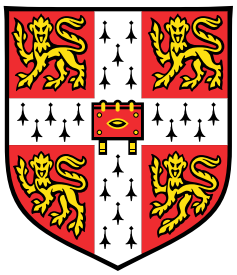
Neural Models for Non-Uniformly Sampled Data



Aliaksandra Shysheya

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Lucy Cavendish College August 2020

I would like to dedicate this thesis to my loving partner and parents, who have made my studies in Cambridge possible.

Declaration

I, Aliaksandra Shysheya of Lucy Cavendish College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

This dissertation contains 14987 words excluding bibliography, photographs and diagrams but including tables, captions, footnotes, appendices and abstract.

We used the following external code for our experiments in Chapter 4:

- torchdiffeq package for the implementation of differentiable ODE solvers.

Aliaksandra Shysheya
August 2020

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Richard E. Turner for providing excellent advice and guidance throughout the project. I have learnt a lot from Richard, and working with him has been always a pleasure. I would also like to thank Wessel Bruinsma, Jonathan Gordon and Will Tebbutt for their help and fruitful discussions of the project.

Finally, I would like to thank my friend Aliaksei Mikhailiuk for his suggestions on the writing of this thesis.

Abstract

Despite the unquestionable success of deep learning models in solving various machine learning problems, they are generally limited to achieving acceptable performance only on highly structured data, such as images or natural language data. One key limitation is that most of these models either cannot handle non-uniformly sampled data or work poorly on it. However, in some areas, like environmental and medical research, the modelled data typically does not reside on a grid, nor is it uniformly sampled. This problem has attracted a lot of attention from the machine learning community lately, and several of successful approaches have been proposed. However, coming from different machine learning domains, such as unsupervised generative modelling and few-shot learning, these approaches lack unified perspective. This perspective is desirable as it could provide a rigorous description of the emerging field of off-the-grid time series modelling.

This work develops a unifying framework for models capable of handling irregularly-spaced time series data. In this work, we consider various models proposed within both unsupervised learning and meta-learning paradigms. Within these paradigms, we are concerned with a wide range of models, from entirely non-amortised to fully amortised ones. We come up with a series of schematics that place this considerable variability of models from disjoint machine learning domains into a common context. These schematics could be beneficial in comprehending the general approaches to data modelling as well as in choosing the appropriate model for a particular task setting. Our qualitative and quantitative evaluation of several models from the proposed framework provides further observational insights as to which model should be used in which setting.

Finally, within the proposed unifying framework, we naturally encountered novel models operating within the meta-learning framework. These models ideally fit in the considered schematics and could be applied to various few-shot learning tasks involving time-series data.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AEVB Auto-Encoding Variational Bayes

AN Autoregressive Network

CDE Controlled Differential Equation

CNN Convolutional Neural Network

CNP Conditional Neural Process

ConvCNP Convolutional Conditional Neural Process

ConvDeepSet Convolutional Deep Set

ConvCNP Convolutional Neural Process

DAG Directed Acyclic Graph

DGM Deep Generative Models

DLVM Deep Lantent Variable Model

ELBO Evidence Lower Bound

EM Expectation-Maximization

EQ Exponentiated-Quadratic

GAN Generative Adversarial Network

GP Gaussian Processes

GRU Gated Recurrent Unit

KL Kullback-Leibler

LSTM Long Short-Term Memory

LVM Latent Variable Model

ML Machine Learning

MSE Mean Squared Error

Neural CDE Neural Controlled Differential Equation

Neural ODE Neural Ordinary Differential Equation

NP Neural Process

ODE Ordinary Differential Equation

ReLU Rectified Linear Unit

RKHS Reproducing Kernel Hilbert Space

RNN Recurrent Neural Network

SGVB Stochastic Gradient Variational Bayes

SP Stochastic Process

SVI Stochastic Variational Inference

TE Translation Equivariance

VAE Variational Autoencoder

VI Variational Inference

MLP Multilayer Perceptron

Chapter 1

Introduction

With recent advances in Machine Learning (ML) a lot of extremely important tasks in different fields, including computer vision, speech recognition, and reinforcement learning, were solved with a sufficiently good quality. Such a breakthrough became possible due to the advances and hardware as well as novel ML methods, scalable to complex and high dimensional data. Deep learning has played a paramount role in these recent advances. In particular, the introduction of Convolutional Neural Networks (CNNs) for images and Recurrent Neural Networks (RNNs) for sequential data not only made it possible to obtain excellent performance on various ML tasks, but also revolutionized the way research community approached those tasks.

However, despite the unquestionable success of neural models in solving various ML problems, they are generally limited to achieving excellent performance only on highly structured data, such as images or natural language data. One key limitation is that most of these models either cannot handle non-uniformly sampled data or work poorly on it. Nonetheless, in some areas, like environmental or medical research, the modeled data typically does not reside on a grid, nor is it uniformly sampled. In practice data of this type is usually modeled with a standard RNN with the difference between each successive time-point appended as a feature. But this approach still shows unsatisfactory performance and more specifically tailored solutions are desirable.

Recently, there has been a surge of interest in solving the considered task, and a lot of different approaches from several ML domains have been proposed. For instance, Neural ODEs [9] leverage a connection between residual networks and ordinary differential equations to handle data in continuous domains [69]. Conversely, convolutional conditional neural processes [28] provide a convolutional form for set-structured data, relaxing the requirement for data to be uniformly sampled. The two methods approach the problem from independent perspectives and in some ways mirror the use of RNNs vs CNNs for sequences.

Coming from different ML domains, the existing models lack of unifying perspective that could be used to compare and contrast models in this field. Towards this end, in this thesis we are mainly concerned with the development of a unifying framework, which provides a rigorous text book understanding of the field as well as explicitly explains main research approaches and directions used to solve this task.

1.1 Thesis Contributions

The main contributions of this work are as follows:

1. A **thorough review of existing machine learning methods** for modelling irregularly-sampled data in chapter 3. We have discussed key approaches within unsupervised and meta-learning paradigms, which are suitable for handling non-uniformly sampled data.

2. A **unifying design model space** that organizes conceptually different approaches and places them into the common context. We organize the model space by devising a set of schematics, which outline the most important modelling dimensions. The discussion of the model design space is provided in chapter 4.

3. **An introduction of novel models** encountered within the proposed design model space. The proposed models, discussed in detail in chapter 4, operate within the meta-learning framework and could be applied to various few-shot learning tasks involving time-series data.

4. **An evaluation of several models from the introduced modelling space**. The experiments were held on both real-world and synthetically generated data. The evaluation of the models is followed by a discussion of the results in chapter 6, where we outline future research directions that can leverage the results of this work.

Chapter 2

Background

In this chapter, we introduce and discuss key ideas and fundamental models which this thesis draws upon in later chapters. In section 2.1, we elaborate on Neural Ordinary Differential Equations (Neural ODEs) [9] and their connection to residual networks. With a continuously-defined hidden function, the Neural ODE model is suitable for handling irregularly-sampled data. For this reason, the model is widely used as an important component in recent methods discussed in later chapters of this work [9, 38, 69].

Other fundamental concepts actively used in this work are Latent Variable Models (LVMs) and Variational Inference (VI) framework, which is commonly used to train LVMs. In particular, several recent approaches [9, 22, 69] that are of utmost importance to our discussion use VI framework for model fitting. For this reason, the discussion of Neural ODEs [9] is followed by the review of latent variable models and variational inference in section 2.2.

2.1 Neural Ordinary Differential Equations

One of the most prominent approaches to modelling continuous time series data is the Neural ODE model [9]. Neural ODE has continuously-defined dynamics, which makes this model particularly appealing for the task of modelling off-the-grid data. Since several successful neural models [35, 69] are built upon the Neural ODE [9] framework, Neural ODEs [9] constitutes one of the key ideas this thesis draws upon. For this reason, a detailed description of Neural ODEs [9] is provided in this section.

Residual Networks and Model Definition

A lot of successful neural network architectures, including RNNs, residual networks [31] and normalizing flows [65], have a similar general structure. These models build their capacity not by learning a complex function f that is applied only once, but by composing a sequence of transformations to a hidden state. At each discrete time step the hidden state is transformed as follows:

$$h_{t+1} = h_t + f(h_t, \theta_t) \tag{2.1}$$

where $t \in \{0, \ldots, T\}$ and f is some learned function of the current hidden state and parameters θ_t. If we define $\theta = [\theta_1, \ldots, \theta_T]^T$ to be a vector comprising all trainable parameters, the hidden transformation can be rewritten as:

$$h_{t+1} - h_t = f(h_t, t, \theta) \tag{2.2}$$

Equation 2.2 is a difference equation with $\Delta t = 1$. If we now increase the number of sequential transformations and decrease the time step size from 1 to some $\Delta t < 1$, we will obtain the following equation:

$$h_{t+\Delta t} - h_t = \Delta t f(h_t, t, \theta) \tag{2.3}$$

where $t \in \{0, \Delta t, \ldots, T\}$. It can be noticed that Equation 2.3 can be viewed as a step of the forward Euler method for the ordinary differential equation (ODE) obtained by taking the limit $\Delta t \to 0$:

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \tag{2.4}$$

where t is continuous and $t \in [0, T]$.

Hence, if we are given initial hidden state value $h(0)$, the output layer $h(T)$ can be defined as a solution to the ODE initial value problem at time T. The computation of the $h(T)$ can then be executed using a black-box ODE numerical solver. So, this black-box ODE solver is essentially a map from initial hidden state value h_{t_0} to hidden state $h(t_1)$ at some time t_1:

$$h(t_1) = \text{ODESolve}(f_\theta, h(t_0), t_0, t_1) \tag{2.5}$$

Equation 2.5 defines the Neural ODE model [9].

Model Training and Adjoint method

Since we are interested in learning parameters θ, we need to be able to perform backpropagation through the ODE solver. One way to do this is to directly backpropagate through the operations of the ODE solver. However, if we use this method, we would have to store

all the intermediate hidden states in memory. Hence, direct backpropagation results in high memory cost of $\mathcal{O}(TH)$, where H is the memory footprint of the vector field and T is the time horizon.

Another way to compute gradients with respect to parameters θ and intermediate hidden states $h(t)$ is using the adjoint sensitivity method [61]. This method treats ODE solver as a black box and calculates gradients by solving another ODE backwards in time. In contrast to direct backpropagation, adjoint method does not require all intermediate hidden states to be stored in memory. It also can be applied to any ODE solver. As a result, this method reduces memory costs from $\mathcal{O}(TH)$ to $\mathcal{O}(H)$. Moreover, the adjoint method [61] provides an explicit control over numerical error.

We now discuss the adjoint method [61] in detail. Assume we are given initial state $h(t_0)$ and internal parameters θ. The output of the Neural ODE [9] model is defined as

$$h(t_1) = \text{ODESolve}(f_\theta, h(t_0), t_0, t_1) \tag{2.6}$$

Then we are given a scalar-valued loss function $\mathscr{L}$ that is evaluated on $h(t_1)$. To optimize $\mathscr{L}$, we need to calculate the derivatives of $\mathscr{L}$ with respect to parameters $\frac{d\mathscr{L}}{d\theta}$ as well as to intermediate hidden states $\frac{d\mathscr{L}}{dh(t)}$. In the adjoint sensitivity method [61], $\frac{d\mathscr{L}}{dh(t)}$ is referred to as the adjoint $a(t)$. The adjoint $a(t)$ determines how the gradient of the loss depends on the hidden state $h(t)$ at each t. It can be shown that the adjoint $a(t)$ is a solution to the following initial value problem:

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial}{\partial h(t)} f(h(t), t, \theta) \tag{2.7}$$

$$a(t_1) = \frac{d\mathscr{L}}{dh(t_1)} \tag{2.8}$$

Hence, we can compute $\frac{d\mathscr{L}}{dh(t_0)}$ by using black box ODE solver again. This time the ODE solver will be run backwards in time since $\frac{d\mathscr{L}}{dh(t_1)}$ is easy to compute. However, it can be seen that for solving the auxiliary ODE (Equation 2.7) we need to know the value of $h(t)$ along its entire trajectory. To solve this issue, $h(t)$ is also recomputed backwards starting from its final value $h(t_1)$.

Given the adjoint $a(t)$, we can find the derivative with respect to parameters θ as:

$$\frac{d\mathscr{L}}{d\theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(h(t), t, \theta)}{\partial \theta} dt \tag{2.9}$$

Let's define $b(t) = \frac{d\mathscr{L}}{d\theta(t)} = -\int_{t_1}^{t} a(t)^T \frac{\partial f(h(t),t,\theta)}{\partial \theta} dt$. From this definition, $b(t_1) = 0$ and $b(t_0) = \frac{d\mathscr{L}}{d\theta(t_0)} = \frac{d\mathscr{L}}{d\theta}$. Then $b(t)$ can be seen as a solution to another backward ODE defined as follows:

$$\frac{db(t)}{dt} = -a(t)^T \frac{\partial}{\partial \theta} f(h(t),t,\theta) \tag{2.10}$$

$$b(t_1) = 0 \tag{2.11}$$

A detailed derivation of Equations 2.7 and 2.9 is provided in Appendix B of [9].

All in all, in order to find all necessary derivatives, $\frac{d\mathscr{L}}{dh(t_0)}$ and $\frac{d\mathscr{L}}{d\theta}$, we need to solve the augmented adjoint backward ODE:

$$\frac{d}{dt}\begin{pmatrix} z(t) \\ a(t) \\ b(t) \end{pmatrix} = \begin{pmatrix} f(h(t),t,\theta) \\ -a(t)^T \frac{\partial f(h(t),t,\theta)}{\partial h(t)} \\ -a(t)^T \frac{\partial f(h(t),t,\theta)}{\partial \theta} \end{pmatrix} \tag{2.12}$$

In this work, a PyTorch [60] implementation from github.com/rtqichen/torchdiffeq was used in all our experiments involving Neural ODEs [9].

2.2 Latent Variable Models and Variational Inference

LVMs are a powerful concept in probabilistic modelling. They provide a way to explain complex relations between observed variables by some simple correspondences between observed and latent variables. Suppose we have a set of manifest variables $\mathscr{X} = (x_1, \ldots, x_n)$ that can be observed and a set of latent variables $\mathscr{Z} = (z_1, \ldots, z_m)$. In general, LVMs consider that the latent variables can explain dependencies between the manifest variables, i.e. given latent variables, manifest ones are assumed to be conditionally independent:

$$p(x_1, \ldots, x_n, z_1, \ldots, z_m) = p(z_1, \ldots, z_m) \prod_{i=1}^{n} p(x_i | z_1, \ldots, z_m) \tag{2.13}$$

In particular, in this work we are mainly concerned with a specific type of LVMs, where latent variable z drives the generation process of the observed data x. The directed acyclic graph (DAG) for the considered process is given in Figure 2.2:

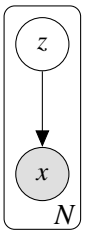


Fig. 2.1 DAG for the considered LVM. Observed variables are shaded in gray.

This model can be formulated as follows:

$$z \sim p_{\theta^*}(z) \tag{2.14}$$

$$x \sim p_{\theta^*}(x|z) \tag{2.15}$$

where $p_{\theta^*}(z), p_{\theta^*}(x|z)$ are some valid distributions and θ^* are parameters of these distributions. Usually the dimensionality of z is chosen to be substantially smaller than that of the observed variable x, such that z incorporates significant high level information about x.

In practice we are only given the observed data $\{x_i\}_{i=1}^N$. Using this data, we want to infer the parameters of the model θ (it can be either a point estimate or parameters of the distribution over θ) as well as find the posterior distribution $p(z|x)$.

The exact inference in this model is only possible in a very limited number of simple cases, e.g. Gaussian mixture or hidden Markov models. If the posterior is tractable, the Expectation-Maximization (EM) algorithm [16] is usually applied. However, if $p_\theta^*(x|z)$ is parameterized with a more complex non-linear model, EM algorithm cannot be used and more powerful methods for approximate inference are required. VI framework [36], discussed below, is one of the most commonly used approaches for approximate inference.

2.2.1 Variational inference

VI [36] is a framework used to perform an approximate inference in models where the posterior distribution is intractable. The core idea behind variational inference is to introduce some tractable family of distributions over the latent variables $q(z)$ and then to find $q^*(z)$ within this family, such that $q^*(z)$ is as close as possible to the true posterior distribution [5]. The proximity of two distributions is measured using Kullback–Leibler (KL) divergence. VI is a particularly appealing approach, as it solves the inference problem by solving the optimization one. This allows VI framework to be easily scaled to larger datasets and models [5].

Within the VI framework, we introduce a family $\mathscr{Q}$ of distributions over latent variables. Our aim is to find $q^*(z) \in \mathscr{Q}$, such that $q^*(z)$ is the closest distribution to the true posterior.

Hence, inference now consists of solving the following optimization problem

$$q^*(z) = \arg\max_{q \in \mathcal{Q}} \mathrm{KL}(q(z)||p(z|x)) \tag{2.16}$$

However, $\mathrm{KL}(q(z)||p(z|x))$ cannot be directly computed since it involves computation of the logarithm of the evidence $\log p(x)$, which is intractable:

$$\mathrm{KL}(q(z)||p(z|x)) = \mathbb{E}_{q(z)}[\log q(z)] - \mathbb{E}_{q(z)}[\log p_\theta(z,x)] + \log p(x) \tag{2.17}$$

Equation 2.17 expands the KL divergence and shows where computation of $\log p(x)$ is involved. However, since $\log p(x)$ does not depend on $q(z)$, instead of directly optimizing the $\mathrm{KL}(q(z)||p(z|x))$ we can optimize the following alternative objective:

$$\mathrm{ELBO}(q) = \mathbb{E}_{q(z)}[\log p_\theta(z,x)] - \mathbb{E}_{q(z)}[\log q(z)] \tag{2.18}$$

The alternative objective from Equation 2.18 is called Evidence Lower BOund (ELBO). The maximization of ELBO with respect to q is identical to minimization of $\mathrm{KL}(q(z)||p(z|x))$.

ELBO can be rewritten as follows:

$$\mathrm{ELBO}(q) = \mathbb{E}_{q(z)}[\log p_\theta(x|z)] - \mathrm{KL}[q(z)||p(z)] \tag{2.19}$$

Equation 2.19 provides intuition about which variational distributions the evidence lower bound favours. The first term in Equation 2.19 is the expected log-likelihood of the observed data. It indicates how well the data is explained by $q(z)$. The second term is the negative KL-divergence between prior and variational distributions. It forces variational distribution to be close to prior.

It can be shown that ELBO is a lower bound for $\log p(x)$. Equation 2.17 implies the following:

$$\log p(x) = \mathrm{ELBO}(q) + \mathrm{KL}(q(z)||p(z|x)) \tag{2.20}$$

Since KL-divergence is always non-negative, then:

$$\log p(x) \geq \mathrm{ELBO}(q) \tag{2.21}$$

There is a clear trade-off in the choice of variational family $\mathcal{Q}$. On the one hand, the more complex the chosen variational family is, the more difficult is the optimization of ELBO will be. On the other hand, the more flexible the chosen variational family is, the better variational approximation can be found.

Traditional VI approaches, such as mean-field approximation [82] and variational EM algorithm [24], suffer from certain drawbacks that restrict their scalability and flexibility. First, in these approaches separate variational parameters are introduced for each data point. This means that for each new data point, inference of the variational parameters must be performed, which may be computationally demanding for large datasets. Moreover, the number of parameters that needs to be stored grows linearly with the number of data points. Second, traditional VI methods usually imply that expectations in ELBO (Equation 2.19) are tractable, thus imposing a restriction on the complexity of variational family.

Fortunately, recent advances in VI framework, including black-box [63] and amortized variational inference [43, 66], found a way to overcome those issues and to scale VI to significantly larger datasets.

Stochastic Variational Inference

Let's assume that variational family $\mathcal{Q}$ is chosen to be parametric and $q(z;\phi)$ is parameterized by variational parameters ϕ. Given observed dataset $\{x_i\}_{i=1}^n$ the main goal is to find variational parameters for each data point $\{\phi_i\}_{i=1}^n$ and generative model parameters θ that maximize ELBO objective for the whole dataset:

$$\text{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i\}_{i=1}^n, \theta) = \sum_{i=1}^n \text{ELBO}(x_i, \phi_i, \theta) \tag{2.22}$$

In traditional VI approaches, which use coordinate ascent inference, at step t the global parameters θ are updated using the gradient of $\text{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i\}_{i=1}^n, \theta)$:

$$\nabla_\theta \text{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i^t\}_{i=1}^n, \theta^{t-1}) = \sum_{i=1}^n \nabla_\theta \text{ELBO}(x_i, \phi_i^t, \theta^{t-1}), \tag{2.23}$$

where $\{\phi_i^t\}_{i=1}^n$ are current estimates of variational parameters when $\theta = \theta^{t-1}$. Equation 2.23 implies that with each update of θ we have to recalculate the variational parameters for each data point x_i. Undoubtedly, this makes traditional VI methods hard to scale to large datasets. Moreover, in large data regime we expect that even a subset of the data can be informative about the global parameters θ.

This scalability issue may be solved by using stochastic optimization [67]. Stochastic optimization is performed by using noisy unbiased estimates of the gradient instead of the full gradient. Under some mild conditions on the step-size schedule, stochastic optimization algorithms provably converge to an optimum of the objective [67]. Stochastic optimization is particularly appealing in the case of VI since ELBO objective is essentially a sum of many

terms that can be independently evaluated. In this setting, we can cheaply compute noisy gradients of ELBO by subsampling only a few of these terms:

$$\nabla_\theta \text{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i^t\}_{i=1}^n, \theta^{t-1}) \approx n \nabla_\theta \text{ELBO}(x_i, \phi_i^t, \theta^{t-1}), i \sim \mathscr{U}\{1,\dots,n\} \tag{2.24}$$

where $\mathscr{U}\{1,\dots,n\}$ stands for discrete uniform distribution over set $\{1,\dots,n\}$.

The other issue in traditional VI methods is that they impose a restrictions on the complexity of generative model as well as variational family, as they consider expectations in ELBO (Equation 2.19) to be tractable. Moreover, the gradient of ELBO w.r.t. the global parameters θ cannot be calculated as well, since it also requires taking expectations over the variational distribution $q(z_i;\phi_i)$:

$$\nabla_\theta \text{ELBO}(x_i, \phi_i, \theta) = \mathbb{E}_{q(z_i;\phi_i)}[\nabla_\theta \log p_\theta(x_i|z_i)] \tag{2.25}$$

Although the Equation 2.25 is intractable in general, it can be approximated using Monte-Carlo sampling:

$$\nabla_\theta \text{ELBO}(x_i, \phi_i, \theta) \approx \frac{1}{L}\sum_{l=1}^{L} \nabla_\theta \log p_\theta(x_i|z_i^l), \text{ where } z_i^l \sim q(z_i;\phi_i) \tag{2.26}$$

As a result, we obtain a scalable way of learning the global parameters θ in generative models with a complex non-linear structure:

$$\begin{gathered}\nabla_\theta \text{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i\}_{i=1}^n, \theta) \approx \frac{n}{L}\sum_{l=1}^{L} \nabla_\theta \log p_\theta(x_i|z_i^l), \\ \text{where } z_i^l \sim q(z_i;\phi_i), i \sim \mathscr{U}\{1,\dots,n\}\end{gathered} \tag{2.27}$$

However, the derivative of ELBO objective w.r.t ϕ_i cannot be evaluated in the same manner via Monte-Carlo estimate, as in Equation 2.26. This is due to the fact that the distribution $q(z_i;\phi_i)$, over which the expectation is taken, depends on the parameters ϕ_i.

Let's assume that KL-divergence in ELBO can be computed and differentiated in closed form. If KL-divergence cannot be evaluated in the closed form, we could apply the same logic as for $\nabla_{\phi_i}\mathbb{E}_{q(z_i;\phi_i)}[\log p_\theta(x_i|z_i)]$ to approximate it.

Now we need to approximate $\nabla_{\phi_i}\mathbb{E}_{q(z_i;\phi_i)}[\log p_\theta(x_i|z_i)]$:

$$\begin{aligned}\nabla_{\phi_i}\mathbb{E}_{q(z_i;\phi_i)}[\log p_\theta(x_i|z_i)] = \int_{\mathscr{Z}} \nabla_{\phi_i} q(z_i;\phi_i)\log p_\theta(x_i|z_i)dz_i = \\ = \int_{\mathscr{Z}} q(z_i;\phi_i)\nabla_{\phi_i}[\log q(z_i;\phi_i)]\log p_\theta(x_i|z_i)dz_i = \\ = \mathbb{E}_{q(z_i;\phi_i)}[\nabla_{\phi_i}[\log q(z_i;\phi_i)]\log p_\theta(x_i|z_i)] \approx \\ \approx \frac{1}{L}\sum_{l=1}^{L}\nabla_{\phi_i}[\log q(z_i^l;\phi_i)]\log p_\theta(x_i|z_i^l),\ \text{where } z_i^l \sim q(z_i;\phi_i)\end{aligned} \tag{2.28}$$

However, this gradient estimator exhibits extremely high variance [58] and could not be used in practice because the procedure does not converge. Hence, either some variance reduction schemes [57], or a more robust gradient estimator is needed.

the Reparameterization Trick

Kingma and Welling [43], Rezende et al. [66] proposed a more practical estimator of the derivative of ELBO w.r.t. the variational parameters. If z is a continuous random variable, it is then often possible to express the random variable $\tilde{z} \sim q(z;\phi)$ as a deterministic differentiable function $\tilde{z} = g(\varepsilon,\phi)$ of an auxiliary noise variable ε:

$$\tilde{z} = g(\varepsilon,\phi), \text{where } \varepsilon \sim p(\varepsilon) \tag{2.29}$$

Using Equation 2.29 and the change-of-variables rule, we can obtain a better estimator for $\nabla_{\phi_i}\mathbb{E}_{q(z_i;\phi_i)}[\log p_\theta(x_i|z_i)]$:

$$\begin{aligned}\nabla_{\phi_i}\mathbb{E}_{q(z_i;\phi_i)}[\log p_\theta(x_i|z_i)] = \nabla_{\phi_i}\int_{\mathscr{Z}} q(z_i;\phi_i)\log_\theta(x_i|z_i)dz_i = \\ = \nabla_{\phi_i}\int_{\mathscr{E}} p(\varepsilon)\log_\theta(x_i|g(\varepsilon,\phi_i))d\varepsilon = \mathbb{E}_\varepsilon[\nabla_{\phi_i}\log_\theta p(x_i|g(\varepsilon,\phi_i))] \approx \\ \approx \frac{1}{L}\sum_{l=1}^{L}\nabla_{\phi_i}\log_\theta p(x_i|g(\varepsilon^l,\phi_i)),\ \text{where } \varepsilon^l \sim p(\varepsilon)\end{aligned} \tag{2.30}$$

As a result, we can formulate an optimization algorithm performing Stochastic Variational Inference (SVI) . The algorithm is similar to the black-box VI [63], but uses reparameterization trick to obtain estimates of ELBO derivatives.

Algorithm 1: Stochastic Variational Inference

Input : Data points $\{x_i\}_{i=1}^n$

Randomly initialize variational parameters $\{\phi_i\}_{i=1}^n$.

repeat

  Draw $i \sim \mathscr{U}\{1,\ldots,n\}$

  $\phi_i^0 = \phi_i$

  for $k = 1,\ldots,K$ **do**

    $\phi_i^k = \phi_i^{k-1} + \alpha \hat{\nabla}_{\phi_i} \mathrm{ELBO}(x_i, \phi_i^{k-1}, \theta)$; // (Equation 2.30)

  $\phi_i = \phi_i^K$

  Update θ based on the $\hat{\nabla}_\theta \mathrm{ELBO}(\{x_i\}_{i=1}^n, \{\phi_i\}_{i=1}^n, \theta)$; // (Equation 2.27)

until *convergence*;

SVI (Algorithm 1) optimizes directly for per-sample variational parameters. This may as well hamper the scalability of VI since the algorithm may require running iterative inference for a large number of steps. What is more, the number of variational parameters needed to be stored grows linearly with the number of data points. This issue can be overcome by using inference networks to approximate $q_\phi(z|x)$, as discussed below.

Inference Networks and Stochastic Backpropagation

A solution to the above problem is to utilize an inference network as the approximation for $q_\phi(z|x)$. Models that incorporate inference network are commonly referred to as *amortized models*.

In this setting, a powerful parametric model is chosen to be an inference network and the model is then trained to map data points x to the variational parameters of the posterior distribution over z. In case of amortized models, ϕ is of fixed dimension and shared across all data points $\{x_i\}_{i=1}^n$, so that memory consumption is constant w.r.t. n. Moreover, new data points can be now easily incorporated to the system, in contrast to SVI (Algorithm 1), where additional optimization is required.

Another appealing quality of amortized models is the ability to perform joint training of the model parameters θ and variational parameters ϕ.

The model is trained using Stochastic Gradient Variational Bayes (SGVB) estimator [43], which is essentially a Monte-Carlo estimator of ELBO:

$$\begin{gathered}\tilde{\mathscr{L}}_{SGVB}(\theta,\phi,x_i) = \frac{1}{L}\sum_{l=1}^{L} \log_\theta(x_i, z_i^l) - \log q_\phi(z_i^l|x_i), \\ \text{where } z_i^l = g(\varepsilon^l, x_i, \phi), \varepsilon^l \sim p(\varepsilon)\end{gathered} \tag{2.31}$$

The employment of the reparametrization trick in Equation 2.31 allows to directly backpropagate through stochastic component and update variational parameters ϕ jointly with θ.

The SGVB estimator can be naturally extended to work with minibatches:

$$\tilde{\mathscr{L}}^{m}_{SGVB}(\theta,\phi,\boldsymbol{x}^{m}) = \frac{n}{m}\sum_{i=1}^{m}\tilde{\mathscr{L}}_{SGVB}(\theta,\phi,x_i), \tag{2.32}$$

where $\boldsymbol{x}^{m}$ is a randomly drawn sample of m datapoints from the full dataset $\{x_i\}_{i=1}^{n}$.

The estimator 2.32 can be efficiently maximized using stochastic optimization methods, such as Adam [42] or Adagrad [18], yielding the Auto-Encoding Variational Bayes (AEVB) algorithm, also known as stochastic backpropagation [66]. The AEVB algorithm is detailed in Algorithm 2.

Algorithm 2: Minibatch version of the AEVB algorithm.

Input : Data points $\{x_i\}_{i=1}^{n}$

Initialize parameters θ,ϕ.

repeat

  Sample minibatch of m datapoints $\boldsymbol{x^m}$

  Sample $\varepsilon^{l} \sim p(\varepsilon)$ for $l = 1,\ldots,L$

  $g \leftarrow \nabla_{\phi,\theta}\tilde{\mathscr{L}}^{m}_{SGVB}(\theta,\phi,\boldsymbol{x}^{m},\{\varepsilon^{l}\}_{l=1}^{L})$

  Update parameters θ,ϕ using gradients g

until *convergence*;

Output : (θ,ϕ)

2.2.2 Variational Autoencoder

One of the most well-known instances of the family of Deep Latent Variable Models (DLVMs) is the Variational Autoencoder (VAE) model. VAE makes the assumption that observed data generation process is driven by a set of latent variables $\{z_i\}_{i=1}^{n}$.

VAE follows the VI framework and leverages stochastic backpropagation for efficient approximate inference. The training procedure of the model is performed via the AEVB algorithm, detailed in Algorithm 2.

The VAE framework parameterizes $q_\phi(z|x)$ and $p_\theta(x|z)$ with neural networks. In practise, the variational posterior takes the form of a diagonal Gaussian:

$$q_\phi(z|x) = \mathscr{N}(z;\mu_\phi(x),\sigma^2_\phi(x)), \tag{2.33}$$

where $\mu_\phi(x), \sigma^2_\phi(x)$ are neural networks that map observed data x to the variational parameters of z. For Normal distribution, the reparametrization trick can be applied since it is straightforward to show that for z sampled as:

$$z = g_\phi(\varepsilon, x) = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \text{ with } \varepsilon \sim \mathcal{N}(\varepsilon; \mathbf{0}, \boldsymbol{I}) \tag{2.34}$$

holds that $z \sim \mathcal{N}(z; \mu_\phi(x), \sigma^2_\phi(x))$.

The Generative distribution $p_\theta(x|z)$ can be any valid distribution over x. Commonly, it is chosen to be Normal distribution for continuous data and Bernoulli one for binary data. The prior $p(z)$ is chosen to be a unit variance isotropic Gaussian $p(z) = \mathcal{N}(z; \mathbf{0}, \boldsymbol{I})$.

It can be noticed that the architecture of VAE is very similar to deterministic autoencoders [81]. Moreover, akin to autoencoders, the latent space $\mathcal{Z}$ is usually chosen to be of substantially lower dimension than $\mathcal{X}$. The main difference between VAEs and standard autoencoders is that autoencoder provide a point estimate for z, whereas VAE forces the approximate posterior not to collapse to point estimate via KL-term in ELBO objective.

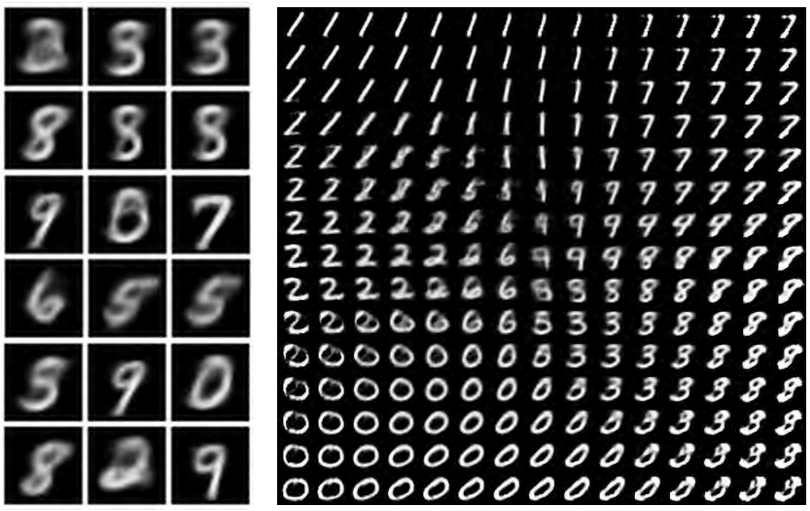
VAEs employ a very efficient and straightforward sampling strategy:

$$\begin{aligned} z &\sim \mathcal{N}(z; \mathbf{0}, \boldsymbol{I}) \\ x &\sim p_\theta(x|z) \end{aligned} \tag{2.35}$$

In many recent works [43, 66] it was empirically shown that latent space serves as a low-dimensional manifold that incorporates important patterns in the data.

Generative capabilities of VAEs as well as structure of the latent space are illustrated in Figure 2.2.

To sum up, this chapter is divided into the discussion of two ML concepts that are extremely useful in understanding of several recently proposed approaches to modelling off-the-grid data. The first concept is Neural ODEs [9], which is used by several models from the design model space introduced in chapter 4. Among those models are ODE-RNN [69] and Latent ODE [9]. These models rest on Neural ODE [9] to model continuously-defined evolution of hidden function. Both models are described in detail in chapter 3. The second part of the current chapter is devoted to the review of LVMs and VI framework. This part was necessary for further investigation of LVMs developed for time-series modelling, such as Latent ODE [9] and Neural Processes (NPs) [22]. These approaches are also investigated in the next chapters of this work.



(a) (b)

Fig. 2.2 (a) Samples from a trained VAE model. (b) Visualization of the latent manifold of the 2-dimensional VAE. Linearly spaced coordinates of the 2d unit hypercube were passed through the inverse cumulative distribution function of the standard normal distribution to produce values of the latent variables z. Each of these values was then decoded using $p_\theta(x|z)$. For both illustrations, we used our implementation of a VAE trained on MNIST with a 2-dimensional latent space.

Chapter 3

Related Work

In the previous chapter, we reviewed general ML concepts, namely Neural ODEs [9] and VI [36] framework. They serve as a foundation for many recent neural approaches to model off-the-grid data. In the present chapter, we discuss these approaches in more detail since they comprise a part of the design model space for irregularly-sampled data that we introduce in this work.

On a global scale recently proposed methods can be divided into two massive groups. In the first group, approaches adhere to unsupervised learning paradigm, where the core goal is to fit the true data distribution. These approaches in the context of time-series data are discussed in section 3.1. The second group comprises models that follow meta-learning framework. According to this framework, each time-series is considered to be a separate dataset and the main aim is to train a model that can generalize well across different datasets. The models belonging to the second group are investigated in section 3.2. Finally, in section 3.3, we discuss autoregressive models that can be attributed to both unsupervised and meta-learning paradigms.

3.1 Unsupervised Learning Paradigm

In this section, we describe the unsupervised learning paradigm in the context of modelling irregularly-sampled time series data. We also describe several recent neural models that employ unsupervised training and are suitable for modelling off-the-grid data [9, 69].

In the unsupervised learning paradigm, the core goal is to learn hidden structure from the unlabelled data as well as detect patterns present in it. Unsupervised learning encompasses a huge number of machine learning tasks, including density estimation [26, 43, 74], clustering [4], feature learning [17, 52] and dimensionality reduction [1].

Perhaps, a central application of unsupervised learning lies in the domain of density estimation, where the main aim is to find an adequate approximation to the true data distribution. With recent advances in deep learning, the most successful models in this domain are Deep Generative Models (DGM), which leverage neural networks to model complex data. In the field of DGMs, there are several commonly used approaches to data modelling: VAEs [43] (section 2.2), Generative Adversarial Networks (GANs) [26] and Autoregressive Networks (ANs) [23] . In this work, we mainly focus on models that explicitly learn data distribution, namely VAEs and ANs.

3.1.1 Variational Autoencoders: Latent ODE

In this section, we discuss the Latent ODE model proposed in [9]. This model builds upon the theory of VAEs (section 2.2) and can be used for modelling irregularly-sampled data.

In more detail, Latent ODE assumes that the data generation process is driven by a latent dynamics that is parameterized by a Neural ODE [9]. As in the case of Neural ODEs [9] initial latent state z_{t_0} determines the entire trajectory, then, given an initial state z_{t_0} and observation times $t_{1:N}$, we can obtain latent states $z_{1:N}$ at each time location $t_{1:N}$. Formally, the Latent ODE [9] model can be formulated as follows:

$$z_0 \sim p(z_0) \tag{3.1}$$

$$z_{1:N} = \text{ODESolve}(f_\theta, z_0, t_0, t_{1:N}) \tag{3.2}$$

$$x_i \sim p_\theta(x_i|z_i), i = 1, \ldots, N \tag{3.3}$$

where f_θ is a neural network that drives the evolution of latent process, prior $p(z_0)$ is taken to be standard normal distribution and $p_\theta(x_i|z_i)$ is a multilayer perceptron (MLP) that models conditional distribution $p(x_i|z_i)$.

Since the posterior is intractable, the VI framework is used to train the model. This requires introduction of the approximate posterior distribution $q_\phi(z_0|x_{1:N}, t_{1:N})$. Originally, Latent ODE [9] model parameterized recognition network as a simple RNN. However, the authors in [69] empirically showed that using ODE-RNN [69] as a recognition network improves the performance of the Latent ODE [9] model on irregularly-sample time series data. Hence, $q_\phi(z_0|x_{1:N}, t_{1:N})$ is defined as:

$$q_\phi(z_0|x_{1:N}, t_{1:N}) = \mathcal{N}(z_0; \mu_{z_0}, \sigma^2_{z_0}), \text{ with } (\mu_{z_0}, \sigma^2_{z_0}) = h_\phi(\text{ODE-RNN}(x_{1:N}, t_{1:N}, t_0)) \tag{3.4}$$

where h_ϕ is a neural network that maps the final hidden state of the ODE-RNN [69] into the variational parameters for z_0. To obtain these parameters at time point t_0, ODE-RNN [69] is run backwards from the latest observation time t_N to t_0.

Similar to standard VAEs, Latent ODE [9] is learnt via stochastic optimization of ELBO, as described in section 2.2.

3.2 Meta-Learning Paradigm

In this section, we frame the task of interpolation/extrapolation for irregularly-sampled time series data as a meta-learning task. We also discuss recently proposed neural models that could be used to handle this task [20–22, 28].

In the meta-learning paradigm [72, 77], we develop a learning algorithm that can generalize and perform well across different tasks (datasets). In contrast to supervised learning, in meta-learning, each data sample is considered to be a separate dataset $\mathscr{D}$. Hence, a good meta-learning model should be trained on a set $\mathscr{S}$ of learning tasks $\mathscr{D} \subseteq \mathscr{S}$ and optimized for the best performance on distribution of learning tasks:

$$\theta^* = \arg\max_{\theta \in \Theta} \mathbb{E}_{\mathscr{D} \sim p(\mathscr{D})}[\mathscr{L}_\theta(\mathscr{D})], \tag{3.5}$$

where $\mathscr{L}_\theta(\mathscr{D})$ is a defined objective, which measures the performance of the model with parameters θ on learning task $\mathscr{D}$.

In case of off-the-grid time series data, each dataset $\mathscr{D}$ is assumed to be a separate time series sample. Each sample is often split into two parts: a context set $D_C = (t^C_{1:N_C}, x^C_{1:N_C})$, where values at time points $t^C_{1:N_C}$ are observed, and target set $D_T = (t^T_{1:N_T}, x^T_{1:N_T})$, where the values at time points $t^C_{1:N_T}$ are to be predicted. The goal is then to train a model to better approximate the conditional distribution $p(x^T_{1:N_T} | D_C, t^T_{1:N_T})$:

$$\theta^* = \arg\max_{\theta \in \Theta} \mathbb{E}_{\mathscr{D} \sim p(\mathscr{D})}[\log p_\theta(x^T_{1:N_T} | D_C, t^T_{1:N_T})] \tag{3.6}$$

In the next section, we discuss a family of neural models that can be used to efficiently model conditional distribution $p(x^T_{1:N_T} | D_C, t^T_{1:N_T})$.

3.2.1 Neural Processes Family

The NP family comprises a variety of neural models that can be applied to solve miscellaneous meta-learning tasks. A unifying feature of models belonging to the NP family is the embedding of the context sets D_C into some representation space, either vector [21, 22]

or functional [20, 28] one, through an encoder $D_C \rightarrow E(D_C)$. The obtained representation then serves as prior knowledge that is used to infer function values $x^T_{1:N_T}$ at target set time locations $t^T_{1:N_T}$. The inference itself is performed via a neural decoder ρ that takes the context set representation $E(D_C)$ and a target set time locations $t^T_{1:N_T}$. In contrast to Gaussian Processes (GPs) [64], which scale as $\mathcal{O}((N_T + N_C)^3)$, NP family models have a linear running time complexity owing to its specification. This ability to generate predictions from new context sets make on the fly makes NP models particularly appealing to be used for solving meta-learning tasks.

In the next subsections we describe various instances of the NP family in the context of modelling off-the-grid time series data.

Conditional Neural Processes

Conditional Neural Processes (CNPs) parametrize conditional stochastic processes given a latent representation of a context set D_C of fixed dimensionality. The representation of a context set is obtained using a DeepSet function approximator [83]. Employing DeepSets [83] as an encoder $E(D_C)$ in CNP implies the following structure of the encoder:

$$r_i = h_\theta(t^C_i, x^C_i), i = 1, \ldots, N_C \tag{3.7}$$

$$E(D_C) = r_1 \oplus r_2 \oplus \ldots r_{N_{C-1}} \oplus r_{N_C}, \tag{3.8}$$

where h_θ is a MLP that maps a pair (t_i, x_i) to the vector embedding space of fixed dimensionality, $\oplus$ is a commutative operation that takes elements from the embedding space and maps them into a single element in that space. In our experiments we take $r_1 \oplus r_2 \oplus \ldots r_{N_T-1} \oplus r_{N_T}$ to be the mean operation $\frac{\sum_{i=1}^{N_C} r_i}{N_C}$.

Then given the embedding $E(D_C)$ of the context set D_C, CNPs model the predictive distribution as:

$$p_\theta(x^T_{1:N_T} | t^T_{1:N_T}, D_C) = \prod_i^{N_T} \mathcal{N}\left(x^T_i; \mu\left(t^T_i, E\left(D_C\right)\right), \sigma^2\left(t^T_i, E\left(D_C\right)\right)\right), \tag{3.9}$$

where the mean $\mu(\cdot, E(D_C))$ and variance $\sigma^2(\cdot, E(D_C))$ are parametrized by a MLP decoder $\rho_\theta = (\mu, \sigma^2)$. The decoder ρ_θ takes the representation of the context set $E(D_C)$ and the time point t^T_i and then predicts the parameters of the distribution over x^T_i at time location t^T_i.

The illustration of the CNP model is provided in Figure 3.1:

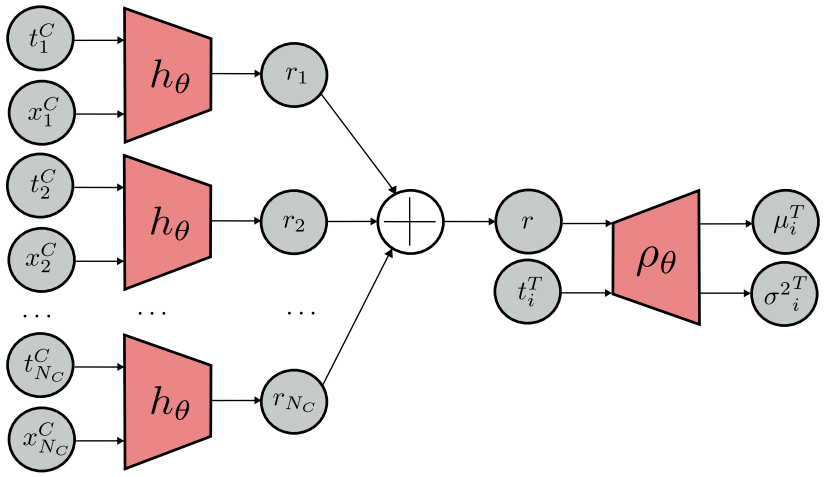


Fig. 3.1 Structure of the CNP model.

The model is trained via a standard maximum likelihood learning of the parameters θ:

$$\theta^* = \arg\max_{\theta\in\Theta} \sum_{\mathscr{D}\in\mathscr{S}} \sum_{(t,x)\in D_T} \log p_\theta(x|t, E(D_C)) \tag{3.10}$$

As compared to training procedures that use VI and amortization, maximum likelihood learning makes a model much easier to train. Moreover, since this learning technique mimics the usage of the system at test time, it leads to a strong performance of the model [27].

Neural Processes

A latent variable extension of Conditional Neural Processes, NPs, was proposed in [22]. The generative model for a NP is defined as follows:

$$p(z, x_{1:N}|t_{1:N}) = p(z)\prod_{i=1}^{N} \mathscr{N}(x_i; \mu_\theta(t_i, z), \sigma_\theta^2(t_i, z)), \tag{3.11}$$

where z is a latent variable with standard normal prior $p(z)$, $\mu_\theta, \sigma_\theta^2$ are MLPs that model conditional distribution $p_\theta(x|t,z)$. The illustration of the specified generative process is given in Figure 3.2:

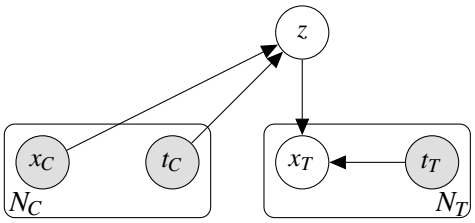


Fig. 3.2 Graphical model of NP. Observed variables are shaded in gray. C stands for context variables, whereas T for target variables. N_C and N_T indicate the number of variable in context and target sets respectively.

Since the posterior distribution $p(x^T_{1:N_T}|D_C, t^T_{1:N_T})$ is intractable due to the structure of the decoder $\rho_\theta = (\mu_\theta, \sigma^2_\theta)$, amortized VI is used to learn a model. Let $q(z|x_{1:N}, t_{1:N}) = q(z|D_C, D_T)$ be a variational posterior parameterized by a DeepSet [83] (Equations 3.7, 3.8).

Instead of using standard ELBO for $\log p(x_{1:N}|t_{1:N})$, the authors proposed to learn a model using approximate lower bound for $p(x^T_{1:N_T}|t^T_{1:N_T}, D_C)$:

$$\log p(x^T_{1:N_T}|t^T_{1:N_T}, D_C) \geq \mathbb{E}_{q(z|D_C,D_T}\left[\sum_{i=1}^{N_T} p(x^T_i|z, t^T_i) + \log \frac{q(z|D_C)}{q(z|D_C, D_T)}\right] \tag{3.12}$$

In contrast to CNPs, which are limited to factorized, parametric predictive distributions, NPs enable to specify more complex predictive distributions by employing latent variables. However, in general, NPs are harder to train in comparison to deterministic CNPs. Moreover, NP training utilizes VI and amortization, which are notorious for exhibiting certain flaws [13, 79].

Although NPs and CNPs have many appealing properties, the authors of [39] showed that NPs tend to underfit the context set D_C. In [39], the authors supposed that this underfitting may be attributed to the mean-aggregation step, since it restricts the ability of the decoder to differentiate which context points are more relevant for a particular target prediction. To tackle this problem, the authors of [39] proposed to incorporate attention mechanism [3, 40] into NP. This allows each input location to choose relevant context points for the prediction. However, it comes with an increased running time complexity of $\mathcal{O}(N_C(N_C + N_T))$.

Convolutional Conditional Neural Processes

In many domains, including modelling of irregularly-sampled time series, ideal solutions to prediction problems should be *translation equivariant* [12, 44]. In more detail, in the context of time series data, translation equivariance property means that if the data are translated in time, then the predictions should be translated accordingly. Hence, building

translation equivariance property into the modelling assumptions may serve as a valuable inductive bias. Unfortunately, standard NP models do not possess this property by default and hence must learn it from the training data. Undoubtedly, such a way of incorporating desired modelling assumptions not only is parameter inefficient but also influences the generalization capabilities of the model. The authors of [28] address this problem by introducing Convolutional Conditional Neural Processes (ConvCNPs), a member of NP family, which accounts for translation equivariance (TE) .

ConvCNP models the predictive distribution over target inputs as:

$$p_\theta(x^T_{1:N_T}|t^T_{1:N_T}, D_C) = \prod_{i=1}^{N_T} \mathcal{N}(x^T_i; \mu_\theta(t^T_i, D_C), \sigma^2_\theta(t^T_i, D_C)), \tag{3.13}$$

where μ_θ and σ^2_θ are parameterized by Convolutional Deep Sets (ConvDeepSets) [28]. Since for a fixed-dimensional vector space the notion of TE is not well-defined, for ConvCNP [28] the context set D_C is embedded into functional space, where TE is well-defined. So, ConvDeepSet [28] is defined to be a flexible parameterization for TE functions that map data set $\mathscr{D} \subseteq \mathscr{S}$ to the space of continuous, bounded functions $C_b(\mathscr{T})$.

ConvDeepSet [28] can be represented as a composition of two functions. Let $\Phi(D_C)$ denote a ConvDeepSet [28]. Then $\Phi(D_C) = \rho(E(D_C))$, where $E(D_C)$ is an encoder that embeds context set D_C into function space and ρ is a decoder that serves as a TE map between two function spaces. The encoder $E(D_C)$ has the following form:

$$E(D_C) = \sum_{i=1}^{N_C} \phi(x^T_i)\psi(\cdot - t^T_i), \tag{3.14}$$

where ψ is a positive-definite kernel associated with a Reproducing Kernel Hilbert Space (RKHS) [2]. Following [28], in our experiments, we set ψ to be the exponentiated-quadratic (EQ) kernel with a learnable length scale parameter. Function $\phi(x^T_i) = [1, x^T_i]^T$, where the first channel ϕ_1 is referred to as a "density channel". This channel provides model information about the locations of observed data points.

A decoder is parameterized by a CNN. However, CNNs cannot handle continuous input and are able to produce only discrete outputs. Hence, we have to discretize the input of the decoder, apply CNN to it and then convert the obtained output to a continuous function. In a more detail, we construct a uniform grid $t^G_{1:N_G}$ covering context and target time locations. The encoder $E(D_C)$ is evaluated at the specified time locations $t^G_{1:N_G}$ and then obtained discretized representation serves as input to the decoder ρ. After that, the output of the decoder is mapped to a continuous function $z(t)$ using the EQ kernel ψ_ρ. Finally, this latent function

$z(t)$ is used to obtain parameters of the predictive distribution at each target time location $t^T_{1:N_T}$. Figure 3.3 illustrates forward pass of the ConvCNP [28] model:

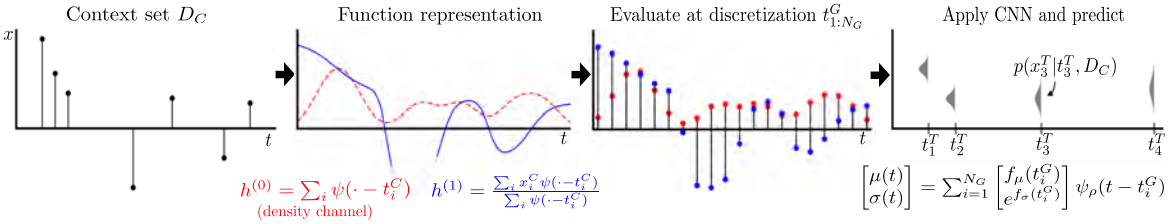


Fig. 3.3 Forward pass of the ConvCNP model.

The model is learnt via maximum-likelihood training (Equation 3.10) using stochastic gradient descent methods [6].

Convolutional Neural Processes

Convolutional Neural Process (ConvNP) [20] is an extension of the ConvCNP [28] that allows for modelling dependencies in the predictive distribution. The correspondence between ConvCNP [28] and ConvNP [20] is similar to the one between NP [22] and CNP [21]. However, while for NPs latent variable z is a fixed dimensional vector, in case of ConvNPs [20] z is a latent function. As in the case of ConvCNPs [28], the transition from finite-dimensional vector space to function space was necessary to incorporate the TE inductive bias into the model structure.

From the definition of ConvCNP [28] (Equation 3.13) it can be seen that ConvCNP [28] represents a map from data sets $D_C \subseteq \mathscr{S}$ to the space of Stochastic Processes (SPs) [68], in particular noise GPs. In more detail, noise GPs are processes with covariance defined as $Cov(t,t') = \sigma^2(t)\delta(t-t')$, where $\sigma^2(t)$ is some continuous, bounded function, $\delta(0) = 1$ and $\delta(\tau) = 0$ for $\tau \neq 0$. Such a mapping has two key limitations. Firstly, it is impossible to acquire coherent function samples from the noise GP process. Secondly, Gaussian distributions have a limited representation capability and cannot model multi-modal or asymmetric distributions. To address this issue the authors of [20] proposed to introduce an additional non-linear, TE map from noise GPs to a more expressive SP family. The composition of ConvCNP [28] and this additional mapping constitutes the ConvNP [20] model. Specifically, the ConvNP [20] model has an encoder-decoder architecture, with an encoder $E(D_C)$ that maps context set D_C to the space of noise GPs and a decoder D that provides a map between two function spaces. Encoder E is parameterized by a ConvCNP [28]. All in all, ConvNP [20] is defined as follows:

$$\text{ConvNP}_{\theta,\phi} = D_\theta \circ E_\phi, \text{with } E_\phi = \text{ConvCNP}_\phi \tag{3.15}$$

Sampling from ConvNP [20] is performed by firstly sampling a function $z \sim \text{ConvCNP}_\phi$ and then computing $f = D_\theta(z)$. The illustration of the process is provided in Figure 3.4.

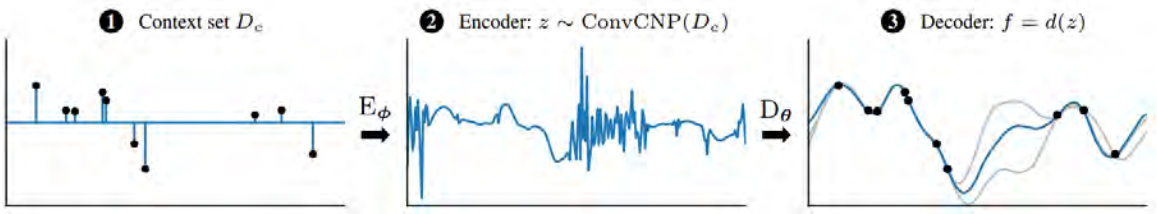


Fig. 3.4 Sampling from a trained ConvNP [20]. Context set D_C serves as input to the encoder (left figure), which outputs a single sample of z (center figure). Then the decoder takes this sample and outputs a predictive one (right figure, function coloured in blue). In the right figure other function samples are shown in gray. The illustration was borrowed from [20].

Since in practice we cannot compute samples from noise GPs, a discrete version of the model should be used. Similar to ConvCNP [28], the domain of z is discretized on a uniform grid $t^G_{1:N_G}$. After discretization latent function sampling boils down to sampling independent Gaussian random variables. Then the decoder D_θ is implemented as a CNN, which takes values of z at a uniform grid $t^G_{1:N_G}$ and outputs values of $D_\theta(z)$ at the same time locations.

The ConvNP [20] model likelihood is defined as follows:

$$p_{\theta,\phi}(x^T_{1:N_T} | t^T_{1:N_T}, D_C) = \mathbb{E}_{z \sim E_\phi(D_C)} \left[\prod_{i=1}^{N_T} \mathcal{N}(x^T_i; D_\theta(z), \sigma_{x^T_i}(t^T_i, z)) \right], \tag{3.16}$$

where $\sigma_{x^T_i}(t^T_i, z)$ is Gaussian observation noise.

Conceptually similar to NPs [22], ConvNPs [20] could be learned with NPs training objective (Equation 3.12). However, the KL term in the objective introduces some technical issues when training the ConvNP model [20]. To begin with, KL divergences between non-discretized SPs cannot be directly computed and should be handled carefully [15, 76]. What is more, for discretized version, KL term will vastly depend on the chosen discretization and there are no scientific results showing that KL divergence on discretized data converges to true KL divergence between SPs. To avoid this issues, the authors of [20] proposed to use Monte-Carlo estimate of $\log p_{\theta,\phi}(x^T_{1:N_T} | t^T_{1:N_T}, D_C)$ and perform maximum-likelihood training on $\mathscr{S}$. The training objective for each $\mathscr{D} \subseteq \mathscr{S}$ is defined as:

$$\mathscr{L}(\theta, \phi; \mathscr{D}) = \log \left[\frac{1}{L} \sum_{l=1}^{L} \exp \left(\sum_{i=1}^{N_T} \log p_\theta(x^T_i | t^T_i, z_l) \right) \right], z_l \sim E_\phi(D_C) \tag{3.17}$$

In their experiments, the authors of [20] demonstrate that this learning objective often performs much better than VI-inspired ones.

3.3 Autoregressive Modelling

Let us assume that we are given a dataset $\mathscr{D}$ consisting of evenly sampled, possibly multidimensional, time series $\boldsymbol{x}^p = \{x_i^p\}_{i=1}^{N_p}$, where $x_i^p \in \mathbb{R}^d$.

By the chain rule of probability, the joint distribution can be factorized over each time series sample as

$$p(\boldsymbol{x}) = p(x_1, \ldots, x_N) = p(x_1) \prod_{i=2}^{N} p(x_i | x_{i-1}, \ldots, x_1) = p(x_1) \prod_{i=2}^{N} p(x_i | \boldsymbol{x}_{<i}) \tag{3.18}$$

where $\boldsymbol{x} \in \mathscr{D}$ and $\boldsymbol{x}_{<i} = [x_1, x_2, \ldots, x_{i-1}]$ denotes a collection of random vectors with index less than i. Bayesian network illustrating the chain rule factorization is provided in Figure 3.5.

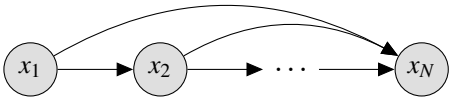


Fig. 3.5 Bayesian network illustrating the chain rule factorization (Equation 3.18).

Making no conditional independence assumptions, autoregressive models use observations from all previous time steps to predict the value at the current time step. One way to introduce such a model could be to specify every conditional $p(x_i|\boldsymbol{x}_{<i})$ in a tabular form. Although such a distribution representation has a general structure and allows to define any possible distribution over N random vectors, the space complexity for such a representation has an exponential growth with N.

To mitigate this issue, conditional probabilities $p(x_i|\boldsymbol{x}_{<i})$ could be parameterized by functions containing limited number of parameters. These models are usually referred to as *autoregressive generative models*. These models cannot represent all possible predictive distributions anymore, but with a flexible enough parameterization, they can serve as an adequate approximation to the true distribution [30, 46, 80].

Perhaps, one of the most widely used generative autoregressive models is RNN [70]. To deal with variable length sequences, RNNs use internal memory state that incorporates information about all observed data points. This internal memory state is updated each time

an observation comes: $h_t = f(x_{t-1}, h_{t-1}; \theta)$. Then given a current memory state h_t the RNN model is trained to predict conditional distribution $p(x_t|\boldsymbol{x}_{<t}) \approx p_\theta(x_t|h_t)$. The illustration of the process is given in Figure 3.6, where we also show how recurrent layer can be unrolled.

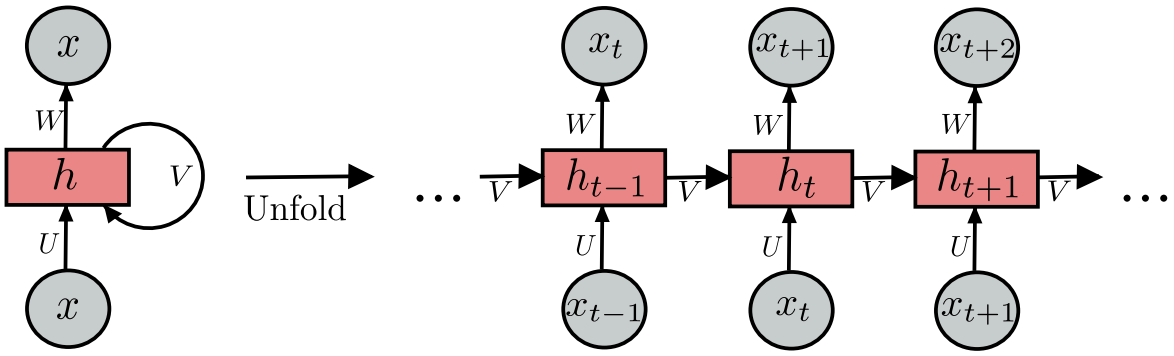


Fig. 3.6 RNN used as a generative model. Given current observation x_t RNN updates the memory state h_t and then predicts parameters of conditional distribution $p(x_{t+1}|\boldsymbol{x}_{<t})$. W, V, U are trainable parameters in the RNN model.

In vanilla RNN models f is usually modelled as a 1-layered MLP:

$$f(x_{t-1}, h_{t-1}; \theta) = \phi(W^{(1)} h_{t-1} + W^{(2)} x_{t-1} + b), \tag{3.19}$$

where $\theta = \{W^{(1)}, W^{(2)}, b\}$ are trainable parameters of the layer, $\phi(\cdot)$ is some activation function, e.g. Rectified Linear Unit (ReLU) , sigmoid or tanh.

However, vanilla RNN cell structure (Equation 3.19) is notoriously known for suffering from exploding and vanishing gradient problems in case of long sequences [25, 59]. To alleviate the issue of vanishing gradient, variations of Gated Recurrent Units (GRUs) were proposed [10, 29, 33]. They circumvent the vanishing gradient problem by introducing gating mechanisms that guide the information flow through time.

RNN-based autoregressive models are commonly trained by optimizing the maximum likelihood objective:

$$\theta^* = \arg\max_{\theta \in \Theta} \frac{1}{|\mathscr{D}|} \sum_{\boldsymbol{x} \in \mathscr{D}} p_\theta(\boldsymbol{x}) = \arg\max_{\theta \in \Theta} \frac{1}{|\mathscr{D}|} \sum_{\boldsymbol{x} \in \mathscr{D}} \sum_{i=1}^{|x|} p_\theta(x_i|\boldsymbol{x}_{<i}) \tag{3.20}$$

Despite being successful in numerous sequential learning tasks [29, 37], standard RNNs show unsatisfactory performance in modelling irregularly-sampled time-series data. A common ad-hoc approach to directly apply a standard RNN model to off-the-grid data is to divide the timeline into bins and then somehow aggregate or impute observations falling into the same bin. Undoubtedly, such data preprocessing eliminates a lot of information, in

particular the exact times of events, that can be very useful for data modelling [8, 50]. A natural way to tackle this problem is to introduce a model with a continuous-time evolution of the latent space. The first step in this direction was the RNN model, where the dynamics between observation is modelled as an exponential decay [7, 8, 55, 62]. A generalized version of such a model called ODE-RNN was proposed in [69]. ODE-RNN models the evolution of the latent state with a Neural ODE [9]. A detailed description of the ODE-RNN model is provided in Algorithm 3:

Algorithm 3: ODE-RNN

Input : Data points $\{x_i\}_{i=1}^N$ and timestamps $\{t_i\}_{i=1}^N$

Set h_0 and t_0.

for $i \leftarrow 1$ **to** N **do**

$h'_i = \text{ODESolve}(f_\theta, h_{i-1}, t_{i-1}, t_i)$

$h_i = \text{RNNCell}(h'_i, x_i)$

$\mu_i, \sigma_i^2 = \text{MLP}(h'_i)$

Output : $\{\mu_i, \sigma_i^2\}_{i=1}^N$ – parameters of the predictive distribution $p(x_i|\boldsymbol{x}_{<i})$

Since ODE-RNN model [69] was shown to adequately perform on irregularly-sampled data, we chose this model for our evaluation. The obtained results are described in chapter 5.

There are a lot of recent follow-up works in autoregressive modelling direction. For instance, the authors of [14] proposed a continuous-time version of GRU [10], whereas [47] introduce an autoregressive model that is build upon the Long Short-Term Memory (LSTM) unit [33]. Another work, similar to ODE-RNN, was introduced in [35]. In this work, the authors proposed an extension to a Neural ODE [9] to allow for modelling temporal point process with a piecewise-continuous latent trajectory.

To conclude, in this chapter, we discuss several existing approaches from an immense model space of models devised for handling off-the-grid data. These approaches were grouped in terms of the learning paradigm they follow. Latent ODE [9] represents models following unsupervised learning paradigm, whereas models from NP family constitute a good example of approaches adhering to meta-learning. In the domain of autoregressive modelling, which can be attributed to both unsupervised and meta-learning paradigms, the ODE-RNN [69] model was introduced.

As we can see, for the task of modelling off-the-grid data, there is a massive amount of works spanning several ML paradigms. On the one hand, it is very appealing to the practitioner since it opens a vast number of choices that vary in terms of computational complexity and memory costs. On the other hand, in current literature, approaches to modelling off-the-grid data are disorganised, so it is hard to deal with this massive amount

of publications. What is more, a lot of models introduced above have never been compared to each other, making the choice of the appropriate model even harder. Towards this end, in the next chapter, we propose a unifying framework, which will highlight the distinctions of the introduced models and make a choice easier. Finally, in chapter 5, we evaluate the performance of some of the models.

Chapter 4

Model Design Space: a Common Context for the Models

Recently, within ML community there has been increased attention to developing novel methods for modelling time-series data [9, 28, 38, 69]. A lot of considerably successful models in this area were proposed within different ML domains. However, some of these models, e.g. Neural ODEs [9], are often misinterpret in the community, which restricts their applicability in practice. What is more, since methods come from disjoint ML domains, for a practitioner it can be hard to comprehend the variability of methods that could be used. For these reasons, a rigorous text book description of this ML area would be instrumental in solving a lot of practical and theoretical issues.

In this chapter, we suggest a way to arrange a variety of neural models that were specifically designed to handle non-regularly sampled time series data. We also propose a series of schematics outlining model design space. These schematics could help identify major characteristics of each model.

What is more, we introduce and formulate new models, e.g. Neural ODE CNP and Neural CDE CNP, that were encountered naturally within the proposed schematics, but have been omitted in the recent literature.

In section 4.1, we propose a global model design space that organizes conceptually different approaches to modelling irregularly-sampled data. Further sections in the present chapter are devoted to a more detailed discussion of a particular approach of time series data modelling. In section 4.2, we specify the model design space for models belonging to Neural Process family. Finally, in section 4.3, we formulate schematic for non-amortized modelling approaches.

4.1 Global Model Space

In recent literature, as outlined in chapter 3, there are several notable tendencies in neural modelling of irregularly-sampled time-series data. Each of these trends introduces a distinctive approach to data modelling, and the majority of recent works in this direction can be attributed to one of these approaches. Thus, recognizing the global modelling approach to which a particular model belongs helps identify limitations as well as inductive biases of the model considered. To the best of our knowledge, there are five primary research directions to modelling non-regularly sampled data:

1. Autoregressive Models (section 3.3);
2. VAEs (section 3.1.1);
3. NP Family. Existing models belonging to NP family are described in section 3.2.1, whereas novel models introduced in this work are outlined in section 4.2;
4. Non-Amortized models: meta-learning models with non-amortized inference. A detailed description of the models is provided in section 4.3;
5. Semi-amortized models: models that use amortization to get the initial estimate for instance-specific parameters and then refine this estimate using iterative optimization.

In Figure 4.1, we introduce a schematic that distinguishes global approaches to modelling irregularly sampled-data as well as puts them into a common context:

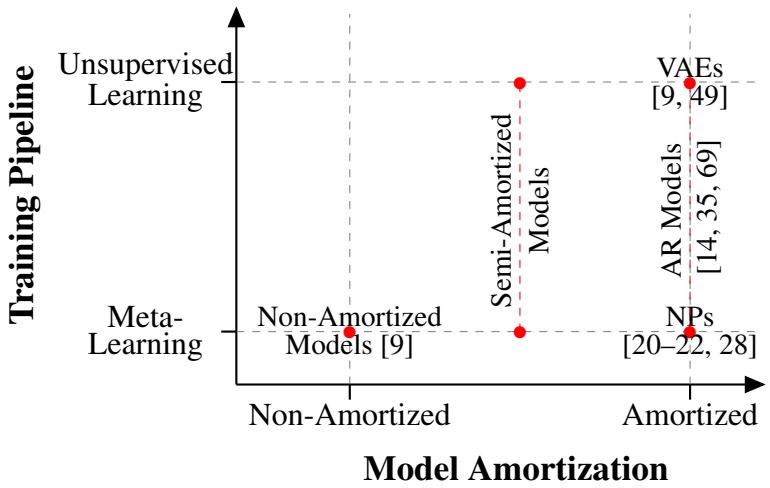


Fig. 4.1 Global design model space outlining principle approaches to modelling off-the-grid time-series data. X-axis indicates whether a model is amortized. Y-axis stands for the learning paradigm that determines the way a model is trained.

The proposed perspective allows the listed research directions to be organized in terms of 1) the choice of the learning paradigm in which a model is fitted, and 2) the level at which the inference in the model is amortized (from an instance-specific local inference to a global inference network).

Training Pipeline

Modelling time series data can be formulated in both meta-learning and unsupervised learning paradigms. In unsupervised learning setting, during training time all data points within each time series sample are assumed to be observed. Then the main task is to model the joint density $p(\boldsymbol{x}|\boldsymbol{t}) = p(x_1,\dots,x_N|t_1,\dots,t_N)$. A number of successful models obeying unsupervised learning paradigm have been proposed, e.g variational autoencoders [9, 49, 69] and autoregressive models [14, 35, 69]. On the contrary, training in meta-learning setting implies that there are two sets of data points, where one of them, context set $(\boldsymbol{x}^C,\boldsymbol{t}^C)$, is observed and the other one, target set $(\boldsymbol{x}^T,\boldsymbol{t}^T)$, is assumed to be missing. Then the task is to adequately approximate the conditional predictive distribution: $p(\boldsymbol{x}^T|\boldsymbol{x}^C,\boldsymbol{t}^T,\boldsymbol{t}^C)$. Non-amortized meta-learning models, autoregressive models [14, 35, 69] as well as NP family [20–22, 28] operate under the meta-learning setting.

Autoregressive models [14, 35, 69] could be attributed to both settings simultaneously. In general, autoregressive models approximate joint density $p(\boldsymbol{x}|\boldsymbol{t})$, which indicates that the unsupervised learning paradigm is followed. On the other hand, on each time step autoregressive models also approximate the conditional distribution $p(x_t|\boldsymbol{x}_{<t})$, which resembles the meta-learning setting with a varying context set.

Model Amortization

Model Amortization refers to the way model inference is performed. Non-amortized models use instance-specific local inference. To train those models SVI is commonly applied. On the contrary, in the case of amortized models, instance-specific parameters are predicted by an inference network (or encoder), which is shared across all dataset. The trade-off here is clear: SVI could give a good local estimate of instance-specific parameters, but requires performing optimization for each data point. Conversely, amortized models avoid per-sample optimization, thus significantly reducing time costs. However, these models impose a restriction on instance-specific parameters, as the parameters are now represented by some parametric function of the observed data. The authors of [13] has shown that such a restriction may be indeed too strict, causing a significant amortization gap for amortized models (underfitting), especially in the large data regime. To mitigate this issues, recent

works have introduced semi-amortized inference for meta-learning [71, 78] and unsupervised learning [32, 41, 45, 54] settings. However, to the best of our knowledge, none of the semi-amortized inference approaches was evaluated on the interpolation/extrapolation task for irregularly-sampled time series data.

4.2 Neural Processes Family

In this section, we propose a design model space to organise the variety of models belonging to the NP family. In this model space, we identify three core dimensions that can be used to compare and contrast existing modelling approaches. The dimensions are as follows:

1. *Type of the decoder used*. Convolution-based and Neural ODE [9]-based decoders can both be used to handle non-uniformly sampled data efficiently. Being conceptually different, these two models imply quite different inductive biases, computational requirements and constraints. Therefore, the type of the decoder forms one of the major discriminating dimensions in the design space;

2. *Complexity of the predictive distribution*. This dimension incorporates the choice between deterministic and latent variable models. Deterministic models are limited to modelling factorized predictive distribution. Consequently, such a model can be used neither to produce coherent predictive samples nor to model more complicated joint distributions. Hence, to enable more expressive predictive distributions, latent variable models could be used. However, the training procedure for latent variable models could be more demanding in terms of computational resources [20] in comparison to deterministic counterpart. Moreover, amortized variational inference, that is commonly used for training latent variable models [22], suffers from certain drawbacks, influencing the quality of a final model [13].

3. *Dimension of the embedding space*. A key component in the NP family is the embedding of data sets into some representation space. The possible choices for embedding space span both finite-dimensional vector as well as infinite-dimensional function spaces. For finite-dimensional embeddings DeepSet function approximation [83] is used, whereas ConvDeepSet [28] embeds data sets in an infinite-dimensional function space. Although DeepSet framework provides a more compact data sets representation, ConvDeepSet allows for modelling TE in the data, which is an important inductive bias for time series modelling[20, 28].

A visualization of the defined model space for NP family is shown in Figure 4.2. On the schematics, each vertex of the cube is attributed to a particular model. The name of the model associated with a specific vertex is indicated near this vertex.

All models mentioned in Figure 4.2 belong to NP family. Hence, they exhibit a common general structure and are defined as a composition of an encoder and a decoder. The encoder outputs some latent representation of a data set, whereas the decoder takes this latent representation as input and outputs parameters of the predictive distribution. However, the structure of the encoder and the decoder changes dramatically for NP models with different specifications.

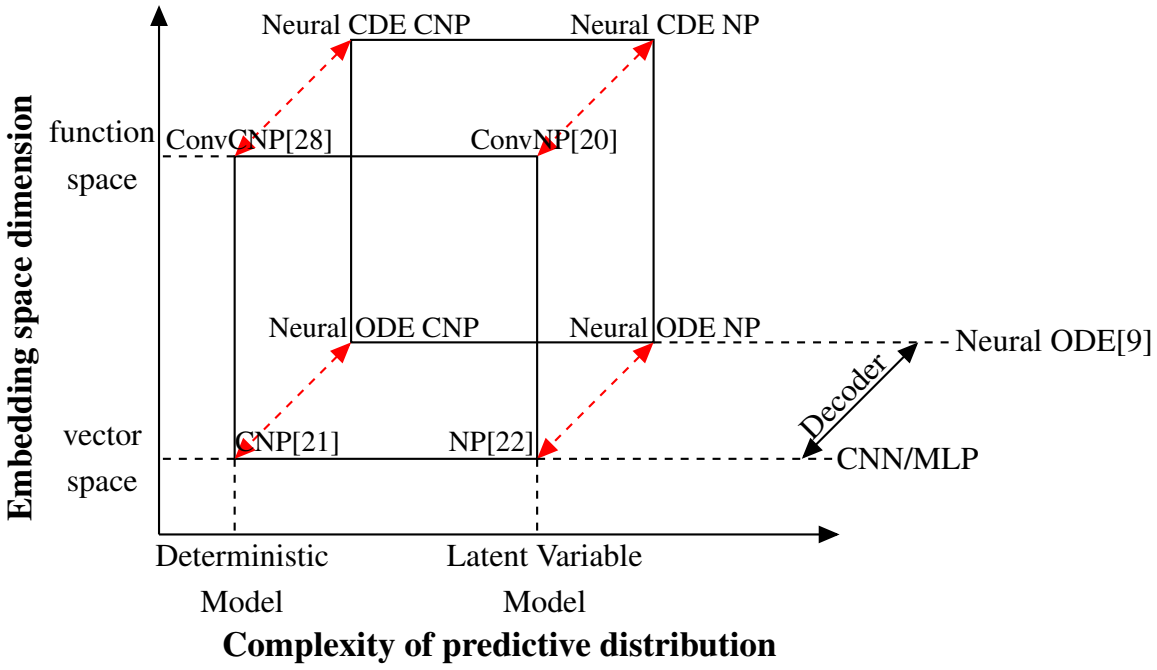


Fig. 4.2 Design model space for models from NP family. Each axis states for one of the dimensions defining the design space. Each vertex of the cube identifies a specific configuration of the NP model. Near each vertex, the name of the NP model complying with the specified configurations is provided.

Further, we will formulate and discuss variations of neural processes, which use Neural ODE model [9] as a decoder. The namings of these models (Neural ODE CNP, Neural ODE NP, Neural CDE CNP and Neural CDE NP) are given to comply with their convolution-based counterpart. The specifications of these models could be understood from the design model space illustrated in Figure 4.2.

We will omit the discussion of existing models (CNP [21], NP [22], ConvCNP [28] and ConvNP [20]) in this chapter, as it can be found in section 3.2.1.

Notation

The notation proposed in this section assumes that models are trained on time series data. Let $\mathscr{X} = \mathbb{R}^{out}$ denote the output space. Let (t,x) be an input-output pair, meaning the value x was observed at time t. Let $\mathscr{S}$ be the collection of all time series samples, with D_C, D_T a context and target sets respectively. Let $D_T = (t^T_{1:N_T}, x^T_{1:N_T}), D_C = (t^C_{1:N_C}, x^C_{1:N_C})$, where N_T and N_C are the number of points in target and context sets accordingly. A single task is denoted as $\xi = (D_C, D_T)$.

Neural ODE CNP

Model Formulation. Neural ODE CNP assumes that given context set D_C, predictive distribution over target outputs is conditionally independent and can be modelled as:

$$p(x^T_{1:N_T} | t^T_{1:N_T}, D_C) = \prod_{i=1}^{N_T} \mathscr{N}(x^T_i | \mu_\theta(z_i), \sigma^2_\theta(z_i)), \tag{4.1}$$

where $\mu_\theta(\cdot), \sigma^2_\theta(\cdot)$ are shared, pointwise MLPs that map z at each $t^T_{1:N_T}$ in the target set to $\mathbb{R}^{out}$. Similar to CNP [21], latent function z is defined as a composition $\rho \circ E$, where E is an encoder that maps D_C to embedding space $\mathbb{R}^e$ and $\rho : R^e \to \mathscr{H}$ is a decoder that maps representation of D_C into an appropriate function space $\mathscr{H}$. In case of Neural ODE CNP, ρ is parameterized as Neural ODE [9], i.e. $z_{1:N_T} = ODESolve(f_\theta, h_0, t^T_0, t^T_{1:N_T})$, where $h_0 = z_{t^T_0}$ is the initial condition of the ODE. Since the solution of an ODE is fully determined by its initial condition, all information about context set D_C should be incorporated into initial condition vector. Hence, an encoder should provide a value for the initial condition: $h_0 = E(D_C)$. All in all, the model can be formulated as:

$$h_0 = E(D_C) \tag{4.2}$$

$$z_{1:N_T} = \text{ODESolve}(f_\theta, h_0, t^T_0, t^T_{1:N_T}) \tag{4.3}$$

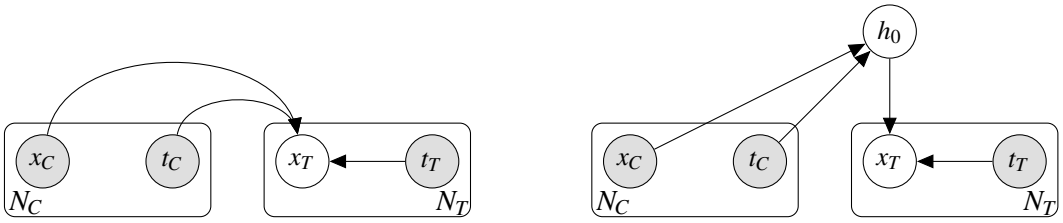
where f_θ is a function that specifies the dynamics of the hidden state, using a neural network with parameters θ. In our experiments, we set t^T_0 to be the time of the earliest observation in the whole training data.

As for the architecture of the encoder, any appropriate model, such as RNN, ODE-RNN [69] or DeepSet [83], could be used.

Training. Training procedure for the Neural ODE CNP model follows standard protocol for training CNPs [21], where the negative conditional log-likelihood of target sets is minimized (Equation 3.10) using stochastic gradient descent methods [6].

Neural ODE NP

Model Formulation. To enable richer joint predictive distributions, a latent variable extension of Neural ODE CNP could be introduced. We refer to this extension as Neural ODE NP model. For Neural ODE NP, we assume that the initial condition vector is a global latent variable with multivariate standard normal prior, rather than a deterministic vector as in Neural ODE CNP. Graphical models for both Neural ODE CNP and Neural ODE NP are illustrated in Figure 4.3:



(a) Neural ODE CNP (b) Neural ODE NP

Fig. 4.3 Graphical models for Neural ODE CNP (4.3a) and Neural ODE NP (4.3b). Observed variables are shaded in gray. C stands for context variables, whereas T for target variables. N_C and N_T indicate the number of variable in context and target sets respectively.

The Neural ODE NP model can be formulated as follows:

$$h_0 \sim p(h_0|D_C) \tag{4.4}$$

$$z_{1:N_T} = \text{ODESolve}(f_\theta, h_0, t_0^T, t_{1:N_T}^T) \tag{4.5}$$

$$x_i^T \sim \mathcal{N}(x_i^T|\mu_\theta(z_i), \sigma_\theta^2(z_i)), i = 1,\ldots,N_T \tag{4.6}$$

Since the Neural ODE decoder is non-linear, we will use amortized VI to learn the model. Let $q_\theta(z|D_C, D_T)$ be a variational posterior of the latent variable z, parameterized by an encoder. The architecture of the encoder is similar to the one used for the Neural ODE CNP model. Moreover, as the posterior distribution $p(h_0|D_C)$ is intractable, following [22], we will approximate it with the variational posterior $q_\theta(h_0|D_C)$ predicted by the encoder.

Training. The model parameters are optimized using the approximate ELBO for $\log p(x_{1:N_T}|t_{1:N_T}, D_C)$ (Equation 3.12), as proposed in [22].

Model Similarity. It is also worth noticing that if we consider all data to be observed (without splitting it into target and context sets), Neural ODE NP model would be converted to a standard VAE, namely the Latent ODE model [9, 69].

Neural CDE CNP

Neural ODE CNP and Neural ODE NP models embed context dataset D_C into a vector embedding space. Thus, all information about the observed data should be compressed into a fixed dimensional representation. However, in some cases, especially in a large data regime, the chosen dimensionality of the embedding space could be insufficient to incorporate all the information needed for the adequate performance of the model. A simple solution to this problem would be just to increase the dimension of the embedding space. However, such an approach may cause some difficulties with stochastic optimization convergence. Moreover, it comes with computational and memory costs.

Another solution to the problem of underfitting could be to embed observed data into infinite-dimensional function space, as proposed for ConvCNP [28] and ConvNP [20]. However, the Neural ODE [9] decoder used for the Neural ODE CNP and Neural ODE NP models is not compatible with function embedding space, since the latent function z produced be Neural ODE [9] decoder is fully determined by a fixed-dimensional initial state vector h_0 and time t_0. This issue can be rectified by the use of the Neural Controlled Differential Equation (Neural CDE) model proposed in [38]. The model is built upon the definition of the Controlled Differential Equation (CDE), which is driven by some continuous function of bounded variation $X : [\tau, T] \to \mathbb{R}^v$:

$$z_t = z_\tau + \int_\tau^t f(z_s) dX(s), \text{ for } t \in (\tau, T] \tag{4.7}$$

$$z_\tau = \gamma \tag{4.8}$$

where the integral is a Riemann–Stieltjes integral, $z_\tau = \gamma$ is an initial hidden state vector at time τ. Function $f : \mathbb{R}^w \to \mathbb{R}^{w \times v}$ is a continuous function that describes the dynamics of the hidden state. CDE defined in (4.7 – 4.8) exists and is unique under global Lipschitz conditions on function f [51].

By introducing parameterization of f as a neural network and taking function X to be the natural cubic spline [56] with knots at observations' times the authors of [38] defined the Neural CDE model. As a result, Neural CDEs [38] allow for adjusting the trajectory based on local observations and hence could be used in models where encoder embeds data into function space.

Model Formulation. Based on the results from [38], we will now formulate the Neural CDE CNP model. This model has a structure similar to ConvCNP [28], but the parameterization of the decoder is different. Opposed to ConvCNP [28], Neural CDE CNP model parameterizes decoder with Neural CDE:

$$z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z_s) dE(D_c)(s) \tag{4.9}$$

$$z_{t_0} = \gamma_\theta(E(D_c)(t_0)) \tag{4.10}$$

where γ_θ, f_θ are neural networks that determine initial hidden state and dynamics of the latent function z respectively. $E(\cdot)$ is the encoder that maps context set D_c into functional representation. The encoder in Neural CDE model is taken to be the same as in ConvCNP [28]. A detailed structure of the encoder is specified in section 3.2.1.

Since $E(D_c)(\cdot)$ is differentiable, Equation 4.9 can be rewritten as:

$$z_t = z_{t_0} + \int_{t_0}^{t} f_\theta(z_s) \frac{dE(D_c)}{ds}(s) ds = z_{t_0} + \int_{t_0}^{t} g_{\theta, E(D_c)}(z_s, s) ds \tag{4.11}$$

Hence, Neural CDE could be solved using the same techniques that are used for solving Neural ODEs [9].

The illustration of the Neural CDE CNP forward pass for irregularly-sampled time series data is provided in Figure 4.4:

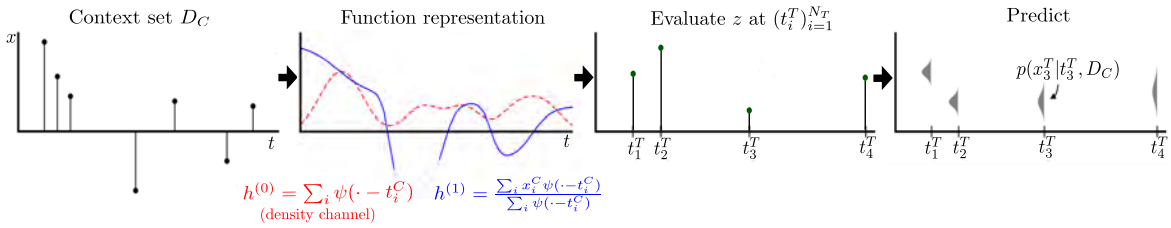


Fig. 4.4 Illustration of the Neural CDE CNP forward pass for irregularly-sampled time series.

It worth noticing that if z_{t_0} doesn't directly depend on t_0, but only on $E(D_c)(t_0)$, the Neural CDE CNP model would be TE.

As in all CNPs [21], Neural ODE CNP assumes that given context set D_C, predictive distribution over target outputs is conditionally independent and can be modelled as:

$$p(x_{1:N_T}^T | t_{1:N_T}^T, D_C) = \prod_{i=1}^{N_T} \mathcal{N}(x_i^T | \mu_\theta(z_i), \sigma_\theta^2(z_i)), \tag{4.12}$$

where $\mu_\theta(\cdot), \sigma^2_\theta(\cdot)$ are shared, pointwise MLPs that map z at each $t^T_{1:N_T}$ in the target set to $\mathbb{R}^{out}$.

Training. The model is trained via optimization of a standard maximum-likelihood objective (Equation 3.10) using stochastic gradient descent methods [6].

Model Similarity. Assume that $f_\theta(z_s) = W \in \mathbb{R}^{w\times(out+1)}$ is independent of the value of z_s. Then

$$z_t = z_{t_0} + \int_{t_0}^{t} W \frac{dE(D_c)}{ds}(s)ds = z_{t_0} + W(E(D_c)(t) - E(D_c)(t_0)) \tag{4.13}$$

If we now assume that $\gamma_\theta(E(D_c)(t_0)) = WE(D_c)(t_0)$, then $z_t = WE(D_c)(t)$. Hence, under introduced assumptions Neural CDE CNP model can be viewed as a ConvDeepSet [28] parameterization with a degenerate decoder. The decoder, in this case, is a one-layered MLP network, which is a degenerate case of a CNN – a one-layered CNN with kernel size 1. The discretization points are then assumed to be target time points themselves since there are no convolutions over time in the decoder and hence the uniform discretization grid is not required.

Neural CDE NP

A step further could be taken to introduce a counterpart of the ConvNP [20] model that is built upon Neural CDE CNP. In contrast to Neural CDE CNP model, Neural CDE NP would allow for dependencies in the predictive distribution. Thus, it could be used in applications where the generation of coherent samples is required. However, an accurate formulation of this model requires further investigation and is beyond the scope of this work.

4.3 Non-Amortized Models

In this section, we propose a schematic to organize the design model space for non-amortized models that can be applied to the task of interpolation/extrapolation of irregularly-sampled time series data. In this work, by the term *non-amortized models* we mean models that employ local instance-specific optimization from a fixed initial parameter value.

In contrast to amortized models, non-amortized ones have instance-specific parameters that are optimized during training. Such models are commonly learnt using SVI [34, 63], detailed in Algorithm 1.

On the one hand, this type of inference employs more flexible parameterization in comparison to amortized one. Hence, it does not suffer from severe underfitting caused by an amortization gap, as opposed to amortized models [13].

On the other hand, SVI requires to perform optimization for each data point individually. Hence, it can be difficult and time-consuming to apply a trained model to new test data sample, because optimization may take a large number of steps to convergence. Moreover, SVI does not provide any reasonable strategy to initialize local parameters for a new data point. The parameters are just randomly initialized. Since the optimized objective is a highly non-convex function, a bad initialization may lead to convergence to a spurious local minimum. Hence, the performance of the trained model vastly depends on the initial value of the instance-specific parameters. This issue may be circumvented by the introduction of MAML [19]-like training, where instance-specific parameters would have constant initialization which is learnt during training. The details of the training pipeline are provided further in the present section.

The illustration of the proposed design model space for non-amortized models is given in Figure 4.5:

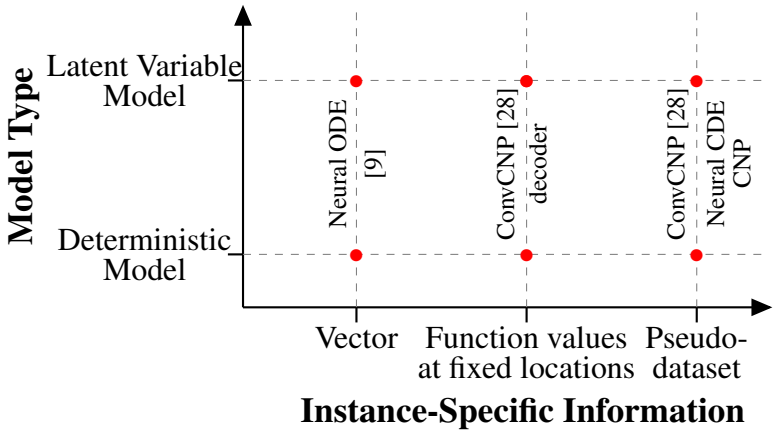


Fig. 4.5 Schematic outlining the design model space for non-amortized models that can be used for interpolation/extrapolation of irregularly-sampled time series data. X-axis stands for the type of local instance-specific information. Y-axis indicates whether instance-specific information is presented as a deterministic variable or a probabilistic one, with some prior distribution imposed. On each vertical dashed line connecting deterministic and probabilistic counterparts a suitable model architecture is indicated.

The proposed schematic introduces two dimensions that can be used to compare and contrast non-amortized models. The first one determines whether instance-specific parameters in the model are deterministic or probabilistic. The second one indicates how instance-specific parameters are represented, e.g. as a vector or as a pseudo-dataset.

Model Type

This dimension differentiates latent variable and deterministic models. For deterministic models, instance-specific parameters are assumed to be non-random and are directly optimized using the maximum-likelihood objective. In contrast, latent variable models consider that observed data were generated from a set of unobserved latent variables. Since we use neural networks for data modelling, the posterior distribution over latent variables is intractable and VI [36] is commonly used to approximate an intractable posterior.

Instance-Specific Information

Based on the model architecture that is used for data modelling, instance-specific parameters can be represented as various data structures:

1. Vector representation. If data are modelled with a Neural ODE [9], then instance-specific parameters are either the initial condition vector itself (deterministic) or parameters of variational distribution over this vector (probabilistic). Such a model is formulated as follows:

$$z_0 = h_0, \text{ if deterministic} \tag{4.14}$$

$$z_0 \sim p(z_0), \text{ if probabilistic} \tag{4.15}$$

$$z_{1:N} = \text{ODESolve}(f_\theta, z_0, t_0, t_{1:N}) \tag{4.16}$$

   where $t_{1:N} = t^T_{1:N_T} \cup t^C_{1:N_C}$ and $N = N_T + N_C$. Since the dynamics of the trained Neural ODE model [9] is entirely determined by the initial condition vector, we assume that the necessary information about a data sample is compressed into this vector.

2. Function values at fixed locations. Another possible way to construct a non-amortized model is to model data using the decoder part of the ConvCNP [28] model. To do this, we first need to fix discretization times $(t^d_i)_{i=1}^{N_d}$. Then function values $(x_i)_{i=1}^{N_d}$ at the chosen time locations would serve as instance-specific parameters and hence optimized separately for each data sample. The formal definition of the model is the following:

$$x_{1:N_d} \sim p(x_{1:N_d}), \text{ if probabilistic} \tag{4.17}$$

$$z^d_{1:N_d} = \text{CNNdecoder}_\theta(x_{1:N_d}) \tag{4.18}$$

$$z_i = \sum_{j=1}^{N_d} z^d_j \psi_\rho(t_i - t^d_j), i = 1, \ldots, N \tag{4.19}$$

where ψ_ρ is the EQ kernel with a learnable length scale parameter ρ.

It is worth mentioning that since this model has a large number of instance-specific parameters, it would work only in the case of abundant training data.

3. Pseudo-dataset. In this case, we assume that an encoder-decoder architecture, such as ConvCNP [28] or Neural CDE CNP, serves as a decoder for a non-amortized model. Then, similar to FITC [75] approach for Gaussian Processes, for each time series sample we introduce pseudo-dataset $(t^{pseudo}_{1:N_p}, x^{pseudo}_{1:N_p})$ that is optimized along with the parameters of the decoder. The formulation of the model is the following:

$$x^{pseudo}_{1:N_p} \sim p(x^{pseudo}_{1:N_p}), \text{ if probabilistic} \tag{4.20}$$

$$z_{1:N} = \text{decoder}_\theta(t_{1:N}, t^{pseudo}_{1:N_p}, x^{pseudo}_{1:N_p}) \tag{4.21}$$

In all non-amortized models introduced above the predictive distribution is assumed to be modelled as:

$$p(x_{1:N}|z_{1:N}) = \prod_{i=1}^{N} \mathcal{N}(x_i|\mu_\theta(z_i), \sigma^2_\theta(z_i)), \tag{4.22}$$

where $\mu_\theta(\cdot), \sigma^2_\theta(\cdot)$ are shared MLPs that map latent function z at some time location t to the output space $\mathbb{R}^{out}$.

Model Training

To learn a non-amortized model, a MAML [19]-like training pipeline could be employed. MAML [19] training aim to learn a model that can be easily adapted to model new samples within a small number of gradient steps. Global and instance-specific parameters of the model are explicitly trained to make the model adapt to new sample within several gradient steps w.r.t. instance-specific parameters. In other words, MAML [19]-like training makes the model to relatively easy to fine-tune. Moreover, in contrast to traditional SVI approaches, this training framework proposes a sensible way of initializing local parameters for new data, i.e. instance-specific parameters have shared constant initialization, which is learnt during training. The MAML [19]-like training algorithm is described in Algorithm 4.

We will further use notation introduced in Algorithm 4. For deterministic models, objective function $\mathscr{L}(\lambda, \theta, D_C)$ is assumed to be the log-likelihood of D_C, whereas for latent variable models $\mathscr{L}(\lambda, \theta, D_C)$ is the ELBO objective. From Algorithm 4, we can see that for training we need to compute total derivative of the final objective function w.r.t. model parameters θ, λ_0. To compute this total derivative we need to backpropagate through the

gradient ascent [41, 53]. This backpropagation step can be done efficiently with Hessian-vector products [48]. Assume that in Algorithm 4 we perform one fine-tuning step ($K = 1$) and $\lambda_1 = \lambda_0 + \alpha \nabla_{\lambda_0} \mathscr{L}(\lambda_0, \theta, D_C)$. To backpropagate through this, we use the chain rule:

$$\begin{aligned} \frac{d\mathscr{L}(\lambda_1, \theta, D_C \cup D_T)}{d\lambda_0} = \frac{d\lambda_1}{d\lambda_0} \frac{d\mathscr{L}}{d\lambda_1} = \\ = \frac{d\mathscr{L}(\lambda_1, \theta, D_C \cup D_T)}{d\lambda_1} + \alpha H_{\lambda_0, \lambda_0} \mathscr{L}(\lambda_0, \theta, D_C) \frac{d\mathscr{L}(\lambda_1, \theta, D_C \cup D_T)}{d\lambda_1} \end{aligned} \tag{4.23}$$

The computation of $\frac{d\mathscr{L}(\lambda_K, \theta, D_C \cup D_T)}{d\theta}$ can be done in a similar manner. The computation of Hessian-vector products (Equation 4.23) can be done via automatic differentiation, which is supported by standard deep learning libraries.

Algorithm 4: MAML [19] training for non-amortized models

Input : Collection of time series samples (tasks) $\xi^i = (D_C, D_T)$, where $\xi^i \in \mathscr{S}$

Let λ_0 be the initialization of instance-specific parameters.

Let θ indicate global parameters of the model.

Let $\mathscr{L}$ be a training objective.

Initialize parameters θ, λ_0.

repeat

  Sample task $\xi = (D_C, D_T)$ from $\mathscr{S}$

  Compute adapted initialization with gradient descent:

  for $k = 1, \dots K$ **do**

    $\lambda_k = \lambda_{k-1} + \alpha \nabla_{\lambda_{k-1}} \mathscr{L}(\lambda_{k-1}, \theta, D_C)$

  Update θ based on $\frac{d\mathscr{L}(\lambda_K, \theta, D_C \cup D_T)}{d\theta}$

  Update λ_0 based on $\frac{d\mathscr{L}(\lambda_K, \theta, D_C \cup D_T)}{d\lambda_0}$

until *convergence*;

Output : (λ_0, θ)

For a more flexible initialization of instance-specific parameters, semi-amortized models [32, 41, 45, 54, 71, 78] could be introduced. Instead of having one shared initialization value, semi-amortized models use amortization to get the initial estimate of instance-specific parameters for each sample individually. Then, they refine this estimate using iterative optimization, similar to Algorithm 4.

To summarise, in the present chapter, we investigate the model design space of neural approaches for modelling off-the-grid time series data. We propose a series of schematics as well as outline crucial modelling dimensions that help to organise a huge model space of existing models [9, 20–22, 28, 35, 38, 69].

Moreover, we formulate new models that were encountered in the context of the proposed schematics. The proposed models, together with the existing ones, could be used to model irregularly-sampled data. For instance, Neural CDE CNP and Neural ODE CNP, introduced in this work, operate within the meta-learning paradigm and can be efficiently applied to solve various few-shot learning tasks, including personalized medical data prediction.

Chapter 5

Implementation, Experiments, and Results

In this chapter, we provide an extensive evaluation of some models from the design model space introduced in chapter 4. Implementation details are provided separately for each experiment. All models were implemented in Pytorch [60] and are compatible with GPU training.

In section 5.1, we compare the performance of three models, namely ConvCNP [28], ODE-RNN [69] and Neural ODE CNP, on synthetic one-dimensional data. In section 5.2, we investigate the choice of encoder for the Neural ODE CNP model as well as the amortization gap problem in this model. Finally, in section 5.3, we evaluate ConvCNP [28] and ODE-RNN [69] on real-world medical data.

5.1 Synthetic 1D Data

In this section we evaluate some of the models from a huge design model space, introduced in chapter 4. Among all models we chose to evaluate ConvCNP [28], ODE-RNN [69] and Neural ODE CNP (section 4.2), since they represent different approaches to how the context set D_C is incorporated to obtain final estimates for the target set D_T. As mentioned earlier, ConvCNP [28] and Neural ODE CNP are encoder-decoder models that embed observed data into functional and vector representations respectively, while ODE-RNN is an autoregressive model that approximates conditional distribution $p(x_i|\boldsymbol{x}_{<i}, \boldsymbol{t}_{<i})$.

5.1.1 Data Generation

In this section, we consider synthetic regression problems. We chose to generate synthetic data from a GP [64] with different kernels as well as from a non-Gaussian challenging sawtooth process. The kernels used for data generation from GPs [64] are as follows:

- **EQ**:

$$k(x,x') = e^{-\frac{1}{2}\left(\frac{x-x'}{0.25}\right)^2} \tag{5.1}$$

- **Matern-$\frac{5}{2}$**:

$$k(x,x') = (1-4\sqrt{5}d+\frac{5}{3}d^2)e^{-\sqrt{5}d} \tag{5.2}$$

  where $d = 4|x-x'|$

- **Noisy mixture**:

$$k(x,x') = e^{-\frac{1}{2}(x-x')^2} + e^{\frac{1}{2}\left(\frac{x-x'}{0.25}\right)^2} + 0.001\delta(x-x') \tag{5.3}$$

- **Weakly periodic**:

$$k(x,x') = e^{-\frac{1}{2}(f_1(x)-f_1(x'))^2 - \frac{1}{2}(f_2(x)-f_2(x'))^2} \cdot e^{-\frac{1}{8}(x-x')^2} \tag{5.4}$$

  where $f_1(x) = \cos(8\pi x), f_2(x) = \sin(8\pi x)$

- **Ornstein-Uhlenbeck (Gauss-Markov)**:

$$k(x,x') = \frac{b_0^2}{2a_0} e^{-a_0|x-x'|} \tag{5.5}$$

  with $a_0 = 1, b_0 = 0.5$ in our experiments.

Non-Gaussian random sawtooth samples are generated from the following function:

$$y_{sawtooth} = \frac{A}{2} - \frac{A}{\pi}\sum_{k=1}^{\infty}(-1)^k \frac{\sin(2\pi k f(t-s))}{k}, \tag{5.6}$$

where we took amplitude $A = 1$, for each sample frequency f was sampled uniformly in $[3,5]$, random shift s in $[-5,5]$. We also truncate the series at an integer K that is uniformly sampled from $[10,20]$.

As for data generation, the number of context and target points per time series is uniformly sampled between 3 and 50 for GPs and between 3 and 100 for the sawtooth process. As ODE-RNN doesn't assume splitting data into context and target sets, the number of observed points per sample is chosen to be comparable with the sum of target and source points, i.e it is randomly sampled between 6 and 100 for GPs and between 6 and 200 for the sawtooth one. Moreover, due to the structure of the models, the data generation process is slightly different for ConvCNP [28] and Neural ODE [9]-based models. In case of ConvCNP [9], the number of points in the context and target sets is equal for all samples in a training batch. After randomly choosing the number of points in each set, this number of context and target points are randomly sampled on the interval $[-2,2]$ from a function sampled from a Gaussian or sawtooth process. In contrast, for the Neural ODE [9]-based models for each training batch we firstly uniformly sampled 256 locations from an interval $[-2;2]$. This locations constitute a set of points where the function values can be observed. This step is necessary because of the way Neural ODE [9] operates, i.e. to evaluate a training batch we have to output the solution of the ODE at the union of all time points in the batch. For this reason, in order to limit the union size, for each batch we sample 256 possible points location. After selecting the set of possible locations, for each sample the number of context and target points is chosen. Then given the number of context and targets points the location for each point is sampled from the set of possible locations. Finally, we sample a function from a Gaussian or sawtooth process and get the values of this at the locations of context and target points.

Finally, in a similar manner, we generate 1000 test tasks on which all models were evaluated.

5.1.2 Model Architectures and Training Details

In this section, we describe in detail the architectures of models used in this experiment as well as give some training details, such as the duration of the training, type of optimizer and learning rates used.

ConvCNP [28]

For all models, we assume that input kernel ψ (section 3.2.1) is an EQ kernel with a learnable length scale parameter. The same assumption holds for the final output layer kernel ψ_ρ. The length scales for the EQ kernels ψ, ψ_ρ are initialized to twice the space between two neighbouring discretization points. For numerical stability, when dividing by the density channel, we add $\varepsilon = 10^{-8}$. The density of discretization is 64 points per unit.

The discretization spans an interval from $\min(t) - 1$ to $\max(t) + 1$, where $\min(t)$ stands for earliest observation time occurring in the union of the context and target sets in the current batch and $\max(t)$ is the latest observation time respectively.

Since the receptive field has a direct influence on the performance of the model, we need to choose considerably wide CNN filters. However, using full 2D convolutions with wide filters would require a lot of memory, which is undesirable. To alleviate this issue, in our experiments we employ depthwise-separable convolutions [11], which help to significantly reduce the number of parameters in CNN decoder while keeping width of CNN filters fixed. What is more, the actual size of the receptive field is a product of the spacing between the discretization points and the width of the CNN filters. This implies that for a fixed receptive field if we increase the density of discretization, we have to increase the width of CNN filters. Hence, the usage of depthwise-separable filters also allows to increase discretization density without reducing receptive field.

We use a 8-layer (excluding an initial and final point-wise linear layers) CNN with 64 channels and depthwise-separable convolutions as a decoder in ConvCNP [28]. The width of the filters depends on the complexity of the data and is chosen such that the receptive filed sized are the following:

- EQ: 2;
- Matern-$\frac{5}{2}$: 2
- Noisy mixture: 4;
- Weakly periodic: 4;
- Gauss-Markov: 2;
- Sawtooth: 16;

The standard deviation is parameterized in the model by passing the output of the decoder through a softplus function. We used ReLU non-linearities in all models.

The number of trainable parameters as well as the size of CNN filter in the ConvCNP model depending on the dataset are given in Table 5.1.

Dataset	Number of trainable parameters	CNN filter width
EQ	42948	17
Matern-$\frac{5}{2}$	42948	17
Noisy mixture	51140	33
Weakly periodic	51140	33
Gauss-Markov	42948	17
Sawtooth	100292	129

Table 5.1 The number of trainable parameters and the size of CNN filters in the ConvCNP [28] model depending on the 1D dataset.

All models were trained using Adam [42] optimizer with learning rate of 5e-4. We also used a weight decay of 10^{-5} applied to all model parameters.

ODE-RNN [69]

In our experiments, we use GRU type of the RNN cell. For all datasets we used a 1-layered MLP with 100 units to model the dynamics of an ODE. The dimension of the hidden function was set to 10. We apply tanh non-linearities, as suggested by the authors of [69]. Tanh non-linearity constraints the output of a neural network and preserves the Lipschitz continuity for an ODE. If a neural network could output huge values for the ODE gradient, such an ODE would be hard to solve with numerical methods, as it would require a huge number of intermediate evaluations to be solved with the required tolerance. We used torchdiffeq [9] package to model neural ODEs. For solving neural ODE, we used standard first-order Euler method with a fixed time step $\Delta t = 0.02$.

A two-layered MLP with 100 intermediate units was employed to map hidden vector to the parameters of the predictive distribution.

Overall, the model has 44352 trainable parameters.

For ODE-RNN [69], we used AdaMax optimizer [42]. We also apply a learning rate decay of 0.999. The learning rate of 1e-2 was used for weakly periodic and EQ kernels, 1e-3 – for Matern-$\frac{5}{2}$ and noisy mixture kernels, and 4e-3 – for the sawtooth process.

Neural ODE CNP

In these experiments, for the Neural ODE CNP model the bottleneck dimension was set to 128. The architectures of encoder and decoder used in our experiments are as follows:

Encoder. We employ the ODE-RNN [69] model with GRU cell as an encoder. For the ODE-RNN encoder we apply a 1-layered MLP with 128 units and tanh non-linearities to

model the dynamics of an ODE. For solving neural ODEs in the encoder, we use standard first-order Euler method with a fixed time step $\Delta t = 0.02$. Then we apply a linear layer to map the output of the encoder to the initial hidden state vector, which serves as input to the decoder.

Decoder. As already mentioned earlier, the decoder is parameterized by the Neural ODE [9] model. A one-layered MLP with 128 units and tanh non-linearities serves as the ODE function. For solving ODE in the decoder we use adaptive fifth-order *dopri5* solver implemented in the torchdiffeq package. We set relative tolerance to 1e-3 and absolute tolerance to 1e-4. A map from hidden function value to the parameters of predictive distribution is parameterized with a simple linear layer.

Overall, the model has 141186 trainable parameters.

For the Neural ODE CNP model, we also used AdaMax optimizer [42]. We employ a learning rate decay of 0.999. The learning rate of 1e-3 was used for all datasets in this experiment.

All models in this experiment were trained for 200 epochs using 256 batches of batch size 16 per epoch.

5.1.3 Results

We evaluate the models on 1000 test tasks. Table 5.2 reports the log-likelihood means and standard errors of the models on the task of interpolation. For interpolation task target and context sets were generated within the interval $[-2;2]$, where training data was observed. Table 5.3 reports Mean Squared Error (MSE) on the save test data. Tables 5.4 and 5.5 provide results for log-likelihood means and MSE for extrapolation task, where target set was generated within the interval $[2;4]$. In this section we test extrapolation capability of the models only for future prediction since ODE-RNN [69] cannot be used to make predictions backwards in time.

Dataset	ConvCNP [28]	ODE-RNN [69]	Neural ODE CNP
EQ	1.477 ± 0.021	0.637 ± 0.0242	0.376 ± 0.019
Matern-$\frac{5}{2}$	0.572 ± 0.018	-0.155 ± 0.018	-0.321 ± 0.019
Noisy Mixture	0.751 ± 0.017	-0.121 ± 0.0168	0.192 ± 0.019
Weakly Periodic	-0.515 ± 0.013	-0.956 ± 0.013	-1.205 ± 0.012
Gauss-Markov	0.874 ± 0.012	0.493 ± 0.011	0.644 ± 0.014
Sawtooth	2.339 ± 0.041	2.016 ± 0.014	-0.616 ± 0.024

Table 5.2 Test log-likelihoods for synthetic one-dimensional data on interpolation task.

Dataset	ConvCNP [28]	ODE-RNN [69]	Neural ODE CNP
EQ	0.119 ± 0.006	0.263 ± 0.009	0.129 ± 0.006
Matern-$\frac{5}{2}$	0.181 ± 0.007	0.368 ± 0.0105	0.213 ± 0.008
Noisy Mixture	0.177 ± 0.009	0.457 ± 0.0159	0.194 ± 0.010
Weakly Periodic	0.327 ± 0.009	0.580 ± 0.012	0.691 ± 0.013
Gauss-Markov	0.019 ± 0.001	0.038 ± 0.001	0.021 ± 0.001
Sawtooth	0.006 ± 0.0003	0.015 ± 0.0005	0.072 ± 0.001

Table 5.3 Test MSE for synthetic one-dimensional data on interpolation task.

Dataset	ConvCNP [28]	ODE-RNN [69]	Neural ODE CNP
EQ	-1.332 ± 0.009	-1.300 ± 0.009	-8.138 ± 0.185
Matern-$\frac{5}{2}$	-1.389 ± 0.011	-1.376 ± 0.009	-8.322 ± 0.302
Noisy Mixture	-1.684 ± 0.013	-1.648 ± 0.0108	-13.141 ± 0.294
Weakly Periodic	-1.359 ± 0.009	-1.377 ± 0.009	-1.609 ± 0.018
Gauss-Markov	-0.284 ± 0.009	-0.255 ± 0.008	-3.039 ± 0.085
Sawtooth	1.215 ± 0.014	-0.682 ± 0.029	-1.809 ± 0.026

Table 5.4 Test log-likelihood for synthetic one-dimensional data on extrapolation task.

Dataset	ConvCNP [28]	ODE-RNN [69]	Neural ODE CNP
EQ	0.917 ± 0.016	0.905 ± 0.016	0.495 ± 0.371
Matern-$\frac{5}{2}$	0.977 ± 0.017	0.974 ± 0.017	1.779 ± 0.358
Noisy Mixture	1.787 ± 0.033	1.783 ± 0.033	0.669 ± 0.155
Weakly Periodic	0.918 ± 0.016	0.936 ± 0.018	0.254 ± 0.128
Gauss-Markov	0.109 ± 0.002	0.102 ± 0.002	0.363 ± 0.103
Sawtooth	0.028 ± 0.001	0.076 ± 0.0009	0.142 ± 0.034

Table 5.5 Test MSE for synthetic one-dimensional data on extrapolation task.

The above tables demonstrate that ConvCNP [28] outperforms other methods in most cases. ODE-RNN [69] shows performance comparable to ConvCNP [28] only on extrapolation tasks for data sampled from GPs. Neural ODE CNP fails catastrophically in terms of log-likelihood metric, however it outperforms ODE-RNN [69] in terms of MSE on some interpolation and extrapolation tasks.

Figures 5.1, 5.2, 5.3 show the interpolation and extrapolation results obtained by the models trained on data sampled from a GP with a specified kernel. Figure 5.4 illustrates the performance of the models trained on the sawtooth process data.

From Figures 5.1, 5.2, 5.3, 5.4, it is clear that ConvCNP [28] outperforms other models and produces much more reasonable estimates for both interpolation and extrapolation tasks. What is more, for GP [64] data, the predictions of ConvCNP [28] in terms of both mean and standard deviation lie very close to the ones made by the corresponding GP [64]. GP predictions correspond the most accurate ones that could be made given a particular context set.

ODE-RNN [69] shows inferior performance in comparison to ConvCNP [28]. The main reason for that is that ODE-RNN [69] provides its estimates only based on the part of the context set. It takes into account only events that happened prior to the time of prediction. This causes abrupt jumps in the mean after a new data point is observed as well as exaggerated uncertainty estimates before new observation.

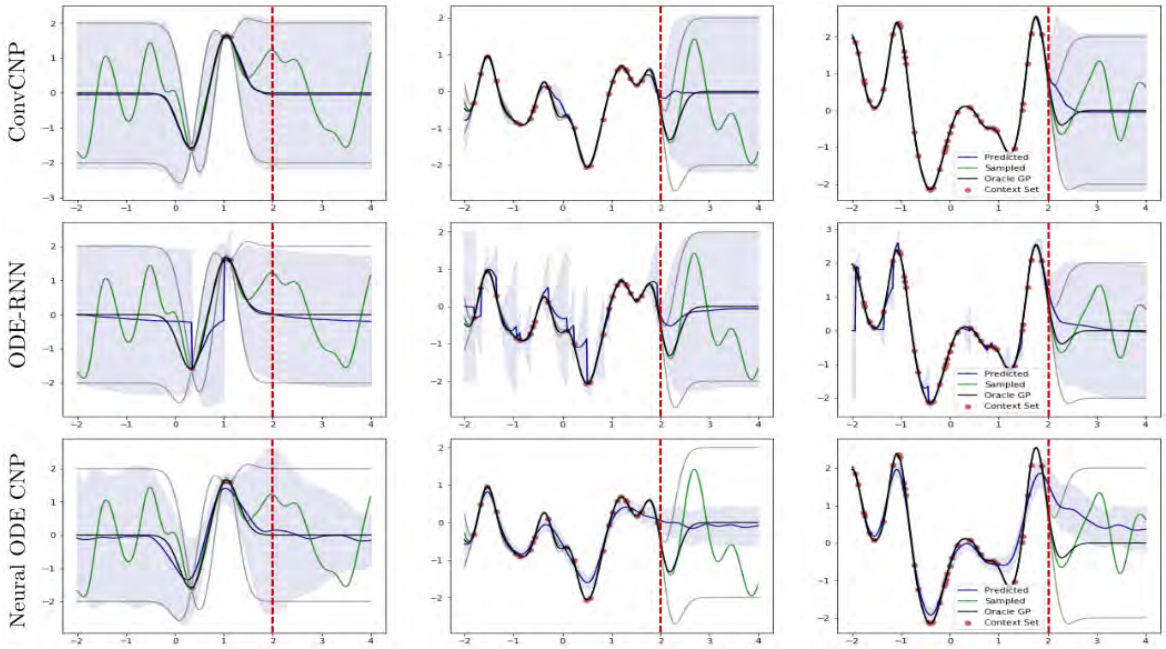
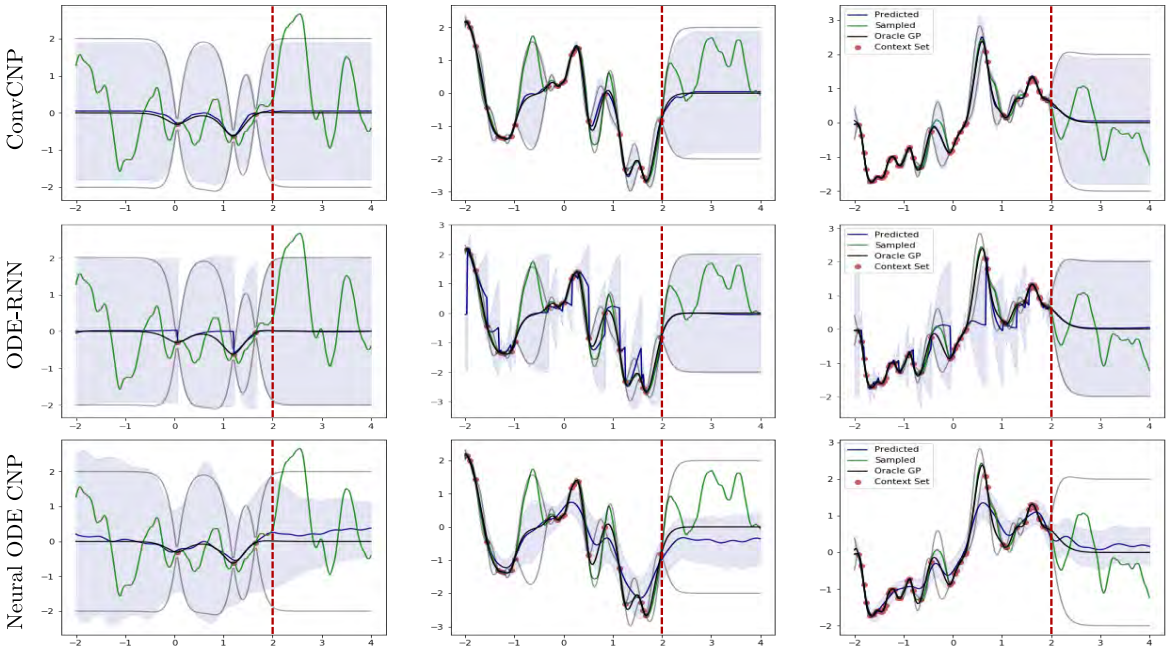
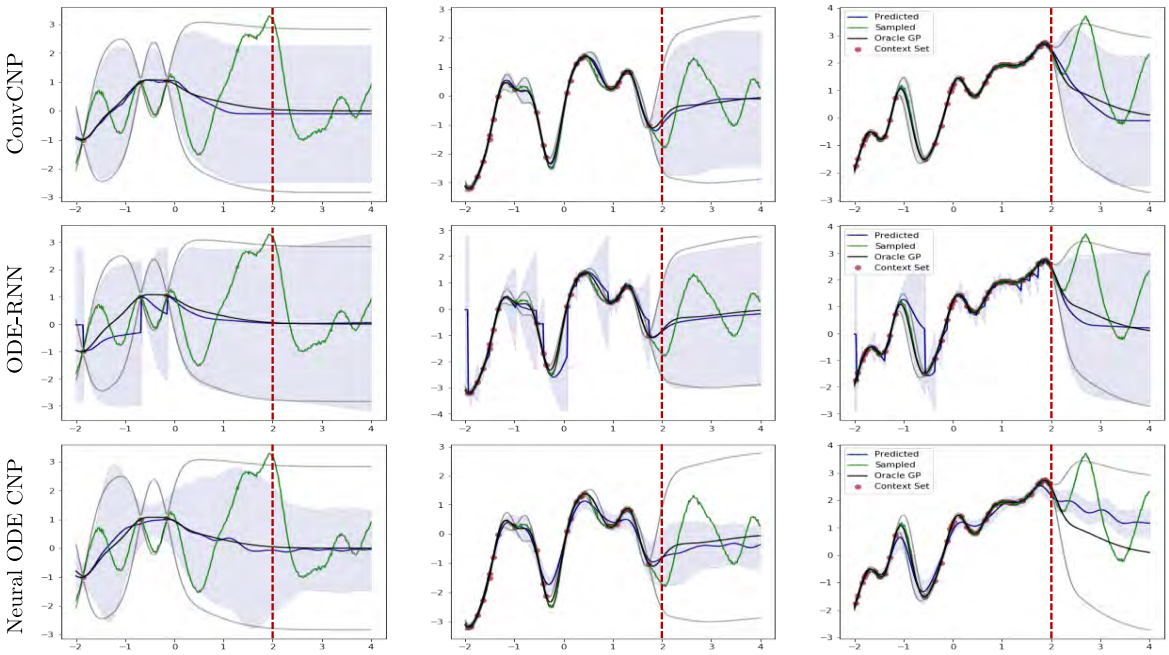


Fig. 5.1 Predictions of ConvCNP [28](top row), ODE-RNN [69] (middle row) and Neural ODE CNP (bottom row) trained on data sampled from a Gaussian Process with EQ kernel. The models are evaluated on both interpolation (interval $[-2;2]$, before the red dotted line) and extrapolation (interval $[2;4]$, after the red dotted line) tasks. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from a Gaussian Process. Black and gray lines are GP mean and $\mu \pm 2\sigma$ correspondingly. Context set is marked with red dots.

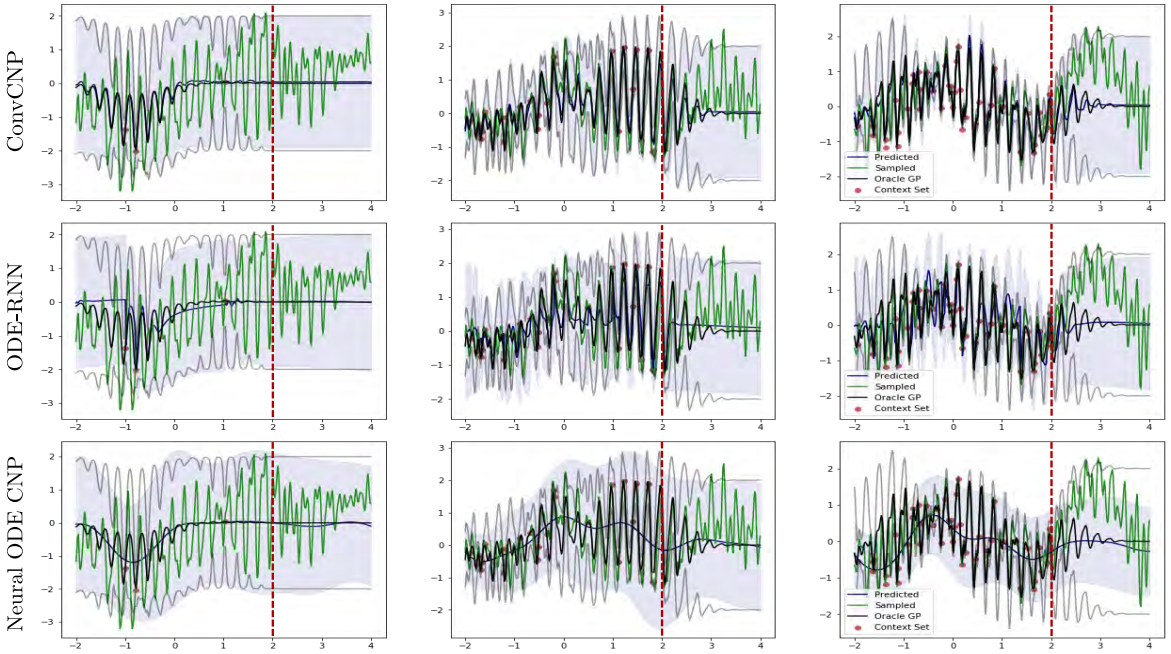


(a) Matern-$\frac{5}{2}$

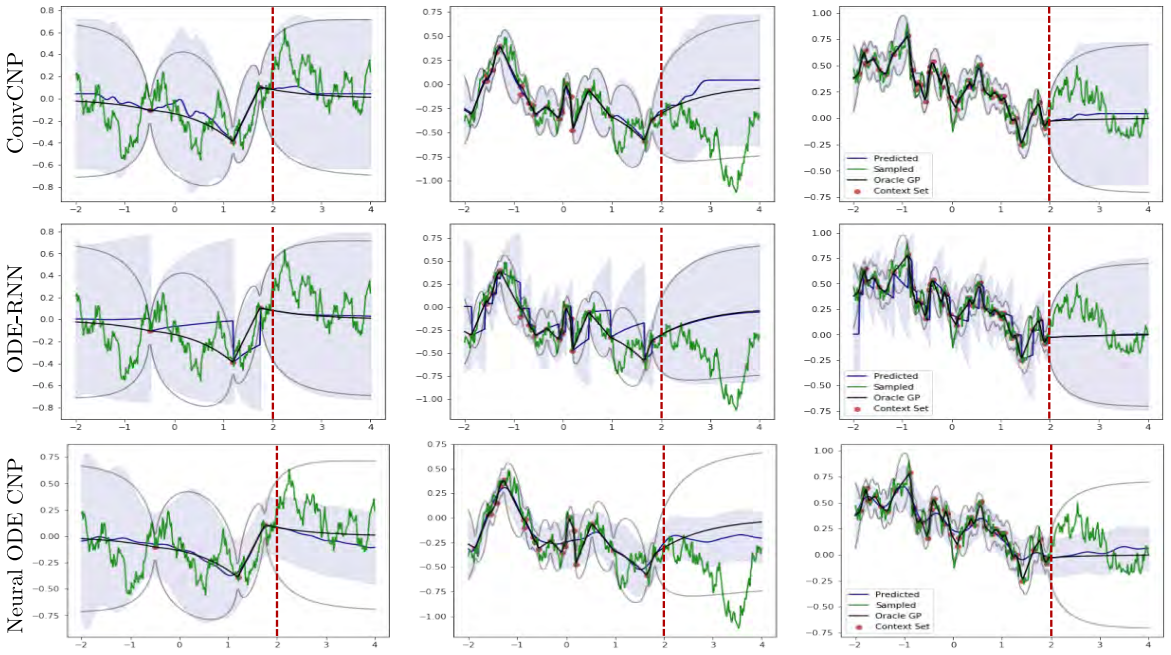


(b) Noisy Mixture

Fig. 5.2 Predictions of ConvCNP [28](top row), ODE-RNN [69] (middle row) and Neural ODE CNP (bottom row) trained on data sampled from a Gaussian Process with Matern$-\frac{5}{2}$ (5.2a) or noisy mixture (5.2b) kernels. The models are evaluated on both interpolation (interval $[-2;2]$, before the red dotted line) and extrapolation (interval $[2;4]$, after the red dotted line) tasks. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from a Gaussian Process. The black and gray lines are GP mean and $\mu \pm 2\sigma$ correspondingly. Context set is marked with red dots.



(a) Weakly Periodic



(b) Gauss-Markov

Fig. 5.3 Predictions of ConvCNP [28](top row), ODE-RNN [69] (middle row) and Neural ODE CNP (bottom row) trained on data sampled from a Gaussian Process with weakly periodic (5.3a) or Gauss-Markov (5.3b) kernels. The models are evaluated on both interpolation (interval $[-2;2]$, before the red dotted line) and extrapolation (interval $[2;4]$, after the red dotted line) tasks. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from a Gaussian Process. The black and gray lines are GP mean and $\mu \pm 2\sigma$ correspondingly. Context set is marked with red dots.

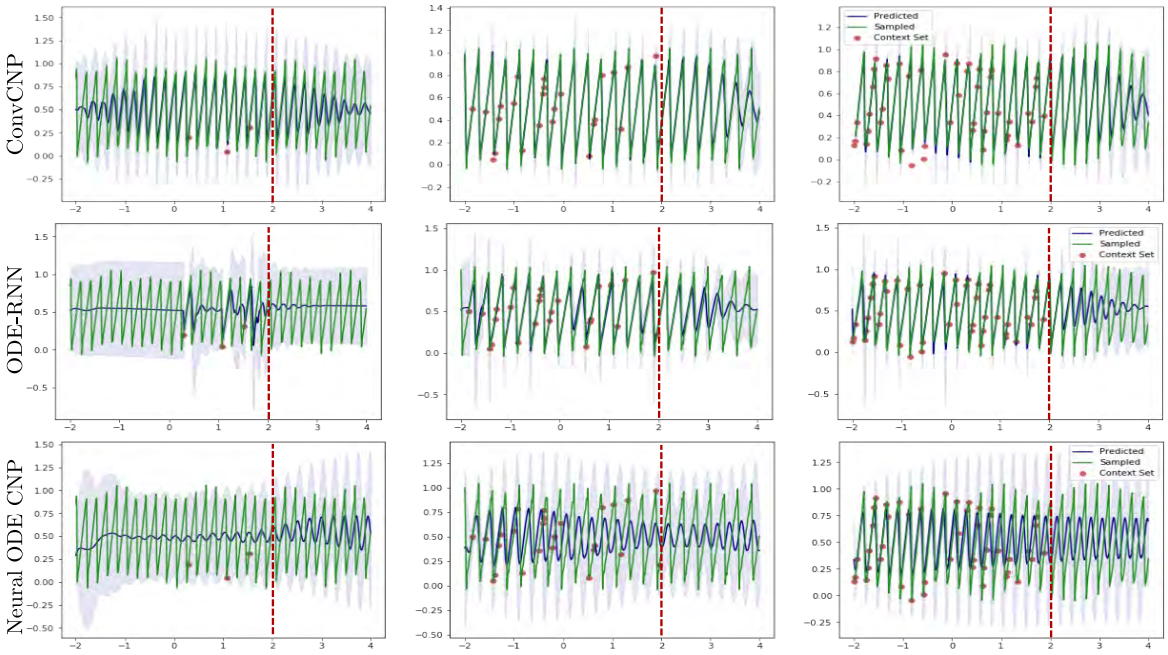


Fig. 5.4 Predictions of ConvCNP [28](top row), ODE-RNN [69] (middle row) and Neural ODE CNP (bottom row) trained on data sampled from the non-Gaussian sawtooth process. The models are evaluated on both interpolation (interval $[-2;2]$, before the red dotted line) and extrapolation (interval $[2;4]$, after the red dotted line) tasks. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from the sawtooth process.Context set is marked with red dots.

Although Neural ODE CNP produces better mean estimates in comparison to ODE-RNN [69], it catastrophically fails to provide reasonable uncertainty bars, particularly for the extrapolation task. We reckon that this limited expressiveness of the model may be due to the bottleneck. In this case, the model is not able to accommodate all the required information in a limited-size vector.

Moreover, the Neural ODE CNP model tends to forget about inputs a long way away from the initial condition. It can be observed in the above figures, where the predictions at the initial time points go directly through the points from context set, in contrast to estimates for the final time points. This issue may be attributed to the use of ODE-RNN [69] as an encoder since recurrent network are notoriously known for having short-term memory.

In the next section, we investigate the Neural ODE CNP model in more detail, providing experiments with different architectures of the encoder. We also compare the performance of Neural ODE CNP with its MLP-based counterpart, namely the CNP [21] model. Finally, we also show that the chosen bottleneck size is enough to encode more information about observations, so that mean function goes directly through the points of the context set.

Therefore, this experiment suggests that the usage of semi-amortized models would be beneficial in this context, making the amortization gap smaller.

5.2 Neural ODE CNP on Synthetic GP Data

By its construction, Neural ODE CNP model provides a lot of freedom in the choice of encoder architecture. So, in section 5.2.1 we examine several possible solutions as well as compare their performance to the standard CNP [21] model, which employs MLP-based decoder instead of Neural ODE [9]-based one. All models in this section were trained and evaluated on data sampled from GP [64] with EQ kernel. The data generation procedure is described in section 5.1.1. In section 5.2.2, we show that representation given by the Neural ODE CNP can be improved, showing that the chosen bottleneck size is enough to incorporate the required information about observed data.

5.2.1 Neural ODE CNP: Encoder Architectures

Model Architectures and Training Details

In this section, we describe in detail the architectures of the models that were used in the following experiments. We inspect the models with bottleneck size of $64, 128$ and 256. For some fixed bottleneck size, we compare the following model architectures:

1. **CNP-small**. This is a standard CNP [21] model. As for the architecture of the model, we used 3-layered MLPs as encoder and decoder with the number of hidden units in each layer being equal to the chosen bottleneck size.

2. **CNP-large**. This model is similar to CNP-small, but has 4-layered MLPs serving as encoder and decoder.

3. **ODE-DS-small** attributes to the Neural ODE CNP model with DeepSet [83] encoder. The output of the DeepSet [83] encoder is used as an initial hidden state for the Neural ODE [38] decoder. 3-layered MLP is used as an encoder, Neural ODE [9] as a decoder. A one-layered MLP with the number of units equal to the bottleneck size with tanh non-linearities is used for the ODE function. A linear layer is used as a mapping from hidden space to the parameters of predictive distribution.

4. **ODE-DS-large**. Similar to ODE-DS-small, this model stands for the Neural ODE CNP model with DeepSet [83] encoder. A 4-layered MLP is used as an encoder. A neural network with 2 hidden layers serves as the ODE function. The number of units

in each layer equals to the chosen bottleneck size. The other parameters are the same as in the **ODE-DS-small** model.

5. **ODE-RNN-small** stands for Neural ODE CNP with ODE-RNN [69] encoder and Neural ODE [9] decoder. Neural ODE [9] decoder has the same architecture as **ODE-DS-small**. We also used a 3-layered MLP with tanh non-linearities as the ODE function. The number of units used is equal to the bottleneck size. As for the ODE-RNN [69] encoder, for the ODE function we employ the same architecture as is used for the decoder. Both mappings from the output of the ODE-RNN [69] to initial hidden state of the decoder as well as from hidden state vector to the parameters of predictive distribution are parameterized with a single linear layer.

6. **ODE-RNN-large** employs the same structure and architecture as **ODE-RNN-small**, however has more trainable parameters. The ODE function for both encoder and decoder are chosen to be 4-layered MLPs with tanh non-linearities and the number of hidden units equal to the bottleneck size.

All in all, for the bottleneck size of 128 the introduced models have the following number of trainable parameters:

Model	Number of trainable parameters
CNP-small	66818
CNP-large	99842
ODE-DS-small	66690
ODE-DS-large	99714
ODE-RNN-small	141186
ODE-RNN-large	178370

Table 5.6 The number of trainable parameters for the CNPs and variations of the Neural ODE CNP model investigated in our experiments.

In this experiment, all models were trained for 200 epochs using 256 batches of batch size 16 per epoch. We used Adam [42] optimizer with learning rate of 1e-3 for all models. We also employ a learning rate decay of 0.999.

Results

We evaluate the models on 1000 interpolation tasks sampled from a GP [64] with EQ kernel. Table 5.7 reports the log-likelihood means and standard errors of the introduced models with

bottleneck size of 64, 128 or 256. We have not managed to train the ODE-RNN-small and ODE-RNN-large models with bottleneck size of 256 because of the unstable training caused by the exorbitant number of trainable parameters in the ODE-RNN [69] encoder. So, these results are absent in Table 5.7.

Models	Bottleneck size		
	64	128	256
CNP-small	-0.890 ± 0.006	-0.664 ± 0.012	-0.560 ± 0.006
CNP-large	-0.760 ± 0.007	-0.551 ± 0.014	-0.350 ± 0.009
ODE-DS-small	-0.782 ± 0.02	-0.521 ± 0.014	-0.699 ± 0.011
ODE-DS-large	-0.781 ± 0.02	-0.387 ± 0.015	-0.626 ± 0.014
ODE-RNN-small	0.130 ± 0.018	$\mathbf{0.370 \pm 0.019}$	–
ODE-RNN-large	0.109 ± 0.02	0.274 ± 0.016	–

Table 5.7 Test log-likelihoods for the considered models with different bottleneck sizes on interpolation task. Models were trained on data sampled from GP [64] with EQ kernel. We have not managed to train the ODE-RNN-small and ODE-RNN-large models with bottleneck size of 256 because of the unstable training caused by the exorbitant number of trainable parameters in the ODE-RNN [69] encoder. The best result among all models is highlighted in bold.

From Table 5.7, it can be seen that models with Neural ODE [9] as a decoder outperform MLP-based one when bottleneck size was equal to 64 and 128. As for bottleneck size of 256, we reckon that the inferior performance of the Neural ODE [9]-based models is caused by the fact that such models are harder to train than ordinary MLPs and they are subject to slower convergence.

What is more, the best performance is achieved by the ODE-RNN-small model, showing that ODE-RNN [69] encoder outperforms DeepSet [83] one. Perhaps, one of the main reasons for that is that ODE-RNN [69] encoder can learn some non-linear interactions between observed data points and use this interaction to produce a final embedding vector. Meanwhile, DeepSet [83] has a specified linear interaction between representations of observed data points, as encoder firstly processes each observation separately and then takes the mean over all encoded observations. However, another possible reason for that is that ODE-RNN [69] simply has more trainable parameters than DeepSet [83].

The illustration of the models' performance is provided in Figure 5.5.

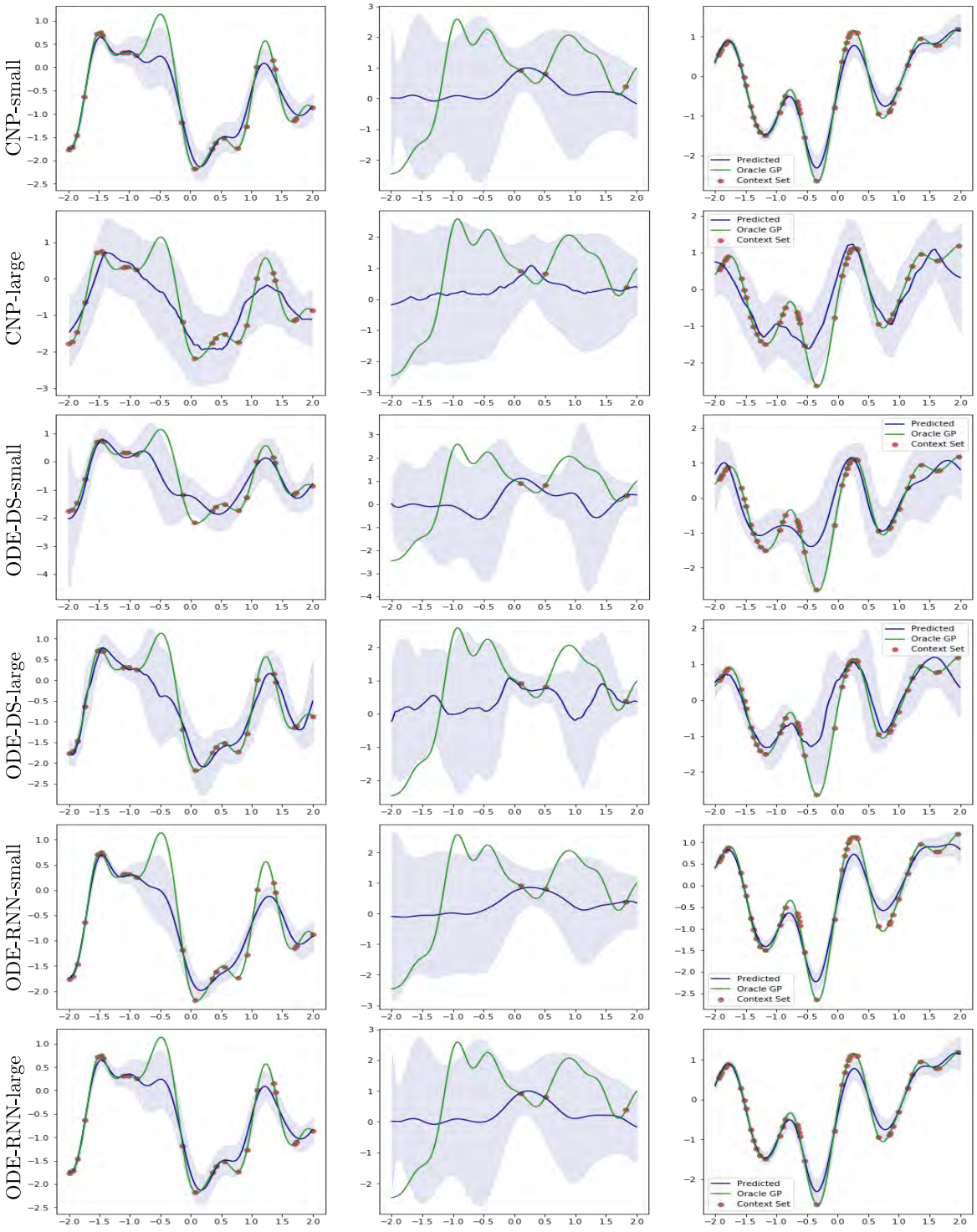


Fig. 5.5 Predictions of CNP-small (first row), CNP-large (second row), ODE-DS-small (third row), ODE-DS-large (forth row), ODE-RNN-small (fifth row), and ODE-RNN-large (last row). All models were trained on data sampled from the GP [64] with EQ kernel. The models are evaluated on interpolation task within the interval $[-2,;2]$. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from a GP [64]. Context set is marked with red dots.

5.2.2 Initial Hidden State Minimization

Although Neural ODE CNP with ODE-RNN [69] encoder shows the best performance among all other models (section 5.2.1), it tends to forget about inputs a long way away from the initial condition. This can be seen in Figure 5.5. This issue can be caused by the amortization gap and the recurrent structure of ODE-RNN [69] decoder. Another reason for that may be the fact that the size of the bottleneck is not enough to incorporate the necessary information about observed data.

To validate the latter, in the following experiment we show that the encoding obtained from the ODE-RNN [69] encoder could be improved and the bottleneck size is enough to encode more information about observations.

In this experiment, we took a trained model with ODE-RNN [69] encoder and obtain an initial estimate for the hidden initial state from the encoder. Then using gradient descent, we optimize the negative log-likelihood of context set w.r.t. bottleneck vector, without optimizing the weights of the decoder. As a result, we were able to achieve a much better encoding where predictive mean goes directly through all context points. Optimization results are provided in Figure 5.6.

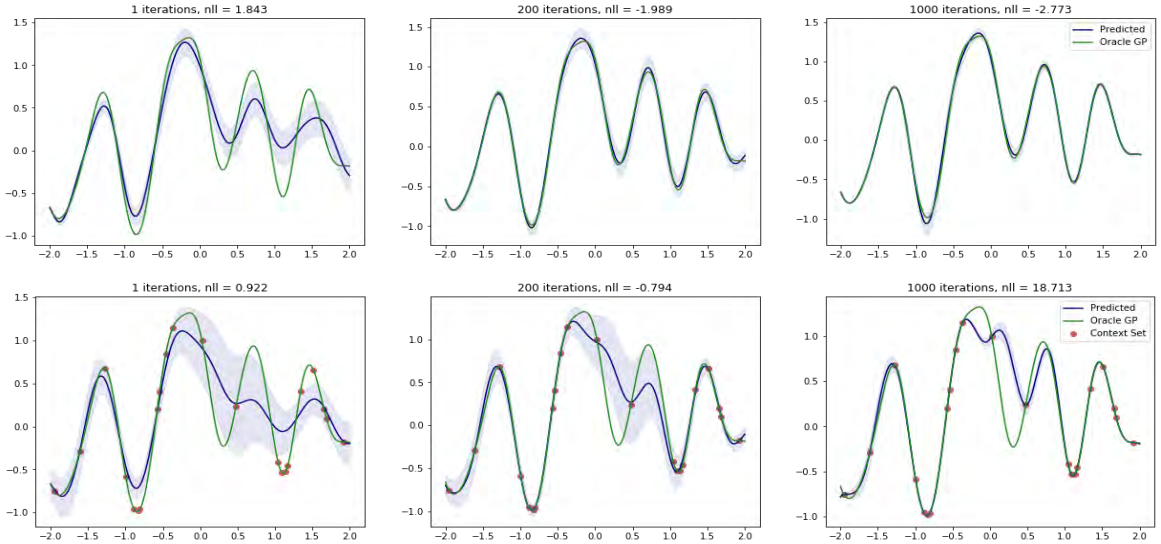


Fig. 5.6 Initial hidden state optimization. Solid blue lines are predictive posterior means μ, whereas shaded blue area is $\mu \pm 2\sigma$. The green solid line is a ground truth sample from a GP [64]. In the first row, the observations are located on the grid with a step of 0.02, whereas in the second row the context set that is observed is indicated with red dots. Left column shows results for the inital hidden state predicted by the encoder. Center and right columns illustrate results after 200 and 1000 optimization steps respectively. In the title of each figure the negative log-likelihood over target set is provided.

It is worth mentioning that in the second row in Figure 5.6, where the number of context points is not that huge, overfitting can be observed and the variance is reduced to almost zero not only at the locations where data points are observed but also at the locations where uncertainty should be presented. However, optimization for a limited number of step, e.g. 200, gives a considerable improvement in the performance of the model, justifying the use of semi-amortized approaches for the problem of irregularly-sampled time series modelling.

5.3 Real Data: PhysioNet

In this section, we investigate the performance of the models on real-world data. The chosen dataset and its preprocessing is described in section 5.3.1. In section 5.3.2, we discuss architecture choices and specific training details. Finally, we discuss the obtained results in section 5.3.3.

5.3.1 Dataset

We evaluated the considered models on the PhysioNet Challenge 2012 [73] dataset. The dataset consists of 8000 time series, where each sample contains measurements from the first 48 hours of a different patient's admission to ICU. Measurements were made at irregular times and at each observation time only some subset of features was observed. There are 37 features that can be observed in this dataset.

We used the version of dataset parsed by Rubanova et.al [69], where the observation times were rounded to the nearest minute. This reduced the possible time measurements two-fold. What is more, to make the split between context and target sets valid for all time series in the dataset, we discard time series samples that contain only one observation. As a result, we obtained a dataset containing 7990 time series samples. What is more, in our experiments we modelled only time-varying features, while static ones (Age, Height, Gender, ICUType and MechVent) were discarded. This results in 32 features that are modelled in our experiments.

We left 70% of data for training, 10% – for validation and the rest – for test. Each feature was normalized such that training data lie within the interval $[0; 1]$. Since some observations at a particular time step are missing, we introduce a binary mask indicating which features are observed at a particular time step. This binary mask was concatenated with feature values and then used as input to the models. Hence, in the experiment where multidimensional data is modelled the dimension of input vector is 64. However, in case of

modelling one-dimensional data the binary mask is redundant. For this reason, it was not used in our experiments where we modelled only one chosen data dimension.

For the ConvCNP [28] model, the number of context points for each sample was chosen randomly betweeb 1 and $N-1$, where N is the number of observed points in the current sample. To form a training batch for ConvCNP [28], the number of points in both context and target sets should be equal for all data samples. In our experiments, we align the lengths of the context and target sets by padding each sample with zeroes to the maximal length within each set.

For the ODE-RNN [69] model, the data is not slit into context and target set. As in the previous experiment, each sample in the training batch was evaluated at the union of all time points in the batch.

For test evaluation we generated 2000 tasks with fixed context and target sets. These tasks were used to evaluate both ConvCNP [28] and ODE-RNN [69] models.

5.3.2 Model Architectures and Training Details

ConvCNP [28]

As in the previous experiment, we assume that kernels ψ, ψ_ρ (section 3.2.1) are EQ kernels with a learnable length scale parameter. The length scales for the EQ kernels are initialized to twice the space between two neighbouring discretization points. The density of discretization is 256 points per unit. The discretization spans an interval from $\min(t)-0.5$ to $\max(t)+0.5$, where $\min(t)$ stands for earliest observation time occurring in the union of the context and target sets in the current batch and $\max(t)$ is the latest observation time respectively.

In the following experiments, we also employ depthwise-separable convolutions [11] in the ConvCNP [28] decoder. In all models, the kernel size is set to 33. We use an 8-layer (excluding an initial and final point-wise linear layers) CNN with 64 channels as a decoder in ConvCNP [28]. We use ReLU non-linearities in all models.

For multidimensional data experiments, we assume that predictive distribution is Gaussian with diagonal covariance matrix. Hence, in multidimensional and one-dimensional data experiments, the standard deviation is parameterized in the model by passing the output of the decoder through a softplus function.

For two-dimensional experiments, we set the predictive distribution to be Gaussian with full covariance matrix. The covariance matrix is parameterized using the Cholesky decomposition, where the decoder of ConvCNP [28] outputs vector of three values that parameterize lower triangular matrix in the Cholesky decomposition.

In the experiment with multidimensional data, the number of trainable parameters is chosen to be comparable with ODE-RNN [69] model, which architecture is described in the next section, and amounts to 60240 parameters.

All models were trained using Adam [42] optimizer with learning rate of 5e-4. We also used a weight decay of 10^{-5} applied to all parameters of the models.

ODE-RNN [69]

We use a neural network with 3 hidden layers comprising 50 units each to model the dynamics of an ODE. The dimension of the hidden function was set to 20. For the ODE-RNN [69] model, we employ tanh non-linearities. For solving neural ODE, we used standard first-order Euler method with a fixed time step $\Delta t = 0.02$. A two-layered MLP with 100 intermediate units was employed to map hidden vector to the parameters of the predictive distribution. Overall, the model has 61184 trainable parameters.

We use AdaMax optimizer [42] with the learning rate of 1e-3. We also apply a learning rate decay of 0.999.

All models are trained for 300 epochs with 200 batches of batch size 50, so that approximately each sample is considered within one epoch.

5.3.3 Results

In this section, we discuss the results of the models' evaluation of PhysioNet [73] data. We firstly evaluate models on multidimensional data, comprising all time-varying features present in the dataset. We show that both models fail to capture and model all dimensions due to the complexity of the dataset, where most dimensions comprise only a couple of observations per sample. For this reason, we propose to model only certain significant features, such as heart rate and blood pressure. What is more, these features were chosen because they have enough observed data for the model to capture patterns presented in the data. Our results in modelling one- and two-dimensional data are reported after the discussion of the experiments on multidimensional data.

Full Dataset

We trained ConvCNP [28] and ODE-RNN [69] using all features of PhysioNet [73] dataset. Then the model were tested on interpolation task. The results are presented in Table 5.8. ODE-RNN [69] and ConvCNP [28] models show comparable results on these data. However, if we visualize the predictions for some features, we can definitely say that the models were

not able to capture any sensible patterns. The illustration of some dimensions is given in Figure 5.7.

ODE-RNN [69]	ConvCNP [28]
2.128 ± 0.014	2.098 ± 0.019

Table 5.8 Test log-likelihood for PhysioNet [73] experiments on interpolation task.

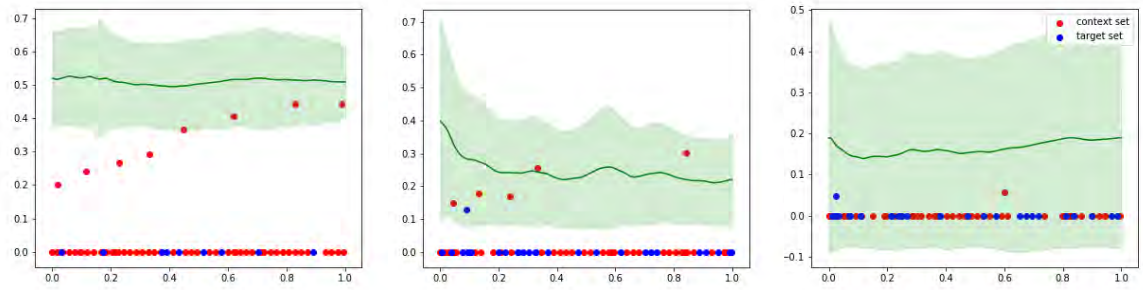


Fig. 5.7 ConvCNP [28] performance on three different features from PhysioNet [73] data. Solid green lines are predictive posterior means μ, whereas shaded green area is $\mu \pm 2\sigma$. Blue dots stand for target set, red ones indicate context set. Blue and red dots with zero value indicate that the feature value is missing, i.e. there is an observation at a particular time point, but the visualized feature is not observed.

From Figure 5.7, we can see that there are dimensions in the PhysioNet [73] data, where the amount of observed data is not sufficient to extract any sensible information to generate predictions. To this end, we further investigate the performance of ConvCNP [28] on one-dimensional data from PhysioNet [73]. We chose only several features, which have enough observations to train the model, and train a separate model for each of the chosen dimensions. The main results are discussed in the next section.

One- and Two-Dimensional Data

For one-dimensional experiments we choose five dimensions from PhysioNet [73], which have the biggest average number of observations per sample. What is more, the chosen dimensions appeared to be crucial for assessing health level. These dimensions are heart rate, diastolic and systolic arterial blood pressure, urine and body temperature.

We compare ConvCNP [28] performance against the performance of a GP [64] with Matern-$\frac{5}{2}$ kernel. The hyperparameters of a GP were optimized either for each individual time-series (GP-individual) or for the whole dataset (GP-dataset). The results are provided in Tables 5.9 and 5.10.

Dataset	ConvCNP [28]	GP-individual	GP-dataset
Heart rate	2.416 ± 0.022	2.106 ± 0.054	2.134 ± 0.038
Diastolic BP	1.575 ± 0.019	0.275 ± 0.169	1.463 ± 0.021
Systolic BP	1.617 ± 0.029	0.401 ± 0.196	1.331 ± 0.048
Urine	2.908 ± 0.105	1.985 ± 0.613	1.666 ± 0.620
Temperature	4.063 ± 0.037	2.753 ± 0.558	2.858 ± 0.333

Table 5.9 Test log-likelihood for one-dimensional PhysioNet [73] experiments on interpolation task. BP stands for blood pressure.

Dataset	ConvCNP [28]	GP-individual	GP-dataset
Heart rate	6.73e-04 $\pm$ 2.96e-05	7.55e-04 $\pm$ 3.42e-05	7.87e-04 $\pm$ 3.57e-05
Diastolic BP	2.61e-03 $\pm$ 9.86e-05	2.95e-03 $\pm$ 1.19e-04	3.01e-03 $\pm$ 9.62e-05
Systolic BP	2.86e-03 $\pm$ 2.22e-04	2.96e-03 $\pm$ 1.67e-04	3.16e-03 $\pm$ 1.68e-04
Urine	1.99e-04 $\pm$ 4.82e-05	2.20e-04 $\pm$ 5.24e-05	2.01e-04 $\pm$ 4.68e-05
Temperature	5.92e-05 $\pm$ 1.58e-05	7.22e-05 $\pm$ 1.69e-05	6.94e-05 $\pm$ 1.67e-05

Table 5.10 Test MSE for one-dimensional PhysioNet [73] experiments on interpolation task. BP stands for blood pressure.

ConvCNP [28] outperforms GP [64] models in terms of log-likelihood as well as MSE metric. Moreover, the ConvCNP [28] model employs more efficient test evaluation since it does not require additional optimization over test data.

In Figure 5.8 we illustrate the performance of ConvCNP [28] and GP [64] models. GP-individual tends to overfit, especially when the context set is scarce, whereas GP-dataset tends to underfit on samples with sufficient amount of data in the context set.

Finally, we evaluate ConvCNP [28] model on two-dimensional data. For these two dimensions we chose diastolic and systolic blood pressure, which obviously should have some mutual correlation. We modelled predictive distribution as Gaussian with full covariance matrix. Test log-likelihood and MSE results are presented in Table 5.11.

Log-likelihood	MSE
2.846 ± 0.053	3.61e-03 $\pm$ 1.49e-04

Table 5.11 Test log-likelihood for PhysioNet [73] two-dimensional experiments on interpolation task for the ConvCNP [28] model.

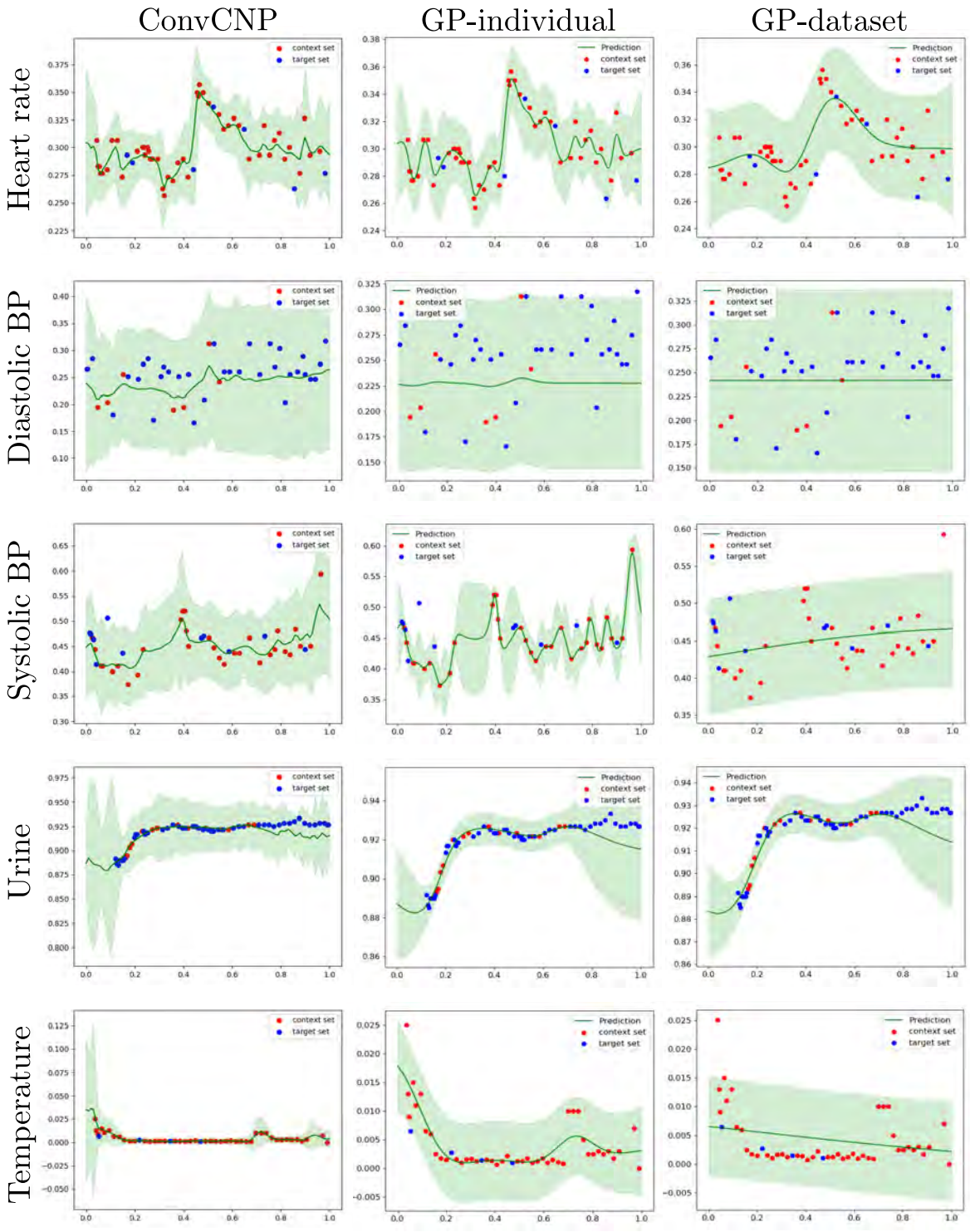


Fig. 5.8 The performance of the ConvCNP [28] (left column), GP-individual (center column) and GP-dataset (right column) models on the chosen dimensions from the Physionet [73] dataset. Each row illustrates models' performance on a single dimension. Solid green lines are predictive posterior means μ, whereas shaded green area is $\mu \pm 2\sigma$. Blue dots stand for target set, red ones indicate context set.

Chapter 6

Conclusion

6.1 Discussion

In this thesis, we have proposed an organization of the existing neural models from the time-series modelling domain. We outlined several critical modelling dimensions that could be used to compare and contrast the models in this area. What is more, we devise a schematic that arranges general modelling directions, such as autoregressive modelling and meta-learning approaches. Further, we also introduce more fine-grained schematics that are helpful to comprehend the NP and non-amortized model spaces.

Within the proposed framework, we were able to encounter new models, that fit naturally to the proposed schematics but have not been introduced in the literature. Being from NP family, Neural ODE CNP and Neural CDE CNP can be applied to solve few-shot learning tasks within the time series domain, e.g. personalized medical data prediction. We also define a series of non-amortized models trained in meta-learning fashion, that could also be used for time-series modelling.

Finally, we conducted a set of experiments that shed light on the comparative performance of several models, which has been lacked in literature. We show the performance of the models on synthetic one-dimensional data as well as on real-world medical data. Moreover, we investigate the Neural ODE CNP model in detail, showing how the choice of an encoder influences the overall model performance. We also empirically prove the existence of the amortization gap in case of Neural ODE CNP model, which motivates further use of semi-amortized models.

Since the growing literature on time-series modelling is disorganized and sometimes obscurely explained, a unifying framework was required. We believe that this work took a step forward in this direction and provide a general framework, which structures main research directions in this area. It could be used to understand better how models relate to

each other as well as to choose the appropriate model for a particular task, which is of great importance to practitioners.

6.2 Future Work

There are a number of future research directions that could complement the present work.

Neural CDE CNP

Since the ConvCNP [28] model in our experimental evaluation has shown the best performance among the considered models, it is particularly interesting to compare ConvCNP [28] model to its Neural-ODE [9]-based counterpart. What is more, the Neural CDE CNP model was firstly introduced in this thesis, so it needs further investigation, where we could assess the performance of the model.

Semi-Amortized Models

In the experimental chapter of the present work, we show that amortized models tend to have amortization gap, influencing the final model performance. Towards this end, it would be interesting to examine semi-amortized models in terms of suitable architectures and training pipeline. In recent literature, there has been proposed several training pipelines [32, 41, 45, 54], which vary in terms of a trade-off between computational complexity and model performance. However, the models were mainly evaluated on image datasets. Thus, it would be interesting to compare the performance of the training pipelines for models working with irregularly-spaced time series data.

Real-World Datasets

Although we provide an extensive evaluation of the models on synthetic one-dimensional data, it is necessary to evaluate the models on several complex real-world datasets from different area. A lot of patterns present in data may be unique to a particular domain. Hence, there may be a dependency between the choice of model and the domain and type of data considered. Moreover, based on our experiments on PhysioNet [73] data, we can see that models can fail to fit the data due to its complexity. Such experiments would be helpful in identifying the limits of the existing models and perhaps give some intuition for further improvement.

References

[1] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *WIREs Comput. Stat.*, 2(4):433–459.

[2] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

[4] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[5] Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

[6] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*.

[7] Cao, W., Wang, D., Li, J., Zhou, H., Li, L., and Li, Y. (2018). Brits: Bidirectional recurrent imputation for time series. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6775–6785. Curran Associates, Inc.

[8] Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2016). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8.

[9] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc.

[10] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

[11] Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. cite arxiv:1610.02357.

[12] Cohen, T. S. and Welling, M. (2016). Group equivariant convolutional networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2990–2999. JMLR.org.

[13] Cremer, C., Li, X., and Duvenaud, D. (2018). Inference suboptimality in variational autoencoders. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1078–1086, Stockholmsmässan, Stockholm Sweden. PMLR.

[14] De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. (2019). Gru-ode-bayes: Continuous modeling of sporadically-observed time series. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 7379–7390. Curran Associates, Inc.

[15] de G. Matthews, A. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the kullback-leibler divergence between stochastic processes. In Gretton, A. and Robert, C. C., editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 231–239, Cadiz, Spain. PMLR.

[16] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38.

[17] Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 766–774, Cambridge, MA, USA. MIT Press.

[18] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

[19] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400.

[20] Foong, A. Y. K., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., and Turner, R. E. (2020). Meta-learning stationary stochastic process prediction with convolutional neural processes. *arXiv preprint arXiv:2007.01332*.

[21] Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., and Eslami, S. M. A. (2018a). Conditional neural processes. *CoRR*, abs/1807.01613.

[22] Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. (2018b). Neural processes. *CoRR*, abs/1807.01622.

[23] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France. PMLR.

[24] Ghahramani, Z. and Beal, M. J. (2001). Propagation algorithms for variational bayesian learning. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 507–513. MIT Press.

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

[26] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[27] Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. (2019). Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*.

[28] Gordon, J., Bruinsma, W. P., Foong, A. Y. K., Requeima, J., Dubois, Y., and Turner, R. E. (2020). Convolutional conditional neural processes. In *International Conference on Learning Representations*.

[29] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4.

[30] Gregor, K., Danihelka, I., Graves, A., Rezende, D., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1462–1471, Lille, France. PMLR.

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

[32] Hjelm, D., Salakhutdinov, R. R., Cho, K., Jojic, N., Calhoun, V., and Chung, J. (2016). Iterative refinement of the approximate posterior for directed belief networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4691–4699. Curran Associates, Inc.

[33] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[34] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14(4):1303–1347.

[35] Jia, J. and Benson, A. R. (2019). Neural jump stochastic differential equations. *CoRR*, abs/1905.10403.

[36] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Mach. Learn.*, 37(2):183–233.

[37] Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks.

[38] Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural Controlled Differential Equations for Irregular Time Series. *arXiv:2005.08926*.

[39] Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, S. M. A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2019). Attentive neural processes. *CoRR*, abs/1901.05761.

[40] Kim, Y., Denton, C., Hoang, L., and Rush, A. M. (2017). Structured attention networks. *CoRR*, abs/1702.00887.

[41] Kim, Y., Wiseman, S., Miller, A., Sontag, D., and Rush, A. (2018). Semi-amortized variational autoencoders. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2678–2687, Stockholmsmässan, Stockholm Sweden. PMLR.

[42] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

[43] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

[44] Kondor, R. and Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the action of compact groups. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2752–2760. PMLR.

[45] Krishnan, R. G., Shalit, U., and Sontag, D. (2017). Structured inference networks for nonlinear state space models. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2101–2109. AAAI Press.

[46] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 29–37, Fort Lauderdale, FL, USA. PMLR.

[47] Lechner, M. and Hasani, R. (2020). Learning long-term dependencies in irregularly-sampled time series.

[48] LeCun, Y., Simard, P. Y., and Pearlmutter, B. (1993). Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 156–163. Morgan-Kaufmann.

[49] Li, X., Wong, T.-K. L., Chen, R. T. Q., and Duvenaud, D. K. (2020). Scalable gradients and variational inference for stochastic differential equations. In Zhang, C., Ruiz, F., Bui, T., Dieng, A. B., and Liang, D., editors, *Proceedings of The 2nd Symposium on Advances in Approximate Bayesian Inference*, volume 118 of *Proceedings of Machine Learning Research*, pages 1–28. PMLR.

[50] Lipton, Z. C., Kale, D., and Wetzel, R. (2016). Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. In Doshi-Velez, F., Fackler, J., Kale, D., Wallace, B., and Wiens, J., editors, *Proceedings of the 1st Machine Learning for Healthcare Conference*, volume 56 of *Proceedings of Machine Learning Research*, pages 253–270, Children's Hospital LA, Los Angeles, CA, USA. PMLR.

[51] Lyons, T., Caruana, M., and Levy, T. (2014). Differential equations driven by rough paths. *Springer*.

[52] Längkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42.

[53] Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2113–2122, Lille, France. PMLR.

[54] Marino, J., Yue, Y., and Mandt, S. (2018). Iterative amortized inference. *CoRR*, abs/1807.09356.

[55] Mei, H. and Eisner, J. M. (2017). The neural hawkes process: A neurally self-modulating multivariate point process. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6754–6764. Curran Associates, Inc.

[56] Opfer, G. and Knott, G. D. (2001). Interpolating cubic splines. *J. Approx. Theory*, 112(2):319–321.

[57] Owen, A. B. (2013). *Monte Carlo theory, methods and examples*.

[58] Paisley, J., Blei, D. M., and Jordan, M. I. (2012). Variational bayesian inference with stochastic search. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1363–1370, Madison, WI, USA. Omnipress.

[59] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

[60] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[61] Pontriagin, L., Boltyanskie, V., Collection, K. M. R., Neustadt, L., Trirogoff, K., Gamkrelidze, R., and Miŝenko, E. (1962). *The Mathematical Theory of Optimal Processes*. Interscience publishers. Interscience Publishers.

[62] Rajkomar, A., Oren, E., Chen, K., Dai, A., Hajaj, N., Liu, P., Liu, X., Sun, M., Sundberg, P., Yee, H., Zhang, K., Duggan, G., Flores, G., Hardt, M., Irvine, J., Le, Q., Litsch, K., Marcus, J., Mossin, A., and Dean, J. (2018). Scalable and accurate deep learning for electronic health records. *npj Digital Medicine*, 1.

[63] Ranganath, R., Gerrish, S., and Blei, D. (2014). Black Box Variational Inference. In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 814–822, Reykjavik, Iceland. PMLR.

[64] Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA.

[65] Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France. PMLR.

[66] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China. PMLR.

[67] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

[68] Ross, S. (1996). *Stochastic processes*. Wiley series in probability and statistics: Probability and statistics. Wiley.

[69] Rubanova, Y., Chen, R. T. Q., and Duvenaud, D. (2019). Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907.

[70] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA.

[71] Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2018). Meta-learning with latent embedding optimization. *CoRR*, abs/1807.05960.

[72] Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany.

[73] Silva, I., Moody, G., Scott, D., Celi, L., and Mark, R. (2012). Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *Computing in cardiology*, 39:245–248.

[74] Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London.

[75] Snelson, E. and Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press.

[76] Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). FUNCTIONAL VARIATIONAL BAYESIAN NEURAL NETWORKS. In *International Conference on Learning Representations*.

[77] Thrun, S. and Pratt, L., editors (1998). *Learning to Learn*. Kluwer Academic Publishers, USA.

[78] Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P., and Larochelle, H. (2019). Meta-dataset: A dataset of datasets for learning to learn from few examples. *CoRR*, abs/1903.03096.

[79] Turner, R., Berkes, P., and Sahani, M. (2008). Two problems with variational expectation maximisation for time-series models. *Bayesian Time Series Models*.

[80] van den Oord, A., Kalchbrenner, N., Espeholt, L., kavukcuoglu, k., Vinyals, O., and Graves, A. (2016). Conditional image generation with pixelcnn decoders. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc.

[81] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408.

[82] Xing, E. P., Jordan, M. I., and Russell, S. J. (2012). A generalized mean field algorithm for variational inference in exponential families. *CoRR*, abs/1212.2512.

[83] Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. (2017). Deep sets. *CoRR*, abs/1703.06114.