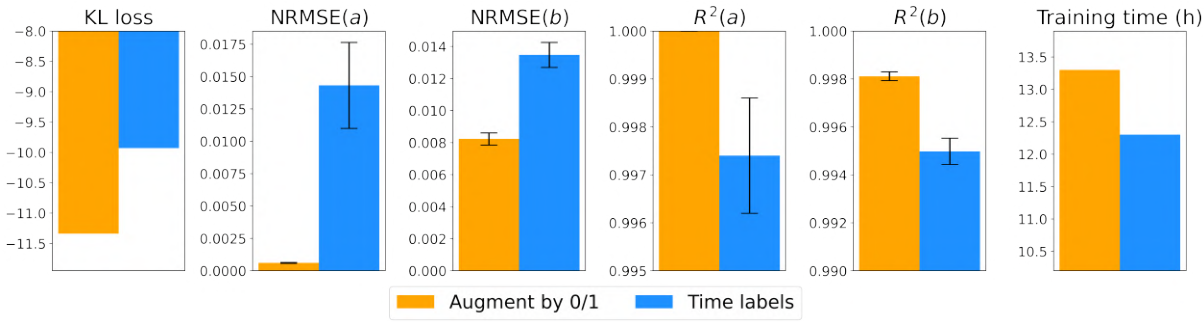


Figure 17: Comparison of KL loss, validation metrics and training time for the tested missing data handling approaches. The KL loss is reported without reconstructing the model-specific correction constant. For NRMSE and R^2, bootstrapped means (± 1 standard error) are calculated based on 500 test datasets with a varying number of missing values.

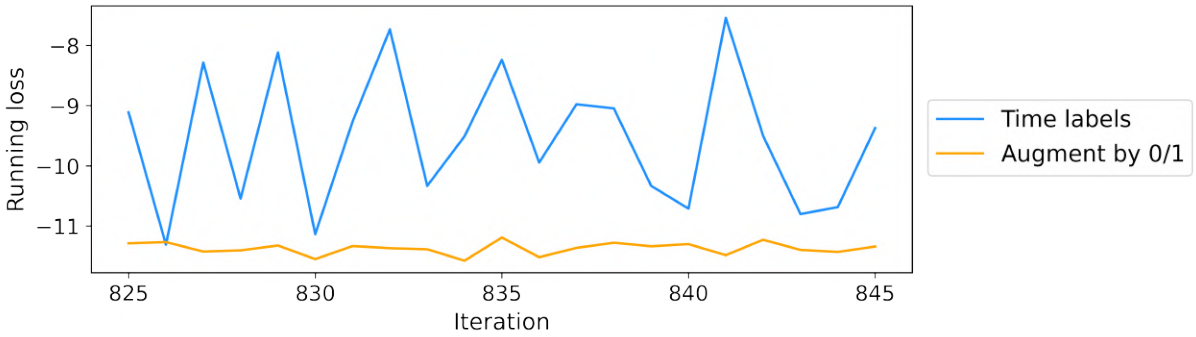


Figure 18: Comparing the behavior of the running loss for the tested missing data handling approaches

Compared to "Augment by 0/1", the running loss of the approach "Time labels" exhibits much stronger jumps from one iteration to another, to an extent that cannot be explained merely by the stochasticity of the gradient descent. Instead, we recall that unlike "Augment by 0/1", the approach "Time labels" is implemented by sampling the number $N_\varnothing$ of missing data points for each entire training batch (cf. Section 4.3). Thus, Monte Carlo estimates of the loss will jump considerably depending on the value $N_\varnothing$ sampled for the respective batch. But since the exact loss is only approximated as an average across many iterations and the network parameters are updated after each iteration, we run into convergence issues for the more challenging loss function of the oscillation model.

We conclude that "Augment by 0/1" performs the most robustly amongst the three proposed approaches. However, it is important to note that the poor performance of "Time labels" here does not imply that this technique to encode missing values itself is bad. It is rather its nature of producing variable-size data (missing values are removed and not replaced) and the current BayesFlow implementation requiring the datasets within a training batch to have the same size (vectorization) that force us to use a possibly inadequate approximation of the loss.

5.3 SIR Epidemiology Model

Having determined "Augment by 0/1" as the most robust missing data handling approach, we apply it to a SIR model, which describes the spread of an infectious disease [Kuhl, 2021]. SIR models are not only interesting in view of the current pandemic situation, but also because their dynamics are much less trivial than the ones of the conversion reaction model.

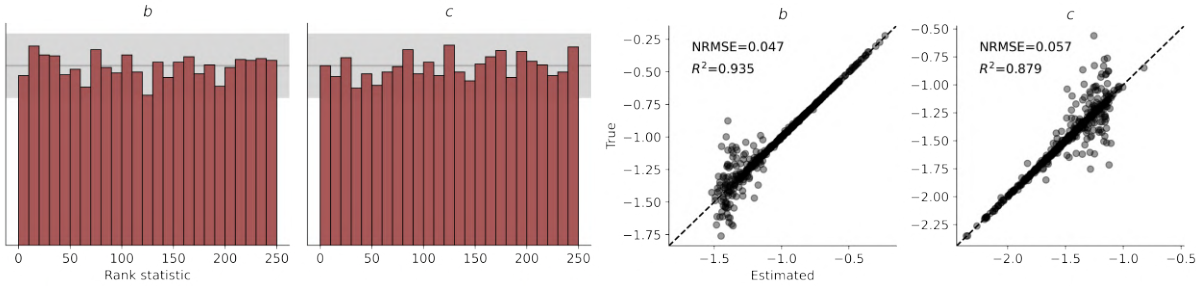
Let S denote the number of susceptible, I the number of infectious and R the number of recovered individuals within a population of constant size $P = S + I + R$. Assuming the elementary reactions

$$S + I \xrightarrow{\beta} 2I, \qquad I \xrightarrow{\gamma} R$$

with infection rate $\beta > 0$ and recovery rate $\gamma > 0$, the infection dynamics are quantified by the following non-linear system of ODEs:

$$\begin{aligned} \frac{\mathrm{d}S}{\mathrm{d}t} &= -\frac{\beta SI}{P}, \\ \frac{\mathrm{d}I}{\mathrm{d}t} &= \frac{\beta SI}{P} - \gamma I, \\ \frac{\mathrm{d}R}{\mathrm{d}t} &= \gamma I \end{aligned}$$

We specify the population size $P = 1000$ as well as the initial state $(S_0, I_0, R_0) = (999, 1, 0)$, and solve this initial value problem using a standard ODE solver in Python with internal error control. The data at time t are obtained by adding normally distributed measurement noise to



(a) SBC (b) Validation metrics

Figure 19: Performance validation for the SIR model

the normalized state vector:

$$y_t = \frac{1}{P}\begin{pmatrix} S(t) \\ I(t) \\ R(t) \end{pmatrix} + \begin{pmatrix} \varepsilon_{t,1} \\ \varepsilon_{t,2} \\ \varepsilon_{t,3} \end{pmatrix} \text{ with } \varepsilon_{t,i} \sim \mathcal{N}(0,\, 0.015^2) \text{ independently in } t \text{ and } i$$

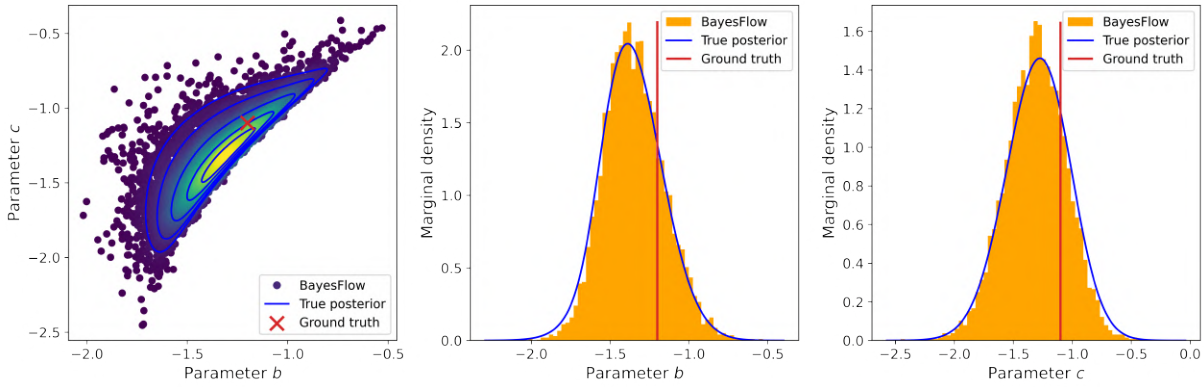
A complete dataset contains $N = 31$ observations at $t_k = 6k$ for $k = 0, 1, \ldots, 30$ from which up to $N_{\varnothing}^{\max} = 15$ may be missing. The inference is done for the log-scale parameters $b = \log_{10}(\beta)$ and $c = \log_{10}(\gamma)$ that are believed to follow normal prior distributions:

$$b \sim \mathcal{N}(-1,\, 0.3^2),\; c \sim \mathcal{N}(-1.5,\, 0.3^2) \text{ independently}$$

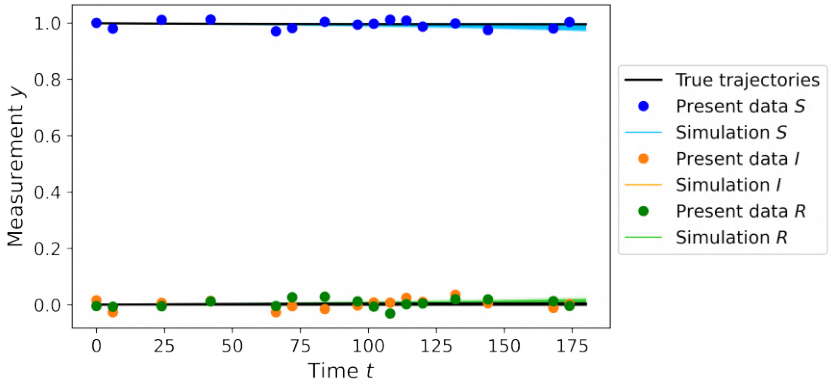
We train a cINN with 5 ACLs of $[128, 128, 128]$ units with a LSTM(128) network and use the technique "Augment by 0/1 with fill-in constant -1" to encode missing data. As the simulation using an ODE solver is computationally rather expensive, we employ a lower batch size of 64.

The SBC plot exhibits uniformity, i.e. no systematic bias or over-/underfitting is detected for the approximate posteriors (Figure 19a). As to the plot displaying the metrics NRMSE and R^2, we observe clear deviations from the "desired" diagonal line, meaning that some ground truth parameters are not recovered well by the means of the approximate posteriors (Figure 19b). However, this does not necessarily imply that the approximate posteriors are incorrect: Notice that deviations occur especially for smaller values of b and larger values of c. And exactly for combinations (b, c) with $b \leq c$, i.e. when the infection rate is lower than the recovery rate, the simulated dynamics are flat and the observed data therefore carry only little information about the ground truth parameters (besides the fact that $b \leq c$ must hold approximately). In these cases, even the true posterior does not need to be centered at the ground truth parameters, so we should not expect the approximate posterior to recover them either. In Figure 20, we look at such a rather challenging scenario and demonstrate that our method still manages to learn the true posterior reasonably well.

We finish with a dataset describing typical infection dynamics, where the number of infectious individuals builds up towards a peak at $t \approx 70$ and then flattens out towards the end of the observation period $t_{30} = 180$ (Figure 21b). Although assuming half of the data points to be missing, we find that the BayesFlow samples fit the true posterior conditioned on the available data accurately (Figure 21a). The same is true for the data fitting through the re-simulated trajectories (Figure 21b). We therefore conclude that our approach "Augment by 0/1" performs well on incomplete datasets from the SIR model.

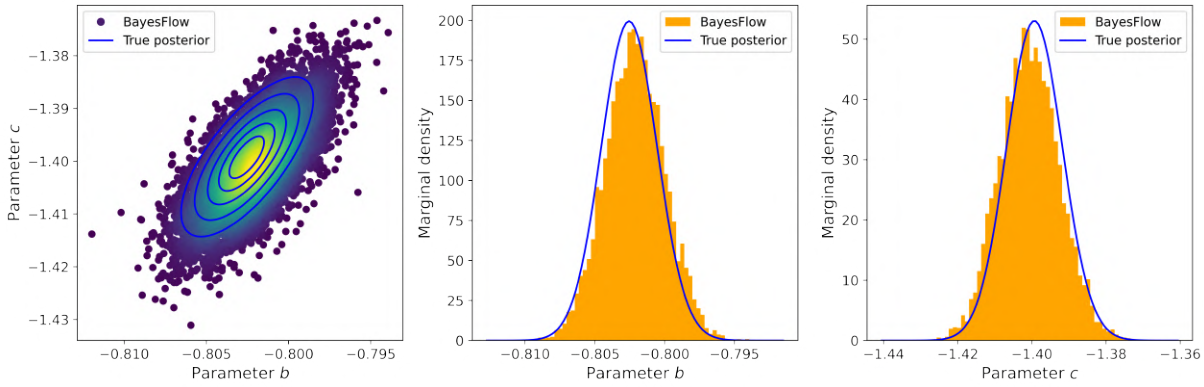


(a) Visual fit (note that the true posterior conveys the information that the parameter b cannot exceed parameter c by a lot, but its center differs from the ground truth parameters on a coarse scale)

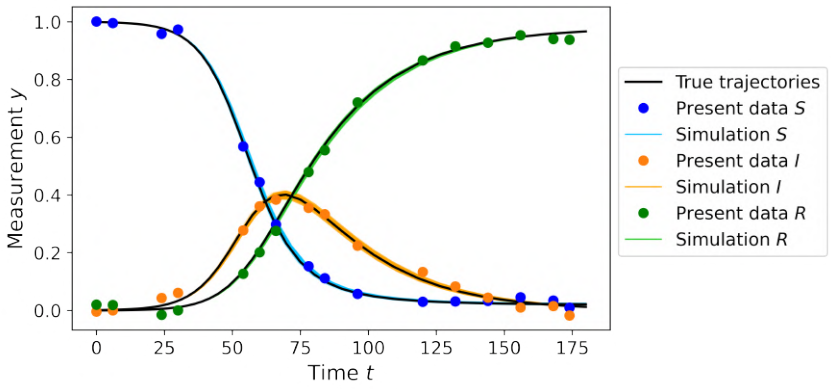


(b) Posterior predictive check (orange trajectories are hidden by the green ones)

Figure 20: Ground truth $(b, c) = (-1.2, -1.1)$ leading to flat dynamics; $N_{\varnothing} = 15$ out of $N = 31$



(a) Visual fit



(b) Posterior predictive check

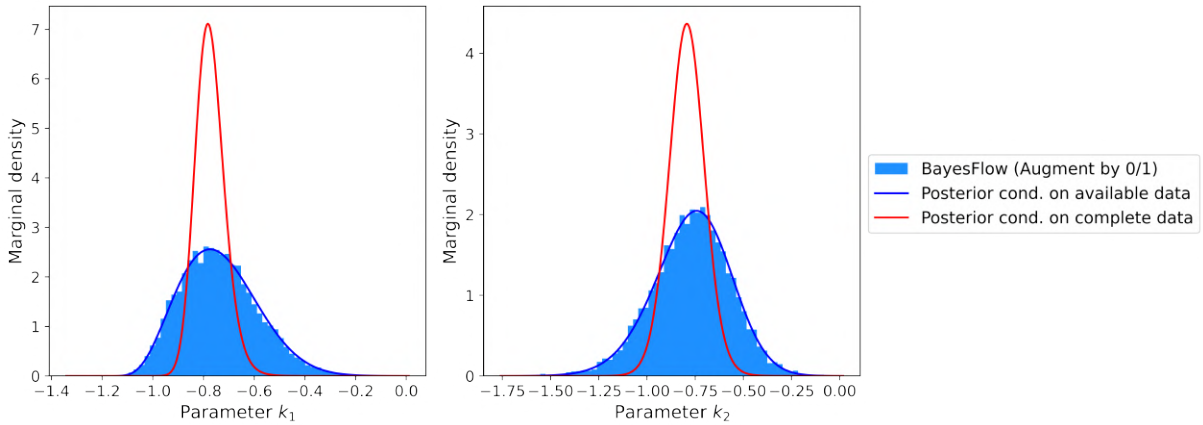
Figure 21: Ground truth $(b, c) = (-0.8, -1.4)$ leading to typical dynamics; $N_{\varnothing} = 15$ out of $N = 31$

5.4 Outlook on Data Imputation Methods

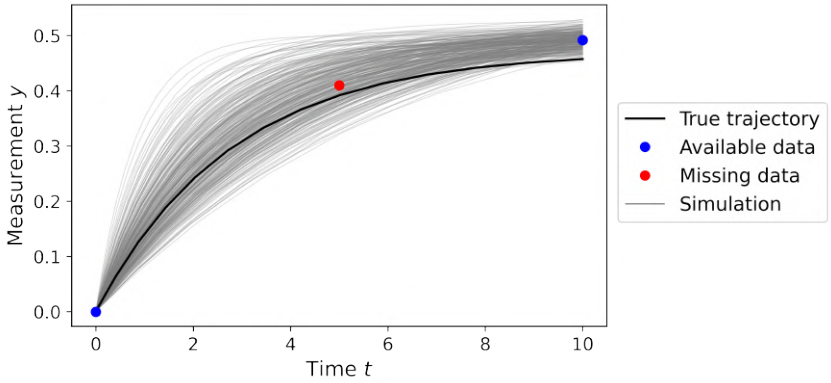
So far, we have focused on estimating the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{\mathcal{T}})$ conditioned on the available data. The key idea is to expose the neural network to datasets with cleverly encoded missing values during training. It is important to notice that none of our proposed approaches has the ability to reconstruct the missing values and to approximate the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$ conditioned on the (only partially known) complete original data (Figure 22).

However, a closer look at Figure 22a readily reveals why the posterior conditioned on the complete data can be of great interest as well: Since more data points and thus more information is taken into account, it usually exhibits stronger posterior contraction than the posterior conditioned only on the available data and thus allows for more reliable point estimates.

The probably simplest way to get a first approximation of $p(\boldsymbol{\theta} \mid \boldsymbol{x}_{1:N})$ is to impute the missing values from the available data using linear interpolation. The imputed dataset is then fed into the BayesFlow workflow trained on complete data. But this naive approach can already lead to severe misapproximations even for relatively simple non-linear dynamics such as the conversion reaction model (Figure 23).

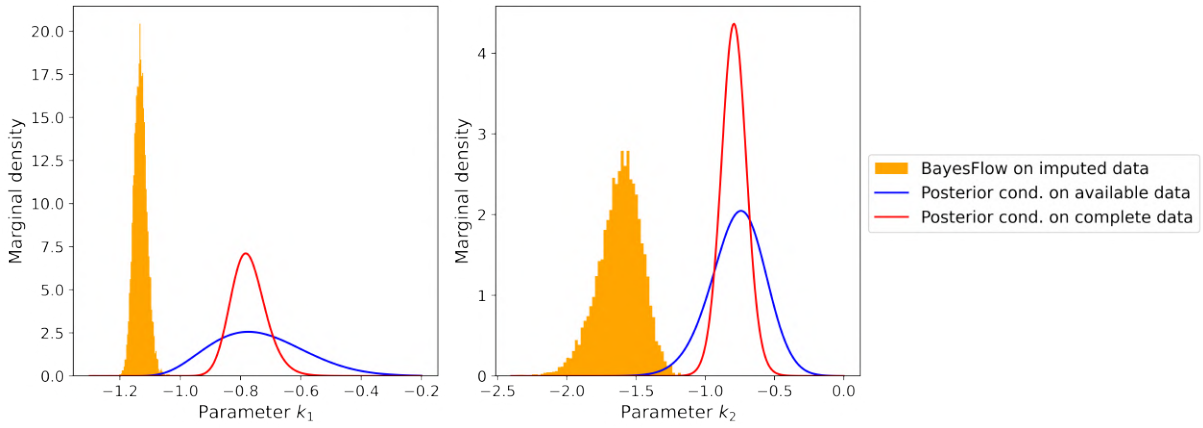


(a) The BayesFlow samples approximate the posterior conditioned on the available data very precisely, but are far away from recovering the posterior conditioned on the complete original data.

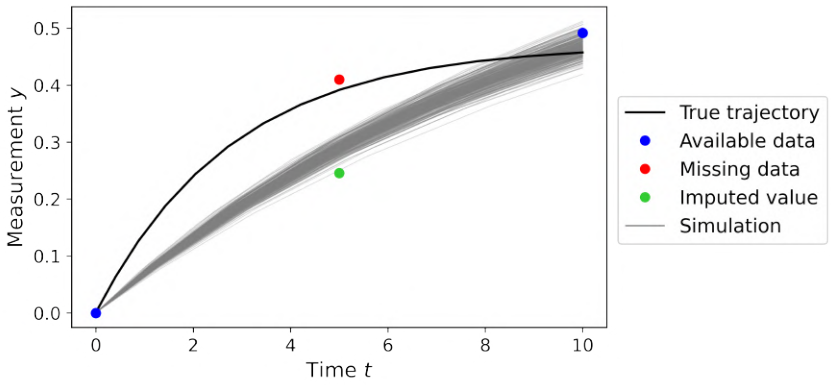


(b) The re-simulated trajectories do not fit the second data point, indicating that our approach has no ability to recover the missing value.

Figure 22: Reduced conversion reaction model with missing data handled with "Augment by 0/1". The results for the other two discussed approaches would be nearly identical and are not displayed here.



(a) The samples inferred from the imputed data by the network trained on complete data provide a very poor approximation of the desired posterior distribution conditioned on the complete data.



(b) The reason is that the linearly interpolated value is far off the true trajectory and true data value. Thus, the re-simulated trajectories are forced to fit the wrongly imputed value instead of the true value of the missing data point.

Figure 23: Reduced conversion reaction model: Imputing missing values via linear interpolation can lead to severe misapproximation of the posterior distribution.

Instead, we could employ more advanced imputation techniques based on (cubic) spline interpolation [Liu et al., 2020] or deep learning approaches using e.g. LSTM networks [Song et al., 2020] or autoencoders [Khadka and Shakya, 2021]. Though being tested on various application examples, these imputation techniques of course do not work flawlessly. Thus, in future work, we would like to address the idea of training the BayesFlow workflow on imputed data with an added binary row to label either imputed or original values. A step beyond this could be to switch from binary labels to continuous weights indicating trustworthyness of imputed values, with confidence e.g. obtainable when using Gaussian process regression [Wang, 2020]. Our hope is that the network learns to extract information from the imputed values with "more caution" and thus performs more robustly, as opposed to the approach where we simply feed imputed data into the BayesFlow workflow trained on complete datasets.

6 Summary and Outlook

6.1 Summary

In this thesis, we evaluated and extended the BayesFlow method that was designed to perform amortized likelihood-free Bayesian inference. In short, a conditional invertible neural network is trained jointly with an optional summary network to learn a normalizing mapping between the parameters of interest and a Gaussian latent variable, conditional on data simulated from the forward model. This enables efficient posterior sampling for any real dataset by sampling the latent variable and transforming back to the parameter space via the inverse mapping.

Based on toy models, we validated the good performance of BayesFlow on complete datasets. In this context, we also explored that LSTM summary networks with approximately as many hidden units as the total number of entries in the data matrix are a practical and reasonable choice for time series models. Further, we saw that uniform priors should be used with caution since the neural network might not always be able to learn the hard boundary of the posterior.

Afterwards, we investigated how BayesFlow can be modified to handle missing data, concretely to infer the posterior distribution conditioned on the available data. To this end, we proposed a learning algorithm in which the BayesFlow workflow is trained on datasets with artificially induced missing values encoded by one of the three techniques "Augment by 0/1", "Insert C" and "Time labels". For the rather simple conversion reaction model, all three approaches lead to similarly good approximations of the desired posterior.

However, for more complicated models that do not permit an easy choice of fill-in value (e.g. no non-simulable value is known or even exists), the approach "Augment by 0/1" is superior to "Insert C". The reason is that the employed fill-in value C might be ambiguous and then, the additional information in the augmented row is essential to correctly distinguish a signal in the neighborhood of C from a value C inserted for a missing data point. Moreover, the currently possible implementation of the "Time labels" approach (fixed number of missing observations for all datasets within a training batch) causes the running loss to jump, which possibly leads to convergence issues when the model is more challenging. In summary, we recommend to use "Augment by 0/1" to handle missing data since among the three proposed approaches, it has provided reasonably good posterior approximation the most robustly across all tested models.

6.2 Outlook

Future work will focus, amongst others, on two goals. Firstly, we want to apply the BayesFlow method combined with the missing data handling technique "Augment by 0/1" to real-world problems from systems biology and epidemiology. A possible starting point could be non-linear mixed-effects models, which are computationally intensive to fit and for which there is great demand for efficient and scalable inference methods [Fröhlich et al., 2018]. In this context, we can certainly make use of parallel computing to speed up the batch simulation during training. Besides, it will be interesting to explore if we can improve the accuracy by choosing a more sophisticated summary network (e.g. multi-layer LSTM network) and integrating potentially known patterns behind the data missingness into the learning algorithm. Further, we should consider models in which data values are missing independently in different observed features. Then, it may become necessary to do the augmentation for each feature row of the data matrix.

Secondly, we seek to incorporate data imputation methods into the BayesFlow workflow and thus to reliably recover the more contracted posterior distribution conditioned on the complete original dataset. The main ideas have been outlined in Section 5.4.

References

L. Ardizzone, J. Kruse, C. Rother, and U. Köthe. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJed6j0cKX.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv: Learning*, 2016. doi: 10.48550/ARXIV.1511.07289. URL https://arxiv.org/abs/1511.07289.

M. Deans. Maximally informative statistics for localization and mapping. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 2, pages 1824–1829 vol.2, 2002. doi: 10.1109/ROBOT.2002.1014806.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

F. Fröhlich, A. Reiser, L. Fink, D. Woschée, T. Ligon, F. J. Theis, J. O. Rädler, and J. Hasenauer. Multi-experiment nonlinear mixed effect modeling of single-cell translation kinetics after transfection. *npj Systems Biology and Applications*, 5(1):1, 2018. doi: 10.1038/s41540-018-0079-7. URL https://doi.org/10.1038/s41540-018-0079-7.

F. Fröhlich, C. Loos, and J. Hasenauer. Scalable inference of ordinary differential equation models of biochemical processes. In G. Sanguinetti and V. A. Huynh-Thu, editors, *Gene Regulatory Networks: Methods and Protocols*, volume 1883 of *Methods in Molecular Biology*, chapter 16, pages 385–422. Humana Press, 1 edition, 2019.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12:2451–2471, 1999.

J. Hasenauer, N. Jagiella, S. Hross, and F. J. Theis. Data-driven modelling of biological multi-scale processes. Technical Report arXiv:1506.06392v1 [q-bio.MN], arXiv, July 2015.

C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows, 2018. URL https://arxiv.org/abs/1804.00779.

N. T. Ingolia and A. W. Murray. The ups and downs of modeling the cell cycle. *Current Biology*, 14(18):R771–R777, 2004. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2004.09.018. URL https://www.sciencedirect.com/science/article/pii/S0960982204006943.

S. K. Khadka and S. Shakya. Imputing block of missing data using deep autoencoder. In J. S. Raj, editor, *International Conference on Mobile Computing and Sustainable Informatics*, pages 697–707, Cham, 2021. Springer International Publishing. ISBN 978-3-030-49795-8. URL https://doi.org/10.1007/978-3-030-49795-8_66.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf.

E. Kuhl. *The classical SIR model*, pages 41–59. Springer International Publishing, Cham, 2021. ISBN 978-3-030-82890-5. doi: 10.1007/978-3-030-82890-5_3. URL https://doi.org/10.1007/978-3-030-82890-5_3.

E. Lesaffre and A. B. Lawson. *Bayesian biostatistics*. Statistics in Practice. John Wiley & Sons, Ltd., Chichester, 2012. ISBN 978-0-470-01823-1. doi: 10.1002/9781119942412. URL https://doi.org/10.1002/9781119942412.

J. Liepe, C. Barnes, E. Cule, K. Erguler, P. Kirk, T. Toni, and M. P. H. Stumpf. ABC-SysBio – approximate Bayesian computation in python with GPU support. *Bioinformatics*, 26(14): 1797–1799, 2010. doi: 10.1093/bioinformatics/btq278.

H. Liu, Y. Wang, and W. Chen. Three-step imputation of missing values in condition monitoring datasets. *IET Generation, Transmission & Distribution*, 14(16):3288–3300, 2020. doi: https://doi.org/10.1049/iet-gtd.2019.1446. URL `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2019.1446`.

L. F. Olsen, U. Kummer, A. L. Kindzelskii, and H. R. Petty. A model of the oscillatory metabolism of activated neutrophils. *Biophysical Journal*, 84(1):69–81, 2003. ISSN 0006-3495. doi: https://doi.org/10.1016/S0006-3495(03)74833-4. URL `https://www.sciencedirect.com/science/article/pii/S0006349503748334`.

J. Oruh, S. Viriri, and A. Adegun. Long short-term memory recurrent neural network for automatic speech recognition. *IEEE Access*, 10:30069–30079, 2022. doi: 10.1109/ACCESS.2022.3159339.

J. Owen, D. Wilkinson, and C. Gillespie. Likelihood free inference for Markov processes: A comparison. *Statistical applications in genetics and molecular biology*, 14, 10 2014. doi: 10.1515/sagmb-2014-0072.

U. Picchini. Inference for SDE models via approximate Bayesian computation. *Journal of Computational and Graphical Statistics*, 23(4):1080–1100, 2014. ISSN 10618600. URL `http://www.jstor.org/stable/43304799`.

J. A. Pitt and J. R. Banga. Parameter estimation in models of biological oscillators: an automated regularised estimation approach. *BMC Bioinformatics*, 20(1):82, 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2630-y. URL `https://doi.org/10.1186/s12859-019-2630-y`.

S. T. Radev, F. Graw, S. Chen, N. T. Mutters, V. M. Eichel, T. Bärnighausen, and U. Köthe. OutbreakFlow: Model-based Bayesian inference of disease outbreak dynamics with invertible neural networks and its application to the COVID-19 pandemics in Germany. *PLoS Comput Biol*, 17(10):e1009472, Oct. 2021.

S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1452–1466, 2022. doi: 10.1109/TNNLS.2020.3042395.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/rezende15.html`.

L. Ruthotto and E. Haber. An introduction to deep generative modeling, 03 2021. URL `https://arxiv.org/abs/2103.05180`.

M. Sangiorgio and F. Dercole. Robustness of LSTM neural networks for multi-step forecasting of chaotic time series. *Chaos Solitons Fractals*, 139:110045, 12, 2020. ISSN 0960-0779. doi: 10.1016/j.chaos.2020.110045. URL `https://doi.org/10.1016/j.chaos.2020.110045`.

Y. Schälte and J. Hasenauer. Efficient exact inference for dynamical systems with noisy measurements using sequential approximate Bayesian computation. *Bioinformatics*, 36(Supplement 1):i551–i559, 7 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btaa397.

M. Schmitt, P.-C. Bürkner, U. Köthe, and S. Radev. Detecting model misspecification in amortized Bayesian inference with neural networks, 12 2021. URL `https://arxiv.org/abs/2112.08866`.

W. Song, C. Gao, Y. Zhao, and Y. Zhao. A time series data filling method based on LSTM

— Taking the stem moisture as an example. *Sensors*, 20(18), 2020. ISSN 1424-8220. doi: 10.3390/s20185045. URL https://www.mdpi.com/1424-8220/20/18/5045.

W. D. Stein, W. D. Figg, W. Dahut, A. D. Stein, M. B. Hoshen, D. Price, S. E. Bates, and T. Fojo. Tumor growth rates derived from data for patients in a clinical trial correlate strongly with patient survival: a novel strategy for evaluation of clinical trial data. *Oncologist*, 13(10): 1046–1054, Oct. 2008.

M. Taboga. Kullback-Leibler divergence. In *Lectures on probability theory and mathematical statistics*. Kindle Direct Publishing, 2021. URL https://www.statlect.com/fundamentals-of-probability/Kullback-Leibler-divergence.

S. Talts, M. Betancourt, D. Simpson, A. Vehtari, and A. Gelman. Validating Bayesian inference algorithms with simulation-based calibration, 2018. URL https://arxiv.org/abs/1804.06788.

J. Wang. An intuitive tutorial to Gaussian processes regression, 09 2020. URL https://arxiv.org/abs/2009.10862.

D. R. Wolf and E. I. George. Maximally informative statistics. 12 2001. doi: 10.48550/ARXIV.PHYSICS/0010039. URL https://arxiv.org/abs/physics/0010039.