The Neural Processes Family: Translation Equivariance and Output Dependencies

James Ryan Requeima

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Sidney Sussex College December 1, 2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

James Ryan Requeima
December 1, 2022

The Neural Processes Family: Translation Equivariance and Output Dependencies

Most contemporary machine learning approaches use a model trained from scratch on a particular task and a learning algorithm designed by hand. This approach has worked very well with the advent of deep learning and in the presence of very large datasets (Goodfellow et al., 2016). Recently, meta-learning has emerged as a machine learning approach to learn both a model and a learning algorithm (Hospedales et al., 2021; Schmidhuber, 1987) directly from data. Neural processes (Garnelo et al., 2018a,b) are a family of meta-learning models which combine the flexibility of deep learning with the uncertainty awareness of probabilistic models. Training using meta-learning allows neural processes to apply deep neural networks to applications with smaller training sets where they would typically overfit. Neural processes produce well-calibrated predictions, enable fast inference at test time, and have flexible data-handling properties that make them a good candidate for messy real-world datasets and applications.

However, this thesis focuses on addressing two shortcomings when applying neural processes to real-world applications by i) incorporating translation equivariance into the architecture of neural processes rather than requiring the model to learn this inductive bias directly from data and ii) developing methods for neural processes to parametrize rich predictive distributions that can model dependencies between output-space variables and produce coherent samples.

This thesis makes four main contributions to the family of neural processes models. First, we introduce the *convolutional conditional neural process* (ConvCNP). The ConvCNP incorporates translation equivariance into its modelling assumptions by using convolutional neural networks and improves training data efficiency and performance when data is approximately stationary. Second, we propose the latent variable version of the ConvCNP, the *convolutional latent neural process* (convLNP) that is able to model epistemic uncertainty and output-space dependencies and able to produce coherent function samples. We also propose an approximate maximum likelihood training procedure for the ConvLNP improving upon the standard VI approximate inference technique used by latent neural processes at the time. Third, we propose the *Gaussian neural process* (GNP) which models the predictive distribution with a full covariance Gaussian. The GNP can model joint output-space dependencies like the ConvLNP but avoids the issues associated with using latent variables. Training GNPs is much more simple than the ConvLNP since it uses the same maximum likelihood technique as standard conditional neural processes. Fourth, we introduce the autoregressive neural process (AR NP). Rather than proposing a new neural process architecture this method produces predictions at test time by evaluating existing neural process models autoregressively via the product rule of probability. This method allows for the use of existing, potentially already trained neural processes to model non-Gaussian predictive distributions and produce coherent samples without any modifications to the architecture or training procedure.

The efficacy of each of these methods is demonstrated through a series of synthetic and real world experiments in climate science, population modelling, and medical science applications. It can be seen in these applications that incorporating translation equivariance as a modelling assumption and generating predictive distributions that model output-space dependencies improves predictive performance.

James Ryan Requeima

For Bertie, Maude and Theo.

Acknowledgements

First I would like to thank my supervisor Rich. It's common knowledge that one's relationship with their advisor will make or break your PhD experience and I couldn't have asked for a better one, so thank you. I would also like to thank my secondary advisor Miguel who gave me my first taste of research and my first publication by bringing me under his wing. I'd like to thank David for inspiring me to go into machine learning and helping me to become a better researcher and writer.

Thanks to my friends and collaborators Wessel, Stratis, Jonathan, John, Andrew, Will, Ambrish, Anna, and Tom. Your curiosity, openness and spirit of sharing has made doing research in the CBL truly a pleasure. Thanks to my good friends Cozmin, Em, Yann, Evan, Brendan and Maja for keeping me sane.

Thanks to my parents and family for the love and support. Thanks to Bertie, Maude and Theo for keeping things interesting during my PhD. Finally thanks to Antonia for helping me through all of the challenges along the way – I couldn't have done this without you.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In this chapter, I outline the main motivations for the work presented in this thesis as well as the main contributions contained within. A list of publications that I have co-authored during my PhD studies is presented.

1.1 Motivation

Recently, meta-learning, often called learning to learn, has emerged as a machine learning approach to simultaneously learn a model and a learning algorithm (Hospedales et al., 2021; Schmidhuber, 1987) for a collection of related tasks. A fundamental goal of machine learning is to automate decision making and meta-learning can be viewed as the next logical step toward this goal. Meta-learning is also better aligned with human and animal learning where it is observed that learning strategies improve on both lifetime and evolutionary timescales (Biggs, 1985; Harlow, 1949; Hospedales et al., 2021; Schrier, 1984). Neural processes (Garnelo et al., 2018a,b) are a family of meta-learning models which combine the flexibility of deep learning with the uncertainty awareness of probabilistic models. Rather than modelling a particular function between inputs and outputs, the usual paradigm of supervised learning, neural process model the *learning the of a map* between training sets and functions (or predictive distributions). In doing so, neural processes alleviate the need for the user to define a prior distribution based on the user's previous knowledge about the function. Instead, neural processes aim to learn properties common to all related tasks via meta-training. This makes neural processes suitable for applications where individual training sets are small.

Physical applications like climate modelling have off-the-grid data. Datasets in medical applications are messy. Both types of datasets are often incomplete due to issues with collection, data corruption, or privacy restrictions requiring modelling methods that are robust to missing or corrupted data. The flexible data-handling capabilities of neural processes (NPs; Garnelo et al. (2018a,b)) make them a promising candidate for application to these and other real-world problems. Due to similar issues, many real-world datasets are small and require data-efficient training techniques such meta-learning or sim-to-real pre-training, again easily handled by NPs.

However, many real-world machine learning applications have a set of requirements that are not met by neural processes. First, ideal solutions to prediction problems with spatio-temporal input variables should be translation equivariant: if the data are translated in time or space, then the predictions should be translated correspondingly (Cohen and Welling, 2016; Kondor and Trivedi, 2018). Although part of the motivation for neural processes is to avoid having to specify model priors, in this case it is useful to be able to incorporate the fundamental inductive bias of translation equivalence into our models as we can do with convolutional neural networks (Cohen and Welling, 2016; LeCun et al., 1998). Initially for conditional neural processes (CNPs; Garnelo et al. (2018a)) translation equivarience had to be learned directly from data requiring large amounts of training data making training inefficient.

Second, the data distributions for real-world applications require non-Gaussian, multi-modal predictive distributions with complex output dependencies across variables. To accurately answer questions such as "What's the probability of enough rain falling across a river basin to lead to a flood?" or "What's the chance of having a week of high temperatures in a particular city?", we require the whole joint distribution over several locations and not just marginal predictions. Many medical science applications, such as patient simulation or treatment schedule modelling, require frameworks for reasoning under uncertainty and a framework for downstream estimation. Standard CNPs are unable to model dependencies across output dimensions. This is due to the modelling assumption that dimensions of the output distribution are independent given the neural network parameters. This assumption results in incoherent samples. As a consequence, the predictions are poorly suited for downstream estimation and real-world applications, such as those in climate science, that require well-calibrated joint uncertainties (Markou et al., 2022). The latent neural process (LNP; Garnelo et al. (2018b)) attempts to address this issue through the use of a latent variable in the model. Unfortunately, this results in a difficult training procedure requiring approximate inference (Foong et al., 2020).

The majority of the work in this thesis is toward developing members of the neural processes family of models that overcome the above challenges, with the goal of applying them to real-world applications. We aim to achieve this by incorporating translation equivariance into neural processes, by making them easier to train, and by making them able to produce more rich and flexible predictive distributions. In the next section, we outline the contributions made in this thesis toward this goal.

1.2 Overview and Main Contributions

This section provides an overview of the contributions this thesis puts forward.

Convolutional Conditional Neural Processes. Chapter 3 introduces the convolutional conditional neural process (ConvCNP), a member of the neural process family that models translation-equivariant maps from training sets to predictive distributions. Translation equivariance is an important inductive bias for many learning problems including time series modelling, spatial data, and images. As a requirement for translation-equivariance, the model embeds data sets into an infinite-dimensional function space as opposed to a finite-dimensional vector space. We extend the theory of neural representations of sets Zaheer et al. (2017a) to functional representations; we introduce the architecture of a convolutional deep set; and we demonstrate that any translation-equivariant embedding can be represented using this construction. We demonstrtate that convolutional neural networks achieve what was state-of-the-art performance at the time of publication compared to contemporary neural processes on synthetic experiments, the PLAsTiCC data set (Allam Jr et al., 2018), a sim2real predator-prey setting, and two-dimensional image completion experiments.

The content in this chapter is based on the publication 'Convolutional Conditional Neural Processes' presented in the International Conference on Learning Representations, 2020 Gordon et al. (2020). The research was a was a collaborative effort conducted with the lead authors Jonathan Gordan, and Wessel P. Bruinsma, collaborators Andrew Y. K. Foong, myself, and Yann Dubois as well as our advisor Richard E. Turner. All of of the work in the paper was a collaborative effort by our entire team.

Convolutional Latent Neural Processes. In chapter 4 we propose the Convolutional Latent Neural Process, which builds off of the previous chapter endowing latent neural processes with translation equivariance. The ConvLNP uses ConvCNPs as part of its architecture and extends ConvCNPs to allow for dependencies in the predictive distribution. This property enables ConvNPs to be deployed in settings which require coherent samples, such as Thompson sampling or conditional image completion. Moreover, we propose a new maximum-likelihood objective to replace the standard ELBO objective in latent neural processes, which conceptually simplifies the framework and empirically improves performance. We demonstrate the strong performance and generalization capabilities of ConvLNPs on 1D regression, image completion, and various tasks with real-world spatio-temporal data.

The content in this chapter is based on the publication 'Meta-Learning Stationary Stochastic Process Prediction with Convolutional Neural Processes' presented in the Conference on Neural Information Processing Systems, 2020 Foong et al. (2020). The research was conducted with the lead authors Andrew Y. K. Foong, Wessel P. Bruinsma, and Jonathan Gordon, collaborators Yann Dubois and myself, as well as our advisor Richard E. Turner. All of of the work in the paper was a collaborative effort by our entire team.

Gaussian Neural Processes. In chapter 5 we introduce the Gaussian neural process (GNP). GNPs directly parameterize a Gaussian predictive distribution via mean and covariance functions defined by neural process functions. The GNP is able to model joint output dependencies, like latent neural processes (Garnelo et al., 2018b). However, latent neural processes require approximate inference for training due to their latent variable. GNPs take advantage of the much more simple training procedure of the CNPs. We also extend GNPs by using invertible output transformations to capture non-Gaussian output distributions. We show that GNPs can be used in downstream estimation tasks which require dependent function samples and that, by accounting for output dependencies, GNPs achieve improved predictive performance on synthetic and real data. In particular, we demonstrate that GNPs outperform their CNP and LNP counterparts on real-world electroencephalogram (EEG) data. We also show that GNPs outperform a standard ensemble of widely used methods in a statistical downscaling, a climate modelling application, while providing spatially coherent temperature samples which are necessary for climate impact studies.

The content in this chapter is based on the publication 'Practical Conditional Neural Processes Via Tractable Dependent Predictions' presented in the International Conference on Learning Representations, 2022 Markou et al. (2021). The research was conducted with my co-lead author Stratis Markou and collaborators Wessel P. Bruinsma, and Anna Vaughan as well as our advisor Richard E. Turner. All of of the work in the paper was a collaborative effort by our entire team.

Autoregressive Neural Processes. In chapter 6, we again focus on developing a neural processes method for modelling rich predictive distributions. Instead of designing a new architecture to do so, we propose changing how CNPs are deployed at test time, without any modifications to the model or training procedure. Using a standard CNP, instead of making predictions independently for every target point, we autoregressively define a joint predictive distribution using the chain rule of probability, taking inspiration from the neural autoregressive density estimator (NADE) literature Uria et al. (2016). We show that this simple procedure allows factorised Gaussian CNPs to model highly dependent, non-Gaussian predictive distributions. In an extensive range of tasks with synthetic and real data, we show that CNPs in autoregressive (AR) mode not only significantly outperform non-AR CNPs, but are also competitive with more sophisticated models that are significantly more expensive and challenging to train. This performance is perhaps surprising since AR CNPs are not explicitly trained to model joint dependencies. To handle the high-resolution data in a computationally tractable manner, we introduce a multi-scale architecture for ConvCNPs. We also combine AR ConvCNPs with a beta-categorical mixture likelihood, producing strong results compared to other neural processes. Viewing the AR NP as a type of neural density estimator (Uria et al., 2016), we highlight their connections to a range of existing methods in the deep generative modelling literature.

The content in this chapter is based on the submission 'Autoregressive Conditional Neural Processes' under review for the International Conference on Learning Representations, 2023. The research was conducted with my co-lead authors Stratis Markou and Wessel P. Bruinsma, and collaborators Andrew Y. K. Foong, Anna Vaughan, Tom Andersson, and Anthony Buonomo as well as our advisors Scott Hosking and Richard E. Turner. All of the work in the paper was a collaborative effort by our entire team.

1.3 List of Publications

The following is a list of publications that I have co-authored during my PhD. Titles in bold indicate material contained in this thesis.

Peer-reviewed Conference Proceedings.

José Miguel Hernández-Lobato*, James Requeima*, Edward O. Pyzer-Knapp, Alán Aspuru-Guzik. 'Parallel and distributed Thompson sampling for large-scale

* indicates equal contribution.

accelerated exploration of chemical space.' In: International Conference on Machine Learning, 2017.

James Requeima*, Will Tebbutt*, Wessel Bruinsma*, Richard E. Turner. 'The Gaussian Process Autoregressive Regression Model (GPAR).' In: International Conference on Artificial Intelligence and Statistics, 2019.

James Requeima*, Jonathan Gordon*, John Bronskill*, Sebastian Nowozin, Richard E. Turner. 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes.' In: Conference on Neural Information Processing Systems, spotlight paper, 2019.

Jonathan Gordon*, Wessel Bruinsma*, Andrew Y. K. Foong, James Requeima, Yann Dubois, Richard E. Turner. '**Convolutional Conditional Neural Processes**.' In: International Conference on Learning Representations, 2020.

John Bronskill*, Jonathan Gordon*, James Requeima, Sebastian Nowozin, Richard E. Turner. 'TaskNorm: Rethinking Batch Normalization for Meta-Learning Meta-Learning.' In: International Conference on Learning Representations, 2020.

Andrew Y. K. Foong*, Wessel Bruinsma*, Jonathan Gordon*, Yann Dubois, James Requeima, Richard E. Turner '**Meta-Learning Stationary Stochastic Process Prediction with Convolutional Neural Processes**.' In: Neural Information Processing Systems, 2020.

Stratis Markou*, James Requeima*, Wessel Bruinsma, Anna Vaughan, Richard E. Turner. '**Practical Conditional Neural Processes Via Tractable Dependent Predictions**.' In: International Conference on Learning Representations, 2022.

Wessel P. Bruinsma*, Stratis Markou*, James Requeima*, Andrew Y. K. Foong, Anna Vaughan, Tom Andersson, Anthony Buonomo, Scott Hosking, Richard E. Turner. '**Autoregressive Conditional Neural Processes**.' In: International Conference on Learning Representations, 2023

Peer-reviewed Workshop Proceedings

Daniel Flam-Shepherd, James Requeima, David Duvenaud. 'Mapping Gaussian Process Priors to Bayesian Neural Networks.' In: NeurIPS Bayesian Deep Learning Workshop, 2017.

Daniel Flam-Shepherd, James Requeima, David Duvenaud. 'Characterizing and Warping the Function space of Bayesian Neural Networks.' In: NeurIPS Bayesian Deep Learning Workshop, 2018.

Wessel Bruinsma, James Requeima, Andrew Y. K. Foong, Jonathan Gordon, Richard E. Turner. 'The Gaussian Neural Process.' In: Advances in Approximate Bayesian Inference Symposium, 2020.

Stratis Markou*, James Requeima*, Wessel Bruinsma, Richard E. Turner. 'Efficient Gaussian Neural Processes for Regression.' In: ICML Uncertainty and Robustness in Deep Learning Workshop, 2021.

Ambrish Rawat, James Requeima, Wessel Bruinsma, Richard Turner. 'Challenges and Pitfalls of Bayesian Unlearning.' In: ICML Updatable Machine Learning Workshop, 2022

Tom R. Andersson, Wessel Bruinsma, Stratis Markou, Daniel C. Jones, J. Scott Hosking, James Requeima, Alejandro Coca-Castro, Anna Vaughan, Anna-Louise Ellis, Matthew Lazzara, Richard E. Turner. ' Active Learning with Convolutional Gaussian Neural Processes for Environmental Sensor Placement.' In: Workshop on Gaussian Processes, Spatiotemporal Modeling, and Decision-making Systems, NeurIPS 2022.

Chapter 2

Background

This chapter provides an introduction the main topics considered in this thesis. We begin by discussing meta-learning and stochastic processes, two motivating topics for neural processes. We then look at neural processes themselves and various members of the neural process family of models. These models will be the starting-off point for the work developed in this thesis and points of comparison for our experimentation.

2.1 Meta-Learning and Stochastic Processes

Most contemporary machine learning approaches train a model from scratch for a particular task using features and a learning algorithm designed by hand. This approach has worked very well especially with the advent of deep learning and in the presence of very large datasets (Goodfellow et al., 2016). However, one goal of machine learning is to automate decision making – to make decisions learning directly from data. Early success in machine learning utilized hand-engineered features (Domingos, 2012; Hospedales et al., 2021; Lowe, 2004). Deep neural networks are flexible approximating functions and part of the success of deep learning can be attributed to their ability to learn features and models directly from data (Krizhevsky et al., 2012). Meta-learning, often called *learning to learn*, can be viewed as the next logical step toward automating descision making: simultaneously learning the features, the model, and the learning algorithm directly from data (Hospedales et al., 2021).

Conventional ML improves model predictions over multiple data instances. Meta-learning improves a learning algorithm over multiple learning episodes, tasks, or datasets. If we have many related tasks that share some common structure, meta-learning attempts to gain information about the shared structure by training over multiple episodes. The training of meta-learning models is often framed as taking place in two stages: the so-called *inner loop* or *base learning* and the *outer loop* or *meta-training* (Hospedales et al., 2021). During the inner-loop, the algorithm learns to solve a task such solving a specific regression task (Krizhevsky et al., 2012) via optimizing a supervised regression loss. During the outer loop, the algorithm updates the inner learning algorithm in order to improve an outer objective. This improvement should lead to an inner learning algorithm that has better performance according to some metric such as generalization or speed of learning. Most current meta-learning algorithms define these two objectives as a meta-level objective that is end-to-end optimized.

This meta-learning framework can lead to models which perform better than those trained from scratch (Finn et al., 2017; Garnelo et al., 2018a; Requeima et al., 2019) and is better aligned with human and animal learning where it has been observed that learning strategies improve on both lifetime and evolutionary timescales (Biggs, 1985; Harlow, 1949; Schrier, 1984). Take few-shot learning (Lake et al., 2015) for image classification for example: the task of performing image classification after seeing a handful of training examples. Meta-learning approaches to few-shot learning like the one used by Requeima et al. (2019) learn a common set of features and learning algorithm to quickly adapt these features to a particular new task after training on repeated episodes. Anecdotally, I've noticed that my young twin toddlers now need very few examples to identify and classify objects – they have honed the learning algorithms required to perform this task after very few training examples having performed similar tasks many times before.

Meta-learning is related to transfer-learning (Zhuang et al., 2020), where knowledge gained in one domain is transferred to another for improved performance on the latter, and multi-task learning (Perez et al., 2018; Rebuffi et al., 2017, 2018), where a model is asked to perform well on multiple tasks. Meta-learning is a framework that can be used to attack either of these problem settings (Requeima et al., 2019). Meta-learning has seen success in few-shot image classification (Finn et al., 2017; Ravi and Larochelle, 2017; Requeima et al., 2019; Snell et al., 2017), hyperparameter optimization Franceschi et al. (2018), unsuperviserd learning (Metz et al., 2018), reinforcement learning (Alet et al., 2020) and neural architecture search (Liu et al., 2018a). In this thesis we focus on continuing the success of contemporary meta-learning methods but, in particular, with the goal of applying them to real-world problems such as climate modelling, population modelling, and medical science applications (see Sections 3.5, 4.5, 5.5 and 6.4).

2.1.1 Meta-Learning Problem Statement

Let $\mathcal{X}$ be a compact input space and let $\mathcal{Y}$ be the output space. Let $\mathcal{D}_N = (\mathcal{X} \times \mathcal{Y})^N$ be the collection of all sets of N input–output pairs, and let $\mathcal{D} = \bigcup_{N=0}^{\infty} \mathcal{D}_N$. We call elements $D \in \mathcal{D}$ *data sets* and denote $D = (\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x} \in \mathcal{X}^N$, $\boldsymbol{y} \in \mathcal{Y}^N$ are the inputs and outputs respectively. In meta-learning we are given a collection of data sets $(D_m)_{m=1}^M$, called a *meta–data set*, with the individual data sets D_m called *tasks* (Vinyals et al., 2016). Every task D_m is split up $D_m = D_m^{(\mathrm{c})} \cup D_m^{(\mathrm{t})}$ into a *context set* $D_m^{(\mathrm{c})} = (\boldsymbol{x}_m^{(\mathrm{c})}, \boldsymbol{y}_m^{(\mathrm{c})})$ and a *target set* $D_m^{(\mathrm{t})} = (\boldsymbol{x}_m^{(\mathrm{t})}, \boldsymbol{y}_m^{(\mathrm{t})})$. Here $\boldsymbol{x}_m^{(\mathrm{c})}$ are called the *context inputs*, $\boldsymbol{y}_m^{(\mathrm{c})}$ the *context outputs*, $\boldsymbol{x}_m^{(\mathrm{t})}$ the *target inputs*, and $\boldsymbol{y}_m^{(\mathrm{t})}$ the *target outputs*. Our goal is to devise an algorithm which takes in a context set $D_m^{(\mathrm{c})}$ and produces the best possible prediction for the target outputs $\boldsymbol{y}_m^{(\mathrm{t})}$ given target inputs $\boldsymbol{x}_m^{(\mathrm{t})}$.

In most real-world examples where we do not have access infinitely many tasks, we subsample our finite dataset D to form context-target pairs. For each iteration m of our training procedure, we sample a context set $D_m^{(c)}$ and target set $D_m^{(t)}$ from D and perform the procedure in Algorithm 1.

Algorithm 1 Meta-Training using Stochastic Gradient Descent and Finite Data.

```
Require: Finite Data $D$, parameters $\theta$, Loss function $\mathcal{L}(\theta, D^{(c)}, D^{(t)})$, Learning rate $\alpha$
1: while not converged do
2:     sample $D^{(c)}, D^{(t)}$ from $D$
3:     $\mathcal{L} \leftarrow \mathcal{L}(\theta, D^{(c)}, D^{(t)})$
4:     $\theta \leftarrow \alpha \nabla \theta \mathcal{L}$
5: end while
6: return Model parameters $\theta$
```

2.1.2 Stochastic Processes

For our purposes, a stochastic process (SP) on $\mathcal{X}$ will be defined[1] as a probability measure on the set of functions from $\mathcal{X} \to \mathcal{Y}$, i.e. $\mathcal{Y}^{\mathcal{X}}$, equipped with the product σ-algebra of the Borel σ-algebra over each index point (Tao, 2011), denoted Σ. The measurable sets of Σ are those which can be specified by the values of the function at a countable subset $I \subset \mathcal{X}$ of its input locations. We model the world as having a ground truth stochastic process $P \in \mathcal{P}(\mathcal{X})$ but in practice we only ever observe data at a finite number of points. We denote the set of all measures on $\mathcal{X}$ as $\mathcal{P}(\mathcal{X})$.

[1]Strictly speaking, this is non-standard terminology, since P is the *law* of a stochastic process. See Ross et al. (1996) or Tao (2011) for a more thorough introduction to stochastic processes.

The approach we will take in this thesis to define a stochastic process is via its finite-dimensional marginal distributions (Garnelo et al., 2018b); we view the process as a random function $F : \mathcal{X} \to \mathcal{Y}$ and for each finite sequence of inputs $x_{1:n} = (x_1, \ldots, x_n)$ we define the marginal distribution over the output function values $y_{1:n} = (F(x_1), \ldots, F(x_n))$. Inspired by Gaussian processes (Rasmussen and Williams, 2006), this is a natural approach; a GP defines all marginal joint distributions as multivariate Gaussians via its mean and covariance functions. The Kolmogorov extension theorem gives us a way of ensuring that our collection of marginal distributions form a stochastic process since, in general, it is not guaranteed.

Theorem 1 (Kolmogorov Extension Theorem (Oksendal, 2013))**.** Let $\mathcal{X}$ be an input space and let $\mathcal{Y}$ be an output space. Let $\{p\}$ be a collection of densities such that for all $n \in \mathbb{N}$, for each finite sequence of distinct variables $x_{1:n} = (x_1, \ldots, x_n)$ from $\mathcal{X}$ there is a unique density $p_{x_1,\ldots,x_n}(y_1, \ldots, y_n)$ in this collection. If $\{p\}$ satisfies

1. **Exchangeability:** For each $n \in \mathbb{N}$, if π is a permutation of $1, \ldots, n$, then:

$$p_{x_1,\ldots,x_n}(y_1, \ldots, y_n) = p_{x_{\pi(1)},\ldots,x_{\pi(n)}}(y_{\pi(1)}, \ldots, y_{\pi(n)}) \tag{2.1}$$

2. **Consistency:** If $1 \leq m \leq n$ then

$$p_{x_1,\ldots,x_m}(y_1, \ldots, y_m) = \int p_{x_1,\ldots,x_n}(y_1, \ldots, y_n)\, \mathrm{d}y_{m+1} \ldots \mathrm{d}y_n \tag{2.2}$$

then there exists a unique measure on $(\mathcal{Y}^{\mathcal{X}}, \Sigma)$ that has these densities as its finite-dimensional distributions. If $\{p\}$ satisfies the above two conditions then we call it *Kolmogorov-consistent*.

In this thesis, we will model the world as having a ground truth stochastic process $P \in \mathcal{P}(\mathcal{X})$.

Consider a Kolmogorov-consistent (i.e. consistent under marginalization) collection of distributions on finite index sets $I \subset \mathcal{X}$. By the Kolmogorov extension theorem, there exists a unique measure on $(\mathbb{R}^{\mathcal{X}}, \Sigma)$ that has these distributions as its finite marginals. Hence we may think of these stochastic processes as defined by their finite-dimensional marginals.

2.1.3 Prediction Map Formulation

Consider rainfall y as a function of position x. To model rainfall, we can view it as a *random* function from $\mathcal{X}$ to $\mathcal{Y}$. Mathematically, this corresponds to a SP on $\mathcal{X}$—a probability distribution over functions from $\mathcal{X}$ to $\mathcal{Y}$—which we denote by P. Given perfect knowledge of P, we could predict rainfall at any location of interest by conditioning P on observations D_c, yielding a *predictive* SP. However, in practice we will only have access to a large collection of sample functions from P. Each function is known only at a finite set of inputs, $D = (\boldsymbol{x}_k, \boldsymbol{y}_k)_{k=1}^{K}$, which we divide into $D^{(c)}, D^{(t)}$ for meta-training. Given sufficient data, we can *meta-learn* the map from context sets $D^{(c)}$ to the ground-truth predictive distribution: $D^{(c)} \mapsto p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})$.

Formally, Let $p(\boldsymbol{y}|\boldsymbol{x})$ denote the density with respect to Lebesgue measure of the finite marginal of the stochastic process P with index set $\boldsymbol{x}$. Assume we have observed P at a finite number of points $(\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)})$, with $p(\boldsymbol{y}^{(c)}|\boldsymbol{x}^{(c)}) > 0$. Let $\boldsymbol{x}^{(t)}$ be another finite index set. Then we define the finite marginal at $\boldsymbol{x}^{(t)}$ conditioned on $D^{(c)}$ as the distribution with density

$$p(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)}) = \frac{p(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)}|\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)})}{p(\boldsymbol{y}^{(c)} \mid \boldsymbol{x}^{(c)})}. \tag{2.3}$$

It can easily be verified that for a fixed $D^{(c)}$, the conditional marginal distributions for different $\boldsymbol{x}^{(t)}$ in Equation (2.3) are Kolmogorov-consistent. Again, the Kolmogorov extension theorem implies there is a unique measure $P_{D^{(c)}}$ on $(\mathcal{Y}^{\mathcal{X}}, \Sigma)$ that has Equation (2.3) as its finite marginals. We now define the *prediction map*, so called because it maps each observed dataset $D^{(c)}$ to the exact predictive stochastic process conditioned on $D^{(c)}$. The meta-learning task may be viewed as learning an approximation to the prediction map.

Definition 1 (Prediction map)**.** A *prediction map* π is a map from sets to SPs, $\pi_P : \mathcal{S} \to \mathcal{P}(\mathcal{X})$, $\pi_P(D^{(c)}) = P_{D^{(c)}}$.

2.1.4 Gaussian Processes

Gaussian Processes (GPs) (Rasmussen and Williams, 2006) are a nonparametric model from which neural processes, the model we examine in the next section, take inspiration. GPs define a stochastic process on $\mathcal{Y}$ via a mean function $\mu : \mathcal{X} \to \mathcal{Y}$, a covariance (or kernel) function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and a likelihood function $p : \mathcal{Y} \to \mathbb{R}^+$. Together, for any index locations $\boldsymbol{x}^{(t)}$, they parametrize:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \int \mathcal{N}\left(f \mid \mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x})\right) p(\boldsymbol{y} \mid f, \boldsymbol{x}) \, \mathrm{d}f \tag{2.4}$$

where $k(\boldsymbol{x}, \boldsymbol{x})$ is the covariance matrix generated by evaluating $k(x_i, x_j)$ for every pair x_i, x_j in $\boldsymbol{x}$.

It can be shown using Bayes rule on equation 2.4 that, if the likelihood function is also a Gaussian, the output of the GP prediction map is also a GP (Rasmussen and Williams, 2006). For a GP with zero mean function we get the prediction map π_{GP} mapping to the GP posterior $p(f \mid \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)})$ with mean function μ_{post} and covariance functions k_{post}:

$$\begin{aligned} \mu_{\text{post}}(\boldsymbol{x}^{(t)}) &= k(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}) k(\boldsymbol{x}^{(c)}, \boldsymbol{x}^{(c)})^{-1} \boldsymbol{y}^{(c)} & (2.5) \\ k_{\text{post}}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}) &= k(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}) - k(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}) k(\boldsymbol{x}^{(c)}, \boldsymbol{x}^{(c)})^{-1} k(\boldsymbol{x}^{(c)}, \boldsymbol{x}^{(t)}) & (2.6) \end{aligned}$$

Computing the posterior in classical GPs is expensive, scaling with $O\left((M+N)^3\right)$ where M and N are the number of observations and targets. Another downside to GPs is that they require the user to explicitly specify prior information about the problem via the kernel and mean functions which, if done poorly, will induce the wrong inductive bias.

2.2 Neural Processes

In this section we examine various members of the neural process family of meta-learning models. Neural Processes (NPs) are an attractive family of models producing well-calibrated[2] marginal predictions and enable fast inference at test time. At their core, CNPs use an encoder/decoder architecture, similar to a Variational Autoencoder (Kingma and Welling, 2014) and are meta-trained to map context sets to conditional predictive distribution. Two key properties of NPs are that they use a neural network architecture to:

1. define a valid stochastic process and to

[2]In this thesis, calibration refers to the statistical consistency between distributional forecasts and observations from the true generative distribution (which is assumed to exist). This calibration will generally be measured using the strictly proper scoring function $\log p$ which is maximized when the distributional forecaster is equal to the generative distribution (Gneiting and Raftery, 2007).

2. take data sets as inputs and return a prediction at any desired input location.

NPs take inspiration from De Finetti's Theorem for the former and rely on the DeepSets neural network architecture for the latter.

2.2.1 Defining a Stochastic Process

De Finetti's theorem (De Finetti, 1937) states that every exchangeable process is a mixture of i.i.d. processes:

$$p(y_1, \ldots, y_n) = \int p(z) \prod_{i=1}^{n} p(y_i \,|\, z) \, \mathrm{d}z \tag{2.7}$$

Where z is a global latent variable conditioned on which the variables y_i are i.i.d.. This formulation gives us a way to construct a stochastic process and highlights a connection between exchangeability and Bayesian statistics: we simply need to define a prior $p(z)$ and a collection of likelihoods $p_{x_i}(y_i \,|\, z)$ for all $x_i \in \mathcal{X}$. As discussed above, this construction defines all finite marginals and its conditional independence assumption ensures that both the prior and posterior predictive distribution are Kolmogorov-consistent. By computing the posterior $p(z \,|\, D^{(c)})$, we get the prediction map

$$\begin{aligned} \pi(D^{(c)})(y_1, \ldots, y_n) &= p(y_1, \ldots, y_n \,|\, D^{(c)}) && (2.8) \\ &= \int p(z \,|\, D^{(c)}) \prod_{i=1}^{n} p(y_i \,|\, z) \, \mathrm{d}z && (2.9) \end{aligned}$$

However, for our meta-learning problem set-up, it is not strictly necessary to define prior predictive since we are looking for an approximation to the prediction map. Instead, neural processes define the collection of finite marginals $p_\theta(y_i \,|\, z)$ and the posterior $p_\theta(z \,|\, D_c)$ functions directly using neural networks. Below, we look at the architecture neural processes use to define the component $p_\theta(z \,|\, D^{(c)})$.

2.2.2 Deep Sets

In order to define the component $p_\theta(z \,|\, D^{(c)})$ we need to define an architecture capable of taking arbitrary sized sets as input and to be permutation invariant on sets. We say that f is a valid function on sets if it satisfies the following:

Property 1 ($\mathbb{S}_n$-invariant and $\mathbb{S}$-invariant functions)**.** Let $\mathcal{X}$ be an input space and let $\mathcal{X}^*$ be the Kleene closure of $\mathcal{X}$ – the set of all finite sequences of elements of $\mathcal{X}$. Let

$f : \mathcal{X}^* \to \mathcal{Y}$ be a function that takes arbitrary sized finite sequences of elements of $\mathcal{X}$ as inputs. If

$$f(X_n) = f(\pi X_n) \text{ for } n \in \mathbb{N}, \text{ and all } \pi \in \mathbb{S}_n, X_n \in \mathcal{X} \tag{2.10}$$

where $\mathbb{S}_n$ is the set of permutations on n elements for $n \in \mathcal{N}$, then we say that f is $\mathbb{S}_n$-invariant. We say that f is an $\mathbb{S}$-invariant function if $f|_{\mathcal{X}_n^*}$, the restriction of f to datasets X_n, is $\mathbb{S}_n$-invariant for all $n \in \mathbb{N}$. We will equivalently call $\mathbb{S}$-invariant functions *permutation invariant*.

Zaheer et al. (2017a) develop an architecture defining neural network approximators that satisfy property 1 which they call *DeepSets*. Throught this thesis, we rely heavily on the concept of DeepSets as the main tool allowing us to parametrize a function mapping from an arbitrarily sized set to predictive distribution. A DeepSet $f : \mathcal{X}^* \to \mathbb{R}$ is constructed by composing

$$f(X) = \rho(E(X)), \;\; E(X) = \sum_{x \in X} \phi(x) \tag{2.11}$$

where $\rho : \mathcal{X} \to \mathbb{R}^d$ and $\phi : \mathbb{R}^d \to \mathbb{R}$ are neural networks. In their work, Zaheer et al. (2017a) prove the following *universal representation* theorems on DeepSets:

Theorem 2 (DeepSets: countable domain case)**.** Let $\mathcal{X}$ be countable. A function $f : \mathcal{X}^* \to \mathbb{R}$ is $\mathbb{S}$-invariant if and only if it can be decomposed as

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right) \tag{2.12}$$

where ρ and ϕ are continuous functions.

Theorem 3 (DeepSets: uncountable domain case)**.** Let $\mathcal{X} = [0, 1]$, $n \in \mathbb{N}$, and $f : \mathcal{X}^n \to \mathbb{R}$. Then f is $\mathbb{S}_n$-invariant if and only if it can be decomposed as

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right) \tag{2.13}$$

where ρ and ϕ are continuous functions.

Note that theorems 2 and 3 do not completely characterize approximating functions on sets as they are either restricted to either countable domains or fixed-sized sets. For

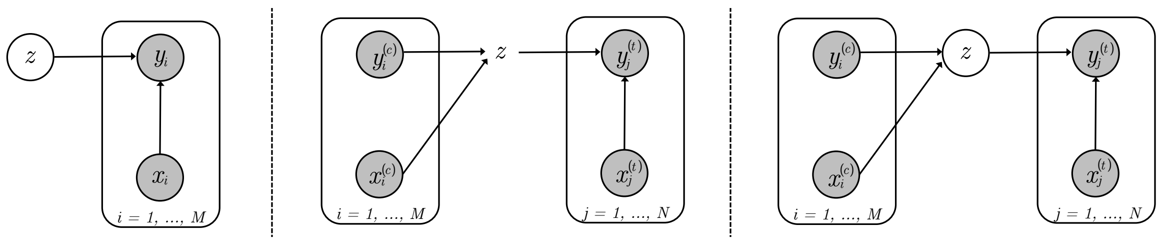


Fig. 2.1 Graphical model for the LNP with no distinction between context and target data (left). Graphical model for the CNP (centre). Graphical model for the LNP interpreting the LNP objective as a valid ELBO (right). The only distinction between the centre and right figures is the treatment of the variable z. All graphical models depict a single dataset or task.

most machine learning applications and, in particular the ones examined in this thesis, we are concerned with functions that take uncountable sets or varying size as inputs and these theorems offer no guarantees on the form of such functions. However, Bloem-Reddy and Teh (2020) extend theorem 3 to varying-sized sets on arbitrary spaces. In Chapter 3 we provide a representation theorem for translation-equivariant functions on sets, extending a key result of Zaheer et al. (2017b) to functional embeddings, including sets of varying size.

2.2.3 Conditional Neural Processes

The simplest examples of a neural processes are conditional neural processes (CNPs) (Garnelo et al., 2018a). CNPs are a class of meta-learning models which leverage the flexibility of deep learning to define a stochastic process on target sets and produce well-calibrated uncertainty. CNPs are end-to-end trained using a simple-to-implement maximum-likelihood procedure. CNPs combine neural networks with features from GPs but note that the architectures considered by Garnelo et al. (2018a) scale as $O(M + N)$ at test time compare to inference required when using classical GPs which scale with $O\left((M + N)^3\right)$ where M and N are the number of observations and targets. As such, CNPs also require significantly less computation and memory than other meta-learning approaches at test time, such as gradient-based fine tuning (Finn et al., 2017; Triantafillou et al., 2019), making them ideal for resource and power-limited applications, such as mobile devices.

CNPs make *factorised* predictions for $\boldsymbol{y}^{(t)}$ using a construction similar to that in equation 2.7. Specifically, given context set $D^{(c)}$ and target set $D^{(t)} = \{\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}\}$ CNPs define the predictive distribution:

$$p_\theta\left(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)}\right) = \int p_\theta\left(z \mid D^{(c)}\right) \prod_{i=1}^{N} p_\theta(y_i^{(t)} \mid x_i^{(t)}, z)\, \mathrm{d}z \tag{2.14}$$

where θ are the network parameters. Instead of parameterizing a distribution, $p_\theta\left(z \mid D^{(c)}\right)$ uses a DeepSet architecture to map the context set to a point estimate of the latent vector $z^{(c)} = f_\theta(D^{(c)})$ defined by the architecture:

$$z_i^{(c)} = h_\theta(x_i, y_i) \text{ for all } x_i, y_i \in D^{(c)} \tag{2.15}$$
$$z^{(c)} = z_1^{(c)} \oplus z_2^{(c)} \oplus \ldots \oplus z_n^{(c)} \tag{2.16}$$

where $h_\theta : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a neural network and $\oplus$ is a commutative operation mapping to a single element $\mathbb{R}^d$, usually the sum or mean operation. The operation in equation 2.16 is often called the *aggregator*. This point estimate can be interpreted as setting $p_\theta\left(z \mid D^{(c)}\right) = \delta\left(z - f_\theta(D^{(c)})\right)$ in equation 2.14. See Figure 2.1 for the graphical model. The intention is for f_θ to map the context relevant task information contained in $D^{(c)}$ to the vector representation such that the remaining CNP architecture is able to condition on this specific task. The network f_θ is often referred to as the *encoder*, mirroring the terminology used in Variational Auto-encoders (Kingma and Welling, 2014). For regression tasks, the *decoder* g_θ is network that takes this task representation $z^{(c)}$ and the target locations $\boldsymbol{x}^{(t)}$ as inputs and parametrizes a distribution which factors over the target points $\boldsymbol{y}^{(t)}$. Typically, g_θ parametrizes a Gaussian distribution via neural networks μ_θ and σ_θ^2:

$$p_\theta(y_i^{(t)} \mid x_i^{(t)}, z) = \mathcal{N}\left(y_i^{(t)} \mid \mu_\theta(x_i^{(t)}, z), \sigma_\theta^2(x_i^{(t)}, z)\right) \tag{2.17}$$

Combining the encoder and decoder networks, CNPs define the predictive distribution via a forward pass through the networks:

$$p_\theta\left(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)}\right) = \prod_{x_i \in \boldsymbol{x}^{(t)}} \mathcal{N}\left(y_i^{(t)} \mid \mu_\theta(x_i^{(t)}, f_\theta(D^{(c)})), \sigma_\theta^2(x_i^{(t)}, f_\theta(D^{(c)}))\right) \tag{2.18}$$

Figure 2.2 illustrates the architecture for a CNP.

For classification tasks, g_θ instead parametrizes the logits of the class probabilities of a categorical distribution. As discussed in 2.2.1, this construction is Kolmogorov-consistent and thus defines a stochastic process.

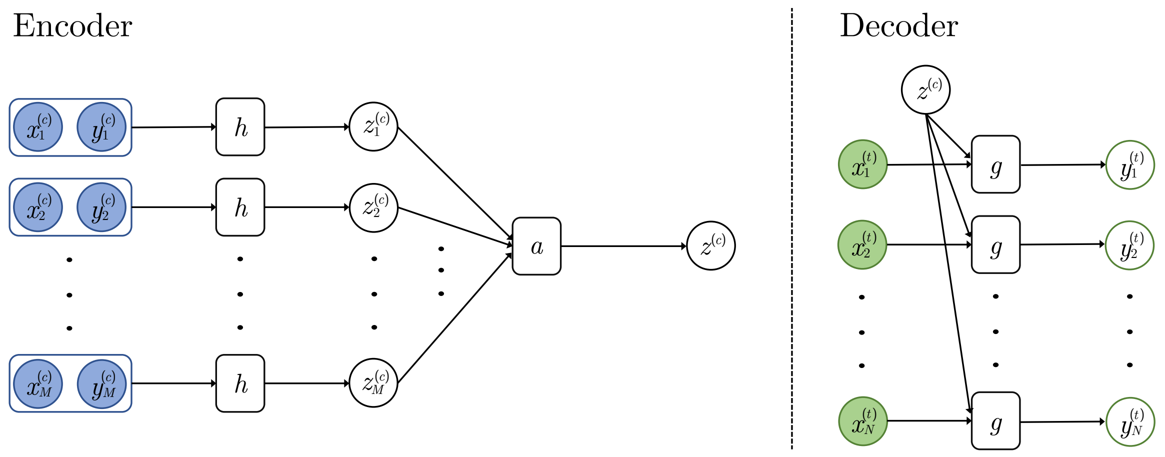


Fig. 2.2 Computation diagram for the CNP. The context set and target set are indicated by blue and green variables respectively with shaded variables being observed. Boxes in the diagram are learned components of the CNP architecture.

As noted by Garnelo et al. (2018a), one of the advantages of meta-training CNPs to target prediction maps is that it shifts the burden of imposing prior knowledge from user defined analytic priors to empirical data.

Training CNPs. Let $(D_m)_{m=1}^M$ be a collection of tasks with context and target splits $D_m = D_m^{(\mathrm{c})} \cup D_m^{(\mathrm{t})}$, and with $D_m^{(\mathrm{c})} = (\boldsymbol{x}_m^{(\mathrm{c})}, \boldsymbol{y}_m^{(\mathrm{c})})$ and $D_m^{(\mathrm{t})} = (\boldsymbol{x}_m^{(\mathrm{t})}, \boldsymbol{y}_m^{(\mathrm{t})})$ as defined in 2.1.1. Note that the context and target datasets may or may not be disjoint depending on the training protocol.

We define

$$\mathcal{L}_{ML}(\theta, D_m) := \log p_\theta \left(\boldsymbol{y}_m^{(\mathrm{t})} \,|\, \boldsymbol{x}_m^{(\mathrm{t})}, D_m^{(\mathrm{c})} \right) \tag{2.19}$$

Assuming the tasks are drawn from some task distribution $p(D)$, we can approximate the expectation over tasks of the conditional log-likelihood of a CNP:

$$\int \mathcal{L}_{ML}(\theta, D)\, \mathrm{d}D \approx \frac{1}{M} \sum_{m=1}^{M} \log p_\theta \left(\boldsymbol{y}_m^{(\mathrm{t})} \,|\, \boldsymbol{x}_m^{(\mathrm{t})}, D_m^{(\mathrm{c})} \right) \tag{2.20}$$

In practice, CNP networks are implemented in automatic differentiation software packages such as PyTorch (Paszke et al., 2017), TensorFlow (Abadi et al., 2016), or JAX (Bradbury et al., 2018). The parameters of the networks θ are then learned by optimizing equation 2.20 using stochastic gradient descent (Bottou, 2010).

Limitations of CNPs: Translation Equivariance. Translation equivariance is an important inductive bias for many learning problems including time series modelling, spatial data, and images. Famously, convolutional neural networks added translation

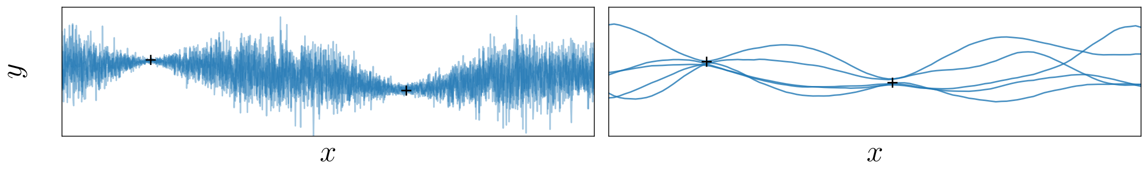


Fig. 2.3 Samples from a CNP (left) which makes independent predictions, and a Gaussian process (right) designed to generate function samples (blue) which are coherent. Coherent samples are often necessary for downstream estimation tasks.

equivariance to standard multilayer perceptrons (Cohen and Welling, 2016; LeCun et al., 1998). Unfortunately, it is not straightforward to incorporate CNNs into CNPs to give us a translation equivariant model. We will discuss translation equivariance and its connection to CNPs further in Chapter 3.

Limitations of CNPs: Coherent Samples. Despite the favourable qualities CNPs exhibit, they are severely limited by the fact that they do not model dependencies in their output. More specifically, given two target input locations x_m and $x_{m'}$, CNPs model their respective outputs y_m and $y_{m'}$ independently. In cases with Gaussian likelihoods, the result is that any two function samples will be independent noise added to the same mean function. We refer to such predictions as *mean-field*. The inability to model dependencies hurts the predictive performance of CNPs and renders it impossible to produce coherent function samples. Since many downstream tasks require dependent function samples, this excludes mean-field CNPs form a range of applications. In heatwave or flood prediction for example, we need to evaluate the probability of the event that the temperature or precipitation remains above some threshold, throughout a region of space and time. As illustrated by Figure 2.3, since mean-field predictions model every location independently they may assign unreasonably low probabilities to such events. If we were able to draw coherent samples from the predictive, the probabilities of such events and similar useful quantities could be more reasonably estimated.

Limitations of CNPs: AR consistency. Generally, when generating training data, context and target sets are drawn from the same pool of data – there is no meaningful distinction between the two (see figure 2.1). However, the CNP modelling assumptions, represented by the graphical model in figure 2.1, treat the context and target data as separate random variables. A consequence of this assumption is that CNPs are not *autoregressively consistent* (AR-consistent) which we describe below. Suppose we have two target points $(x_1^{(\mathrm{t})}, y_1^{(\mathrm{t})})$, $(x_2^{(\mathrm{t})}, y_2^{(\mathrm{t})})$ and context set $D^{(\mathrm{c})}$. CNPs specify that the correct way to evaluate the conditional joint distribution at the target points is:

$$p_\theta\left(y_1^{(\mathrm{t})}, y_2^{(\mathrm{t})} \,|\, x_1^{(\mathrm{t})}, x_2^{(\mathrm{t})}, D^{(c)}\right) = p_\theta\left(y_1^{(t)} \,|\, x_1^{(t)}, f_\theta(D^{(c)})\right) \cdot p_\theta\left(y_2^{(t)} \,|\, x_2^{(t)}, f_\theta(D^{(c)})\right). \quad (2.21)$$

Under the product rule of probability we can write:

$$p_\theta\left(y_1^{(\mathrm{t})}, y_2^{(\mathrm{t})} \,|\, x_1^{(\mathrm{t})}, x_2^{(\mathrm{t})}, D^{(c)}\right) = p_\theta\left(y_1^{(\mathrm{t})} \,|\, y_2^{(\mathrm{t})}, x_1^{(\mathrm{t})}, x_2^{(\mathrm{t})}, D^{(c)}\right) \cdot p_\theta\left(y_2^{(\mathrm{t})} \,|\, x_2^{(\mathrm{t})}, D^{(c)}\right). \quad (2.22)$$

The conditional independence assumption of CNPs gives us that

$$p_\theta\left(y_1^{(\mathrm{t})} \,|\, y_2^{(\mathrm{t})}, x_1^{(\mathrm{t})}, x_2^{(\mathrm{t})}, D^{(c)}\right) = p_\theta\left(y_1^{(\mathrm{t})} \,|\, x_1^{(\mathrm{t})}, D^{(c)}\right) \quad (2.23)$$

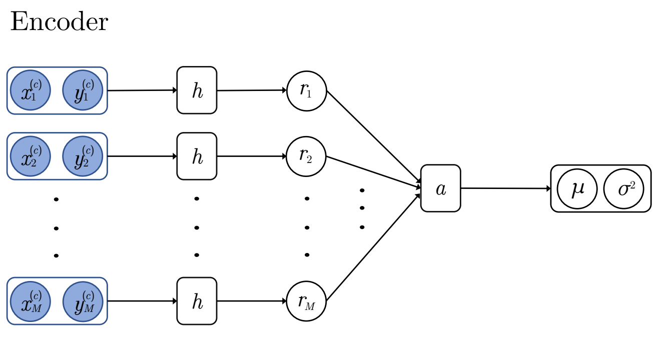
However, if context and target data are fundamentally the same random variable under our model, from equation 2.22 we should be able to evaluate:

$$\begin{aligned} &p_\theta\left(y_1^{(\mathrm{t})}, y_2^{(\mathrm{t})} \,|\, x_1^{(\mathrm{t})}, x_2^{(\mathrm{t})}, D^{(c)}\right) \\ &\qquad = p_\theta\left(y_2^{(t)} \,|\, x_2^{(t)}, f_\theta(D^{(c)} \cup \{(x_1^{(\mathrm{t})}, y_1^{(\mathrm{t})})\})\right) \cdot p_\theta\left(y_1^{(t)} \,|\, x_1^{(t)}, f_\theta(D^{(c)})\right). \quad (2.24)\end{aligned}$$

This type of autoregressive evaluation is a way to bootstrap dependencies between outputs and potentially induce coherent samples. Otherwise, any joint information for outputs must be solely contained in the point estimate for z. Unfortunately, the distribution generated by this autoregressive evaluation is not guaranteed to be (and generally is not) consistent under permutation and marginalization thus violating Kolmogorov-consistency, meaning that the distributions do not define a valid stochastic process. We return to examine the AR perspective of CNPs in chapter 6.

2.2.4 Latent Neural Processes

Conditional neural processes are unable to produce coherent samples or model epistemic uncertainty due to the point estimate of the global latent variable z in combination with the conditional independence assumption in the graphical model. Latent neural processes (LNPs; Garnelo et al., 2018b) are a generalization of conditional neural processes that address these issues by estimating a conditional distribution over the latent variable z in equation 2.14. This latent variable is able to capture global uncertainty allowing us to sample at a global level – function samples are produced by sampling from $p_\theta(z \,|\, D^{(c)})$ and observation noise at each target location is generated by the likelihoods $p_\theta(y^{(t)} \,|\, x^{(t)}, z)$. Garnelo et al. (2018b) propose viewing latent



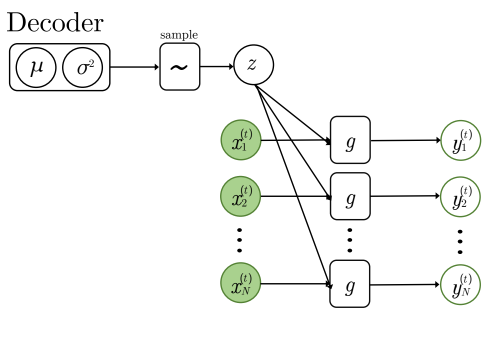


Fig. 2.4 Computation diagram for the LNP. The context set and target set are indicated by blue and green variables respectively with shaded variables being observed. Boxes in the diagram are learned components of the CNP architecture and $\sim$ indicates a sampling procedure to produce samples of the latent variable z.

neural processes as performing amortized approximate Bayesian inference and learning (Kingma and Welling, 2014) in the following latent variable model, illustrated in Figure 2.1:

$$z \sim p_{\boldsymbol{\theta}}(z); \quad y(x) = f_{\boldsymbol{\theta}}(x; z); \tag{2.25}$$

LNPs use and encoder/decoder architecture similar to that of CNPs illustrated in figure 2.4:

- The *encoder* h_θ is a neural network that maps context pairs $(x_i^{(c)}, y_i^{(c)})$ to a vector representation $r_i^{(c)} = h_\phi(x_i^{(c)}, y_i^{(c)})$. The *aggregator* operation then summarises the encoded inputs into a single global representation $r^{(c)} = r_1^{(c)} \oplus r_2^{(c)} \oplus \ldots \oplus r_n^{(c)}$ where $\oplus$ is a commutative operation mapping to a single element $\mathbb{R}^d$, usually the sum or mean operation. Note that the encoder and aggregator are ρ and ϕ from equation 2.11 comprising a DeepSet architecture. Together, they are generally referred to as just the *encoder*, mirroring the terminology used by CNPs.

- The global representation $r^{(c)}$ is used to parametrize the distribution $q_\phi(z \mid D^{(c)}) = \mathcal{N}\left(z \mid \mu(r^{(c)}), \sigma^2(r^{(c)})\right)$. This is often called the *amortized inference network*.

- The *(conditional) decoder* takes sampled global variables $z \sim q_\phi(z \mid D^{(c)})$ and target locations and parametrizes $p_\theta(y_i^{(t)} \mid x_i^{(t)}, z) = \mathcal{N}\left(y_i^{(t)} \mid g_\theta(x_i^{(t)}, z), \sigma^2\right)$ via the neural network g_θ.

Training LNPs. Due to the intractability of $p_\theta(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(t)})$ an approximate inference method needs to be used during training. Garnelo et al. (2018b) propose using amortized variational inference (Kingma and Welling, 2014; Rezende and Mohamed, 2015) to train the LNP end-to-end. For a task D with context and target splits $D = D^{(\mathrm{c})} \cup D^{(\mathrm{t})}$ the standard evidence lower-bound (ELBO) is given by:

$$\log p_\theta(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)} \mid \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}) \geq \mathbb{E}_{q_\phi(z \mid D)} \left[\log p_\theta(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)} \mid z, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}) + \log \frac{p_\theta(z)}{q_\phi(z \mid D)} \right] \tag{2.26}$$

where q_ϕ is the amortized variational posterior parametrized by a neural network. In order to train an LNP using this objective, we would need an analytic prior $p_\theta(z)$, shifting the burden of incorporating prior knowledge back onto the user instead of learning from empirical data. An alternative objective that better reflects the model use at test time is the conditional ELBO:

$$\log p_\theta(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)}) \geq \mathbb{E}_{q_\phi(z \mid D)} \left[\log p_\theta(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, z) + \log \frac{p_\theta(z|D^{(c)})}{q_\phi\left(z \mid D\right)} \right]. \tag{2.27}$$

However, this objective still requires access to the (intractable) conditional prior $p_\theta(z|D^{(c)})$. Garnelo et al. (2018b) propose replacing this factor with the variational posterior $q_\phi\left(z \mid D^{(c)}\right)$, which gives,

$$\log p_{\theta,\phi}(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)}) \geq \mathbb{E}_{q_\phi(z \mid D)} \left[\log p_\theta(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, z) + \log \frac{q_\phi(z|D^{(c)})}{q_\phi\left(z \mid D\right)} \right]. \tag{2.28}$$

A first glance, there is no guarantee that this inequality holds once we replace one of the factors with an approximation. However, let us reinterpret our model in the following way: for any context set $D^{(c)}$ define the prior distribution over z using the variational posterior $q_\phi(z \mid D^{(c)})$, illustrated in Figure 2.1. In this case, equation 2.28 is a valid evidence lower-bound for a different model for each context and target set pair, highlighted by writing the log-likelihood as $\log p_{\theta,\phi}$. We define:

$$\mathcal{L}_{LNP}(\theta, \phi, D) = \mathbb{E}_{q_\phi(z \mid D)} \left[\log p_\theta(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, z) + \log \frac{q_\phi(z|D^{(c)})}{q_\phi\left(z \mid D\right)} \right]. \tag{2.29}$$

Again, assuming the collection of tasks $(D_k)_{k=1}^K$ are drawn from some task distribution $p(D)$, we can approximate the expectation of the lower bound in equation 2.28 over $p(D)$ using Monte-Carlo integration as in Equation (2.20). Meta-training by maximizing equation 2.29 asks q_ϕ to simultaneously be a good amortized prior network $q_\phi\left(z \mid D^{(c)}\right)$ and a good amortized posterior $q_\phi\left(z \mid \boldsymbol{y}^{(t)}, \boldsymbol{x}^{(t)}, D^{(c)}\right)$ when the former means we are constantly redefining our model during training and we really

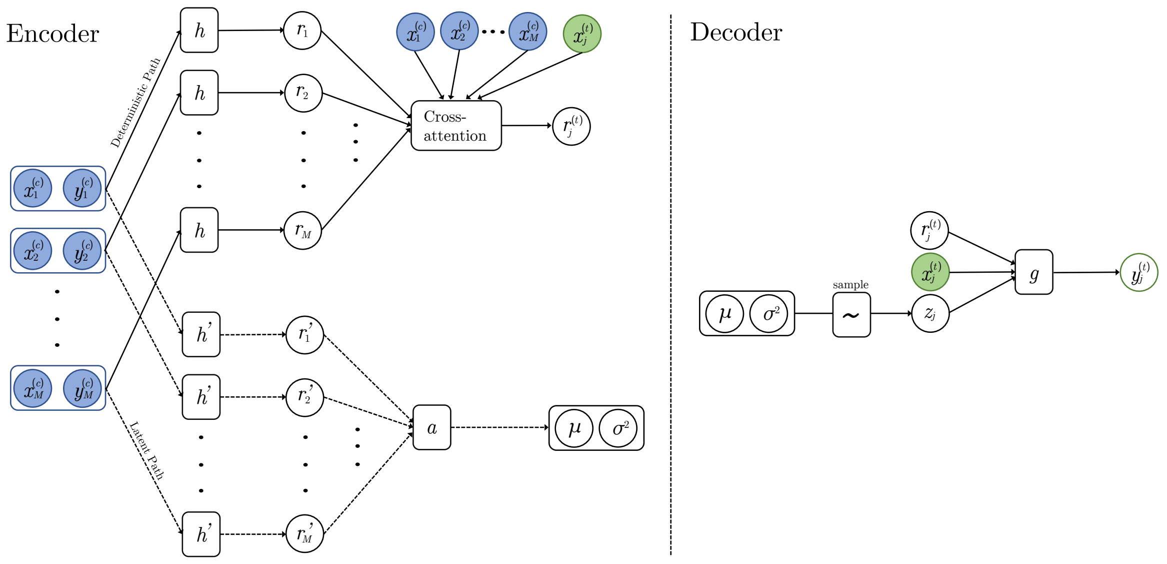


Fig. 2.5 Computation diagram showing the prediction for a single target point $x_j^{(t)}$ for the ALNP. The context set is indicated by blue and a single target point is indicated by the green variable with shaded variables being observed. Boxes in the diagram are learned components of the CNP architecture and $\sim$ indicates the sampling procedure to produce samples of the latent variable z.

only care about its performance as the latter. It also has the potential to overfit the prior to the posterior if too few tasks are seen during meta-training.

LNPs allow for coherent samples and a provide a way of modelling global uncertainty – two important features missing from CNPs. In exchange, LNPs give up the simple maximum likelihood training procedure of CNPs at the cost of the more difficult variational inference using the modified ELBO objective. We return to the discussion of training LNPs in Sections 4.3.2 and 4.4.1.

2.2.5 Attentive Neural Processes

The representational power of CNPs scales poorly with the capacity of the encoder and decoder resulting in models that tent to underfit (Gordon et al., 2020; Kim et al., 2019; Louizos et al., 2019). An important point of comparison that will be considered throughout this thesis is the attentive neural process (ANP) which addresses this issue. Kim et al. (2019) introduce two variants of the attentive neural process, a conditional version that we will call the attentive conditional neural process (ACNP) and a latent variable version that we will call the attentive latent neural process (ALNP). The ACNP uses an *attention mechanism* (Bahdanau et al., 2014; Vaswani et al., 2017; Xu et al., 2015) to replace the general context point representations $r = f_\theta(D^{(c)})$ with *target-specific* context point representations $r'_j = f_\theta(D^{(c)}, x_j^{(t)})$ for each $x_j^{(t)} \in D^{(t)}$. Let $r_i = h_\theta(x_i^{(c)}, y_i^{(c)}), i = 1, \ldots M$ be the context point encodings defined in equation 2.15. Then for each target point $x_j \in D^{(t)}$ we define:

$$r'_j = \sum_{i=1}^{M} \frac{w_\theta(x_j^{(t)}, x_i^{(c)}) r_i}{\sum_{k=1}^{M} w_\theta(x_j^{(t)}, x_k^{(c)})} \tag{2.30}$$

where w_θ is an attention function with learned parameters θ. Notice that setting $w_\theta(x_j^{(t)}, x_i^{(c)}) = 1$ recovers the standard CNP. Kim et al. (2019) show that using *self attention* layers (Parmar et al., 2018; Vaswani et al., 2017) improves performance of the ACNP.

The attentive latent neural process is the latent variable analog of the ACNP. However, instead of replacing the point estimate of the variable r with a distribution $p(r \mid D^{(c)})$ mirroring the relationship between CNPs and LNPs, Kim et al. (2019) add a *latent variable path* to the architecture parametrizing a conditional distribution $p_\theta(z \mid D^{(c)})$ for an additional, but *target-independent*, latent variable z. In this case, the part of the architecture producing r_j defined in equation 2.30 is called the *deterministic path*. The architecture for the ACNP and ALNP are illustrated in figure 2.5. ACNPs and ALNPs are trained using the same objectives as CNPs (equation 2.20) and LNPs (equation 2.29) respectively. Note that the addition of attention to the architecture raises the computations cost from $O(M + N)$ to $O(M(M + N))$ during test time.

2.3 Conclusion and Discussion

In this chapter we presented the main concepts considered in this thesis, in particular, the meta-learning machine learning framework, the prediction map formulation of the meta-learning problem, stochastic processes, and the neural process family of models. These will be the building blocks for the work presented in the remainder of this thesis. In the next chapter, we return to the topic of translation equivariance and extend the conditional neural process family to include a member that is translation equivariant. We show that this inductive bias helps CNPs achieve improved predictive performance over standard CNPs and ACNPs on synthetic, real-world, and image data.

Chapter 3

Convolutional Conditional Neural Processes

As discussed in Section 1.1, when modelling many real-world datasets, such as those with spatio-temporal input variables or images, it is advantageous in terms of training efficiency and model performance to incorporate translation equivariance into our modelling assumptions, as we will see in Section 3.5. Our goal for the following chapter is to extend the conditional neural process family to include a member that is translation equivariant. Convolutional neural networks (CNNs) are neural network architectures that are translation equivariant and we will present a CNP architecture that build CNNs into its encoder and decoder to define the convolutional conditional neural process (ConvCNP). ConvCNPs incorporate translation equivariance into their inductive bias for more effecient training and performance on suitable applications such as spatio-temporal data modelling.

3.1 Introduction

As discussed in chapter 2, Neural processes (NPs; Garnelo et al., 2018a,b) are a rich class of meta-learning models that produce conditional distributions $p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})$ over target variables $\boldsymbol{y}^{(t)}$ given input variables $\boldsymbol{x}^{(t)}$, parameters $\boldsymbol{\theta}$, and a set of observed data points in a *context set* $D^{(c)} = \{\boldsymbol{x}_m^{(c)}, \boldsymbol{y}_m^{(c)}\}_{m=1}^{M}$. Natural application areas of NPs include time series, spatial data, and images with missing values. Consequently, such domains have been used extensively to benchmark NPs (Garnelo et al., 2018a,b; Kim et al., 2019). Often, ideal solutions to prediction problems in such domains should be translation equivariant: if the data are translated in time or space, then the predictions should be translated correspondingly (Cohen and Welling, 2016; Kondor and Trivedi, 2018). As such, NPs would ideally have translation equivariance built directly into the modelling assumptions as an inductive bias. Previously, NP models must learn this structure directly from the data set, which is sample and parameter inefficient as well as impacting the ability of the models to generalize.

In this chapter, we build translation equivariance into CNPs. Famously, convolutional neural networks added translation equivariance to standard multilayer perceptrons (Cohen and Welling, 2016; LeCun et al., 1998). However, it is not straightforward to generalize NPs in an analogous way: (i) CNNs require data to live "on the grid" (e.g. image pixels form a regularly spaced grid), while many of the above domains have data that live "off the grid" (e.g. time series data may be observed irregularly at any time $t \in \mathbb{R}$). (ii) NPs operate on partially observed context sets whereas CNNs typically do not. (iii) NPs rely on embedding sets into a finite-dimensional vector space for which the notion of equivariance with respect to input translations is not natural, as we detail in Section 3.3.

The work in this chapter is based on the publication 'Convolutional Conditional Neural Processes' presented in the International Conference on Learning Representations, 2020 (Gordon et al., 2020). The research was conducted with the lead authors Jonathan Gordan, and Wessel P. Bruinsma, collaborators Andrew Y. K. Foong, and Yann Dubois as well as our advisor Richard E. Turner. I contributed to the conceptualisation and development of the model, writing software[1], and helped develop the theory. I led the experiments on complex time series (see Sections 3.5.2 and 3.5.3), and wrote these sections of the original paper and appendix. I contributed to the implementation of the Synthetic 1D Experiments and conducted the final versions of these experiments (see Section 3.5.1). I also wrote these sections of the original paper and appendix. The above work conducted by me, and all other work in the paper, including the writing and editing, was a collaborative effort by our entire team.

In this chapter, we introduce the ConvCNP, a member of the NP family that accounts for translation equivariance. This is achieved by extending the theory of learning on sets (see 2.2.2) to include *functional representations*, which in turn can be used to express any translation-equivariant NP model. The key contributions in this chapter can be summarized as follows:

(i) We provide a representation theorem for translation-equivariant functions on sets, extending a key result of Zaheer et al. (2017b) to *functional embeddings*, including sets of varying size.

[1]Source code available at https://github.com/cambridge-mlg/convcnp

(ii) We extend the NP family of models to include translation equivariance.

(iii) We evaluate the ConvCNP and demonstrate that it exhibits excellent performance on several synthetic and real-world benchmarks.

The outline of this chapter closely follows these contributions: we first outline the theoretical foundations for the ConvCNP. Next we define the ConvCNP model and describe the architecture choices guided by the earlier theory. Finally, we evaluate the performance of the ConvCNP on both on-the-grid (continuous) and off-the-grid (descrete) settings.

3.1.1 Notation

For convenience, we outline the notation that will be used in this chapter: let $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^d$, $\mathcal{Y}$ compact, be the spaces of inputs and outputs (though to ease notation, we often assume $\mathcal{Y} \subseteq \mathbb{R}$). Define $\mathcal{Z}_M = (\mathcal{X} \times \mathcal{Y})^M$ as the collection of M input–output pairs, $\mathcal{Z}_{\leq M} = \bigcup_{m=1}^{M} \mathcal{Z}_M$ as the collection of at most M pairs, and $\mathcal{Z} = \bigcup_{m=1}^{\infty} \mathcal{Z}_M$ as the collection of finitely many pairs. Since we will consider $\mathbb{S}$-invariant functions (property 1), also called *permutation-invariant*, functions on $\mathcal{Z}$, we may refer to elements of $\mathcal{Z}$ as sets or data sets. Furthermore, we will use the notation $[n] = \{1, \ldots, n\}$.

We will write the predictive distributions of CNPs as $p(\boldsymbol{y}|\boldsymbol{x}, Z) = p(\boldsymbol{y}|\Phi(\boldsymbol{x}, Z), \boldsymbol{\theta})$. Φ is defined as a composition $\rho \circ E$ of an encoder $E\colon \mathcal{Z} \to \mathbb{R}^d$ mapping into the *embedding space* $\mathbb{R}^d$ and a decoder $\rho\colon \mathbb{R}^d \to C_b(\mathcal{X}, \mathcal{Y})$. Here $E(Z) \in \mathbb{R}^d$ is a *vector representation* of the set Z, and $C_b(\mathcal{X}, \mathcal{Y})$ is the space of continuous, bounded functions $\mathcal{X} \to \mathcal{Y}$ endowed with the supremum norm.

3.2 Translation Equivariance

The focus of this work is on models that are *translation equivariant*: if the input locations of the data are translated by an amount $\boldsymbol{\tau}$, then the predictions should be translated correspondingly. Translation equivariance for functions operating on sets is formalized in Property 2.

Property 2 (Translation equivariant mappings on sets)**.** Let $\mathcal{H}$ be an appropriate space of functions on $\mathcal{X} = \mathbb{R}^n$, and define T and T' as follows:

$$
\begin{aligned}
T &: \mathcal{X} \times \mathcal{Z} \to \mathcal{Z}, & T_{\boldsymbol{\tau}} Z &= ((\boldsymbol{x}_1 + \boldsymbol{\tau}, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_m + \boldsymbol{\tau}, \boldsymbol{y}_m)), \\
T' &: \mathcal{X} \times \mathcal{H} \to \mathcal{H}, & T'_{\boldsymbol{\tau}} h(\boldsymbol{x}) &= h(\boldsymbol{x} - \boldsymbol{\tau}).
\end{aligned}
$$

Then a mapping $\Phi\colon \mathcal{Z} \to \mathcal{H}$ is called *translation equivariant* if $\Phi(T_{\boldsymbol{\tau}} Z) = T'_{\boldsymbol{\tau}}\Phi(Z)$ for all $\boldsymbol{\tau} \in \mathcal{X}$ and $Z \in \mathcal{Z}$.

3.3 Convolutional Deep Sets

We are interested in translation equivariance (Property 2) with respect to translations on $\mathcal{X}$. The CNP encoder E maps sets Z to an embedding in a vector space $\mathbb{R}^d$, for which the notion of equivariance with respect to input translations in $\mathcal{X}$ is not well defined. For example, a function f on $\mathcal{X}$ can be translated by $\boldsymbol{\tau} \in \mathcal{X}$: $f(\,\boldsymbol{\cdot} - \boldsymbol{\tau})$. However, for a vector $\boldsymbol{x} \in \mathbb{R}^d$, which can be seen as a function $[d] \to \mathbb{R}$, $\boldsymbol{x}(i) = x_i$, the translation $\boldsymbol{x}(\,\boldsymbol{\cdot} - \boldsymbol{\tau})$ is not well-defined. To overcome this issue, we enrich the encoder $E\colon \mathcal{Z} \to \mathcal{H}$ to map into a *function space* $\mathcal{H}$ containing functions on $\mathcal{X}$. Since functions in $\mathcal{H}$ map from $\mathcal{X}$, our notion of translation equivariance (Property 2) is now also well defined for $E(Z)$. As we demonstrate below, every translation-equivariant function on sets has a representation in terms of a specific functional embedding.

Definition 2 (Functional mappings on sets and functional representations of sets)**.** Call a map $E\colon \mathcal{Z} \to \mathcal{H}$ a *functional mapping on sets* if it maps from sets $\mathcal{Z}$ to an appropriate space of functions $\mathcal{H}$. Furthermore, call $E(Z)$ the *functional representation* of the set Z.

Considering functional representations of sets leads to the key result of this work, which can be summarized as follows. For an appropriate $\mathcal{Z}' \subseteq \mathcal{Z}$, a continuous function $\Phi\colon \mathcal{Z}' \to C_b(\mathcal{X}, \mathcal{Y})$ satisfies properties 1 and 2 if and only if it has a representation of the form

$$\Phi(Z) = \rho\left(E(Z)\right), \quad E(Z) = \textstyle\sum_{(\boldsymbol{x},\boldsymbol{y})\in Z}\phi(\boldsymbol{y})\psi(\,\boldsymbol{\cdot} - \boldsymbol{x}) \in \mathcal{H}, \tag{3.1}$$

for some continuous and translation-equivariant $\rho\colon \mathcal{H} \to C_b(\mathcal{X}, \mathcal{Y})$, and appropriate ϕ and ψ. Note that ρ is a map between function spaces. We also remark that continuity of Φ is not in the usual sense; we return to this below.

Equation (3.1) defines the encoder used by our proposed model, the ConvCNP. In Section 3.3.1, we present the theoretical results in more detail. In particular, Thm 1 establishes equivalence between any function satisfying properties 1 and 2 and the representational form in Equation (3.1). In doing so, we provide an extension of the key result of Zaheer et al. (2017b) discussed in Section 2.2.2 to functional representations on sets, and show that it can naturally be extended to handle varying-size sets. The practical implementation of ConvCNPs — the design of ρ, ϕ, and ψ — is informed by our results in Section 3.3.1 (as well as the proofs, provided in Section A.1), and is discussed for domains of interest in Section 3.4.

3.3.1 Representations of Translation Equivariant Functions on Sets

In this section we establish the theoretical foundation of the ConvCNP. We begin by stating a definition that is used in our main result.

Definition 3 (Multiplicity)**.** A collection $\mathcal{Z}' \subseteq \mathcal{Z}$ is said to have *multiplicity* K if, for every set $Z \in \mathcal{Z}'$, every $\boldsymbol{x}$ occurs at most K times:

$$\operatorname{mult} \mathcal{Z}' := \sup \{ \sup \{ \underbrace{|\{i \in [m] : \boldsymbol{x}_i = \hat{\boldsymbol{x}}\}| : \hat{\boldsymbol{x}} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m}_{\text{number of times every } \boldsymbol{x} \text{ occurs}} \} : (\boldsymbol{x}_i, y_i)_{i=1}^m \in \mathcal{Z}' \} = K.$$

For example, in the case of real-world data like time series and images, we often observe only one (possibly multi-dimensional) observation per input location, which corresponds to multiplicity one. We are now ready to state our key theorem.

Theorem 1. *Consider a collection $\mathcal{Z}'_{\leq M} \subseteq \mathcal{Z}_{\leq M}$ with multiplicity K such that, for all $m \in [M]$, $\mathcal{Z}'_{\leq M} \cap \mathcal{Z}_m$ is topologically closed and closed under permutations and translations. Then a function $\Phi \colon \mathcal{Z}'_{\leq M} \to C_b(\mathcal{X}, \mathcal{Y})$ is continuous*[2]*, permutation invariant (Property 1)), and translation equivariant (Property 2) if and only if it has a representation of the form*

$$\Phi(Z) = \rho\left(E(Z)\right), \quad E((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) = \textstyle\sum_{i=1}^m \phi(y_i)\psi(\bullet - \boldsymbol{x}_i)$$

for some continuous and translation-equivariant $\rho \colon \mathcal{H} \to C_b(\mathcal{X}, \mathcal{Y})$ and some continuous $\phi \colon \mathcal{Y} \to \mathbb{R}^{K+1}$ and $\psi \colon \mathcal{X} \to \mathbb{R}$, where $\mathcal{H}$ is an appropriate space of functions that includes the image of E. We call a function Φ of the above form a convolutional DeepSet (ConvDeepSet).

The proof of Thm 1 is provided in Section A.1. We here discuss several key points from the proof that have practical implications and provide insights for the design of ConvCNPs: (i) For the construction of ρ and E, ψ is set to a flexible positive-definite

[2]For every $m \in [M]$, the restriction $\Phi|_{\mathcal{Z}'_{\leq M} \cap \mathcal{Z}_m}$ is continuous.

kernel associated with a Reproducing Kernel Hilbert Space (RKHS; Aronszajn (1950)), which results in desirable properties for E. (ii) Using the work by Zaheer et al. (2017b), we set $\phi(y) = (y^0, y^1, \cdots, y^K)$ to be the powers of y up to order K. (iii) Thm 1 requires ρ to be a powerful function approximator of continuous, translation-equivariant maps between functions. In Section 3.4, we discuss how these theoretical results inform our implementations of ConvCNPs.

Thm 1 extends the result of Zaheer et al. (2017b), discussed in Chapter 2, by embedding the set into an infinite-dimensional space — the RKHS — instead of a finite-dimensional space. Beyond allowing the model to exhibit translation equivariance, the RKHS formalism allows us to naturally deal with finite sets of varying sizes, which turns out to be challenging with finite-dimensional embeddings. Furthermore, our formalism requires $\phi(y) = (y^0, y^1, y^2, \ldots, y^K)$ to expand up to order no more than the *multiplicity* of the sets K; if K is bounded, then our results hold for sets up to any arbitrarily large finite size M, while fixing ϕ to be only $(K+1)$-dimensional.

3.4 Convolutional Conditional Neural Processes

In this section we discuss the architectures and implementation details for ConvCNPs. Similar to CNPs, ConvCNPs model the conditional distribution as

$$
\begin{aligned}
p(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(t)}, D^{(c)}) &= \prod_{n=1}^{N} p(y_n^{(t)} | \Phi_{\boldsymbol{\theta}}(D^{(c)})(x_n^{(t)})) && (3.2) \\
&= \prod_{n=1}^{N} \mathcal{N}(y_n^{(t)}; \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n) \text{ with } (\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n) = \Phi_{\boldsymbol{\theta}}(D^{(c)})(x_n^{(t)}), && (3.3)
\end{aligned}
$$

where $D^{(c)}$ is the context set and Φ a ConvDeepSet. The key considerations are the design of ρ, ϕ, and ψ for Φ. We provide separate models for data that lie on-the-grid and data that lie off-the-grid.

Form of ϕ. The applications considered in this work have a single (potentially multi-dimensional) output per input location, so the multiplicity of $\mathcal{D}$ is one (i.e., $K = 1$) where context sets $D^{(c)} \subseteq \mathcal{D}$. It then suffices to let ϕ be a power series of order one, which is equivalent to appending a constant to y in all data sets, i.e. $\phi(y) = [1 \; y]^\top$. The first output ϕ_1 thus provides the model with information regarding where data has been observed, which is necessary to distinguish between no observed datapoint at x and a datapoint at x with $y = \mathbf{0}$. Denoting the functional representation as $\boldsymbol{h}$, we can think of the first channel $\boldsymbol{h}^{(0)}$ as a "density channel". We found it helpful to divide the remaining channels $\boldsymbol{h}^{(1:)}$ by $\boldsymbol{h}^{(0)}$ (Algorithm 2, line 4), as this improved performance

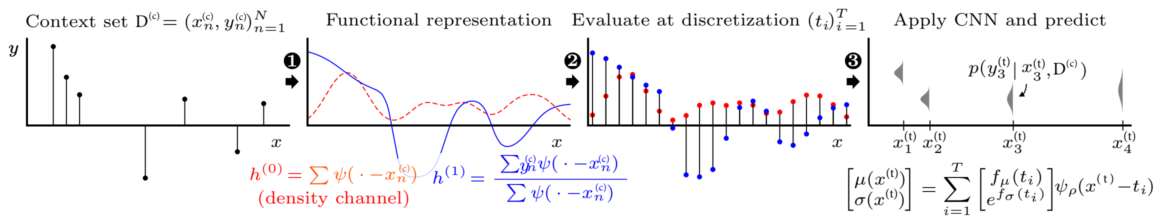

Context set $D^{(c)} = (x_n^{(c)}, y_n^{(c)})_{n=1}^N$
Functional representation
Evaluate at discretization $(t_i)_{i=1}^T$
Apply CNN and predict
y
x
❶
❷
❸
$h^{(0)} = \sum \psi(\cdot - x_n^{(c)})$
(density channel)
$h^{(1)} = \frac{\sum y_n^{(c)} \psi(\cdot - x_n^{(c)})}{\sum \psi(\cdot - x_n^{(c)})}$
$p(y_3^{(t)} \mid x_3^{(t)}, D^{(c)})$
$x_1^{(t)}$ $x_2^{(t)}$ $x_3^{(t)}$ $x_4^{(t)}$
$\begin{bmatrix}\mu(x^{(t)})\\ \sigma(x^{(t)})\end{bmatrix} = \sum_{i=1}^{T} \begin{bmatrix} f_\mu(t_i) \\ e^{f_\sigma(t_i)}\end{bmatrix} \psi_\rho(x^{(t)} - t_i)$

Fig. 3.1 Illustration of the ConvCNP forward pass in the off-the-grid case

when there is large variation in the density of input locations. In the image processing literature, this is known as *normalized convolution* (Knutsson and Westin, 1993). The normalization operation can be reversed by ρ and is therefore not restrictive.

ConvCNPs for off-the-grid data. Having specified ϕ, it remains to specify the form of ψ and ρ. Our proof of Thm 1 suggests that ψ should be a stationary, non-negative, positive-definite kernel. The exponentiated-quadratic (EQ) kernel with a learnable length scale parameter is a natural choice. This kernel is multiplied by ϕ to form the functional representation $E(D^{(c)})$ (Algorithm 2, line 3; and Figure 3.1, arrow 1).

Next, Thm 1 suggests that ρ should be a continuous, translation-equivariant map between function spaces. Kondor and Trivedi (2018) show that, in deep learning, any translation-equivariant model has a representation as a CNN. However, CNNs operate on discrete (on-the-grid) input spaces and produce discrete outputs. In order to approximate ρ with a CNN, we discretize the input of ρ, apply the CNN, and finally transform the CNN output back to a continuous function $\mathcal{X} \to \mathcal{Y}$. To do this, for each context and test set, we space points $(t_i)_{i=1}^n \subseteq \mathcal{X}$ on a uniform grid (at a pre-specified density) over a hyper-cube that covers both the context and target inputs. We then evaluate $(E(D^{(c)})(t_i))_{i=1}^n$ (Algorithm 2, lines 1–2; Figure 3.1, arrow 2). This discretized representation of $D^{(c)}$ is then passed through a CNN (Algorithm 2, line 5; Figure 3.1, arrow 3).

To map the output of the CNN back to a continuous function $\mathcal{X} \to \mathcal{Y}$, we use the CNN outputs as weights for evenly-spaced basis functions (again employing the EQ kernel), which we denote by ψ_ρ (Algorithm 2, lines 6–7; Figure 3.1, arrow 3). The resulting approximation to ρ is not perfectly translation equivariant, but will be approximately so for length scales larger than the spacing of $(E(D^{(c)})(t_i))_{i=1}^n$. The resulting continuous functions are then used to generate the (Gaussian) predictive mean and variance at any input. This, in turn, can be used to evaluate the log-likelihood.

ConvCNP for on-the-grid data. While ConvCNP is readily applicable to many settings where data live on a grid, in this work we focus on the image setting. As

Algorithm 2 ConvCNP Forward Pass for Off-the-grid Data.

```
Require: $\rho = (\mathrm{CNN}, \psi_\rho)$, $\psi$, and density $\gamma$
Require: context set $(\boldsymbol{x}_n^{(c)}, y_n^{(c)})_{n=1}^N$, target points $(\boldsymbol{x}_m^{(t)})_{m=1}^M$
 1: lower, upper $\leftarrow$ range$\Big((\boldsymbol{x}_n)_{n=1}^N \cup (\boldsymbol{x}_m^*)_{m=1}^M\Big)$
 2: $(\boldsymbol{t}_i)_{i=1}^T \leftarrow$ uniform_grid(lower, upper; $\gamma$)
 3: $\boldsymbol{h}_i \leftarrow \sum_{n=1}^N \begin{bmatrix} 1 & y_n \end{bmatrix}^\top \psi(\boldsymbol{t}_i - \boldsymbol{x}_n)$
 4: $\boldsymbol{h}_i^{(1:)} \leftarrow \boldsymbol{h}_i^{(1:)} / \boldsymbol{h}_i^{(0)}$
 5: $(f_\mu(\boldsymbol{t}_i), f_\sigma(\boldsymbol{t}_i))_{i=1}^T \leftarrow \mathrm{CNN}((\boldsymbol{t}_i, \boldsymbol{h}_i)_{i=1}^T)$
 6: $\boldsymbol{\mu}_m \leftarrow \sum_{i=1}^T f_\mu(\boldsymbol{t}_i)\psi_\rho(\boldsymbol{x}_m^* - \boldsymbol{t}_i)$
 7: $\boldsymbol{\sigma}_m \leftarrow \sum_{i=1}^T \mathrm{pos}(f_\sigma(\boldsymbol{t}_i))\psi_\rho(\boldsymbol{x}_m^* - \boldsymbol{t}_i)$
 8: return $(\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m)_{m=1}^M$
```

such, the following description uses the image completion task as an example, which is often used to benchmark NPs (Garnelo et al., 2018a; Kim et al., 2019). Compared to the off-the-grid case, the implementation becomes simpler as we can choose the discretization $(t_i)_{i=1}^n$ to be the pixel locations.

Let $\mathrm{I} \in \mathbb{R}^{H \times W \times C}$ be an image — H, W, C denote the height, width, and number of channels, respectively — and let M_c be the context mask, which is such that $[\mathrm{M}_c]_{i,j} = 1$ if pixel location (i, j) is in the context set, and 0 otherwise. To implement ϕ, we select all context points, $D^{(c)} := \mathrm{M}_c \odot \mathrm{I}$, and prepend the context mask: $\phi = [\mathrm{M}_c, D^{(c)}]^\top$ (Algorithm 3, line 3).

Next, we apply a convolution to the context mask to form the density channel: $\boldsymbol{h}^{(0)} = \textsc{conv}_{\boldsymbol{\theta}}(\mathrm{M}_c)$ (Algorithm 3, line 3). To all other channels, we apply a normalized convolution: $\boldsymbol{h}^{(1:C)} = \textsc{conv}_{\boldsymbol{\theta}}(y)/\boldsymbol{h}^{(0)}$ (Algorithm 3, line 4), where the division is element-wise. The filter of the convolution is analogous to ψ, which means that $\boldsymbol{h}$ is the functional representation, with the convolution performing the role of E (the summation in Algorithm 2, line 3). Although the theory suggests using a non-negative, positive-definite kernel, we did not find significant empirical differences between an EQ kernel and using a fully trainable kernel restricted to positive values to enforce non-negativity (see Sections A.4.4 and A.4.5 for details).

Lastly, we describe the on-the-grid version of $\rho(\cdot)$, which consists of two stages. First, we apply a CNN to $E(D^{(c)})$ (Algorithm 3, line 5). Second, we apply a shared, pointwise MLP that maps the output of the CNN at each pixel location in the target set to $\mathbb{R}^{2C}$, where we absorb MLP into the CNN (MLP can be viewed as an 1×1 convolution). The first C outputs are the means of a Gaussian predictive distribution and the second C the standard deviations, which then pass through a positivity-enforcing function

(Algorithm 3, line 6–7). To summarise, the on-the-grid algorithm is given by

$$(\boldsymbol{\mu}, \text{pos}^{-1}(\boldsymbol{\sigma})) = \underbrace{\text{CNN}}_{\rho}(\overbrace{[\underbrace{\textsc{conv}(\text{M}_c)}_{\text{density channel}}\,; \textsc{conv}(\text{M}_c \odot \text{I}) / \underbrace{\textsc{conv}}_{\text{multiplies by } \psi \text{ and sums}}(\text{M}_c)]^\top}^{E(\text{context set})}), \tag{3.4}$$

where $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ are the image mean and standard deviation, ρ is implemented with CNN, and E is implemented with the mask M_c and convolution $\textsc{conv}$.

Training. Denoting the data sets $\{D_n\}_{n=1}^N \subseteq \mathcal{D}$, and the parameters by $\boldsymbol{\theta}$, maximum-likelihood training involves (Garnelo et al., 2018a,b)

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta} \in \Theta} \textstyle\sum_{n=1}^N \sum_{(x^{(t)}, y^{(t)}) \in D_n^{(t)}} \log p(y \mid \Phi_{\boldsymbol{\theta}}(D_n^{(c)})(x)), \tag{3.5}$$

where we have split D_n into context ($D_n^{(c)}$) and target ($D_n^{(t)}$) sets. This is standard practice in the NP (Garnelo et al., 2018a,b) and meta-learning settings (Finn et al., 2017; Gordon et al., 2019) and relates to neural auto-regressive models (Requeima et al., 2019), as discussed in Section 2.2.3. Note that the context set and target set are disjoint ($D_n^{(c)} \cap D_n^{(t)} = \emptyset$), which differs from the protocol for the CNP (Garnelo et al., 2018a). Practically, stochastic gradient descent methods (Bottou, 2010) are used for optimization.

Algorithm 3 ConvCNP Forward Pass for On-the-grid Data.

```
Require: $\rho$ = CNN and $E = \textsc{conv}_{\boldsymbol{\theta}}$
Require: image I, context mask $\text{M}_c$, and target mask $\text{M}_t$
1: $D^{(c)} \leftarrow \text{M}_c \odot \text{I}$
2: $\boldsymbol{h} \leftarrow \textsc{conv}_{\boldsymbol{\theta}}([\text{M}_c, D_n^{(c)}]^\top)$
3: $\boldsymbol{h}^{(1:C)} \leftarrow \boldsymbol{h}^{(1:C)} / \boldsymbol{h}^{(0)}$
4: $f_t \leftarrow \text{M}_t \odot \text{CNN}(\boldsymbol{h})$
5: $\boldsymbol{\mu} \leftarrow f_t^{(1:C)}$
6: $\boldsymbol{\sigma} \leftarrow \text{pos}(f_t^{(C+1:2C)})$
7: return $(\boldsymbol{\mu}, \boldsymbol{\sigma})$
```

3.5 Experiments and Results

We evaluate the performance of ConvCNPs in both on-the-grid and off-the-grid settings focusing on two central questions: (i) Do translation-equivariant models improve performance in appropriate domains? (ii) Can translation equivariance enable ConvCNPs to generalize to settings outside of those encountered during training? We use several off-the-grid data-sets which are irregularly sampled time series ($\mathcal{X} = \mathbb{R}$), comparing to Gaussian processes (GPs; Williams and Rasmussen (2006)) and ACNP (which is the version of the Attentive Neural Process (Kim et al., 2019) without the latent path in the encoder, see Section 2.2.4), the best performing member of the CNP family during the development of the ConvCNP. We then evaluate on several on-the-grid image data sets ($\mathcal{X} = \mathbb{Z}^2$). In all settings we demonstrate substantial improvements over existing neural process models. For the CNN component of our model, we propose a small and large architecture for each experiment (in the experimental sections named ConvCNP and ConvCNPXL, respectively). We note that these architectures are different for off-the-grid and on-the-grid experiments, with full details regarding the architectures given in appendices A.3 and A.4.

3.5.1 Synthetic 1D Experiments

First we consider synthetic regression problems. At each iteration, a function is sampled, followed by context and target sets. Beyond EQ-kernel GPs (as in Garnelo et al. (2018a); Kim et al. (2019)), we consider more complex data arising from Matern–$\frac{5}{2}$ and weakly-periodic kernels, as well as a challenging, non-Gaussian sawtooth process with random shift and frequency (see Figure 3.2, for example). ConvCNP is compared to the CNP (Garnelo et al., 2018a) and ACNP. Training and testing procedures are fixed across all models. Full details on models, data generation, and training procedures are provided in Section A.3.

Model	Params	EQ	Weak Periodic	Matern	Sawtooth
CNP	66818	-0.86 ± 3e-3	-1.23 ± 2e-3	-0.95 ± 1e-3	-0.16 ± 1e-5
ACNP	149250	0.72 ± 4e-3	-1.20 ± 2e-3	0.10 ± 2e-3	-0.16 ± 2e-3
ConvCNP	6537	0.70 ± 5e-3	-0.92 ± 2e-3	0.32 ± 4e-3	1.43 ± 4e-3
ConvCNPXL	50617	**1.06 ± 4e-3**	**-0.65 ± 2e-3**	**0.53 ± 4e-3**	**1.94 ± 1e-3**

Table 3.1 Log-likelihood from synthetic 1-dimensional experiments.

Table 3.1 reports the log-likelihood means and standard errors of the models over 1000 tasks. The context and target points for both training and testing lie within the interval $[-2, 2]$ where training data was observed (marked "training data range" in Figure 3.2). Table 3.1 demonstrates that, even when extrapolation is not required, ConvCNP significantly outperforms other models in all cases, despite having fewer parameters.

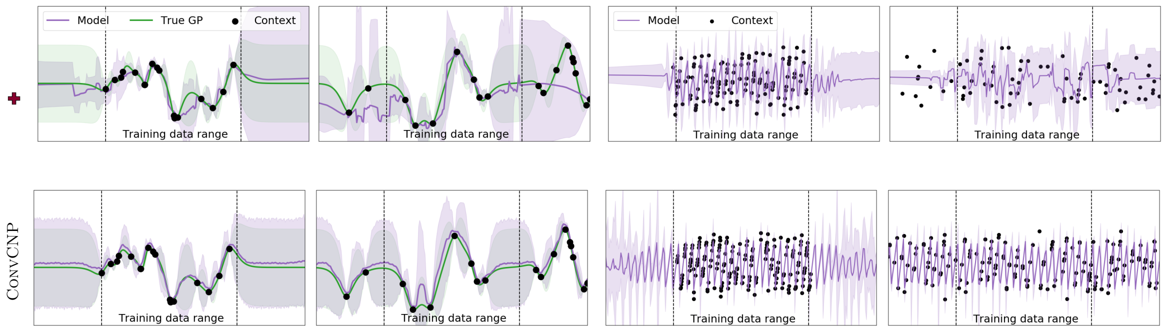


Fig. 3.2 Example functions learned by the AttnCNP (top row), and ConvCNP (bottom row), when trained on a Matern–$\frac{5}{2}$ kernel with length scale 0.25 (first and second column) and sawtooth function (third and fourth column). Columns one and three show the predictive posterior of the models when data is presented in same range as training, with predictive posteriors continuing beyond that range on either side. Columns two and four show model predictive posteriors when presented with data outside the training data range. Plots show means and two standard deviations.

Figure 3.2 demonstrates that ConvCNP generates excellent fits, even for challenging functions such as from the Matern–$\frac{5}{2}$ kernel and sawtooth. Moreover, Figure 3.2 compares the performance of ConvCNP and ACNP when data is observed outside the range where the models were trained: translation equivariance enables ConvCNP to elegantly generalize to this setting, whereas ACNP is unable to generate reasonable predictions.

3.5.2 PLAsTiCC Experiments

The PLAsTiCC data set (Allam Jr et al., 2018) is a simulation of transients observed by the LSST telescope under realistic observational conditions. The data set contains 3,500,734 "light curves", where each measurement is of an object's brightness as a function of time, taken by measuring the photon flux in six different astronomical filters. The data can be treated as a six-dimensional time series. The data set was introduced in a Kaggle competition,[3] where the task was to use these light curves to classify the variable sources. The winning entry (Avocado, Boone, 2019) modeled the light curves with GPs and used these models to generate features for a gradient boosted decision tree classifier. We compare a multi-input–multi-output ConvCNP with the GP models used in Avocado.[4] ConvCNP accepts six channels as inputs, one for each astronomical filter, and returns 12 outputs — the means and standard deviations of six Gaussians.

[3]https://www.kaggle.com/c/PLAsTiCC-2018

[4]Full code for Avocado, including GP models, is available at https://github.com/kboone/avocado.

Full experimental details are given in Section A.3.3. The mean squared error of both approaches is similar, but the held-out log-likelihood from the ConvCNP is far higher (see Table 3.2).

Model	Log-likelihood	MSE
Kaggle GP (Boone, 2019)	-0.335 ± 0.09	0.037 ± 4e-3
ConvCP (ours)	**1.31** ± 0.30	0.040 ± 5e-3

Table 3.2 Mean and standard errors of log-likelihood and root mean squared error over 1000 test objects from the PLastiCC dataset.

3.5.3 Predator-Prey Models: Sim2Real

The ConvCNP model is well suited for applications where simulation data is plentiful, but real-world training data is scarce (Sim2Real). The ConvCNP can be trained on a large amount of simulation data and then be deployed with real-world training data as the context set. We consider the Lotka–Volterra model (Wilkinson, 2011), which is used to describe the evolution of predator–prey populations. This model has been used in the approximate Bayesian computation literature where the task is to infer the parameters from samples drawn from the Lotka–Volterra process (Papamakarios and Murray, 2016). These methods do not simply extend to prediction problems such as interpolation or forecasting. In contrast, we train ConvCNP on synthetic data sampled from the Lotka–Volterra model and can then condition on real-world data from the Hudson's Bay lynx–hare data set (Leigh, 1968) to perform interpolation (see Figure 3.3; full experimental details are given in Section A.3.4). The ConvCNP performs accurate interpolation as shown in Figure 3.3.

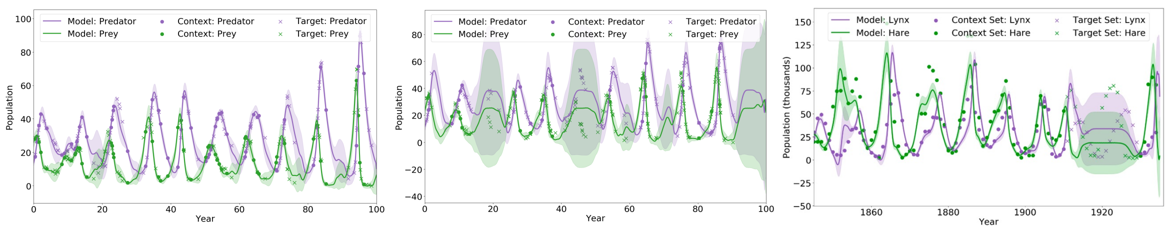


Fig. 3.3 Left and centre: two samples from the Lotka–Volterra process (sim). Right: ConvCNP trained on simulations and applied to the Hudson's Bay lynx-hare dataset (real). Plots show means and two standard deviations.

We attempted to train an ACNP for comparison, but due to the nature of the synthetic data generation, many of the training series end before 90 time units, the length of the Hudson's Bay lynx-hare series. Effectively, this means that the ACNP was asked to predict outside of its training interval, a task that it struggles with, as shown in Section 3.5.1. The plots in Figure 3.4 show that the ACNP is able to learn the first part of the time series but is unable to model data outside of the first 20 or so time units. Perhaps with more capacity and training epochs the ACNP training would be more successful. Note from Figure 3.3 that our model does better on the synthetic data than on the real data. This could be due to the parameters of the Lotka–Volterra model used being a poor estimate for the real data.

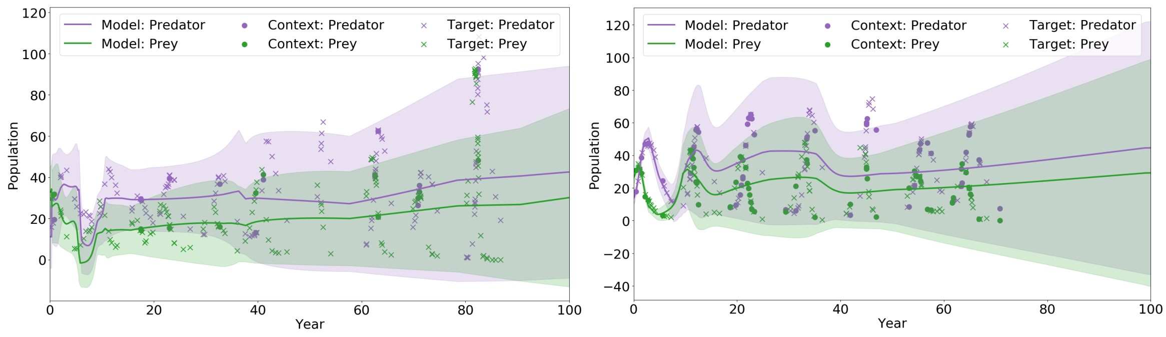


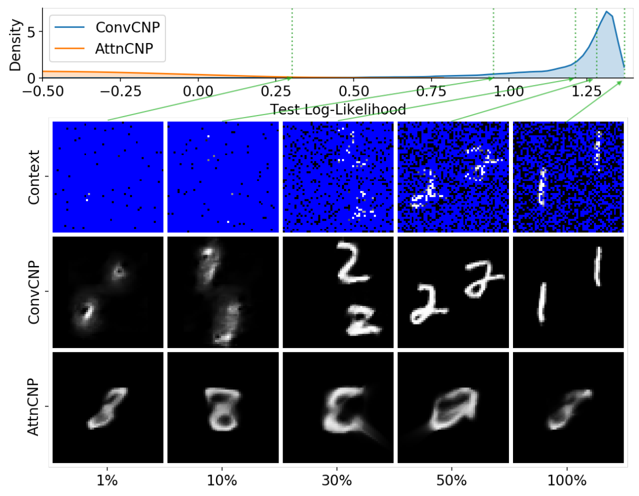
Fig. 3.4 ACNP performance on two samples from the Lotka–Volterra process (sim).

3.5.4 2D Image Completion Experiments

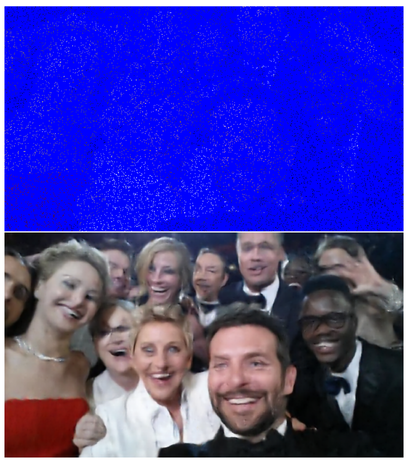
To test ConvCNP beyond one-dimensional features, we evaluate our model on on-the-grid image completion tasks and compare it to ACNP. Image completion can be cast as a prediction of pixel intensities $y_i^{(t)}$ ($\in \mathbb{R}^3$ for RGB, $\in \mathbb{R}$ for greyscale) given a target 2D pixel location $x_i^{(t)}$ conditioned on an observed (context) set of pixel values $D^{(c)} = (x_n^{(c)}, y_n^{(c)})_{n=1}^N$. In the following experiments, the context set can vary but the target set contains all pixels from the image. Further experimental details are in Section A.4.

Standard benchmarks. We first evaluate the model on four common benchmarks: MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), and 32 × 32 and 64 × 64 CelebA (Liu et al., 2018b). Importantly, these data sets are biased towards images containing a single, well-centered object. As a result, perfect translation-equivariance might hinder the performance of the model when the test data are similarly structured. We therefore also evaluated a larger ConvCNP that can learn such non-stationarity, while still sharing parameters across the input space (ConvCNPXL).

Table 3.3 shows that ConvCNP significantly outperforms ACNP when it has a large receptive field size, while being at least as good with a small receptive field size.



(a) Log-likelihood and qualitative results on ZSMM. The top row shows the log-likelihood distribution for both models. The images below correspond to the context points (top), ConvCNP target predictions (middle), and ACNP target predictions (bottom). Each column corresponds to a given percentile of the ConvCNP distribution.



(b) Qualitative evaluation of a ConvCNPXL trained on the unscaled CelebA (218×178) and tested on Ellen's Oscar unscaled (337×599) selfie (DeGeneres, 2014) with 5% of the pixels as context (top).

Fig. 3.5 Zero shot generalization to tasks that require translation equivariance.

Model	Params	MNIST	SVHN
ACNP	410k	1.08 ±0.04	3.94 ±0.02
ConvCNP	113k	1.21 ±0.00	3.89 ±0.01
ConvCNPXL	400k	**1.27 ±0.01**	**3.97 ±0.02**

Model	CelebA32	CelebA64	ZSMM
ACNP	3.18 ±0.02		-0.83 ±0.08
ConvCNP	3.22 ±0.02	3.66 ±0.01	**1.18 ±0.04**
ConvCNPXL	**3.39 ±0.02**	**3.73 ±0.01**	0.86 ±0.12

Table 3.3 Log-likelihood from image experiments (6 runs).

Qualitative samples for various context sets can be seen in Figure 3.6. Further qualitative comparisons and ablation studies can be found in Section A.4.3 and Section A.4.4 respectively.

Generalization to multiple, non-centered objects. The data sets from the previous paragraphs were centered and contained single objects. Here we test whether ConvCNPs trained on such data can generalize to images containing multiple, non-centered objects.

The last column of Table 3.3 evaluates the models in a zero-shot multi MNIST (ZSMM) setting, where images contain multiple digits at test time (Section A.4.2). ConvCNP significantly outperforms ACNP on such tasks. Figure 3.5a shows a histogram of the image log-likelihoods for ConvCNP and ACNP, as well as qualitative results at different percentiles of the ConvCNP distribution. ConvCNP is able to extrapolate to this out-of-distribution test set, while ACNP appears to model the bias of the training data and predict a centered "mean" digit independently of the context. Interestingly, ConvCNPXL does not perform as well on this task. In particular, we find that, as the receptive field becomes very large, performance on this task decreases. We hypothesize that this has to do with behavior of the model at the edges of the image. CNNs with larger receptive fields — the region of input pixels that affect a particular output pixel — are able to model non-stationary behavior by looking at the distance from any pixel to the image boundary. We expand on this discussion and provide further experimental evidence regarding the effects of receptive field on the ZSMM task in Section A.4.6.

Although ZSMM is a contrived task, note that our field of view usually contains multiple independent objects, thereby requiring translation equivariance. As a more realistic example, we took a ConvCNP model trained on CelebA and tested it on a natural image of different shape which contains multiple people (Figure 3.5b). Even with 95% of the pixels removed, the ConvCNP was able to produce a qualitatively reasonable reconstruction. A comparison with ACNP is given in Section A.4.3.

Computational efficiency. Beyond the performance and generalization improvements, a key advantage of the ConvCNP is its computational efficiency. The memory and time complexity of a single self-attention layer grows quadratically with the number of inputs M (the number of pixels for images) but only linearly for a convolutional layer.

Empirically, with a batch size of 16 on 32×32 MNIST, ConvCNPXL requires 945MB of VRAM, while ACNP requires 5839 MB. For the 56×56 ZSMM ConvCNPXL increases its requirements to 1443 MB, while ACNP could not fit onto a 32GB GPU. Ultimately, ACNP had to be trained with a batch size of 6 (using 19139 MB) and we were not able to fit it for CelebA64. Recently, restricted attention has been proposed to overcome this computational issue (Parmar et al., 2018), but we leave an investigation of this and its relationship to ConvCNPs to future work.

3.6 Conclusion and Discussion

In this chapter, we introduced ConvCNP, a member of the CNP family that leverages embedding sets into function space to achieve translation equivariance. The relationship to the NP family, and representing functions on sets, give rise to novel extensions.

On the topic of Deep Sets, two key issues in the existing theory on learning with sets (Qi et al., 2017a; Wagstaff et al., 2019; Zaheer et al., 2017b) are (i) the restriction to fixed-size sets, and (ii) that the dimensionality of the embedding space must be no less than the cardinality of the embedded sets. Our work implies that by considering appropriate embeddings into a function space, both issues are alleviated.

Another line of related research focuses on 3D point-cloud modelling (Qi et al., 2017a,b). While original work focused on permutation invariance (Qi et al., 2017a; Zaheer et al., 2017b), more recent work has considered translation equivariance as well (Wu et al., 2019), leading to a model closely resembling ConvDeepSets. The key differences with our work are the following: (i) Wu et al. (2019) implement ψ as an MLP with learned weights, resulting in a more flexible parameterization of the convolutional weights. (ii) Wu et al. (2019) interpret the computations as Monte Carlo approximations to an underlying continuous convolution, whereas we consider the problem of function approximation directly on sets. (iii) Wu et al. (2019) only consider the point-cloud application, whereas our derivation and modelling work considers general sets.

In the predictive distribution of ConvCNP (Section 3.4), predicted outputs y are conditionally independent given the context set. Consequently, samples from the predictive distribution are incoherent – they lack correlations and appear noisy. One solution is to instead define the predictive distribution in an autoregressive way, like e.g. PixelCNN++ (Salimans et al., 2017). Although samples are now correlated, the quality of the samples depends on the order in which the points are sampled. Moreover, the predicted outputs y are then not consistent under marginalization (Garnelo et al., 2018b; Kim et al., 2019). Consistency under marginalization is more generally an issue for neural autoregressive models (Parmar et al., 2018; Salimans et al., 2017), although consistent variants have been devised (Louizos et al., 2019). To overcome the consistency issue for ConvCNP, exchangeable neural process models (e.g. Korshunova et al., 2020; Louizos et al., 2019) may provide an interesting avenue. Another way to introduce dependencies between outputs y is to employ latent variables as is done in neural processes as in the LNP (Garnelo et al., 2018b). This has the added benefit of overcoming the limited Gaussian parametrization of the predictive distribution used by both CNPs and ConvCNPs which cannot model multi-modality, heavy-tailedness, or asymmetry. However, such an approach only achieves *conditional consistency*: given a context set, the predicted outputs y will be dependent and consistent under marginalization, but this does not lead to a consistent joint model that also includes the context set itself, previously described as AR-consistency. The next chapter will propose a method of combining the benefits of translation equivariance with the latent variable architecture in LNPs to achieve conditional consistency. In fact, the remainder of this thesis will continue to explore these issues of consistancy, coherent model samples and more flexible output space parametrizations.

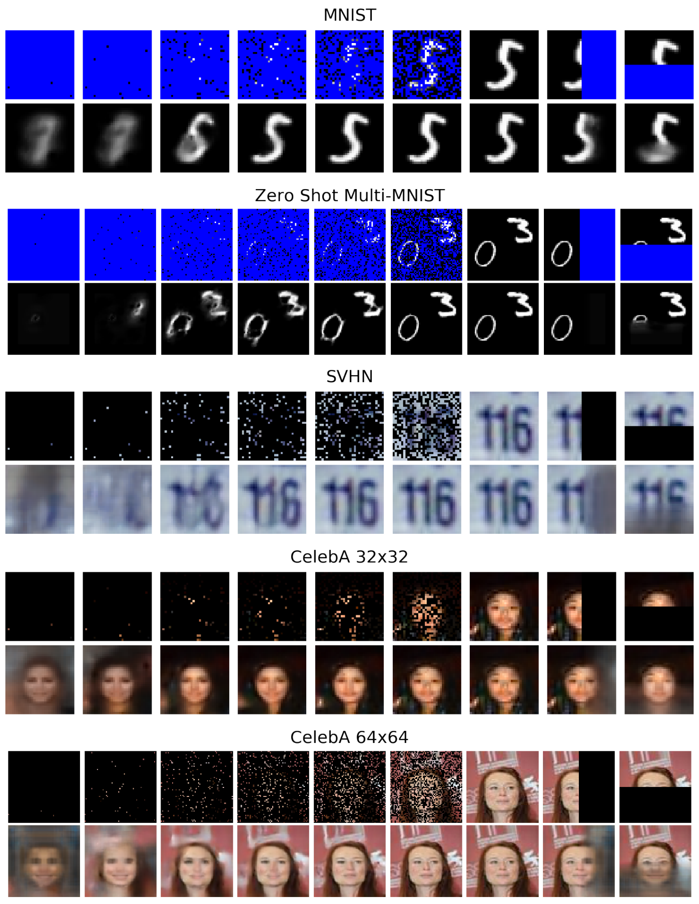


Fig. 3.6 Qualitative evaluation of the ConvCNP(XL). For each dataset, an image is randomly sampled, the first row shows the given context points while the second is the mean of the estimated conditional distribution. From left to right the first seven columns correspond to a context set with 3, 1%, 5%, 10%, 20%, 30%, 50%, 100% randomly sampled context points. In the last two columns, the context sets respectively contain all the pixels in the left and top half of the image. ConvCNPXL is shown for all datasets besides ZSMM, for which we show the fully translation equivariant ConvCNP.

Chapter 4

Convolutional Latent Neural Processes

ConvCNPs incorporated translation equivariance into conditional neural processes for better performance on spatio-temporal and image data. However, as a conditional neural process, it is only able to produce mean-field predictions poorly suited for downstream estimation and real-world applications, such as those in climate science, that require well-calibrated joint uncertainties (Markou et al., 2022). Building off of ConvCNPs introduced in Chapter 3, in this chapter we will introduce a latent variable version of the ConvCNP to tackle this weaknesses of the ConvCNP. We introduce Convolutional Latent Neural Process, a member of the neural process family that is translation equivarient like the ConvCNP but is able to produce coherent function samples model epistemic uncertainty unlike its conditional counterpart, the ConvCNP.

4.1 Introduction

Incorporating appropriate inductive biases into machine learning models is key to achieving good generalization performance. Consider, for example, predicting rainfall at an unseen test location from rainfall measurements nearby. As we saw in Chapter 3, a powerful inductive bias for this task is *stationarity*: the assumption that the generative process governing rainfall is spatially homogeneous. Given only observations in a limited part of the space, stationarity allows the model to extrapolate to yet unobserved regions. And closely related to stationarity is *translation equivariance* (TE). TE formalizes the intuitive idea that if observations are shifted in time or space, then the resulting predictions should be shifted by the same amount. In Chapter 3, we were able to incorporate translation equivariance into CNPs with the development of the ConvCNP model and we saw that, when stationarity or TE is appropriate, like in time-series (Roberts et al., 2013), images (LeCun et al., 1998), and spatio-temporal modelling (Cressie, 1990; Delhomme, 1978), incorporating them into CNPs yields significant benefits. However, CNPs suffer from drawbacks that inhibit their use in scenarios where other SP models, e.g. Gaussian processes (GPs; (Rasmussen and Williams, 2006)), often succeed. For example, both CNPs and ConvCNPs are limited to factorized, parametric predictive distributions. This makes them unsuitable for producing coherent predictive function samples or modelling complicated likelihoods. *Latent Neural Processes* (LNPs; (Garnelo et al., 2018b)) (see Section 2.2.4), a latent variable extension of CNPs, were introduced to enable richer joint predictive distributions. However, the LNP training procedure uses variational inference (VI) and amortization, which are known to suffer from certain drawbacks (Cremer et al., 2018; Turner and Sahani, 2011). Moreover, existing LNPs did not incorporate TE at the time of publication of the work in this chapter.

In this chapter, we build upon both ConvCNPs and LNPs to develop *convolutional latent neural processes* (ConvLNPs). ConvLNPs are a map from data sets to predictive SPs that is both TE *and* capable of expressing complex joint distributions. As training ConvLNPs with VI poses technical and practical issues, we instead propose a simplified maximum-likelihood objective, which directly targets the predictive SP. We show that ConvLNPs produce compelling samples and generalize effectively, making them suitable for a broad range of spatio-temporal prediction tasks.

The work in this chapter is based on the publication 'Meta-Learning Stationary Stochastic Process Prediction with Convolutional Neural Processes' presented in the Conference on Neural Information Processing Systems, 2020 (Foong et al., 2020). The research was conducted with the lead authors Andrew Y. K. Foong, Wessel P. Bruinsma, and Jonathan Gordon, collaborator Yann Dubois as well as our advisor Richard E. Turner. I developed the model and architecture with Jonathan Gordon, and contributed to the writing of software[1], experiments, and the of the original paper and appendix. The bulk of the work on this paper was done by the lead authors but all ideas and work in the paper was a collaborative effort by our entire team.

The key contributions of this chapter are:

(i) We introduce ConvLNPs, extending ConvCNPs to model rich joint predictive distributions.

[1]Code to reproduce the 1D regression experiments can be found at https://github.com/wesselb/NeuralProcesses.jl, and code to implement the image-completion experiments can be found at https://github.com/YannDubs/Neural-Process-Family.

(ii) We propose a simplified training procedure, discarding VI in favor of an approximate maximum-likelihood procedure, which improves performance for ConvLNPs.

(iii) We demonstrate the usefulness of ConvLNPs on toy time-series experiments, image-based sampling and extrapolation, and real-world environmental data sets.

4.2 Notation and Background

For convenience, we review some of the background concepts relevant for this chapter. Let $\mathcal{X} = \mathbb{R}^{d_{\text{in}}}, \mathcal{Y} = \mathbb{R}$ denote the input and output spaces, and let (x, y) be an input-output pair. Let $\mathcal{S}$ be the collection of all finite data sets, with $D^{(c)}, D^{(t)} \in \mathcal{S}$ a *context* and *target* set respectively. In this chapter, we will mainly be concerned with with the regression task of predicting the target set from the context set as in previous chapters. Let $x^{(c)}, y^{(c)}$ be the inputs and corresponding outputs of $D^{(c)}$, with $x^{(t)}, y^{(t)}$ defined analogously. We denote a single *task* as $\xi = (D^{(c)}, D^{(t)}) = ((\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}), (\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}))$. Let $\mathcal{P}(\mathcal{X})$ denote the collection of stochastic processes on $\mathcal{X}$, and let $C_b(\mathcal{X})$ denote the collection of continuous, bounded functions on $\mathcal{X}$.

We are concerned with meta-learning the map from context sets $D^{(c)}$ to the ground-truth predictive distribution called the prediction map (see Definition 1) $\pi_P : D^{(c)} \mapsto p(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(t)}, D^{(c)}) = p(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)} | \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}) / p(\boldsymbol{y}^{(c)} | \boldsymbol{x}^{(c)})$. The general prediction problem may then be viewed as learning to approximate π_P.

4.2.1 Translation Equivariance and Stationarity

The prediction map π_P possesses two important symmetries. First, π_P is *invariant* to permutations of $D^{(c)}$ (Gordon et al., 2020; Zaheer et al., 2017b). Second, if the ground truth process P is *stationary*, then π_P is *translation equivariant*: whenever an input to the map is translated, its output is translated by the same amount (see Section B.2 for formal definitions and proofs). This simple statement highlights the intimate relationship between stationarity and TE. Moreover, it suggests that models for the prediction map should also be TE and permutation invariant. As such models are a small subset of the space of *all* models, building in these properties can greatly improve data efficiency and generalization for stationary SP prediction. In Section 4.3, we extend the TE maps of the ConvCNP to construct a rich class of models which incorporate these inductive biases.

4.2.2 Convolutional Conditional Neural Processes

The ConvCNP from Chapter 3 is an important building block in the model proposed in this chapter. ConvCNPs can be viewed from the perspective of SP prediction, revealing their key limitations. Recall that the ConvCNP models the predictive distribution over target outputs as:

$$p_\phi(\boldsymbol{y}^{(t)}|\boldsymbol{y}^{(t)}, D^{(c)}) = \textstyle\prod_{(x^{(t)},y^{(t)})\in D^{(t)}} \mathcal{N}(y^{(t)}; \mu(x^{(t)}, D^{(c)}), \sigma^2(x^{(t)}, D^{(c)})). \tag{4.1}$$

The mean $\mu(\,\cdot\,, D^{(c)})$ and variance $\sigma^2(\,\cdot\,, D^{(c)})$ are parametrized by ConvDeepSets: a flexible parametrization for TE maps from $\mathcal{S}$ to $C_b(\mathcal{X})$. As we saw in Section 3.3.1, ConvDeepSets introduce the idea of *functional representations*: whereas the standard DeepSets framework embeds data sets into a finite-dimensional vector space (Zaheer et al., 2017b), a ConvDeepSet embeds data sets in an infinite-dimensional function space.

We observe that Equation (4.1) defines a map from context sets $D^{(c)}$ to predictive SPs. Specifically, let $\mathcal{P}_\mathrm{N}(\mathcal{X}) \subset \mathcal{P}(\mathcal{X})$ denote the set of *noise GPs*: GPs on $\mathcal{X}$ whose covariance is given by $\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2(\boldsymbol{x})\delta[\boldsymbol{x} - \boldsymbol{x}']$, where $\sigma^2 \in C_b(\mathcal{X})$ and $\delta[0] = 1$ with $\delta[\,\cdot\,] = 0$ otherwise. Then the ConvCNP is a map $\mathrm{ConvCNP} : \mathcal{S} \to \mathcal{P}_\mathrm{N}(\mathcal{X})$ with Equation (4.1) defining its finite-dimensional distributions. Since ConvDeepSets are TE, and the means and variances of ConvCNPs are ConvDeepSets, it follows that ConvCNPs are also TE as maps from $\mathcal{S} \to \mathcal{P}_\mathrm{N}(\mathcal{X})$. Unfortunately, processes in $\mathcal{P}_\mathrm{N}(\mathcal{X})$ possess two key limitations discussed in Section 3.6. First, it is impossible to obtain coherent function samples as each point of the function is generated independently. Second, Gaussian distributions cannot model multi-modality, heavy-tailedness, or asymmetry.

4.3 The Convolutional Latent Neural Process

We now present the ConvLNP, which addresses the above weaknesses of ConvCNPs. We introduce the parametrization (Section 4.3.1) and a maximum-likelihood meta-training procedure (Section 4.3.2).

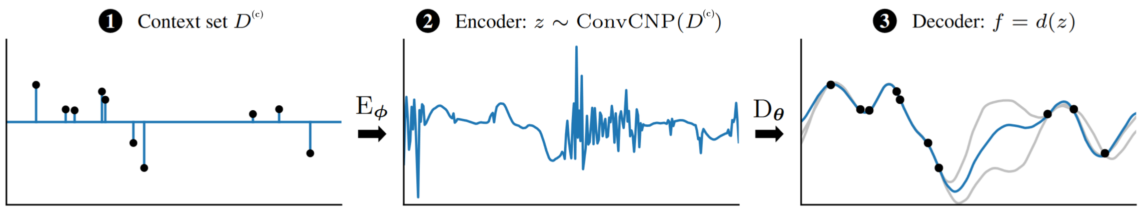

❶ Context set $D^{(c)}$
❷ Encoder: $z \sim \mathrm{ConvCNP}(D^{(c)})$
❸ Decoder: $f = d(z)$
$\mathrm{E}_{\boldsymbol{\phi}}$
$\mathrm{D}_{\boldsymbol{\theta}}$

Fig. 4.1 ConvLNP encoder-decoder architecture. The encoder is a ConvCNP which takes the context set as input (left panel) and outputs a single sample of z (center panel). The decoder takes this as input and outputs a predictive sample (right panel blue; two other samples shown in grey).

4.3.1 Parametrizing Translation Equivariant Maps to Stochastic Processes Using ConvLNPs

The convolutional latent neural process (ConvLNP) extends the ConvCNP by parametrizing a map to predictive SPs more expressive than $\mathcal{P}_{\mathrm{N}}(\mathcal{X})$, allowing for coherent sampling and non-Gaussian predictives. It achieves this by passing the output of a ConvCNP through a non-linear, TE map between function spaces. Specifically, the ConvLNP uses an encoder–decoder architecture, where the encoder $\mathrm{E}\colon \mathcal{S} \to \mathcal{P}_{\mathrm{N}}(\mathcal{X})$ is a ConvCNP and the decoder $d\colon \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^{\mathcal{X}}$ is TE (here $\mathbb{R}^{\mathcal{X}}$ denotes the set of all functions from $\mathcal{X}$ to $\mathbb{R}$). Conditioned on $D^{(c)}$, ConvLNP samples can be obtained by sampling a function $z \sim \mathrm{ConvCNP}(D^{(c)})$ and then computing $f = d(z)$. This is illustrated in Figure 4.1. Importantly, d takes functions to functions and does not necessarily act point-wise: letting $f(x)$ depend on the value of z at multiple locations is crucial for inducing dependencies in the predictive. This sampling procedure induces a map between SPs, $\mathrm{D}\colon \mathcal{P}_{\mathrm{N}}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$ (see Section B.3). Putting these together, with explicit parameter dependence in E and D, the ConvLNP is constructed as

$$\mathrm{ConvLNP}_{\boldsymbol{\theta},\boldsymbol{\phi}} = \mathrm{D}_{\boldsymbol{\theta}} \circ \mathrm{E}_{\boldsymbol{\phi}}, \quad \mathrm{E}_{\boldsymbol{\phi}} = \mathrm{ConvCNP}_{\boldsymbol{\phi}}, \quad \mathrm{D}_{\boldsymbol{\theta}} = (d_{\boldsymbol{\theta}})_*,$$

where $(d_{\boldsymbol{\theta}})_*$ is the pushforward[2] under $d_{\boldsymbol{\theta}}$. In Section B.3, we prove that $\mathrm{ConvLNP}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ is indeed TE.

In practice, we cannot compute samples of noise GPs ($\mathcal{P}_{\mathrm{N}}$) because they comprise uncountably many independent random variables. Instead, we consider a discrete version of the model, which enables computation. Following Gordon et al. (2020), we discretize the domain of z on a grid $(\boldsymbol{x}_i)_{i=1}^K$, with $\boldsymbol{z} := (z(\boldsymbol{x}_i))_{i=1}^K$. As a consequence, the

[2]i.e., $(d_{\boldsymbol{\theta}})_*(\mathrm{E}_{\boldsymbol{\phi}})$ is the measure induced on $\mathbb{R}^{\mathcal{X}}$ by sampling a function from $\mathrm{E}_{\boldsymbol{\phi}}$ and passing it through $d_{\boldsymbol{\theta}}$.

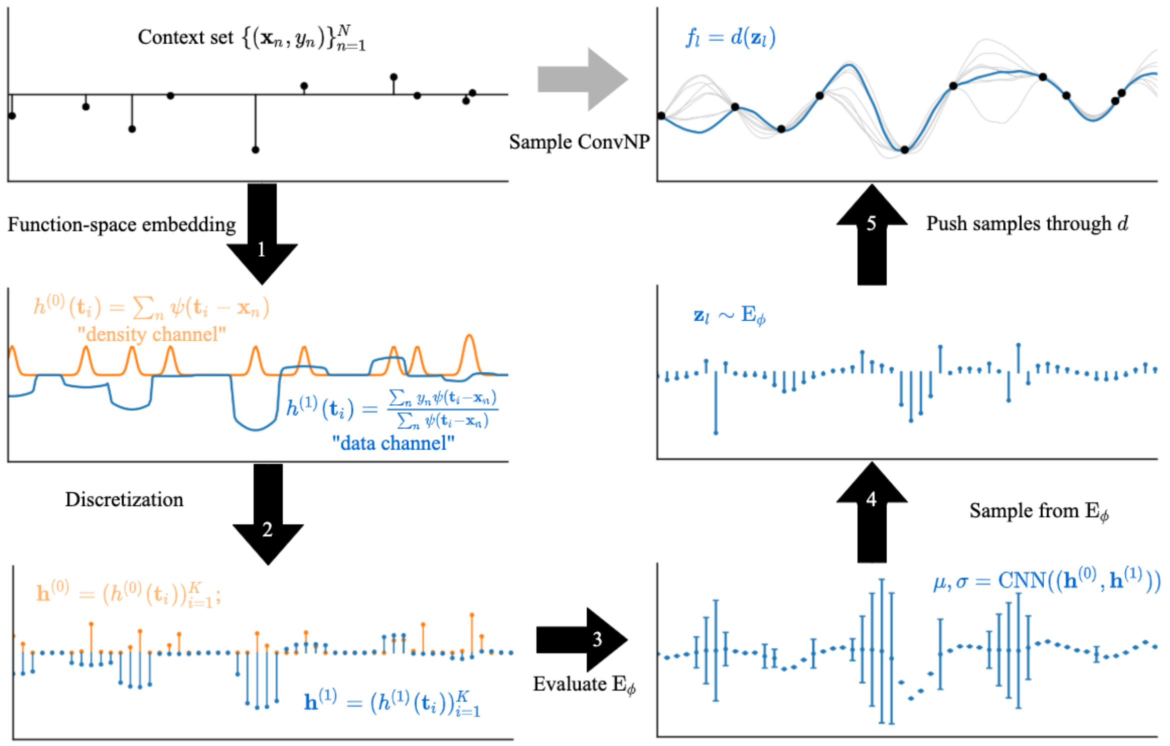

Context set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$
Sample ConvNP
$f_l = d(\mathbf{z}_l)$
Function-space embedding
1
$h^{(0)}(\mathbf{t}_i) = \sum_n \psi(\mathbf{t}_i - \mathbf{x}_n)$
"density channel"
$h^{(1)}(\mathbf{t}_i) = \frac{\sum_n y_n \psi(\mathbf{t}_i - \mathbf{x}_n)}{\sum_n \psi(\mathbf{t}_i - \mathbf{x}_n)}$
"data channel"
5
Push samples through d
$\mathbf{z}_l \sim \mathrm{E}_\phi$
Discretization
2
4
Sample from E_ϕ
$\mathbf{h}^{(0)} = (h^{(0)}(\mathbf{t}_i))_{i=1}^K;$
$\mathbf{h}^{(1)} = (h^{(1)}(\mathbf{t}_i))_{i=1}^K$
3
Evaluate E_ϕ
$\mu, \sigma = \mathrm{CNN}((\mathbf{h}^{(0)}, \mathbf{h}^{(1)}))$

Fig. 4.2 Forward pass of a ConvLNP. Steps (1)-(4) depict sampling from the encoder E_ϕ, which is a ConvCNP. This involves: (1) computing a functional representation of the context set, with separate 'density' and 'data' channels (described in Algorithm 2), (2) discretizing the representation, (3) passing the representation through a CNN, which outputs the parameters of independent Gaussian distributions spaced on a grid, and (4) sampling from these distributions. However, the samples at each grid point are independent of each other, hence in (5) the samples are passed through *another* CNN, the decoder, to induce dependencies, and then are smoothed out.

model can only be equivariant up to shifts on this discrete grid. With this discretization, sampling $\boldsymbol{z} \sim \text{ConvCNP}_{\boldsymbol{\phi}}(D^{(c)})$ amounts to sampling independent Gaussian random variables, and $d_{\boldsymbol{\theta}}$ is implemented by passing $\boldsymbol{z}$ through a CNN. The forward pass of a trained ConvLNP is illustrated in Figure 4.2. Note that CNNs are not always entirely TE due to the zero padding that occurs at each layer. In practice, we find that this is not an issue.[3] Following Kim et al. (2019), we define the model likelihood by adding heteroskedastic Gaussian observation noise $\sigma_y^2(\boldsymbol{x}, \boldsymbol{z})$ to the predictive function draws $f = d_{\boldsymbol{\theta}}(\boldsymbol{z}) \in \mathbb{R}^{\mathcal{X}}$:

$$p_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)}) = \mathbb{E}_{\boldsymbol{z}\sim \mathrm{E}_{\boldsymbol{\phi}}(D^{(c)})}\left[\prod\nolimits_{(x^{(t)},y^{(t)})\in D^{(t)}} \mathcal{N}\left(y^{(t)} \mid d_{\boldsymbol{\theta}}(\boldsymbol{z})(x^{(t)}), \sigma_y^2(x^{(t)}, \boldsymbol{z})\right)\right]. \tag{4.2}$$

Although the product in the expectation factorizes, $p_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})$ does not: $\boldsymbol{z}$ induces dependencies in the predictive, in contrast to Equation (4.1). Below we describe the details for the off-the-grid and on-the-grid ConvLNPs.

Off-the-grid ConvLNP. As described above, the encoder $\mathrm{E}_{\boldsymbol{\phi}}$ is defined by a ConvCNP, which provides a distribution over latent functions z. In practice, we consider the discretized version, where we denote the grid of discretization locations as $(\boldsymbol{t}_i)_{i=1}^K$, with $\boldsymbol{t}_i \in \mathcal{X}$. Let $p_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{t}_i, D^{(c)})$ denote the density of the latent function at the ith position, i.e. at $\boldsymbol{z}_i = z(\boldsymbol{t}_i)$. Then in order to sample $\boldsymbol{z} \sim \mathrm{E}_{\boldsymbol{\phi}}$ we specify the density of the entire discretized latent function $\boldsymbol{z}$ as:

$$p_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}) = \prod_{i=1}^{K} p_{\boldsymbol{\phi}}(\boldsymbol{z}_i|\boldsymbol{t}_i, D^{(c)}) = \prod_{i=1}^{K} \mathcal{N}(\boldsymbol{z}_i; \mu(\boldsymbol{t}_i, D^{(c)}), \sigma^2(\boldsymbol{t}_i, D^{(c)})), \tag{4.3}$$

where μ and σ^2 are parametrized by ConvDeepSets.

We express the ConvDeepSets as the composition of two functions $\Phi = \rho \circ E$. E maps a data set D to its functional representation via

$$E(D) = \sum_{(x,y)\in D} \phi(y)\psi(\,\cdot - x).$$

As with the ConvCNPs parametrization in Chapter 3, we set $\phi(y) = [1, y]^\intercal \in \mathbb{R}^2$, and ψ to be a radial basis function. $E(D)$ is itself discretized by evaluating it on a grid (which for simplicity we can also take to be $(\boldsymbol{t}_i)_{i=1}^K$).

Next, ρ maps the discretized $E(D)$ to a continuous function, which we denote $f = \rho(E(D))$. E is itself implemented in two stages. First a deep CNN maps the discretized $E(D)$ to a discretized output. Second, this discrete output is mapped to

[3]See Section A.4.6 for a discussion.

a continuous function by using the CNN outputs as weights for evenly-spaced basis functions (again employing radial basis functions), which we denote by ψ_ρ.

Whenever models output standard deviations, we enforce positivity via a function (e.g. the soft-plus function), which we denote $\mathrm{pos}(\cdot)$. Pseudo-code for a forward pass through an off-the-grid ConvCNP is provided in Algorithm 2. Note the forward pass involves the computation of a *density channel* $\boldsymbol{h}^{(0)}$, whose role intuitively is to allow the model to know where it has observed datapoints.

On-the-grid ConvLNP. Next, we describe the ConvLNP for on-the-grid data, which is used in our image and environmental experiments. This version is simpler to implement in practice, and is applicable whenever the input data is confined to a regular grid. As in Chapter 3 we choose the discretization $(\boldsymbol{t}_i)_{i=1}^K$ to be the pixel locations.

Let $\mathrm{I} \in \mathbb{R}^{H \times W \times C}$ be an image of dimensions H, W, C (height, width, and channels, respectively). We define a mask M_c, which is such that $[\mathrm{M}_c]_{i,j} = 1$ if pixel location (i, j) is in the context set, and 0 otherwise. Masking an image is then achieved via element-wise multiplication, denoted $\mathrm{M}_c \odot \mathrm{I}$. This allows us to flexibly define context and target sets for an image (target sets are typically considered as the complete image, so the masks M_c are simply binary-valued tensors with the same dimensions as the image). In this setting, we implement ϕ, by selecting the context points, and prepend the context mask: $\phi = [\mathrm{M}_c, \mathrm{Z}_c]^\top$. We then implement E by a simple convolutional layer, which we denote $\textsc{conv}_{\boldsymbol{\theta}}$ to emphasize that we use a standard 2d convolutional layer. Full pseudo-code for the on-the-grid ConvCNP is provided in Algorithm 3.

The ConvLNP can be implemented very simply by passing samples from the ConvCNP through an additional CNN decoder, which we denote $d_{\boldsymbol{\theta}}$. For an "off-the-grid" ConvLNP, similarly to the ConvCNP, we must map the output of a standard CNN back to functions on a continuous domain $\mathcal{X}$. This can be achieved via an RBF mapping, similar to the off-the-grid ConvCNP, e.g. Algorithm 3 lines 5–6. Pseudo-code for off- and on-the-grid ConvLNPs are provided in Algorithms 4 and 5, respectively. Note that for the ConvLNP, the discretization of the latent function $\boldsymbol{z}$ is typically on a pre-specified grid, and therefore lines 5 and 6 of Algorithm 2 are unnecessary when calling the ConvCNP (Algorithm 4, line 1).

Algorithm 4 Forward pass through ConvLNP (off-the-grid)

```
Require: $d = (\mathrm{CNN}, \psi_d)$, $\mathrm{E}_{\boldsymbol{\phi}}$ (off-the-grid ConvCNP), and number of samples $L$
Require: context $(x^{(c)}, y^{(c)})_{n=1}^{N}$, target $(x^{(t)})_{m=1}^{M}$
1: $\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z \leftarrow \mathrm{E}_{\boldsymbol{\phi}}(D^{(c)})$
2: for $\ell = 1, \ldots, L$ do
3:     $\boldsymbol{z}_\ell \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2)$
4:     $(f_\mu(\boldsymbol{t}_i), f_\sigma(\boldsymbol{t}_i))_{i=1}^{K} \leftarrow \mathrm{CNN}(\boldsymbol{z}_\ell)$
5:     $\boldsymbol{\mu}_{m,\ell} \leftarrow \sum_{i=1}^{T} f_\mu(\boldsymbol{t}_i)\psi_d(x_m^{(t)} - \boldsymbol{t}_i)$
6:     $\boldsymbol{\sigma}_{m,\ell} \leftarrow \mathrm{pos}\,(f_\sigma(\boldsymbol{t}_i))$
7: end for
8: return $(\boldsymbol{\mu}, \boldsymbol{\sigma})$
```

Algorithm 5 Forward pass through ConvLNP (on-the-grid)

```
Require: $d = \mathrm{CNN}$, $\mathrm{E}_{\boldsymbol{\phi}}$ (on-the-grid ConvCNP), and number of samples $L$
Require: image I, context mask $\mathrm{M}_c$, and target mask $\mathrm{M}_t$
1: $\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z \leftarrow \mathrm{E}_{\boldsymbol{\phi}}(I, M_c)$
2: for $\ell = 1, \ldots, L$ do
3:     $\boldsymbol{z}_\ell \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2)$
4:     $(f_\mu(\boldsymbol{t}_i), f_\sigma(\boldsymbol{t}_i))_{i=1}^{K} \leftarrow \mathrm{CNN}(\boldsymbol{z}_\ell)$
5:     $\boldsymbol{\mu} \leftarrow f_t^{(1:C)}$
6:     $\boldsymbol{\sigma} \leftarrow \mathrm{pos}\left(f_t^{(C+1:2C)}\right)$
7: end for
8: return $(\boldsymbol{\mu}, \boldsymbol{\sigma})$
```

4.3.2 Maximum Likelihood Learning of ConvLNPs

In this section we propose a maximum-likelihood training procedure for ConvLNPs. Let the ground truth task distribution be $p(\xi) = p(D^{(c)}, D^{(t)})$. Let

$$\mathcal{L}_{\mathrm{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) := \log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(t)}, D^{(c)}) \tag{4.4}$$

be the single-task likelihood, and let

$$\mathcal{L}_{\mathrm{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}) := \mathbb{E}_{p(\xi)}[\log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(t)}, D^{(c)})] \tag{4.5}$$

be the task-averaged likelihood. The following proposition shows that maximizing $\mathcal{L}_{\mathrm{ML}}$ recovers the prediction map π_P in a suitable limit:

Assume tasks $\xi = (D^{(c)}, D^{(t)})$ are generated as follows: first, some finite number of input locations $\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}$ are sampled. Assume that $\Pr(|\boldsymbol{x}^{(t)}| = n) > 0$ for all $n \in \mathbb{Z}_{\geq 0}$, where $|\boldsymbol{x}^{(c)}|$ denotes the number of datapoints in $\boldsymbol{x}^{(t)}$, and assume the same is true of

$\Pr(|\boldsymbol{x}^{(c)}| = n)$. Further assume that for each $n > 0$, the distribution of $\boldsymbol{x}$ given $|\boldsymbol{x}| = n$ has a continuous density with support over all of $\mathbb{R}^{n \times d_{\text{in}}}$. Next, we sample $\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)}$ from the finite marginal of the ground truth stochastic process P, which has density $p(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)}|\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)})$. Finally, we set $(D^{(c)}, D^{(t)}) := ((\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}), (\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}))$.

Proposition 1. Let $\Psi : \mathcal{S} \to \mathcal{P}(\mathcal{X})$ be any map from data sets to stochastic processes, and let $\mathcal{L}_{\text{ML}}(\Psi) := \mathbb{E}_{p(\xi)}[\log p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})]$, where the density p_Ψ is that of $\Psi(D^{(c)})$ evaluated at $\boldsymbol{x}^{(t)}$. Then Ψ globally maximises $\mathcal{L}_{\text{ML}}$ if and only if $\Psi = \pi_P$, the prediction map.

Proof. We have:

$$\mathcal{L}_{\text{ML}}(\Psi) = \mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)}, \boldsymbol{y})}\left[\log p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})\right] \tag{4.6}$$

$$= \mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)})}\left[\mathbb{E}_{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}\left[\log p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})\right]\right] \tag{4.7}$$

$$= \mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)})}\left[\mathbb{E}_{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}\left[\log \frac{p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})} + \log p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})\right]\right] \tag{4.8}$$

$$= \mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)})}\left[\mathbb{E}_{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}\left[\log \frac{p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}\right]\right] + \mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)})}\left[\mathbb{E}_{p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})}\left[\log p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})\right]\right] \tag{4.9}$$

$$= -\mathbb{E}_{p(D^{(c)}, \boldsymbol{x}^{(t)})}\left[\mathrm{KL}\left(p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)}) \middle\| p_\Psi(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})\right)\right] + \text{constant}, \tag{4.10}$$

where the additive constant is constant with respect to Ψ. First note that the KL-divergence is non-negative, and that the prediction map sends all the KL-divergences to zero, globally optimising $\mathcal{L}(\Psi)$. Furthermore, the KL-divergence is equal to zero if and only if the two distributions are equal, and this must hold for (almost) all $\boldsymbol{x}^{(t)}, D^{(c)}$. For, if this were not the case, the KL-divergence would contribute a non-zero amount to the expectation in Equation (4.10). □

Note that $\log p$ is a strictly proper scoring rule (Gneiting and Raftery, 2007) so it is maximized exactly when

$$p_\Psi(\cdot|\boldsymbol{x}^{(t)}, D^{(c)}) = \pi_P(D^{(c)})(\cdot|\boldsymbol{x}^{(t)}),$$

as shown. Strictly speaking, this argument only shows that the finite marginals of the prediction map and Ψ must be equal for almost all $(D^{(c)}, \boldsymbol{x}^{(t)})$ with respect to $p(D^{(c)}, \boldsymbol{x}^{(t)})$. Since the task generation procedure outlined in this section assumes a finite probability of generating any finite-sized context and target set, this is not very restrictive. However, in practice we often limit the maximum size of the sampled data sets, and also their range in $\mathcal{X}$ space. Hence we can only expect the model to learn reasonable predictions within the ranges seen during train time.

In practice, we do not have infinite flexibility in our model or infinite data to compute expectations over $p(\xi)$, but Proposition 1 shows that maximum-likelihood training is sensible with an expressive model and sufficient data. Letting $\mathcal{D} = \{\xi_n\}_{n=1}^{N_{\text{tasks}}}$ be a *meta-training* set, we can train a ConvLNP by stochastic gradient maximization of $\mathcal{L}_{\text{ML}}$ with tasks sampled from $\mathcal{D}$. Unfortunately, for non-linear decoders, $\log p_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})$ is intractable due to the expectation over $\boldsymbol{z}$ (Equation (4.2)). For a given task ξ, we instead optimize the following Monte Carlo estimate of $\mathcal{L}_{\text{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi)$, which is conservatively biased, consistent, and monotonically increasing in L (in expectation) (Burda et al., 2015):

$$\hat{\mathcal{L}}_{\text{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) := \log \left[\tfrac{1}{L} \textstyle\sum_{\ell=1}^{L} \exp\left(\sum_{(x^{(t)},y^{(t)})\in D^{(t)}} \log p_{\boldsymbol{\theta}}(y^{(t)}|x^{(t)}, \boldsymbol{z}_\ell)\right)\right]; \quad \boldsymbol{z}_\ell \sim \mathrm{E}_{\boldsymbol{\phi}}(D^{(c)}). \tag{4.11}$$

One drawback of this objective is that single sample estimators are not useful, as they drive $\boldsymbol{z}$ to be deterministic.

4.4 The Latent Variable Interpretation of ConvLNPs

We now describe an alternative approach to training the ConvLNP via variational lower bound maximization. This serves the dual purpose of relating ConvLNPs to the NP family, and contrasting the existing NP framework with our simplified, maximum-likelihood approach from Section 4.3.2.

4.4.1 A Variational Lower Bound Approach to ConvLNPs

In this section we review the VI objective for the LNP discussed in Section 2.2.4. Garnelo et al. (2018b) propose viewing Neural Processes as performing approximate Bayesian inference and learning in the following latent variable model, illustrated in Figure 2.1:

$$\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}); \quad y(x) = f_{\boldsymbol{\theta}}(x; \boldsymbol{z}); \quad p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) = \textstyle\prod_{(x^{(t)},y^{(t)})\in D^{(t)}} \mathcal{N}\left(y^{(t)}; f_{\boldsymbol{\theta}}(x^{(t)}; \boldsymbol{z}), \sigma_y^2\right). \tag{4.12}$$

To train the model, they propose using *amortized* VI (Kingma and Welling, 2014; Rezende and Mohamed, 2015). This approach involves introducing a variational approximation $q_{\boldsymbol{\phi}}$ which maps data sets $S \in \mathcal{S}$ to distributions over $\boldsymbol{z}$, and maximizing a lower bound (ELBO) on $\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(c)}|\boldsymbol{x}^{(t)}, D^{(c)})$. We can define a similar procedure for ConvLNPs. For ConvLNPs, $\boldsymbol{z}$ is a latent *function*, $q_{\boldsymbol{\phi}}$ is a map from data sets to SPs, and $f_{\boldsymbol{\theta}}$ is a map between function spaces. A natural choice is to use a ConvCNP and CNN for $q_{\boldsymbol{\phi}}$ and $f_{\boldsymbol{\theta}}$, respectively. This results in the same parameterization as in Section 4.3, but a different modelling interpretation and meta-training objective. Given a task $\xi = (D^{(c)}, D^{(t)})$, the ELBO for this model is:

$$\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}\cup D^{(t)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(c)}|\boldsymbol{x}^{(c)}, \boldsymbol{z})\right] - \mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)} \cup D^{(t)}) \| \, p(\boldsymbol{z}|D^{(c)})).$$

As $p(\boldsymbol{z}|D^{(c)})$ is intractable to compute, Garnelo et al. (2018b) instead propose the following objective:

$$\mathcal{L}_{\mathrm{LNP}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) \coloneqq \\ \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}\cup D^{(t)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(c)}|\boldsymbol{x}^{(c)}, \boldsymbol{z})\right] - \mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)} \cup D^{(t)}) \| q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})), \quad (4.13)$$

where the intractable term $p(\boldsymbol{z}|D^{(c)})$ has been substituted with our variational approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})$. Due to this substitution, $\mathcal{L}_{\mathrm{LNP}}$ is no longer a valid ELBO for the original model (Equation (4.12)). Rather, if we define *separate* models for each context set $D^{(c)}$, and *define* the conditional prior for each model as $p(\boldsymbol{z}|D^{(c)}) \coloneqq q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})$, then $\mathcal{L}_{\mathrm{LNP}}$ may be thought of as performing VI in this *collection* of models. However, there is no guarantee that these conditional priors are consistent in the sense that they correspond to a single Bayesian model as in Equation (4.12).

For the non-discretized ConvLNP, Equation (4.13) involves KL divergences between SPs which cannot be computed directly and must be treated carefully (Matthews et al., 2016; Sun et al., 2019). On the other hand, for the discretized ConvLNP, the KL divergences can be computed, but grow in magnitude as the discretization becomes finer, and it is not clear that the KL divergence between SPs is recovered in the limit. This raises practical issues for the use of Equation (4.13) with the ConvLNP, as the balance between the two terms depends on the choice of discretization.

4.4.2 Maximum-Likelihood vs Variational Lower Bound Maximization for Training NPs

We argue that the VI interpretation is unnecessary when focusing on predictive performance, and particularly detrimental for ConvLNPs, where $\boldsymbol{z}$ has many elements. Let $D \coloneqq D^{(c)} \cup D^{(c)}$, and let $Z = \int p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}) \, \mathrm{d}\boldsymbol{z}$. Note that:

$$
\begin{aligned}
\mathcal{L}_{\mathrm{LNP}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) &\coloneqq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z})] - \mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|D) \| q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})) && (4.14) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) + \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)] && (4.15) \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)}\left[\log Z + \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})}{Z} - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)\right] && (4.16) \\
&= \log Z - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|D) \middle\| \frac{1}{Z} p_{\boldsymbol{\theta}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, \boldsymbol{z}) q(\boldsymbol{z}|D^{(c)})\right). && (4.17)
\end{aligned}
$$

If we identify the approximate posterior $q_{\boldsymbol{\phi}}$ with the encoder of the maximum-likelihood ConvLNP, (which in the maximum-likelihood framework does not have an approximate inference interpretation), then $\log Z = \mathcal{L}_{\mathrm{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi)$:

$$
\mathcal{L}_{\mathrm{LNP}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) = \mathcal{L}_{\mathrm{ML}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)} \cup D^{(t)}) \middle\| p_{\boldsymbol{\theta}}(D^{(t)}|\boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)})/Z\right), \tag{4.18}
$$

we see that $\mathcal{L}_{\mathrm{LNP}}$ is equal to $\mathcal{L}_{\mathrm{ML}}$ up to an additional KL term. This KL term encourages consistency among the $q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)$ in the sense that Bayes' theorem is respected if the target set is subsumed into the context set. In the infinite capacity/data limit, $\mathcal{L}_{\mathrm{LNP}}$ is globally maximized if the ConvLNP recovers (i) the prediction map π_P for $\boldsymbol{y}^{(t)}$ and (ii) exact inference for $\boldsymbol{z}$. This follows from (i) Proposition 1, since π_P globally optimizes $\mathcal{L}_{\mathrm{ML}}$; and (ii) that exact inference for $\boldsymbol{z}$ is Bayes-consistent, sending the KL term to zero. In most applications, only the distribution over $\boldsymbol{y}^{(t)}$ is of interest. Given only finite capacity/data, it can be advantageous to not expend capacity in enforcing Bayes-consistency for $\boldsymbol{z}$, which suggests it could be beneficial to use $\mathcal{L}_{\mathrm{ML}}$ over $\mathcal{L}_{\mathrm{LNP}}$. Further, $\mathcal{L}_{\mathrm{ML}}$ has the advantage of being easy to specify for any map parameterizing a predictive process, posing no conceptual issues for the ConvLNP. In Section 4.5 we find that $\mathcal{L}_{\mathrm{ML}}$ significantly outperforms $\mathcal{L}_{\mathrm{LNP}}$ for ConvLNPs, and often also for ALNPs.

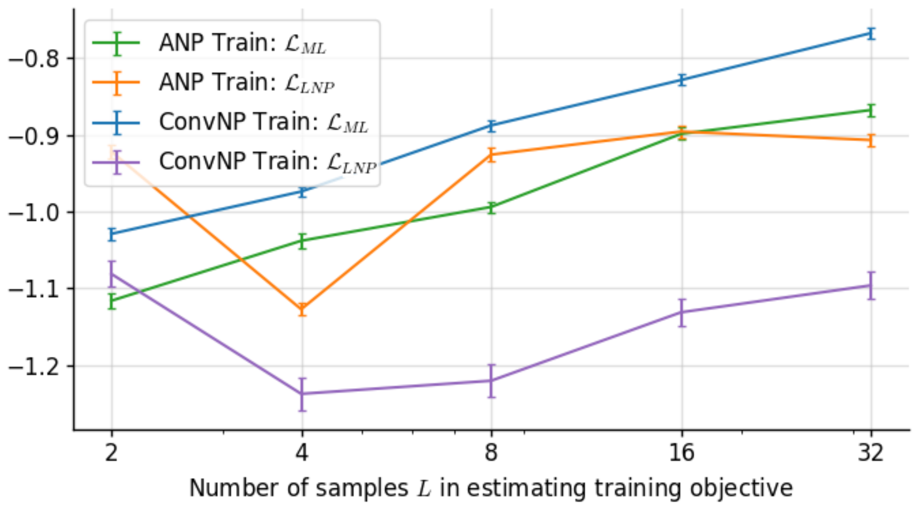


Fig. 4.3 Interpolation performance (within training range) for context set sizes uniformly sampled from $\{0, \ldots, 50\}$ of the ConvLNP and ALNP on Matérn–$\frac{5}{2}$ samples. The models are trained with $\mathcal{L}_{\mathrm{ML}}$ and $\mathcal{L}_{\mathrm{LNP}}$ for various number of samples L. Models trained with $\mathcal{L}_{\mathrm{ML}}$ are evaluated with $\mathcal{L}_{\mathrm{ML}}$, while models trained with $\mathcal{L}_{\mathrm{LNP}}$ are evaluated with $\mathcal{L}_{\mathrm{ML}}$. At evaluation, all bounds are estimated using 2,048 samples.

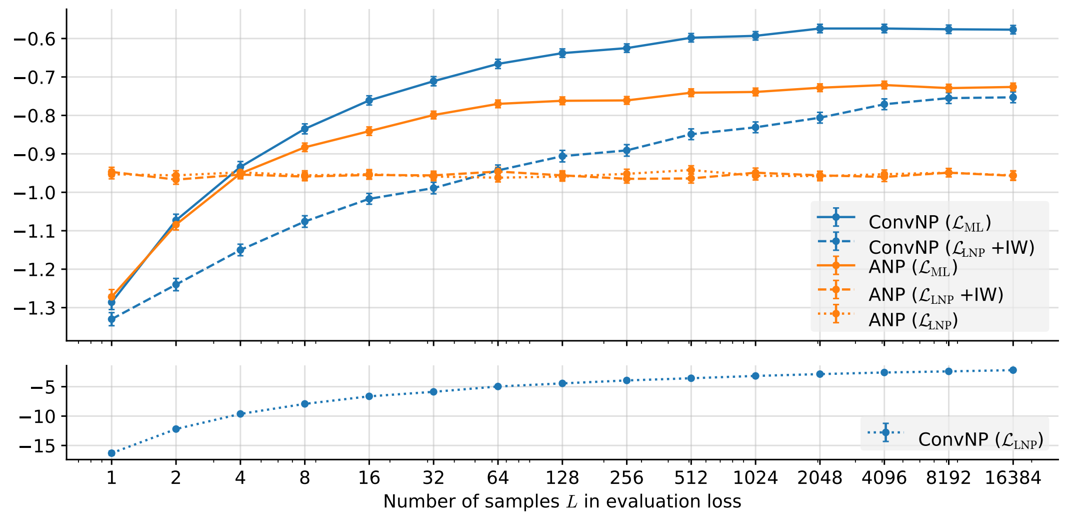
4.4.3 Effect of Number of Samples Used During Training

Figure 4.3 shows the effect of the number of samples L in the training objectives on the performance of the ConvLNP and ALNP. Observe that the performance of $\mathcal{L}_{\mathrm{ML}}$ reliably increases with the number of samples L and that $\mathcal{L}_{\mathrm{ML}}$ outperforms $\mathcal{L}_{\mathrm{LNP}}$. The performance for $\mathcal{L}_{\mathrm{LNP}}$ does not appear to increase with the number of samples L and appears more noisy than $\mathcal{L}_{\mathrm{ML}}$. Note that the models used for Figure 4.3 were trained with homoskedastic observation noise. This is achieved by pooling f_σ over the time dimension.

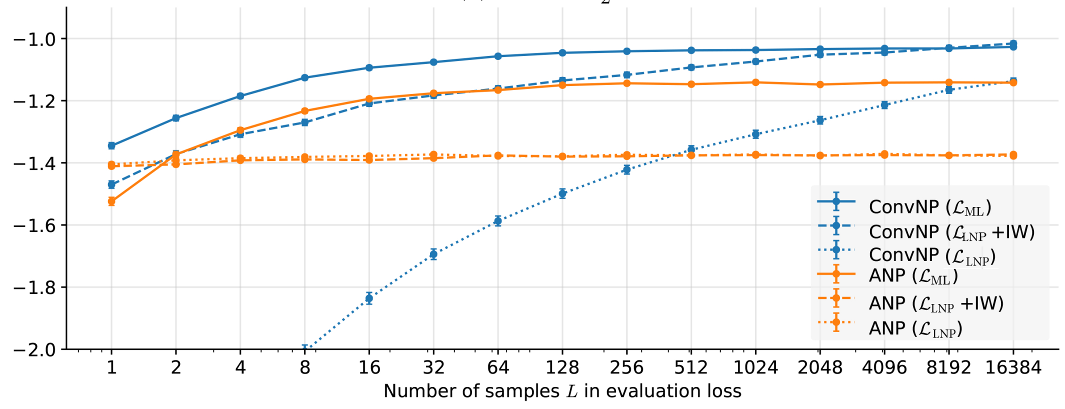
In our experiments in Section 4.5, we set L between 16 and 32.

4.4.4 Effect of Number of Samples Used for Evaluation

As the true log-likelihoods of NP-based models are intractable, quantitative evaluation and comparison of models is challenging. Instead, we compare models by using an estimate of the log-likelihood. A natural candidate is $\mathcal{L}_{\mathrm{ML}}$. However, unless large L is used, $\mathcal{L}_{\mathrm{ML}}$ is conservative and tends to significantly underestimate the log-likelihood. One way to improve the estimate of $\mathcal{L}_{\mathrm{ML}}$ is through importance weighting (IW) (Le



(a) Matérn–$\frac{5}{2}$



(b) Weakly periodic kernel

Fig. 4.4 Log-likelihood bounds achieved by various combination of models and training objectives when evaluated with $\mathcal{L}_{\mathrm{ML}}$ and $\mathcal{L}_{\mathrm{IW}}$ for various numbers of samples L. Color indicates model. Solid lines correspond to models trained and evaluated with $\mathcal{L}_{\mathrm{ML}}$. Dashed lines correspond to models trained with $\mathcal{L}_{\mathrm{LNP}}$ and evaluated with $\mathcal{L}_{\mathrm{IW}}$. Dotted lines correspond to models trained with $\mathcal{L}_{\mathrm{LNP}}$ and evaluated with $\mathcal{L}_{\mathrm{LNP}}$.

et al., 2018; Wu et al., 2016). The encoder $\mathrm{E}_{\boldsymbol{\phi}}(D)$ can be used as a proposal distribution:

$$\hat{\mathcal{L}}_{\mathrm{IW}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \xi) := \log \left(\frac{1}{L} \sum_{\ell=1}^{L} \exp \left(\log w(\boldsymbol{z}_\ell) + \sum_{(x^{(t)}, y^{(t)}) \in D^{(t)}} \log p_{\boldsymbol{\theta}}(y^{(t)} | x^{(t)}, \boldsymbol{z}_\ell) \right) \right), \tag{4.19}$$

where $\boldsymbol{z}_\ell \sim \mathrm{E}_{\boldsymbol{\phi}}(D)$, and where the importance weights are given by $\log w(\boldsymbol{z}_\ell) := \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|D^{(c)}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)$. Here $q_{\boldsymbol{\phi}}(\boldsymbol{z}|D)$ is the density of the encoder distribution. We find that training models with $\mathcal{L}_{\mathrm{ML}}$ results in encoders that are ill-suited as proposal distributions, so we only use $\mathcal{L}_{\mathrm{IW}}$ to evaluate models trained with $\mathcal{L}_{\mathrm{LNP}}$.

Figure 4.4 demonstrates the effect of the number of samples L used to estimate the evaluation objective for the ConvLNP and ALNP trained with $\mathcal{L}_{\mathrm{ML}}$ and $\mathcal{L}_{\mathrm{LNP}}$. The models used to generate Figure 4.4 are the same models used in Section 4.5.1, i.e. having heteroskedastic noise. Observe the general trend that the log-likelihood estimates tend to increase with L, as expected. The ALNP trained with $\mathcal{L}_{\mathrm{LNP}}$ collapsed to a conditional ALNP, meaning that the encoder became deterministic; in that case, $\mathcal{L}_{\mathrm{ML}}$ is exact, which means that larger L and importance weighting will not increase the estimate. In contrast, the ALNP trained with $\mathcal{L}_{\mathrm{ML}}$ did not collapse, and we see that there the estimate increases with L. For the ConvLNP trained with $\mathcal{L}_{\mathrm{LNP}}$, evaluating with $\mathcal{L}_{\mathrm{IW}}$ yields a significant increase, showing that the bound estimated with $\mathcal{L}_{\mathrm{IW}}$ is very loose. The models trained with $\mathcal{L}_{\mathrm{ML}}$ tend to be the best performing, although the ConvLNP trained with $\mathcal{L}_{\mathrm{LNP}}$ is best for weakly periodic kernel and appears to still be increasing with L.

In both the main and the supplement, all log-likelihood lower bounds reported are computed with $\mathcal{L}_{\mathrm{ML}}$ if the model was trained using $\mathcal{L}_{\mathrm{ML}}$ and with $\mathcal{L}_{\mathrm{IW}}$ if the model was trained using $\mathcal{L}_{\mathrm{LNP}}$.

4.5 Experiments and Results

We evaluate ConvLNPs on a broad range of tasks. Our main questions are: (i) Does the ConvLNP produce coherent, meaningful predictive samples? (ii) Can it leverage translation equivariance to outperform baseline methods within and beyond the training range (generalization)? (iii) Does it learn expressive non-Gaussian predictive distributions?

Evaluation and baselines. We use several approaches for evaluating NPs. First, as in (Garnelo et al., 2018b; Kim et al., 2019), we provide qualitative comparisons of samples. These allow us to see if the models display meaningful structure, quantify

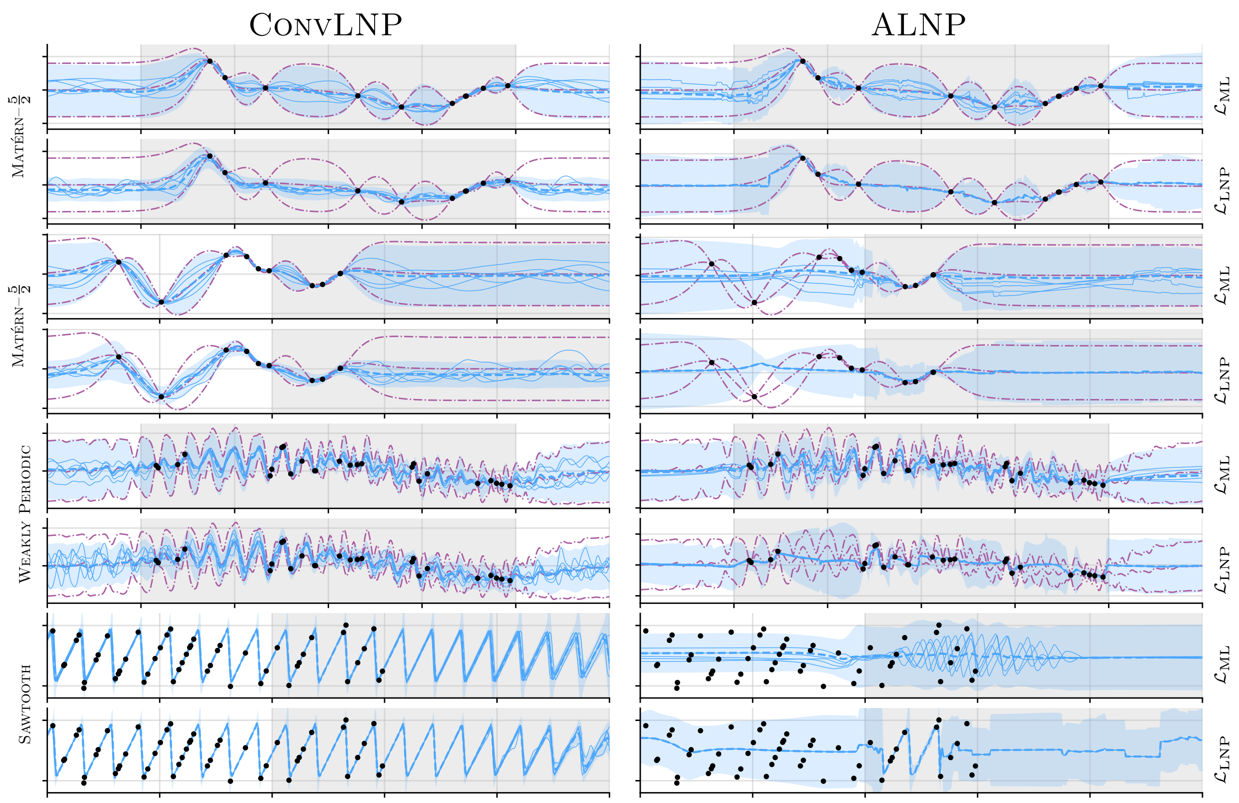


Fig. 4.5 Predictions of ConvNPs and ALNPs trained with $\mathcal{L}_{\text{ML}}$ and $\mathcal{L}_{\text{LNP}}$, showing interpolation and extrapolation within (grey background) and outside (white background) the training range. Solid blue lines are samples, dashed blue lines are means, and the shaded blue area is $\mu \pm 2\sigma$. Purple dash–dot lines are the ground-truth GP mean and $\mu \pm 2\sigma$. ConvNP handles points outside the training range naturally, whereas this leads to catastrophic failure for the ALNP. Note ALNP with $\mathcal{L}_{\text{LNP}}$ tends to collapse to deterministic samples, with all uncertainty explained with the heteroskedastic noise. In contrast, models trained with $\mathcal{L}_{\text{ML}}$ show diverse samples that account for much of the uncertainty.

uncertainty, and are able to generalize spatially. Second, NPs lack closed-form likelihoods, so we evaluate *lower bounds* on their predictive log-likelihoods via importance sampling (Le et al., 2018). As these bounds can be quite loose (Section 4.4.4), they are primarily useful to show when NPs outperform baselines with *exact* likelihoods, such as GPs and ConvCNPs. Finally, in Section 4.5.3 we consider Bayesian optimization to evaluate the usefulness of ConvLNPs for downstream tasks. In Sections 4.5.1 and 4.5.2, we compare against the Attentive LNP (ALNP; (Kim et al., 2019)), which in prior work is trained with $\mathcal{L}_{\text{LNP}}$. The ALNP architectures used here are comparable to those in Kim et al. (2019), and have a parameter count comparable to or greater than the ConvLNP. Full details provided in Section B.4.

		WITHIN TRAINING RANGE			BEYOND TRAINING RANGE		
		Matérn-$\frac{5}{2}$	Weakly Per.	Sawtooth	Matérn-$\frac{5}{2}$	Weakly Per.	Sawtooth
GP	(full)	$1.22 \pm 6\text{E}{-3}$	$-0.06 \pm 5\text{E}{-3}$	N/A	$1.22 \pm 6\text{E}{-3}$	$-0.06 \pm 5\text{E}{-3}$	N/A
ConvLNP*	($\mathcal{L}_{\text{ML}}$)	$\mathbf{-0.58} \pm 0.01$	$\mathbf{-1.02} \pm 6\text{E}{-3}$	$\mathbf{2.30} \pm 0.01$	$\mathbf{-0.58} \pm 0.01$	$\mathbf{-1.03} \pm 6\text{E}{-3}$	$\mathbf{2.29} \pm 0.02$
ALNP*	($\mathcal{L}_{\text{ML}}$)	-0.73 ± 0.01	$-1.14 \pm 6\text{E}{-3}$	$0.09 \pm 3\text{E}{-3}$	$-1.39 \pm 7\text{E}{-3}$	$-1.35 \pm 4\text{E}{-3}$	$-0.17 \pm 1\text{E}{-3}$
ALNP*	($\mathcal{L}_{\text{LNP}}$)	-0.96 ± 0.01	$-1.37 \pm 6\text{E}{-3}$	$0.20 \pm 9\text{E}{-3}$	$-1.48 \pm 4\text{E}{-3}$	-1.66 ± 0.01	$-0.30 \pm 4\text{E}{-3}$
GP	(diag)	$-0.84 \pm 9\text{E}{-3}$	$-1.17 \pm 5\text{E}{-3}$	N/A	$-0.84 \pm 9\text{E}{-3}$	$-1.17 \pm 5\text{E}{-3}$	N/A
ConvCNP		-0.88 ± 0.01	$-1.19 \pm 7\text{E}{-3}$	1.15 ± 0.04	-0.87 ± 0.01	$-1.19 \pm 7\text{E}{-3}$	1.11 ± 0.04

Table 4.1 Log-likelihoods on 1D regression tasks. Lower bounds marked with asterisk. Highest non-GP values in bold.

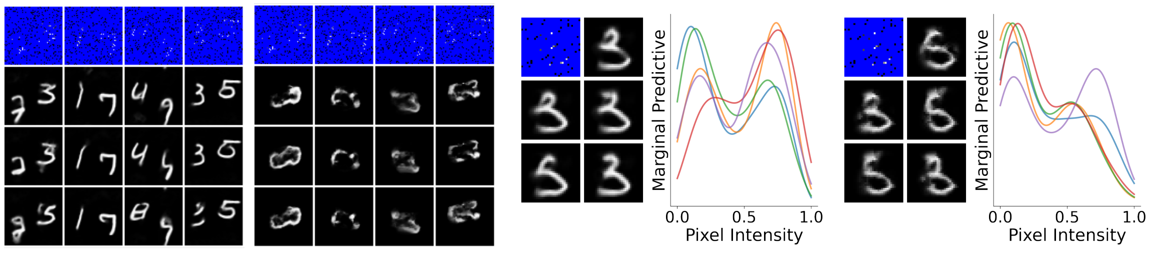
4.5.1 1D Regression

We train on samples from (i) a Matérn-$\frac{5}{2}$ GP, (ii) a weakly periodic GP, and (iii) a non-Gaussian sawtooth process with random shifts and frequency (see Section B.4 for details). Figure 4.5 shows predictive samples, where during training the models only observe data within the grey regions (training range). While samples from the ALNP exhibit unnatural "kinks" and do not resemble the underlying process, the ConvLNP produces smooth samples for Matérn–$\frac{5}{2}$ and samples exhibiting meaningful structure for the weakly periodic and sawtooth processes. The ConvLNP also generalizes gracefully beyond the training range, whereas ALNP fails catastrophically. The ALNP with $\mathcal{L}_{\text{LNP}}$ collapses to deterministic samples, with the epistemic uncertainty explained using the heteroskedastic noise $\sigma_y^2(\boldsymbol{x}, \boldsymbol{z})$. This was also noted in Le et al. (2018). This behaviour is alleviated when training with $\mathcal{L}_{\text{ML}}$, with much of the predictive uncertainty due to variations in the sampled functions.

Table 4.1 compares lower bounds on the log-likelihood for ConvLNP with our proposed $\mathcal{L}_{\text{ML}}$ objective and ALNP with both $\mathcal{L}_{\text{ML}}$ and the standard $\mathcal{L}_{\text{LNP}}$ objective. We also show three *exact* log-likelihoods: (i) the ground-truth GP (full) (ii) the ground-truth GP with diagonalised predictions (diag), and (iii) ConvCNP. The ConvCNP performs on par with GP (diag), which is the optimal factorized predictive. The ConvLNP lower bound is consistently higher than the GP (diag) and ConvCNP log-likelihoods, demonstrating that its correlated predictives improve predictive performance. Further, the ConvLNP performs similarly inside and outside its training range, demonstrating that TE helps generalization; this is in contrast to the ALNP, which fails catastrophically outside its training range. In Section B.5, we provide a thorough comparison for multiple models, training objectives, and data sets.

	MNIST		CelebA32		SVHN		ZSMM	
	$\mathcal{L}_{\text{ML}}$	$\mathcal{L}_{\text{LNP}}$	$\mathcal{L}_{\text{ML}}$	$\mathcal{L}_{\text{LNP}}$	$\mathcal{L}_{\text{ML}}$	$\mathcal{L}_{\text{LNP}}$	$\mathcal{L}_{\text{ML}}$	$\mathcal{L}_{\text{LNP}}$
ConvLNP	**2.11** $\pm$ 0.01	0.99 $\pm$ 0.42	**6.92** $\pm$ 0.10	-0.27 ± 0.00	**9.89** $\pm$ 0.09	0.17 $\pm$ 0.00	**4.58** $\pm$ 0.04	0.14 $\pm$ 0.00
ALNP	1.66 $\pm$ 0.03	1.64 $\pm$ 0.03	5.98 $\pm$ 0.08	6.04 $\pm$ 0.10	9.18 $\pm$ 0.08	8.91 $\pm$ 0.06	-10.8 ± 1.99	-6.45 ± 0.99

Table 4.2 Test log-likelihood lower bounds for image completion (5 runs).



(a) ConvLNP (b) ALNP (c) ConvLNP (d) ALNP

Fig. 4.6 Left two plots: predictive samples on zero-shot multi MNIST. Right two plots: samples and marginal predictives on standard MNIST. We plot the density of the five marginals that maximize Sarle's bimodality coefficient Ellison (1987). We use $\mathcal{L}_{\text{ML}}$ for training. Blue pixels are not in the context set.

4.5.2 Image Completion

We evaluate ConvLNPs on image completion tasks focusing on spatial generalization. To test this, we consider zero-shot multi MNIST (ZSMM), where we train on single MNIST digits but test on two MNIST digits on a larger canvas. We randomly translate the digits during training, so the generative SP is stationary. The black background on MNIST causes difficulty with heteroskedastic noise, as the models can obtain high likelihood by predicting the background with high confidence whilst ignoring the digits. Hence for MNIST and ZSMM we use homoskedastic noise $\sigma_y^2(\boldsymbol{z})$. Figures 4.6a and 4.6b show that the ALNP fails to generalize spatially, whereas this is naturally handled by the ConvLNP.

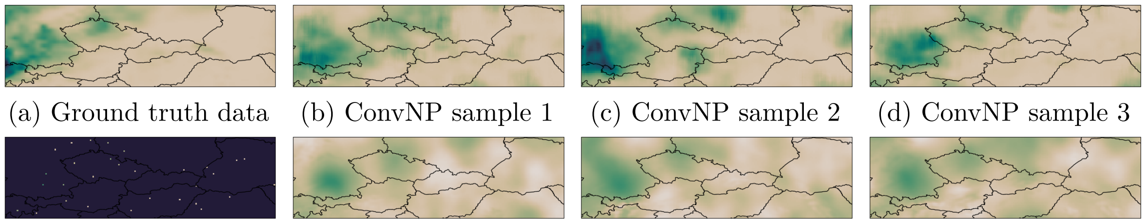
We also test the ConvLNP's ability to learn non-Gaussian predictive distributions. Figure 4.6c shows that the ConvLNP can learn highly multimodal predictives, enabling the generation of diverse yet coherent samples. A quantitative comparison of models using log-likelihood lower bounds is provided in Table 4.2, where ConvLNP trained with $\mathcal{L}_{\text{ML}}$ consistently achieves the highest values. Section B.6 provides details regarding the data, architectures, and protocols used in our image experiments. In Section B.7, we provide samples and further quantitative comparisons of models trained on SVHN (Netzer et al., 2011), MNIST LeCun et al. (1989), and 32×32 CelebA Netzer et al. (2011) in a range of scenarios, along with full experimental details.

		Central (train)	West (test)	East (test)	South (test)
LL	ConvLNP	**4.47** $\pm$ 0.07	**4.55** $\pm$ 0.08	**5.07** $\pm$ 0.07	**4.65** $\pm$ 0.08
	GP	3.33 $\pm$ 0.06	3.65 $\pm$ 0.06	4.07 $\pm$ 0.06	3.34 $\pm$ 0.06
RMSE ($\times 10^{-2}$)	ConvLNP	**5.72** $\pm$ 0.33	**5.77** $\pm$ 0.37	**3.23** $\pm$ 0.22	**6.92** $\pm$ 0.39
	GP	**6.26** $\pm$ 0.30	**5.75** $\pm$ 0.29	**3.10** $\pm$ 0.18	7.94 $\pm$ 0.44

Table 4.3 Joint predictive log-likelihoods (LL) and RMSEs on ERA5-Land, averaged over 1000 tasks.

4.5.3 Environmental Data

We next consider a real-world data set, ERA5-Land (Service, 2020), containing environmental measurements at a ~9 km spacing across the globe. We consider predicting daily precipitation y at position $\boldsymbol{x}$. We also provide the model with orography (elevation) and temperature values. We choose a large region of central Europe as our train set, and use regions east, west and south as held-out test sets. For such tasks, models must be able to make predictions at locations spanning a range different from the training set, inhibiting the deployment of NPs not equipped with TE. To sample a task at train time, we sample a random date between 1981 and 2020, then sample a sub-region within the train region, which is split into context and target sets. In this section, we train using $\mathcal{L}_{\mathrm{ML}}$. See Section B.8 for details.


(a) Ground truth data (b) ConvNP sample 1 (c) ConvNP sample 2 (d) ConvNP sample 3

(e) Context set (f) GP sample 1 (g) GP sample 2 (h) GP sample 3

Fig. 4.7 Predictive samples overlaid on central Europe. Darker colours show higher precipitation. In (e), coloured pixels represent context points. GP samples often take negative values (lighter than ground truth data, see Section B.8.2 for a discussion), whereas the LNP has learned to produce non-negative samples which capture the *sparsity* of precipitation. The model is trained on subregions roughly the size of the lengthscale of the precipitation process. More samples in Section B.9.

Prediction. We first evaluate the ConvLNP's predictive performance, comparing to a GP trained individually on each task as a baseline. In about 10% of tasks, the GP obtains a poor likelihood (< 0 nats); we remove these outliers from the evaluation. The results are shown in Table 4.3. The ConvLNP and GP have comparable RMSEs except on south, where the ConvLNP outperforms the GP. However, the ConvLNP

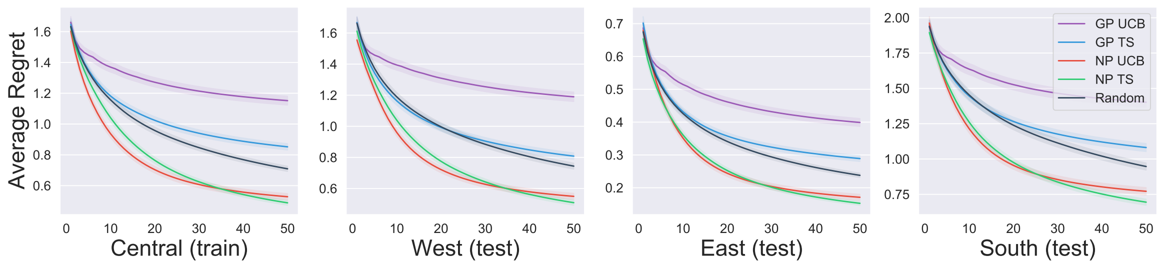


Fig. 4.8 Average regret plotted against number of points queried, averaged over 5000 tasks.

consistently outperforms the GP in log-likelihood, which is expected for the following reasons: (i) the GP does not share information between tasks and hence is prone to overfitting on small context sets, resulting in overconfident predictions; and (ii) the ConvLNP can learn non-Gaussian predictive densities (illustrated in Section B.9). Figure 4.7 shows samples from the predictive process of a ConvLNP and GP, over the whole of the train region. This demonstrates spatial extrapolation, as the ConvLNP is trained only on random subregions.

Bayesian optimization. We demonstrate the ConvLNP in a downstream task by considering a toy Bayesian optimisation problem, where the goal is to identify the location with heaviest rainfall on a given day. We also test the ConvLNP's spatial generalization, by optimising over larger regions (for central, west, and south) than the model was trained on. We test both Thompson sampling (TS) (Thompson, 1933) and upper confidence bounds (UCB) (Auer, 2002) as methods for acquiring points. Note that TS requires coherent samples. The results are shown in Figure 4.8. On all data sets, ConvLNP TS and UCB significantly outperform the random baseline by the 50th iteration; the GP does not reliably outperform random. We hypothesize this is due to its overconfidence, in line with the results on prediction.

4.6 Conclusion and Discussion

In this chapter, we have introduced the ConvLNP, a TE map from observed data sets to predictive SPs. Within the NP framework, ConvLNPs bring together three key considerations.

Expressive joint densities. ConvLNPs extend ConvCNPs to allow for expressive joint predictive densities. A powerful alternative approach is to combine *autoregressive* (AR) models such as PixelCNN++ (Salimans et al., 2017) and the Image Transformer (Parmar et al., 2018)) with CNPs. A difficulty in introducing AR sampling to CNPs is the need to specify a sampling ordering, which is in tension with permutation invariance and relates to the discussion on Bayes-consistency (Section 4.4.2). We return to this idea in Chapter 6.

Translation equivariance. There has been much interest in incorporating equivariance with respect to symmetry groups into neural networks, e.g. Cohen et al. (2019); Cohen and Welling (2016); Kondor (2008); Kondor and Trivedi (2018), with a comprehensive treatment provided by Bloem-Reddy and Teh (2020). ConvLNPs leverage a simple relationship between translation equivariance and stationarity to construct a model particularly well suited to stationary SPs. Similar ideas have been explored for 3D point-cloud modelling (Qi et al., 2017a,b). For example, the models proposed in Wang et al. (2018); Wu et al. (2019) perform convolutions over continuous domains, which are both TE and permutation invariant, achieving excellent performance in point-cloud classification. In contrast with ConvLNPs, point-cloud models (i) are generally used as classification function approximators, rather than meta or few-shot learners; (ii) are typically tailored towards point clouds, making heavy use of specific properties for function design; and (iii) have not considered latent variable or stochastic generalizations.

Neural Process training procedures. One of the key benefits of CNPs is their simple maximum-likelihood training procedure (Garnelo et al., 2018a; Gordon et al., 2020). In contrast, LNPs are usually trained with VI-inspired objectives (Garnelo et al., 2018b), variants of which are empirically investigated in Le et al. (2018). We propose an alternative training procedure that discards VI in favor of a (biased) maximum-likelihood approach that focuses on directly optimizing predictive performance. This approach has two benefits: (i) it does not require carefully designed inference procedures, and works "out-of-the-box" for a range of models; and (ii) empirically, we find that it leads to improved performance for ConvLNPs and, often, for ALNPs.

Despite the improvements in modelling performance, the training of the latent variable ConvLNP is still more difficult than its conditional counterpart, the ConvCNP, and predictions require Monte-Carlo sampling to produce uncertainties. In the next chapter we investigate an approach that maintains the training benefits of CNPs while allowing for modelling correlations in output space and the production of coherent samples.

Chapter 5

Gaussian Neural Processes

In this chapter, we continue our investigation into neural process models able to model joint output dependencies. We introduce a conditional neural process that can take advantage of the much simpler training procedure of the CNPs, avoiding the issues associated with the treatment of the latent variable in LNPs. We introduce the Gaussian neural process which takes inspiration from GPs (Rasmussen and Williams (2006)) to directly parameterize a Gaussian predictive distribution via mean and kernel functions. Our model will be much more scalable than standard GPs, however, as the predictive distribution will be generated via a forward pass through a neural network and kernel function evaluations making them more suitable for large scale applications.

5.1 Introduction

As discussed in Section 2.2.3, CNPs are severely limited by the fact that they do not model dependencies in their outputs. To address this issue, follow-up work introduced latent neural processes models (NPs; Foong et al., 2020; Garnelo et al., 2018b; Kim et al., 2019), such as the ConvLNP, which use latent variables to model output dependencies. However, the likelihood for these models is not analytically tractable, so approximations are required for training (Foong et al., 2020; Le et al., 2018).

My collaborators and I introduced the fully convolutional Gaussian neural process (FullConvGNP) in our publication "The Gaussian Neural Process" presented in the Advances in Approximate Bayesian Inference Symposium, 2020 (Bruinsma et al., 2021)[1]. The FullConvGNP directly parametrizes the covariance of a Gaussian predictive over the output variables. In this way the FullConvGNP models statistical dependencies in

[1]Material from this publication is not included in this thesis.

the output, and can be trained by an exact maximum-likelihood objective, without requiring approximations. However, for D-dimensional data, the architecture of the FullConvGNP involves $2D$-dimensional convolutions, which can be very costly, and, for $D > 1$, poorly supported by most Deep Learning libraries.

The work in this chapter is based on the publication 'Practical Conditional Neural Processes Via Tractable Dependent Predictions' presented in the International Conference on Learning Representations, 2022 (Markou et al., 2021). The research was conducted with my co-lead author Stratis Markou and collaborators Wessel P. Bruinsma, and Anna Vaughan as well as our advisor Richard E. Turner. I contributed and was closely involved with all aspects of the work including the formulation of the model, software development[2], experimentation and writing of the paper. All of of the work in the paper was a collaborative effort by our entire team.

In this chapter, we introduce the Gaussian neural processes (GNPs), a class of model which directly parametrises the covariance of a Gaussian predictive process, thereby circumventing the costly convolutions of the FullConvGNP, and is applicable to higher-dimensional input data. GNPs have analytic likelihoods making them substantially easier to train than their latent variable counterparts. In this chapter:

(i) we show that GNPs can be easily applied to multi-output regression, as well as composed with invertible marginal transformations to model non-Gaussian data;

(ii) we demonstrate that modelling correlations improves performance on experiments with both Gaussian and non-Gaussian synthetic data, including a downstream estimation task that mean-field models cannot solve;

(iii) we demonstrate that GNPs outperform their mean-field and latent variable counterparts on real-world electroencephalogram (EEG) data and climate data;

(iv) in climate modelling, GNPs outperform a standard ensemble of widely used methods in statistical downscaling, while providing spatially coherent temperature samples which are necessary for climate impact studies.

5.2 Gaussian Neural Processes

A central problem with CNP predictive distributions is that they are mean-field: CNPs do not model correlations between $y_m^{(t)}$ and $y_{m'}^{(t)}$ for $m \neq m'$. However, many

[2]Source code available at https://github.com/requeima/kernelcnp

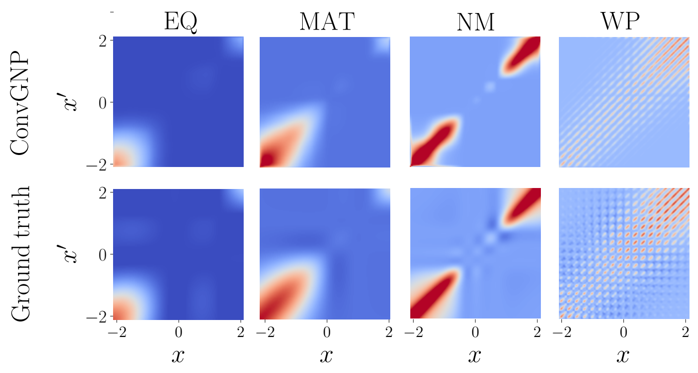


Fig. 5.1 The ConvGNP model, introduced in this work, can recover intricate predictive covariances. Columns show the posterior covariances produced by a ConvGNP, after training with synthetic data drawn from a Gaussian Process with a different covariance (exponentiated quadratic, Matern, noisy mixture or weakly periodic), and conditioned on a randomly sampled dataset.

tasks require modelling dependencies in the output variable. To remedy this, we consider parameterising a correlated multivariate Gaussian using the following *Gaussian prediction map* (see Section 2.1.3):

$$\pi_\theta\left(\boldsymbol{y}^{(t)}; \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}\right) = p_\theta\left(\boldsymbol{y}^{(t)} \,|\, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}\right) = \mathcal{N}\left(\boldsymbol{y}^{(t)}; \mathbf{m}, \mathbf{K}\right) \tag{5.1}$$

where, instead of the expressions for the Bayesian GP posterior, we use neural networks to parameterise the mean $\mathbf{m} = m(\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}, \boldsymbol{x}^{(t)})$ and covariance $\mathbf{K} = K(\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}, \boldsymbol{x}^{(t)})$. We refer to this class of models as Gaussian neural processes (GNPs). In an earlier work, we introduced the FullConvGNP (Bruinsma et al., 2021) with promising results. Unfortunately, the FullConvGNP relies on $2D$-dimensional convolutions for parameterising $\mathbf{K}$, applying the sequence of computations

$$(\boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)}) \xrightarrow{①} (\tilde{\mathbf{x}}, \mathbf{h}) \xrightarrow{②} \mathbf{r} = \mathrm{PSD}(\mathrm{CNN}_{2D}(\mathbf{h})) \xrightarrow{③} \mathbf{K}_{ij} = \textstyle\sum_{\ell=1}^{L} \psi(x_i^{(t)}, \tilde{x}_\ell)\; r_\ell\; \psi(\tilde{x}_\ell, x_j^{(c)}) \tag{5.2}$$

where ① maps $(x^{(c)}, y^{(c)})$ to a $2D$-dimensional grid $\mathbf{h}$ at locations $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_L)$, $\tilde{x}_\ell \in \mathbb{R}^{2D}$, using a ConvDeepSet layer (see Section 3.3), ② maps $\mathbf{h}$ to $\mathbf{r}$ through a CNN with $2D$-dimensional convolutions, followed by a PSD map which ensures $\mathbf{r}$ is positive-definite, and ③ aggregates $\mathbf{r}$ using an RBF ψ. The CNN at ② requires expensive $2D$-dimensional convolutions, which are challenging to scale to higher dimensions (see Section 5.4). To overcome this difficulty, we propose parameterising $\mathbf{m}$ and $\mathbf{K}$ by

$$\mathbf{m}_i = f(x_i^{(t)}, \mathbf{r}), \quad \mathbf{K}_{ij} = k(g(x_i^{(t)}, \mathbf{r}), g(x_j^{(t)}, \mathbf{r})) \tag{5.3}$$

where $\mathbf{r} = r(x^{(c)}, y^{(c)})$, f and g are neural networks with outputs in $\mathbb{R}$ and $\mathbb{R}^{D_g}$, and k is an appropriately chosen positive-definite function. Note that, since k models a posterior covariance, it cannot be stationary. The special case where $\mathbf{K}_{ij} = \sigma_i^2 \mathbf{I}_{ij}$ is diagonal corresponds to a mean-field CNP as presented in (Garnelo et al., 2018a) and Section 2.2.3. Equation (5.3) defines a class of GNPs which, unlike the FullConvGNP, do not require costly convolutions. GNPs can be readily trained via the log-likelihood

$$\theta^* = \arg\max_\theta \log p_\theta \left(\boldsymbol{y}^{(t)} \mid \boldsymbol{x}^{(t)}, D^{(c)} \right), \tag{5.4}$$

where θ collects all the parameters of the neural networks f, g, and r. In this work, we consider two methods to parameterise $\mathbf{K}$, which we discuss next.

Linear covariance: The first method we consider is the `linear` covariance

$$\mathbf{K}_{ij} = g(x_i^{(t)}, \mathbf{r})^\top g(x_j^{(t)}, \mathbf{r}) \tag{5.5}$$

which can be seen as a linear-in-the-parameters model with D_g basis functions and a unit Gaussian distribution on their weights. This model meta-learns D_g context-dependent basis functions, which approximate the true distribution of the target, given the context. By Mercer's theorem (Rasmussen and Williams, 2006), up to regularity conditions, every positive-definite function k can be decomposed as

$$k(z, z') = \textstyle\sum_{d=0}^{\infty} \phi_d(z)\phi_d(z') \tag{5.6}$$

where $(\phi_d)_{d=1}^{\infty}$ is a set of orthogonal basis functions. We therefore expect Equation (5.5) to be able to recover arbitrary (sufficiently regular) GP predictives as D_g grows large. Further, the `linear` covariance has the attractive feature that sampling from it scales linearly with the number of target locations. A drawback is that the finite number of basis functions may limit its expressivity.

Kvv covariance: An alternative covariance which sidesteps this issue, is the `kvv` covariance

$$\mathbf{K}_{ij} = k(g(x_i^{(t)}, \mathbf{r}), g(x_j^{(t)}, \mathbf{r}))v(x_i^{(t)}, \mathbf{r})v(x_j^{(t)}, \mathbf{r}), \tag{5.7}$$

where k is the Exponentiated Quadratic (EQ) covariance with unit lengthscale and v is a scalar-output neural network. The v modulate the magnitude of the covariance, which would otherwise not be able to shrink near the context. Unlike `linear`, `kvv` is not limited by a finite number of basis functions, but the cost of drawing samples from it scales cubically in the number of target points.

Multi-output regression: Extending this approach to the multi-output setting where $y_m^{(t)} \in \mathbb{R}^{D_y}$ with $D_y > 1$, can be achieved by learning functions $m_1, \dots, m_{D_y}$ and $g_1, \dots, g_{D_y}$ for each dimension of the output variable. We can represent covariances across different target points and different target vector entries, by passing those features through either the `linear` or the `kvv` covariance

$$\mathbf{K}_{ijab} = g_a(x_i^{(t)}, \mathbf{r})^\top g_b(x_j^{(t)}, \mathbf{r}), \tag{5.8}$$
$$\mathbf{K}_{ijab} = k(g_a(x_i^{(t)}, \mathbf{r}), g_b(x_j^{(t)}, \mathbf{r}))v_a(x_i^{(t)}, \mathbf{r})v_b(x_j^{(t)}, \mathbf{r}), \tag{5.9}$$

where $\mathbf{K}_{ijab}$ denotes the covariance between entry a of $y_i^{(t)}$ and entry b of $y_j^{(t)}$.

Neural architectures: This discussion leaves room for choosing f, g, and r, producing different models belonging to the *GNP family*, of which the FullConvGNP is also a member. For example, we may choose these to be DeepSets, attentive Deepsets or CNNs, giving rise to Gaussian neural processes (GNPs), attentive GNPs (AGNPs) or convolutional GNPs (ConvGNPs) respectively. Particularly, in the ConvGNP, the feature function g takes the form

$$(x^{(c)}, y^{(c)}) \xrightarrow{①} (\tilde{\mathbf{x}}, \mathbf{h}) \xrightarrow{②} \mathbf{r} = \mathrm{CNN}_D(\mathbf{h}) \xrightarrow{③} g(x_i^{(t)}, \mathbf{r}) = \textstyle\sum_{\ell=1}^{L} \psi(x_i^{(t)}, x_{r,\ell})\, r_\ell, \tag{5.10}$$

where, crucially, $\mathbf{h}$ are values on a D-dimensional grid at $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_L)$, $\tilde{x}_\ell \in \mathbb{R}^D$, and ② uses D-dimensional rather than a $2D$-dimensional CNN. This renders the ConvGNP much cheaper than the FullConvGNP in both compute and memory, while retaining translation equivariance (see Section C.1.1 for proof), making the former a scalable alternative to the latter.

5.3 Non-Gaussian prediction maps

For many tasks joint-Gaussian models are sufficient. However, many tasks require non-Gaussian marginal distributions instead, for example because of non-negative or heavy-tailed variables.

Gaussian Copula Neural Processes: To address this issue, we draw inspiration from copula models (Elidan, 2013). Copulae use a dependent base distribution to model correlations, and adjust its marginals using invertible transformations to better approximate the data at hand. Wilson and Ghahramani (2010) and Jaimungal and Ng (2009) have extended copulae to the stochastic process setting, by using a GP as a base measure, and transforming its marginals appropriately. Adapting the approach of

Jaimungal and Ng, we consider the following transformation to the output of a GNP

$$\boldsymbol{y}^{(t)} = \Phi_M^{-1}(\mathbf{u}^{(t)}, \boldsymbol{\psi}), \text{ where } \mathbf{u}^{(t)} = \Phi_G(\mathbf{v}^{(t)}) \text{ and } p(\mathbf{v}^{(t)}) = \pi_G\left(\mathbf{v}^{(t)}; \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(c)}, \boldsymbol{x}^{(c)}\right). \tag{5.11}$$

where Φ_G is the CDF of the standard Gaussian, Φ_M^{-1} is the inverse CDF of a chosen distribution with parameters $\boldsymbol{\psi} = \psi(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)}, \boldsymbol{y}^{(c)})$, and π_G is a Gaussian prediction map. We refer to models of this form as Gaussian copula neural processes (GCNPs). The log-likelihood of this model can be computed exactly using the change of variables formula (Kobyzev et al., 2020). Since the transformation in Equation (5.11) is dimension-wise, the resulting Jacobian is diagonal, and the log-likelihood takes the form

$$\log \pi\left(\boldsymbol{y}^{(t)}; \mathbf{x}^{(c)}, \mathbf{y}^{(c)}, \mathbf{x}^{(t)}\right) = \log \pi_G(\mathbf{v}^{(t)}; \mathbf{x}^{(c)}, \mathbf{y}^{(c)}, \mathbf{x}^{(t)}) + \textstyle\sum_{m=1}^{M} \log\left|\Theta'(\mathbf{y}_m^{(t)})\right|, \tag{5.12}$$

where $\Theta(\cdot) = \Phi_G^{-1}(\Phi_M(\cdot, \boldsymbol{\psi})), \mathbf{v}^{(t)} = \Theta(\mathbf{y}^{(t)})$. Generally, Θ can be any arbitrary invertible map such as a Normalising Flow (NF). However, requiring marginalisation consistency places limitations on the form of Θ. While point-wise NFs are consistent under marginalisations, it is unclear if non-marginal NFs can be used in a consistent model (Section C.1.2). Last, while fully learnable marginal NFs (Durkan et al., 2019) can be used, the Θ presented here was sufficient for our experiments.

5.4 Computation time and memory comparison

In this section we provide quantitative measurements for the computational and memory costs of the models. Table 5.1 shows the runtime and memory footprint of the GNP, AGNP, ConvGNP and FullConvGNP models applied to data with one-dimensional inputs, $D = 1$. In particular, we measure the runtime and memory required to perform a single forward pass through the neural architecture of each model that was used for the one-dimensional Gaussian tasks, on an NVIDIA GeForce RTX 2080 Ti GPU.

The ConvGNP is cheaper than the FullConvGNP: In Table 5.1 we see that the FullConvGNP model requires a factor of two more runtime and a factor of five larger memory footprint for this example. It should be noted that the performance difference between the ConvGNP and FullConvGNP is less pronounced in the one-dimensional setting, compared to the higher-dimensional settings for two reasons. First, both the runtime and memory footprint of the FullConvGNP increase much quicker than those of the ConvGNP, due to the fact that the FullConvGNP uses $2D$-dimensional convolutions. Second, the parallelisation of operations in the GPU may wash out some of the differences in runtime in favour of the FullConvGNP. More specifically, in cases where some of the GPU workers are idle, i.e. the GPU is not at its parallelisation limit, additional computations can be carried out at a small overhead by using these idle workers. Some of the additional computations required by the FullConvGNP come at a reduced runtime cost to what we would expect if the parallelisation limit of the GPU was reached. Since we expect this limit to be reached for larger D, we also expect the performance difference to be more pronounced for higher dimensions, for this reason.

	GNP	AGNP	ConvGNP	FullConvGNP
Runtime ($\times 10^{-3}$ sec)	0.58 ± 0.00	1.62 ± 0.00	1.74 ± 0.00	4.59 ± 0.01
Memory (MB)	0.476	2.971	0.231	1.23

Table 5.1 Computational and memory costs for one-dimensional models, during a forward using a batch size of one, that is, a single task was passed to each model.

Forward pass through a CNN: To further highlight the how the runtime and memory costs scale with the convolution dimension D_c, we measured the runtime and memory footprint of a CNN with $D_c = 1, 2$ and 3. In particular we used a depth-wise separable CNN (Chollet, 2017) similar to that used in Bruinsma et al. (2021), consisting of twelve hidden convolutional layers in $D_c = 1, 2$ or 3 dimensions, each with a kernel size of 5, measuring the runtime and memory cost of a forward pass through the network. Thus for each dimension, the CNN is applied to a tensor with D_c dimensions, each with a size of $N = 128$, plus an additional channel dimension which we set to 2. Figure 5.2 shows that the runtime and memory required by the CNN increase exponentially with D_c. Extrapolating to $D_c = 4$, we observe that the runtime and memory costs become extremely large, making the FullConvGNP difficult to apply, even to data in $D = 2$ dimensions, since the FullConvGNP requires $D_c = 2D$ dimensional convolutions. For $D = 3$ the FullConvGNP would require $D_c = 6$ dimensional convolutions, which is would be significantly above the runtime and memory we can afford with existing GPUs.

5.5 Experiments and Results

We apply the proposed models to synthetic and real data. Our experiments with synthetic data comprise four Gaussian tasks and a non-Gaussian task. In our experiments with real data, we evaluate our models on electroencephalogram data as well as climate data. We train our models and the FullConvGNP, whenever applicable, using the maximum-likelihood objective in Equation (5.4). We also train the ALNP and

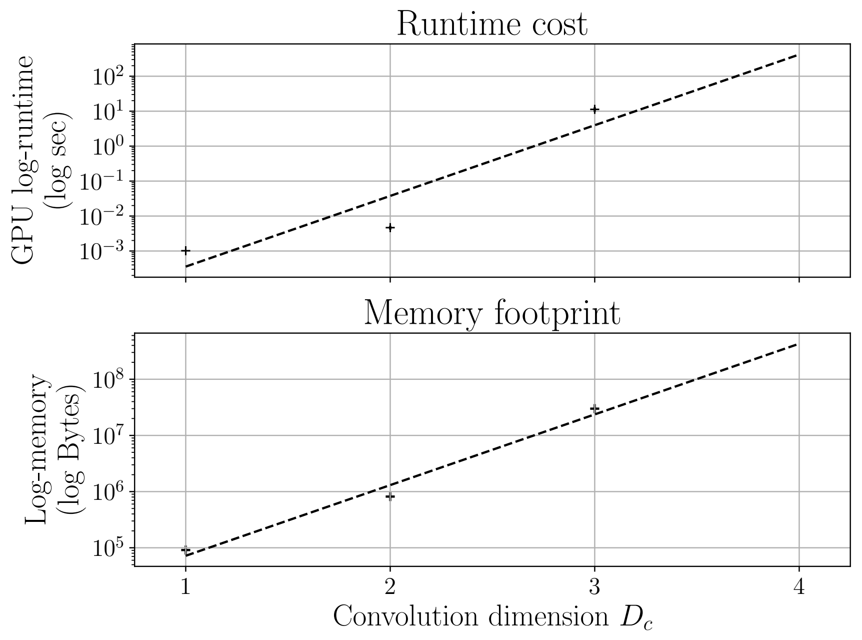


Fig. 5.2 Scaling of the runtime cost and memory footprint of a CNN as a function of the convolution dimension. See text for discussion. *Error bars have been included for the runtime, but are too small to be seen in this plot.*

ConvLNP models. These are latent variable models which place a distribution q over $\mathbf{r}$ and rely on q for modelling output dependencies. As in Chapter 4 we train the ALNP and ConvLNP via a biased Monte Carlo estimate of the objective

$$\theta^* = \arg\max_\theta \log \left[\mathbb{E}_{\mathbf{r}\sim q(\mathbf{r})}\left[p\left(\mathbf{y}^{(t)}; \mathbf{x}^{(c)}, \mathbf{y}^{(c)}, \mathbf{x}^{(t)}, \mathbf{r}\right)\right]\right]. \tag{5.13}$$

5.5.1 Gaussian synthetic experiments

We apply the above models to synthetic datasets generated from GPs with four different covariance functions. Note that in our experiments AGNP has a deterministic path only. In these experiments we have access to the ground truth predictive posterior, which we can use to assess the performance of our models. For each task, we generate multiple datasets using a fixed covariance function, and sub-sample these datasets into context and target sets, to which we fit the models (Figure 5.3). We consider tasks with one and two-dimensional inputs, the latter being a problem where the FullConvGNP cannot be feasibly applied to. Figures 5.4 and 5.5 compare the predictive log-likelihood of the models on these tasks, from which we observe the following trends.

Dependencies improve performance: We expected that modelling dependencies would allow the models to achieve better log-likelihoods. Indeed, for a fixed architecture,

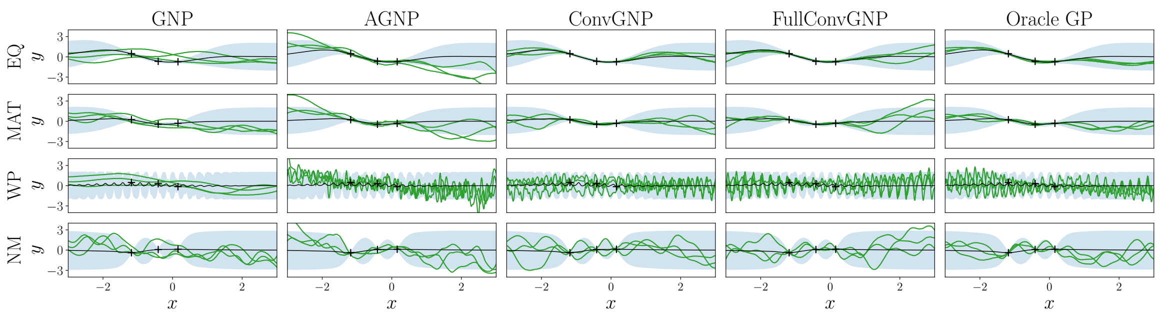


Fig. 5.3 Samples drawn from the models' predictive posteriors (green) compared to the ground truth marginals (blue), using the `kvv` covariance.

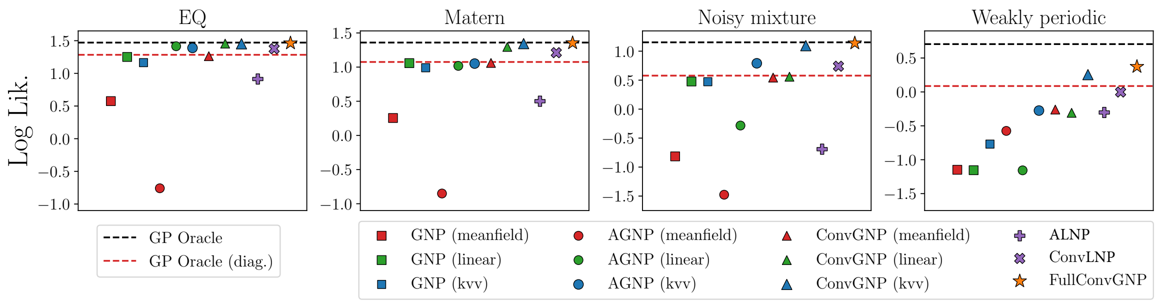


Fig. 5.4 Predictive log-likelihoods across datasets for the 1D Gaussian tasks. The oracle GP performance is shown in dashed black. The dashed red line marks the performance of the diagonal GP oracle, where the off-diagonal covariance terms are 0. *Error bars too small to be seen in the plots.*

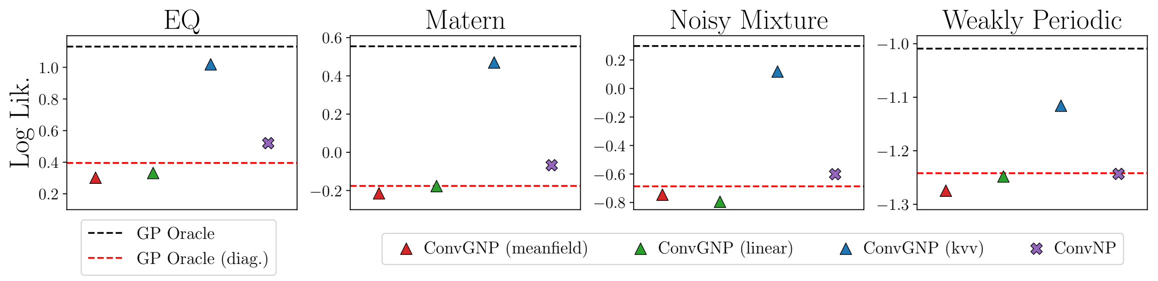


Fig. 5.5 Predictive log-likelihood performance of the models, across datasets for the 2D Gaussian experiments, where the FullConvGNP is not applicable. *Error bars too small to be seen in the plots.*

the correlated GNPs (🟩, 🟦, 🟢, 🔵, ▲, ▲) typically outperform their mean-field counterparts (🟥, 🔴, ▲). This suggests that our models can learn meaningful dependencies in practice, in some cases recovering oracle performance.

Correlated ConvGNPs compete with the FullConvGNP: The correlated ConvGNPs (▲, ▲) are often competitive with the FullConvGNP (⭐). The `kvv` ConvGNP

(▲) is the only model, from those examined here, which competes with the FullConvGNP in all tasks. Unlike the latter, however, the former is scalable to higher input dimensions, and remains the best performing model in the two-dimensional tasks (Figure 5.5). For further details on the runtime and memory costs, see Section 5.4.

Correlated GNPs outperform latent variable models: Correlated GNPs typically outperform the latent-variable ALNP (✚) and ConvLNP (✖) models, which could be explained by the fact that the GNPs have a Gaussian predictive while ALNP and ConvLNP do not, and all tasks are Gaussian. Despite experimenting with different architectures, and even allowing for many more parameters in the ALNP and ConvLNP compared to the AGNP (●, ●) and ConvGNP (▲, ▲), we found it difficult to make the latent variable models competitive with correlated GNPs. We typically found the GNP family significantly easier to train than the latent variable models.

`Kvv` outperformed `linear`: We generally observed that the `kvv` models (■, ●, ▲) performed as well, and occasionally better than, their `linear` counterparts (■, ●, ▲). To test whether the `linear` models were limited by the number of basis functions D_g, we experimented with various settings $D_g \in \{16, 128, 512, 2048\}$. We did not observe a performance improvement for large D_g, suggesting that the models are not limited by this factor. This is surprising because, as $D_g \to \infty$ and assuming flexible enough f, g, and r, the `linear` models should, by Mercer's theorem, be able to recover any (sufficiently regular) GP posterior. We leave open the possibility that the `linear` models might be more difficult to optimise and thus struggle to compete with `kvv`.

Predictive samples: Figure 5.3 shows samples drawn from the predictives, from which we qualitatively observe that, like the FullConvGNP, the ConvGNP produces good quality function samples, which capture the behaviour of the underlying process. The ConvGNP is the only conditional model (other than the FullConvGNP) which produces high-quality posterior samples. The predictive log-likelihood (Figure 5.4) can be used as an objective measure for sample quality.

5.5.2 Predator-Prey experiments

To assess performance on a non-Gaussian synthetic task, we generate data from a Lotka–Volterra predator-prey model (Arnold, 1992). This model describes the evolution of the populations of a predator and a prey species, which are related via a stochastic non-linear difference equation. These time series are non-Gaussian, since neither of the populations can fall below zero. To encode this prior knowledge, we use marginal transformations, enforcing non-negativity in the output variable.

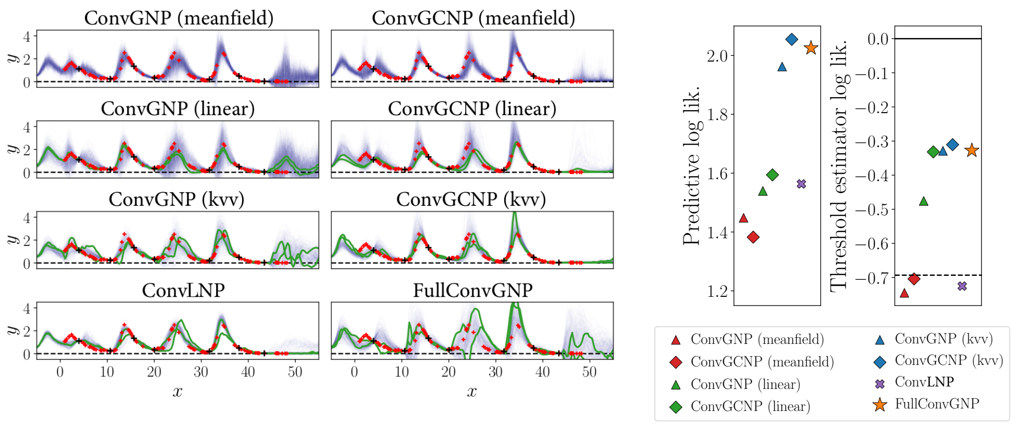


Fig. 5.6 The predator modelling task. Model fits (left) where black and red crosses show the context and target sets of a dataset, the blue regions show the marginals, and the green lines are samples from the predictive. The dashed line marks $y = 0$. Predictive log likelihoods and threshold estimator log likelihoods for the tested models (right). The dashed line indicates random prediction. *Error bars for per-datapoint predictive log-likelihoods and threshold estimation task log-likelihoods too small to be seen in the plots.*

We generate synthetic data following the method in Chapter 3 (see Section C.3). We train and evaluate the ConvGNP, ConvLNP and FullConvGNP models. We also train ConvGCNPs with an exponential CDF transformation $\Phi_M(u) = 1 - e^{-u/\psi}, \psi = \psi(\mathbf{x}^{(c)}, \mathbf{y}^{(c)}, \mathbf{x}^{(t)})$. The marginals of these ConvGCNPs explicitly encode the prior knowledge that the populations are non-negative quantities. Figure 5.6 shows the model fits on a fixed predator time series. It also shows the predictive log-likelihood on test data and the log-likelihood on a downstream estimation task, explained below.

Dependencies improve performance: Similarly to the Gaussian tasks, we observe that modelling output dependencies (▲, ◆, ▲, ◆) improves the predictive log-likelihood over mean field models (▲, ◆). Further, the `kvv` covariance (▲, ◆) performs better than the `linear` covariance (▲, ◆).

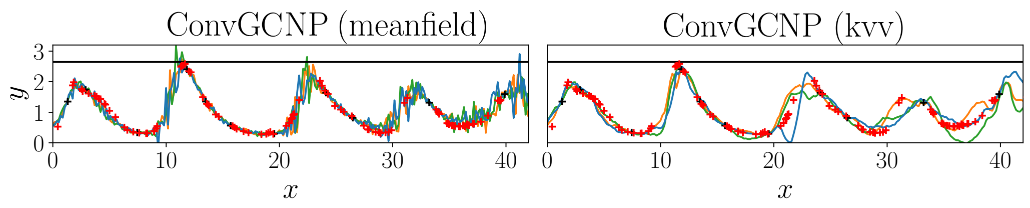


Fig. 5.7 Illustration of the failure mode of mean-field models in the threshold estimation task. The context and target are shown in black and red crosses, and the black line shows the threshold for this context set. In each plot, three samples from the predictive are shown in orange, green and blue.

Marginal transformations: We observe that exponential marginal transformations improve performance in the correlated models, as the ConvGCNP models (◆, ◆) typically exhibit better performance than their ConvGNP counterparts (▲, ▲). Further, the ConvGCNPs produce non-negative posterior samples, which are arguably more plausible than the samples produced the other models.

Comparison with the ConvLNP and FullConvGNP: The ConvGNP with a `kvv` covariance (▲) is competitive with the FullConvGNP (⬟), while adding marginal transformations can further improve the model's performance (◆). Both models outperform the ConvLNP (✕) by a considerable margin.

Correlated GNPs and downstream estimation: We assessed the performance of the models on a downstream estimation task, illustrated in Figure 5.7. Given a context set, we use the trained models to estimate the probability that the population will exceed the maximum population observed in the context, by a factor of at least 1.1. Mean-field models make independent predictions, and thus the proportion of function samples which exceed the threshold is unreasonably large. By contrast, a correlated model avoids this failure mode as their function samples are coherent. Figure 5.6 shows that correlated models (▲, ◆, ▲, ◆, ⬟) are substantially more accurate than mean-field models (▲, ◆) in this threshold estimation task, with the latter being marginally worse than a random prediction.

5.5.3 Electroencephalogram experiments

We applied the proposed and competing models on a real electroencephalogram (EEG) dataset. This dataset comprises of 7632 multivariate time series, collected by placing electrodes on different subjects' scalps (Zhang et al., 1995). Each time series exhibits correlations across channels, as the levels of activity at different regions of the subjects' brain are correlated. It is plausible that there also exist patterns which are shared between different time series from a single subject, as well as across subjects, making this an ideal task for a multi-output meta-learning model such as the ConvGNP. We showcase the ability of the ConvGNP to perform correlated multi-output predictions and handle missing data. We train the models on time series with 7 channels, where we occlude randomly chosen windows of 3 of the channels, using the occluded values as targets. Table 5.2 shows the models' performance on held-out test data, and Figure 5.8 shows an example fit of a ConvGNP. To demonstrate the benefits of meta-learning, the table includes a multi-output GP baseline (MOGP, Bruinsma et al., 2020), which is trained from scratch for every task, without a meta-learning component (see Section C.4). Observe that this baseline is significantly outperformed by all meta-models.

Correlated models improve predictive log-likelihood: We observe that the correlated ConvGNPs show a considerable improvement in terms of log-likelihood, compared to the mean-field ConvGNP as well as the ConvLNP. This improvement in log-likelihood suggests that the correlated ConvGNPs learn to represent meaningful correlations in the target outputs.

	ConvGNP			ConvLNP	MOGP
	mean-field	`linear`	`kvv`		
Log Lik.	$-5.27{\pm}0.01$	$\mathbf{-1.39{\pm}0.01}$	$\mathbf{-1.24{\pm}0.00}$	$-3.96{\pm}0.01$	$-12.7{\pm}0.42$

Table 5.2 Per-datapoint log-likelihood on the held-out EEG test set.

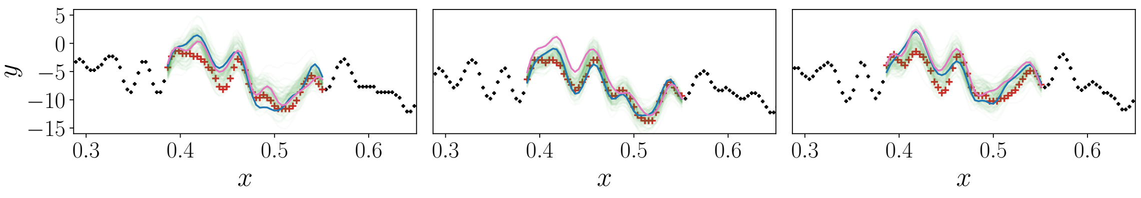


Fig. 5.8 Fit of the ConvGNP (`kvv`) on the EEG data. Each pane shows one of the three channels with unobserved targets (red crosses). All other data (black crosses), including the remaining four channels are observed. Marginals are shown in green, and two samples are shown in blue and pink.

ConvGNPs can model multivariate and partially observed data: Figure 5.8 shows that the `kvv` ConvGNP is able to infer unobserved values from the EEG data of a held-out test example. The model produces both calibrated marginals as well as plausible function samples, and represents not only temporal, but also cross-channel correlations.

5.5.4 Temperature downscaling for environmental modelling

Lastly, we apply the models on a environmental modelling task. In climate modelling, future projections are obtained by simulating the atmospheric equations of motion on a spatio-temporal grid. Unfortunately, computational constraints typically limit spatial resolution to around 100-200km (Eyring et al., 2016), which is insufficient to resolve extreme events and produce local projections (Allen et al., 2016; Maraun et al., 2017). To address this issue, so-called *statistical downscaling* is routinely applied. A mapping from low-resolution simulations to historical data is learnt, and is then applied to future simulations to produce high-resolution predictions (Maraun and Widmann, 2018a).

While numerous data-driven approaches to statistical downscaling exist, they are often limited to making predictions on a fixed set of points at which historical observations are available (Baño-Medina et al., 2020; Bhardwaj et al., 2018; Liu et al., 2020; Misra et al., 2018; Pan et al., 2019; Sachindra et al., 2018; Vandal et al., 2018, 2019, 2017; White et al., 2019). Recently, Vaughan et al. (2022) have applied ConvCNPs to temperature downscaling, enabling predictions at arbitrary locations. Though the ConvCNP outperforms an ensemble of existing methods, it is unable to generate coherent samples, limiting its practical applicability.

Experimental setup: We modify the model of Vaughan et al. (2022), which maps a low-resolution grid of reanalysis data (Dee et al., 2011) together with local orographic information, to a set of features used to parameterise a Gaussian predictive. We use reanalysis and historical station data (Dee et al., 2011) throughout Europe, to set up three experiments, where we train on data from the years 1979-2002, and make predictions on a held out test set from the years 2003-2008, emulating realistic prediction tasks using future climate simulations. The first experiment uses a training set of 86 specific stations in Europe, following a standardised experimental protocol known as the VALUE framework (Maraun et al., 2015), for which extensive baselines are available. Note that we train and test the models on the same stations, but at different time periods. The second is a larger experiment consisting of a training set of 3043 stations across Europe and the 86 VALUE stations as a held out test set. The third experiment uses 713 training and 250 held out test stations, all located in Germany, which has densest station coverage in Europe. Locations of training and test stations for each experiment are shown in Figure 5.9.

For further details on our setup, see Section C.5.

This experiment highlights the benefits of correlated models, since the stations are near one another and are thus highly correlated.

Correlations improve performance: Across all three experiments, modelling correlations improves the predictive log-likelihood (Table 5.3). Since small differences are observed in the MAE, we conclude that the correlated models have learnt to model meaningful statistical output dependencies. The log-likelihood improvement is greatest for the Germany experiment, where the stations are near one another and thus more strongly correlated. In the VALUE experiment, the correlated ConvGNPs improve on

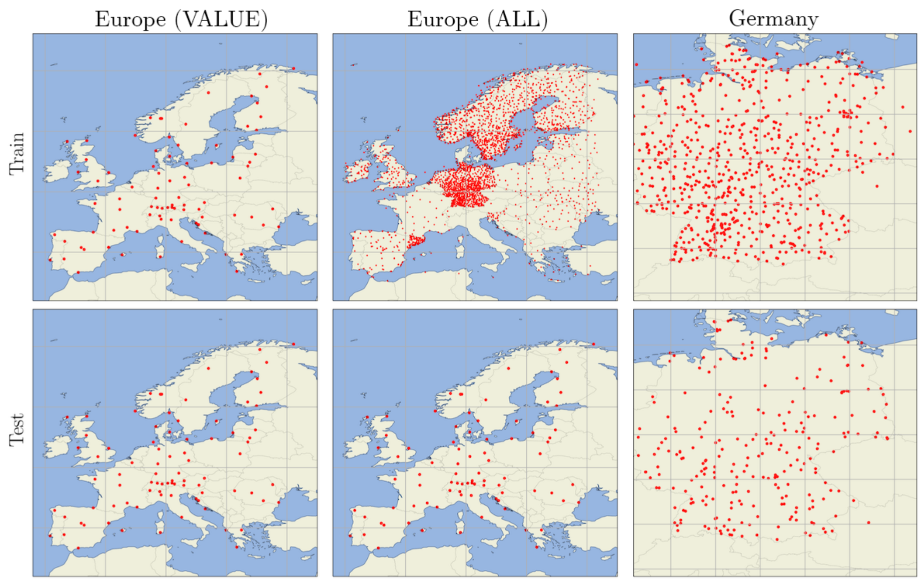


Fig. 5.9 Locations of the training (top) and test (bottom) target locations for Europe (VALUE), Europe (all) and Germany.

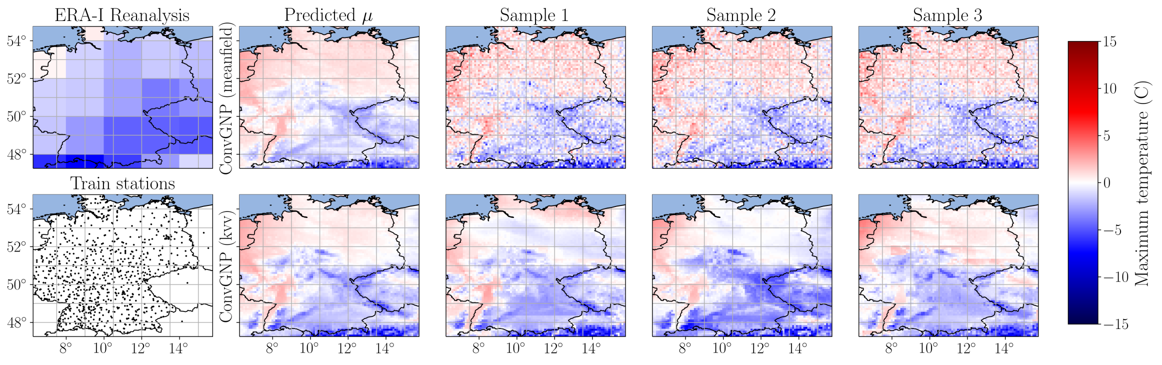


Fig. 5.10 Illustration of sampled temperature fields. After training on low-res. simulations (top left) and observed data (bottom left), the models are conditioned on future low-res. simulations, to make predictions. The remaining columns show the predicted mean and three samples from the predictive.

the ConvLNP, as well as on the mean-field ConvGNP, which has itself been shown (Vaughan et al., 2022) to outperform an ensemble of widely used statistical downscaling methods.

Sampling coherent temperature fields: Correlated ConvGNP models can be used to sample coherent temperature fields, something that mean-field models cannot perform. Figure 5.10 shows that the sampled fields are not only coherent, but also exhibit non-trivial correlation structure.

	Europe (VALUE)	Europe (all)	Germany
ConvGNP (mean-field)	−148.12 / 1.06	−176.93 / 1.41	−263.90 / 0.94
ConvGNP (linear)	−131.58 / 1.03	**−153.23** / 1.33	−224.74 / **0.92**
ConvGNP (kvv)	**−131.47** / **1.02**	−157.25 / **1.30**	**−209.06** / 1.00
ConvLNP	−144.14 / 1.08	−163.11 / 1.38	−252.94 / 1.13
VALUE baseline (MAE)	1.32		

Table 5.3 Predictive log-likelihood and MAE for the three temperature prediction experiments.

5.6 Conclusion and Discussion

In this chapter we presented the Gaussian neural process, a tractable method for modelling statistical dependencies in the outputs of a CNP. GNPs parameterise the covariance of a predictive Gaussian by passing context-dependent feature vectors through positive definite covariance functions. The resulting models, account for output correlations, but unlike previous methods, they can be applied to data with higher-dimensional inputs while maintaining an analytically tractable log-likelihood, which makes them especially easy to train. GNPs can be extended to multi-output regression, and also combined with invertible marginal transformations to model non-Gaussian data.

Similar approaches which model output correlations include the FullConvGNP model of Bruinsma et al. (2021) which, however, cannot be feasibly scaled beyond one-dimensional input data, due to its use of $2D$-dimensional convolutions. Rudner et al. (2018) noted that an NP with an affine decoder and Gaussian conditional prior defines a GP predictive similar to our `linear` covariance. The GNP covariances we presented bear similarities to deep kernels (DKs; Calandra et al., 2016; Patacchiola et al., 2019; Wilson et al., 2015). DKs also use neural networks in the context of GPs but, unlike GNPs which define a GP *predictive*, DKs define a GP *prior*, which is conditioned on data using Bayes' rule. Thus, the computational cost of DKs at test-time is cubic in the context points, whereas that of GNPs is linear, enabling the latter to scale to larger datasets.

ConvGCNP models can be regarded as meta-learning versions of the copula processes of Wilson and Ghahramani (2010) and Jaimungal and Ng (2009). These transform the marginals of a GP, defining a non-Gaussian prior process, and perform inference over it, which requires compute which scales cubically with the size of the context. By contrast, ConvGCNPs directly parameterise a predictive, making their test-time complexity linear in the context.

In this chapter, we saw that modelling correlations improves predictive performance over mean-field models on Gaussian and non-Gaussian synthetic data, including a downstream estimation task that mean-field models cannot solve. GNPs also show improved performance over their mean-field and latent counterparts on real EEG and climate tasks. In statistical temperature downscaling, our models outperform a standard ensemble of widely used methods in statistical downscaling, while providing spatially coherent temperature samples. This renders our models suitable for application to climate impact studies.

GNPs are successful in modelling output space correlations by modifying the CNP architecture to parameterize a full covariance Gaussian. In the next chapter, we explore the idea of using unmodified CNP architectures trained using standard CNP training to produce mean-field predictions but evaluated autoregressively to also produce correlated predictions.

Chapter 6

Autoregressive Neural Processes

Much of this thesis has been concerned with developing novel neural process architectures, whether conditional or latent, to model dependencies in their output – a shortcoming of the original CNP. The following chapter takes a different approach, utilizing existing models and architectures, but modifying their application at test time. Taking advantage of the product rule for joint probability distributions, neural process models, even ones trained to produce mean field predictions, can be applied autoregressively to achieve a predictive distribution modelling dependencies between output variables. The following chapter investigates autoregressive neural process models and compares them with existing approaches discussed in previous chapters.

6.1 Introduction

In the previous chapters, we discussed how CNPs are a family of meta-learning models which combine the flexibility of deep learning with the uncertainty awareness of probabilistic models. They are trained to produce well-calibrated predictions via a simple maximum-likelihood procedure, and naturally handle off-the-grid and missing data, making them ideally suited for tasks in climate science and healthcare. However, initial CNP architectures were not able to model statistical dependencies (Figure 6.1; left) harming their predictive performance and making it impossible to draw coherent function samples necessary for downstream estimation tasks (Markou et al., 2022). LNPs use a latent variable to induce dependencies and model non-Gaussianity but this approach comes with its own set of challenges including an intractable likelihood necessitating approximate inference.

As discussed in Chapter 5, the fully convolutional Gaussian neural process (FullConvGNP) maintains tractability at the cost of only allowing Gaussian predictions.

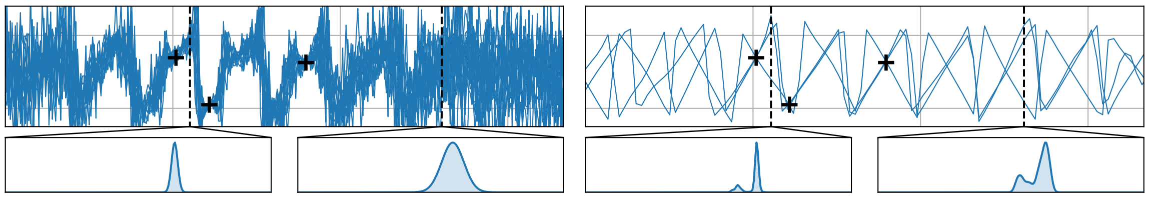


Fig. 6.1 A ConvCNP trained on random sawtooth functions and applied in standard mode (*left*) and in our proposed autoregressive (AR) mode (*right*). The black crosses denote observed data points, the blue lines show model samples, and the bottom plots show the marginal predictive distributions at the locations marked by the dashed vertical lines. In standard mode, the CNP models each output with an independent Gaussian (*left*). However, the *same* CNP, when run in AR mode, can produce coherent samples and model multimodality (*right*).

It uses a neural network to define the mean and covariance function of a predictive Gaussian process (GP) which models dependencies. However, it uses a much more complex architecture and is only practically applicable to problems with one-dimensional inputs, limiting its adoption compared to the more lightweight CNP. Also discussed in Chapter 5, the Gaussian neural process (GNP) is considerably simpler but sacrifices performance relative to the FullConvGNP.

In this chapter we examine a much simpler method for modelling dependencies with neural processes that has been largely overlooked: autoregressive (AR) sampling. AR sampling requires *no changes* to the architecture *or* training procedure. Instead, we *change how the CNP is deployed at test time*, extracting predictive information that would ordinarily be ignored. Instead of making predictions at all target points simultaneously, we autoregressively feed samples back into the model. AR CNPs trade the fundamental property of *consistency under marginalisation and permutation*, which is foundational to many neural process models, for non-Gaussian and correlated predictions. In Table 6.1 we place AR CNPs within the framework of other neural process models.

The work in this chapter is based on the submission 'Autoregressive Conditional Neural Processes' presented at the International Conference on Learning Representations, 2023. The research was conducted with my co-lead authors Stratis Markou and Wessel P. Bruinsma, and collaborators Andrew Y. K. Foong, Anna Vaughan, Tom Andersson, and Anthony Buonomo as well as our advisors Scott Hosking and Richard E. Turner. I contributed and was closely involved with all aspects of the work including the formulation of the model, software development[1], experimentation and writing of the paper. All of of the work in the paper was a collaborative effort by our entire team.

[1]Source code available at https://github.com/wesselb/neuralprocesses.

Class	Consistent	Dependencies	Non-Gaussian	Exact Training
AR CNPs	✗	✓	✓	✓
CNPs	✓	✗	✓	✓
GNPs	✓	✓	✗	✓
LNPs	✓	✓	✓	✗

Table 6.1 Comparison of various classes of neural processes. Each row shows whether a model class produces consistent predictions, models dependencies, can produce non-Gaussian predictions, and can be trained without approximating the likelihood. For CNPs, even though the presentation by Garnelo et al. (2018a) assumes Gaussian predictions, it is simple to relax this Gaussianity assumption; this is not the case for GNPs.

The key contributions of this chapter are:

- We show that CNPs used in AR mode capture rich, non-Gaussian predictive distributions and produce coherent samples (Figure 6.1). This is remarkable as the CNPs used have Gaussian likelihoods, are not trained to model joint dependencies or non-Gaussianity, and are significantly cheaper to train than LNPs and FullConvGNPs (Figure 6.2).

- We prove that given sufficient data and model capacity, the performance of AR CNPs is at least as good as that of GNPs, which explicitly model correlations in their predictions.

- Viewing AR CNPs as a type of neural density estimator (Uria et al., 2016), we highlight their connections to a range of existing methods in the deep generative modelling literature.

- In an extensive range of Gaussian and non-Gaussian regression tasks, we show that AR CNPs are consistently competitive with, and often significantly outperform, all other neural process models in terms of predictive log-likelihood.

- We deploy AR CNPs on a range of tasks involving real-world climate data. To handle the high-resolution data in a computationally tractable manner, we introduce a novel multi-scale architecture for ConvCNPs. We also combine AR ConvCNPs with a beta-categorical mixture likelihood, producing strong results compared to other neural processes.

AR CNPs represent a promising first application of this procedure to the simplest class of neural processes, and motivates future work on applications of AR sampling to other neural process models.

6.2 Autoregressive Conditional Neural Processes

Let us revisit some of the notation that we will use in this chapter. Let $\mathcal{P}$ be the set of all $\mathcal{Y}$-valued stochastic processes on $\mathcal{X}$. Neural processes (NPs) directly and flexibly parametrise a prediction map $\pi_\theta \colon \mathcal{D} \to \mathcal{Q}$ where $\mathcal{Q} \subseteq \mathcal{P}$ and where θ are learnable parameters. CNPs set $\mathcal{Q}$ to be the collection of GPs f such that $\operatorname{cov}(f(x), f(y)) = 0$ for $x \neq y$. GNPs let $\mathcal{Q}$ be the collection of continuous GPs. Latent NPs (LNPs; Garnelo et al., 2018b) let $\mathcal{Q}$ be a collection of non-Gaussian processes by making use of a latent variable. Let $P_{\boldsymbol{x}_m^{(\mathrm{t})}}\pi(D_m^{(\mathrm{c})})$ denote the finite-dimensional distribution of the process $\pi(D_m^{(\mathrm{c})})$ evaluated at inputs $\boldsymbol{x}_m^{(\mathrm{t})}$, and denote its density by $q_\theta(\,\boldsymbol{\cdot}\,|\, \boldsymbol{x}_m^{(\mathrm{t})}, D_m^{(\mathrm{c})})$. To learn the parameters θ, NPs seek to maximise:

$$\mathcal{L}_M(\pi) = \tfrac{1}{M} \textstyle\sum_{m=1}^M \log q_\theta(\boldsymbol{y}_m^{(\mathrm{t})} \,|\, \boldsymbol{x}_m^{(\mathrm{t})}, D_m^{(\mathrm{c})}). \tag{6.1}$$

For CNPs and GNPs, $\mathcal{L}_M$ can be computed exactly since q_θ is Gaussian.[2]. However, for LNPs, $\mathcal{L}_M$ must be approximated (Foong et al., 2020; Garnelo et al., 2018b), typically impacting performance.

Autoregressive CNPs. Our proposal is to take an existing CNP and run it in an autoregressive fashion, feeding predictions for earlier outputs back into the model. Inspired by the product rule, we define the joint predictive as a product of conditionals, modelling each conditional with a CNP. For example, in the case of three target points, $q_\theta(y_3^{(\mathrm{t})} \,|\, \boldsymbol{y}_{1:2}^{(\mathrm{t})}, D_m^{(\mathrm{c})}) q_\theta(y_2^{(\mathrm{t})} \,|\, y_1^{(\mathrm{t})}, D_m^{(\mathrm{c})}) q_\theta(y_1^{(\mathrm{t})} \,|\, D_m^{(\mathrm{c})})$. To enable a theoretical analysis of this procedure, we now proceed to set up more formal notation. Suppose that $\pi_\theta \colon \mathcal{D} \to \mathcal{Q}$ is an NP, and we wish to predict at some target inputs $\boldsymbol{x}^{(\mathrm{t})}$ given a context set $D^{(\mathrm{c})}$. Standard NPs would output the predictive $P_{\boldsymbol{x}^{(\mathrm{t})}}\pi_\theta(D^{(\mathrm{c})})$ which, for CNPs, would be a factorised Gaussian. We propose to instead roll out the NP autoregressively, as described in Proc. 6.2.1.

Procedure 6.2.1 (Autoregressive application of neural processes)**.** For a neural process π_θ, context set $D^{(\mathrm{c})} = (\boldsymbol{x}^{(\mathrm{c})}, \boldsymbol{y}^{(\mathrm{c})})$, and target inputs $\boldsymbol{x}^{(\mathrm{t})}$, let $\mathrm{AR}_{\boldsymbol{x}^{(\mathrm{t})}}(\pi_\theta, D^{(\mathrm{c})})$ be the distribution defined as follows:

$$\text{for } i = 1, \dots, N, \quad y_i^{(\mathrm{t})} \sim P_{x_i^{(\mathrm{t})}} \pi_\theta(\boldsymbol{x}^{(\mathrm{c})} \oplus \boldsymbol{x}_{1:(i-1)}^{(\mathrm{t})}, \boldsymbol{y}^{(\mathrm{c})} \oplus \boldsymbol{y}_{1:(i-1)}^{(\mathrm{t})}), \tag{6.2}$$

where $\boldsymbol{a} \oplus \boldsymbol{b}$ concatenates two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. See Figure D.1 in Section D.3 for an illustration.

[2]Unless otherwise specified, we assume CNPs use Gaussian likelihoods, as in Garnelo et al. (2018a). However, it is straightforward to modify them to use non-Gaussian likelihoods, as we do in Section 6.4.4.

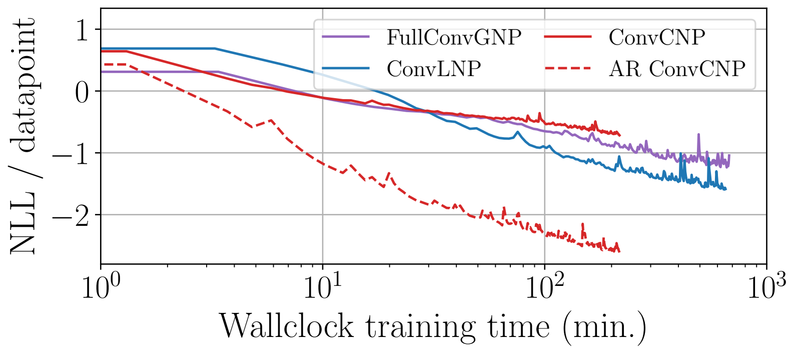


Fig. 6.2 Negative log-likelihoods on non-Gaussian sawtooth data. Deploying the ConvCNP in AR mode dramatically improves performance, and outperforms state-of-the-art NPs with Gaussian (FullConvGNP) and non-Gaussian (ConvLNP) predictive distributions, at a fraction of the training cost.

Since earlier samples $y_i^{(\mathrm{t})}$ feed back into later applications of π_θ, the whole sample $\boldsymbol{y}^{(\mathrm{t})}$ is correlated, *even if* π_θ *does not model dependencies between target outputs*, as with CNPs. At test time, when evaluating the corresponding the density $q_\theta^{(\mathrm{AR})}$ of $\mathrm{AR}_{\boldsymbol{x}^{(\mathrm{t})}}(\pi_\theta, D^{(\mathrm{c})})$ at $\boldsymbol{y}^{(\mathrm{t})}$, we use the formula

$$\log q_\theta^{(\mathrm{AR})}(\boldsymbol{y}^{(\mathrm{t})} \,|\, \boldsymbol{x}^{(\mathrm{t})}, D^{(\mathrm{c})}) = \textstyle\sum_{i=1}^N \log q_\theta(y_i^{(\mathrm{t})} \,|\, x_i^{(\mathrm{t})}, D^{(\mathrm{c})} \oplus (\boldsymbol{x}_{1:(i-1)}^{(\mathrm{t})}, \boldsymbol{y}_{1:(i-1)}^{(\mathrm{t})})). \tag{6.3}$$

Whilst any NP can be used in AR, we focus on CNPs as they are the computationally cheapest class.

Understanding the infinite data limit. To better understand why AR CNPs successfully model dependencies, we analyse the idealised case of infinite data and model capacity. Let $p(f)$ be the data-generating stochastic process, and let $p(\varepsilon)$ be the stochastic process representing observation noise, defined by letting $\varepsilon(\boldsymbol{x})$ be a vector of i.i.d. noise variables for all $\boldsymbol{x}$. We assume

$$\boldsymbol{y}_m^{(\mathrm{c})} = y_m(\boldsymbol{x}_m^{(\mathrm{c})}) \quad \text{and} \quad \boldsymbol{y}_m^{(\mathrm{t})} = y_m(\boldsymbol{x}_m^{(\mathrm{t})}) \quad \text{where} \quad y_m(\,\cdot\,) = f_m(\,\cdot\,) + \varepsilon_m(\,\cdot\,), \tag{6.4}$$

$(f_m)_{m=1}^M$ are i.i.d. draws from $p(f)$, and $(\varepsilon_m)_{m=1}^M$ are i.i.d. draws from $p(\varepsilon)$. Define the *prediction map* $\pi_y \colon \mathcal{D} \to \mathcal{P}$ as the mapping from a data set to the posterior over y, $\pi_y(D) = p(y \,|\, D)$. Then $\mathcal{L}_M$ is a Monte Carlo approximation of the following infinite-sample objective (Foong et al., 2020):

$$\mathcal{L}_\infty(\pi) = -\mathbb{E}_{p(D^{(\mathrm{c})})p(\boldsymbol{x}^{(\mathrm{t})})}[\mathrm{KL}(P_{\boldsymbol{x}^{(\mathrm{t})}}\pi_y(D^{(\mathrm{c})}) || P_{\boldsymbol{x}^{(\mathrm{t})}}\pi(D^{(\mathrm{c})}))] + \text{const.} \tag{6.5}$$

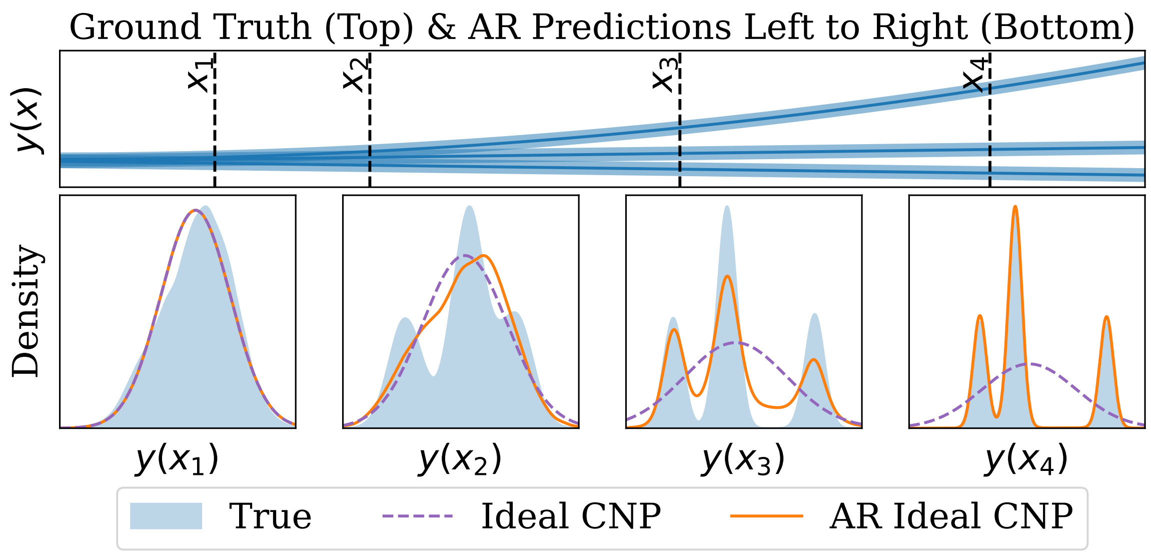


Fig. 6.3 *Top:* generative process: mixture model of three deterministic functions with additive Gaussian noise. *Bottom:* at the four target locations indicated by dashed lines, the panes show the true distribution and predictions by the ideal CNP and the ideal CNP applied autoregressively at the targets from left to right. Notice that the first first AR predictive distribution (left) is a Gaussian and the AR predictive distributions improve as we take more AR samples (left to right). Details in Section D.5.

Under appropriate regularity assumptions, $\mathcal{L}_\infty(\pi)$ is maximised over all π when the expected KL divergence term is zero, which occurs if and only if $\pi = \pi_y$. In practice, NPs do not maximise $\mathcal{L}_\infty(\pi)$ over all π, but (i) use a finite-sized meta–data set and (ii) restrict $\mathcal{Q} \subseteq \mathcal{P}$.

$$\overbrace{\pi_M \in \operatorname*{arg\,max}_{\text{all } \pi\colon \mathcal{D}\to\mathcal{Q}} \mathcal{L}_M(\pi)}^{\text{what we compute in practice}} \xrightarrow[M\to\infty]{} \overbrace{\pi_\infty \in \operatorname*{arg\,max}_{\text{all } \pi\colon \mathcal{D}\to\mathcal{Q}} \mathcal{L}_\infty(\pi)}^{\text{ideal NP}} \xrightarrow[\mathcal{Q}\to\mathcal{P}]{} \overbrace{\pi_y = \operatorname*{arg\,max}_{\text{all } \pi\colon \mathcal{D}\to\mathcal{P}} \mathcal{L}_\infty(\pi)}^{\text{exact prediction map}} \tag{6.6}$$

Here π_M is an NP trained on the practical objective equation 6.1, which, in the limit of infinite data, approximates the so-called *ideal NP* π_∞. The ideal NP depends on the choice of $\mathcal{Q}$, *i.e.* the class of NPs under consideration, and, in turn, approximates π_y. For CNPs and GNPs, using the fact that minimising $\mathrm{KL}(p, q)$ over q matches moments (Minka, 2001), we can readily determine and even practically compute the ideal NP for these two classes of NPs. The *ideal CNP* predicts a diagonal-covariance GP whose mean function and *marginal variance function* match π_y: $\pi_\infty(D) = \mathcal{GP}(m, k)$ where $m(x) = \mathbb{E}[y(x) \mid D]$, and $k(x, x') = \mathbb{V}[y(x) \mid D]$ if $x = x'$ and $k(x, x') = 0$ otherwise. On the other hand, the *ideal GNP* predicts a GP whose mean function and *full covariance function* match π_y: $\pi_\infty(D) = \mathcal{GP}(m, k)$ where $m(x) = \mathbb{E}[y(x) \mid D]$, $k(x, x') = \operatorname{cov}(y(x), y(x') \mid D)$. The main result of this subsection is that the ideal CNP, despite not modelling correlations, becomes superior to the ideal GNP *when deployed in AR mode*:

Proposition 2 (Advantage of AR CNPs over GNPs)**.** Assume appropriate regularity conditions on y. Let π_{C} be the ideal CNP and let π_{G} be the ideal GNP. Then, for all inputs $\boldsymbol{x}$ and data sets $D \in \mathcal{D}$,

$$\mathrm{KL}(P_{\boldsymbol{x}}\pi_y(D), \mathrm{AR}_{\boldsymbol{x}}(\pi_{\mathrm{C}}, D)) \leq \mathrm{KL}(P_{\boldsymbol{x}}\pi_y(D), P_{\boldsymbol{x}}\pi_{\mathrm{G}}(D)). \tag{6.7}$$

We provide a proof in Section D.1. Intuitively, the advantage of AR CNPs comes from their ability to model non-Gaussian dependencies. Proposition 2 shows that to outperform the GNP, it suffices to train a CNP to model the *marginals* of π_y, and rely on the AR procedure to induce dependencies. A visualisation of the ideal CNP and the ideal CNP applied autoregressively can be seen in Figure 6.3.

Consistency and the AR design space. As shown in Table 6.1, AR CNPs give up the fundamental property of *consistency*, as the distributions $\{\mathrm{AR}_{\boldsymbol{x}}(\pi_\theta, D^{(\mathrm{c})}_m) : \boldsymbol{x} \in \mathcal{X}^N,\ N \in \mathbb{N}\}$ are not consistent under permutation or marginalisation: permuting $\boldsymbol{x}$ can change the distribution, as can introducing and marginalising new target points. This violates the conditions of the Kolmogorov extension theorem (Oksendal, 2013), preventing the distributions from defining a consistent stochastic process. There is thus a large design space involved when deploying AR CNPs, where choices that have no effect on the predictions of other NPs can now significantly affect performance.

One such choice is how many points to sample at a time. Sampling one at a time induces dependencies between all points, but requires N forward passes. Alternatively, we could divide the N inputs in $\boldsymbol{x}^{(\mathrm{t})}$ into blocks of K points each, and sample each block with a single CNP forward pass. This requires N/K forward passes, with points in the same block conditionally independent. If $K = N$, this is the standard CNP prediction, and if $K = 1$, we recover Procedure 6.2.1. This provides a knob for practitioners to trade off between faster, consistent, but less expressive standard CNP predictions, and slower, less consistent, but more expressive AR predictions. In this paper, we use full AR mode with $K = 1$, and leave an investigation of block AR sampling to future work.

Obtaining smooth samples. Due to the lack of consistency in AR mode, the spacing chosen between target points can significantly affect performance. For example, care must be taken so the number of target points is not much greater than the size

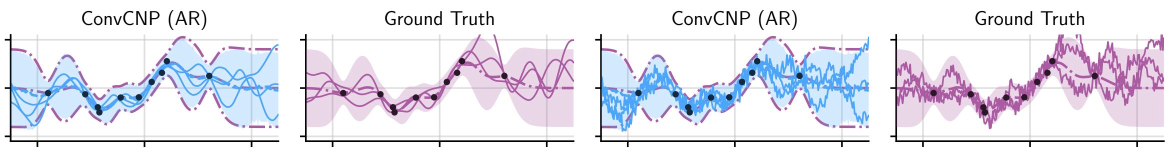


Fig. 6.4 Comparison of noiseless (*left*) and noisy (*right*) samples from an AR ConvCNP trained on data sampled from a GP with an exponentiated-quadratic kernel, and the ground truth GP. The noiseless AR samples were generated from the noisy AR samples using the procedure suggested by Proposition 3.

of the context sets seen during train time, to avoid confronting the model with an out-of-distribution context set size at test time. This raises the question of how to sample functions on a very fine grid. Furthermore, since CNPs do not differentiate between epistemic and aleatoric uncertainty, it is not clear how to obtain smooth, noiseless samples, that is, samples for f uncorrupted by the i.i.d. noise ε in equation 6.4. The following proposition shows that, for a smooth sample corrupted by additive noise, the smooth component can be approximated with the predictive mean conditioned on noisy observations:

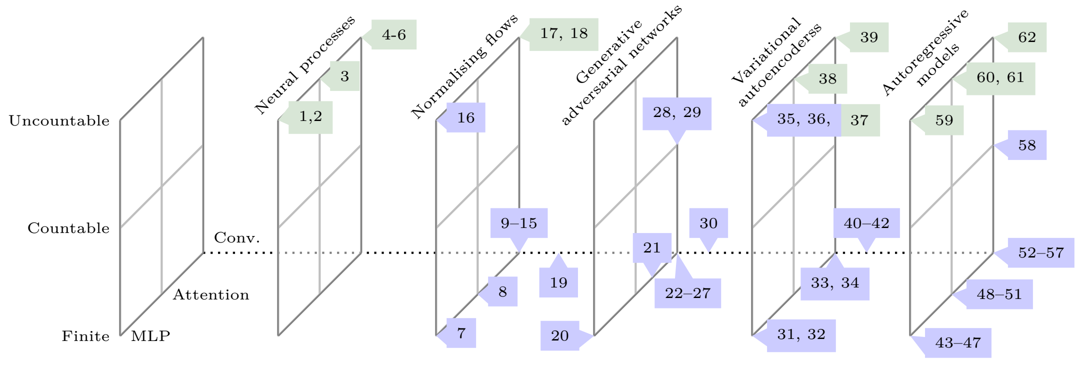
Proposition 3 (Recovery of smooth samples)**.** Let $\mathcal{X} \subseteq \mathbb{R}$ be compact, and let f be a stochastic process with surely continuous sample paths and $\sup_{x\in\mathcal{X}}\|f(x)\|_{L^2} < \infty$. Let $(\varepsilon_n)_{n\geq 0}$ be i.i.d. (potentially non-Gaussian) random variables such that $\mathbb{E}[\varepsilon_0] = 0$ and $\mathbb{V}[\varepsilon_0] < \infty$. Consider any sequence $(x_n)_{n\geq 1} \subseteq \mathcal{X}$, and let $x^* \in \mathcal{X}$ be a limit point of $(x_n)_{n\geq 1}$. If $y(x^*) = f(x^*) + \varepsilon_0$ and $y_n = f(x_n) + \varepsilon_n$ are noisy observations of f, then

$$\lim_{n\to\infty} \mathbb{E}[y(x^*)\,|\,y_1,\dots,y_n] = f(x^*) \quad \text{almost surely.} \tag{6.8}$$

We provide a proof in Section D.2. Equation (6.8) suggests the following two-step procedure for obtaining smooth samples from AR CNPs. **Step 1:** Let $\boldsymbol{x}_{1:n}$ be a number target inputs that does not exceed the number of points seen during training. Sample $\boldsymbol{y}_{1:n} \sim \mathrm{AR}_{\boldsymbol{x}_{1:n}}(\pi_\theta, D^{(\mathrm{c})}_m)$. This sample includes observation noise. **Step 2:** Remove the noise from the sample by passing it through the model once more: $\mathcal{N}(\boldsymbol{\mu}_{1:n}, \mathbf{D}) = P_{\boldsymbol{x}_{1:n}}\pi_\theta(D^{(\mathrm{c})}_m \oplus (\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}))$. Here the predictive mean $\boldsymbol{\mu}_{1:n}$ forms the noiseless sample. To produce a sample at arbitrarily many inputs, one may also evaluate $\mathcal{N}(\boldsymbol{\mu}'_{1:n}, \mathbf{D}) = P_{\boldsymbol{x}'_{1:n}}\pi_\theta(D^{(\mathrm{c})}_m \oplus (\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n}))$ where $\boldsymbol{x}'_{1:n}$ is arbitrary. This result of this procedure is illustrated in Figure 6.4, and was used to generate the noiseless samples shown in Figure 6.1 (right). Figure D.1 in Section D.3 also illustrates this two-step procedure in a pictorial step-by-step fashion.

6.3 Connections to Other Neural Distribution Estimators

Various paradigms have been developed for neural distribution estimators, including normalising flows (Dinh et al., 2014), generative adversarial networks (GANs;



Neural processes

1. CNP (Garnelo et al., 2018a)
2. GNP (Markou et al., 2022)
3. ACNP (Kim et al., 2019)
4. ConvCNP Gordon et al. (2020)
5. ConvGNP (Markou et al., 2022)
6. FullConvGNP (Bruinsma et al., 2021)

Normalising flows

7. NICE (Dinh et al., 2014)
8. Flow++ (Ho et al., 2019)
9. RealNVP (Dinh et al., 2016)
10. FFJORD (Grathwohl et al., 2018)
11. Glow (Kingma and Dhariwal, 2018)
12. i-ResNets (Behrmann et al., 2018)
13. Res. Flows (Chen et al., 2019)
14. IAF (Kingma et al., 2016)
15. NAF (Huang et al., 2018)
16. TGP (Maroñas et al., 2021)
17. BRUNO (Korshunova et al., 2020)
18. Copula GNP (Markou et al., 2022)
19. FlowGAN (Grover et al., 2018)

GANs

20. GAN (Goodfellow et al., 2014)
21. SAGAN (Zhang et al., 2019)
22. DCGAN (Radford et al., 2016)
23. WGAN (Arjovsky et al., 2017)
24. InfoGAN (Chen et al., 2016a)
25. SNGAN (Miyato et al., 2018)
26. BigGAN (Brock et al., 2018)
27. StyleGAN (Karras et al., 2019)
28. Spatial GAN (Jetchev et al., 2016)
29. ∞-GAN (Lu et al., 2020)
30. PresGAN (Dieng et al., 2019)

Variational autoencoders

31. VAEs (Kingma and Welling, 2014)
32. IWAE (Burda et al., 2015)
33. Conv. VAE (Salimans et al., 2015)
34. VDVAE (Child, 2020)
35. GP-VAE (Fortuin et al., 2020)
36. SGP-VAE (Ashman et al., 2020)
37. LNP (Garnelo et al., 2018b)
38. ALNP (Kim et al., 2019)
39. ConvLNP (Foong et al., 2020)
40. VQ-VAE (Oord et al., 2017)
41. PixelVAE (Gulrajani et al., 2016)
42. VLAE (Chen et al., 2016b)

Autoregressive models

43. RNADE (Uria et al., 2013)
44. NADE (Uria et al., 2016)
45. DeepNADE (Uria et al., 2014)
46. EoNADE (Uria et al., 2014)
47. MADE (Germain et al., 2015)
48. PixelSNAIL (Chen et al., 2018)
49. Sparse Transformer (Child et al., 2019)
50. DEformer (Alcorn and Nguyen, 2021)
51. XLNet (Yang et al., 2019)
52. ConvNADE (Uria et al., 2016)
53. WaveNet (Oord et al., 2016a)
54. PixelCNN (Oord et al., 2016b)
55. PixelCNN++ (Salimans et al., 2017)
56. Fast PixelCNN++ (Ramachandran et al., 2017)
57. Scalable Pixel Net (Menick and Kalchbrenner, 2018)
58. ARDMs (Hoogeboom et al., 2021)
59. AR CNP
60. AR ACNP
61. Transformer NP (Nguyen and Grover, 2022)
62. AR ConvCNP

Fig. 6.5 Conceptual diagram showing the relationships between AR CNPs and various neural distribution estimators. The vertical axis denotes whether the model learns a distribution over a finite number of random variables, a countably infinite number, or an uncountably infinite number. The axis into the page denotes whether the architecture is MLP-based, or uses attention or convolutions. From left to right, we show different modelling paradigms. Fruitful exchanges occur when NPs (highlighted in green) are introduced into other modelling paradigms. The proposed AR CNPs can be viewed as introducing NPs to the AR modelling paradigm.

Goodfellow et al., 2014), variational autoencoders (VAEs; Kingma and Welling, 2014), autoregressive models (Uria et al., 2016), and diffusion models (Ho et al., 2020; Sohl-Dickstein et al., 2015). We argue that NPs should be viewed as neural distribution estimators alongside these models, revealing interesting connections between them. Figure 6.5 visualises the landscape of neural distribution estimators, highlighting the dimensionality of the target distribution, and the use of attention or convolution.

Our proposed AR CNPs can be viewed both as a kind of NP, and as an instance of autoregressive models. As such, they inherit the strengths of AR models, such as their ability to model complex dependencies with a tractable likelihood, but also some of their weaknesses, most notably slow test-time sampling. Slow sampling is the main drawback of applying AR CNPs to large target sets, though it may be possible to adapt techniques for speeding up other AR models to AR CNPs (Ramachandran et al., 2017). One major difference between AR CNPs and existing AR models is that AR CNPs model an uncountably infinite set of variables, which allows querying arbitrary input locations but leads to the consistency issues discussed in Section 6.2.

Another distinguishing feature of the AR CNP is that it is trained to be order agnostic with respect to the product rule decomposition of the joint distribution: there is no preferred ordering of the input values during train time. DEformer (Alcorn and Nguyen, 2021), EoNADE (Uria et al., 2014), and XLnet (Yang et al., 2019) are other examples of AR models designed for order agnosticism. The AR CNP shares design choices with other AR models to achieve this goal: (i) a shared architecture is used to produce each conditional distribution, similar to WaveNet (Oord et al., 2016a) and PixelCNN (Oord et al., 2016b); (ii) the datapoint index is given as input to the network as in the DEformer model (Alcorn and Nguyen, 2021); and (iii) training maximises a log-likelihood including all decompositions of the joint distribution, similar to EoNADE (Uria et al., 2014) and XLnet (Yang et al., 2019).

In recent work, Nguyen and Grover (2022) proposed the Transformer NP (TNPs), which uses a causally-masked transformer architecture with an autoregressive likelihood. In contrast, rather than proposing a new AR architecture, our work focuses on running *existing CNPs* in AR mode to obtain coherent samples and improved likelihoods, without modifying the architecture or training procedure. In prior work, Volpp et al. (2021) used AR sampling in order to visualise samples from CNPs. However, their work focuses on proposing a novel context aggregation mechanism for NPs, and they do not evaluate the likelihood of CNPs autoregressively or investigate any performance gains.

Figure 6.5 also shows the connections betweeen NPs and VAEs and normalising flows. Just as VAEs typically use decoders that parametrise a factorised distribution, and rely on the latent variable to induce dependencies, the same is true for LNPs. Again, the key difference with standard VAEs is that LNPs model a distribution over an uncountable number of variables. In addition, models like conditional BRUNO (Korshunova et al., 2020) and copula GNPs (Markou et al., 2022) combine ideas from NPs and normalising flows, by transforming a stochastic process with an invertible transformation. Finally, GAN models such as Spatial GAN (Jetchev et al., 2016) and $\infty-$GAN (Lu et al., 2020) model countable numbers of variables, such as images of arbitrary size. Inspecting Figure 6.5, we see that GANs are the only class of models depicted that do not currently have an NP version, or a version that models an uncountable number of variables. This suggests adversarial training of NPs as an interesting avenue for future investigation.

6.4 Experiments and Results

In this section we investigate the performance of AR CNPs on synthetic and real data. Across a wide range of tasks, the AR CNP is competitive with much more sophisticated approaches. Throughout, we train LNPs with both the ELBO and ML objective (see Sections 4.3.2 and 4.4.1). For all experiments, we use a random ordering of the target points inProcedure 6.2.1; see Section D.4 for a justification.

6.4.1 Synthetically Generated Gaussian and Non-Gaussian Data

Synthetic experiment setup. We evaluate an extensive collection of NP models on a wide range of Gaussian and non-Gaussian synthetic regression tasks. We consider tasks with functions drawn from (i) different GPs; (ii) a non-Gaussian sawtooth process (as in Figure 6.1); (iii) a non-Gaussian mixture task, where, with equal probability, we sample the function from one of three possible GPs or the sawtooth process. We also consider various versions of the tasks for different input and output dimension d_x, d_y, with dependencies across the output channels. To ensure a fair comparison, we choose architectural details to make the parameter counts comparable between all models.

Results. Table 6.2 highlights the best performing models on some representative tasks; for further results across all twenty synthetic tasks and further experimental details, see Section D.8. The AR procedure dramatically improves the performance of the ConvCNP, with the AR ConvCNP being the best performing model for most tasks, except on the Gaussian EQ task where it performs marginally worse than the FullConvGNP. In particular, the AR ConvCNP outperforms the FullConvGNP and ConvGNP on non-Gaussian tasks, in agreement with Proposition 2, while having a faster training time than the other convolutional models (Figure 6.2). For the sawtooth task, Figure D.5 in Section D.8.2 illustrates that predictions by the AR ConvCNP can be multi-modal and non-Gaussian, even when using a Gaussian likelihood. Finally, we note that in tasks with $d_x = 2$, where the FullConvGNP cannot be used (as discussed in Section 6.1), the AR ConvCNP far outperforms all competing approaches.

6.4.2 Sim-to-Real Transfer with the Lotka–Volterra Equations

Predator-prey data. We next investigate sim-to-real transfer, where the models are trained on simulated data and tested on real data. NPs are well-suited to this setting, since a large meta-data set can be easily generated to train them. We consider the Hudson's Bay hare–lynx data set, which is a population time series of Snowshoe hares and Canadian lynx (MacLulich, 1937). To generate simulated data, we use a stochastic version of the Lotka–Volterra equations (Lotka, 1910; Volterra, 1926):

$$\mathrm{d}X_t = \alpha X_t \,\mathrm{d}t - \beta Y_t X_t \,\mathrm{d}t + \sigma X_t^{\nu} \,\mathrm{d}W_t^{(1)}, \quad \mathrm{d}Y_t = -\gamma X_t \,\mathrm{d}t + \delta Y_t X_t \,\mathrm{d}t + \sigma Y_t^{\nu} \,\mathrm{d}W_t^{(2)}. \tag{6.9}$$

Under these equations, the prey population X_t grows exponentially with rate α, the predator population Y_t decays exponentially with rate γ, and the predators hunt the prey. $W^{(1)}$ and $W^{(2)}$ are independent Brownian motions introducing noisy behaviour. These equations generate non-Gaussian data with both within-channel as well as cross-channel dependencies. We simulate the Lotka-Volterra equations on a dense grid, and use them to generate meta–data sets in three different ways. *Interpolation*: we randomly subsample the data into context and target sets. *Forecasting*: we choose a random time, before which all data are contexts, and all future data are targets. *Reconstruction:* we randomly choose between the X_t or Y_t, split the chosen series as in forecasting, and append the other series to the context. In training, for every batch, we choose one of these tasks uniformly at random.

Results. Table 6.3 shows the results of the best performing models. The AR ConvCNP performs best both on the simulated as well as the real data, again demonstrating that running CNPs in AR mode improves performance and can even outperform strong GNP and LNP baselines. For full experimental details and additional results see Section D.9.

	EQ		Sawtooth		Mixture	
	KL to ground truth (↓ better)		Pred. log-likelihood (↑ better)		Pred. log-likelihood (↑ better)	
	$d_x, d_y = 1$	$d_x, d_y = 2$	$d_x, d_y = 1$	$d_x, d_y = 2$	$d_x, d_y = 1$	$d_x, d_y = 2$
ConvCNP	0.41 ±0.01	0.41 ±0.00	2.38±0.04	0.12 ±0.01	−0.23±0.04	−0.85 ±0.01
ConvCNP (AR)	0.01 ±0.00	**0.03** ±0.00	**3.60**±0.01	**0.38** ±0.00	**0.45**±0.04	**−0.62** ±0.01
ConvGNP	0.01 ±0.00	0.19 ±0.00	2.62±0.05	0.26 ±0.01	−0.24±0.02	−0.74 ±0.01
FullConvGNP	**0.00** ±0.00		2.16±0.04		−0.05±0.03	
ConvLNP (ML)	0.25 ±0.01	0.39 ±0.00	3.06±0.04	0.31 ±0.01	−0.06±0.03	−0.78 ±0.02
ConvLNP (ELBO)	0.06 ±0.00	0.79 ±0.00	3.51±0.02	0.04 ±0.00	0.12±0.04	−0.92 ±0.01
Diagonal GP	0.40 ±0.01	0.40 ±0.00				
Trivial	1.19 ±0.00	0.79 ±0.00	−0.18±0.00	−0.32 ±0.00	−1.32±0.00	−1.46 ±0.00

Table 6.2 Performance of NPs training on the GP EQ task, sawtooth task, and mixture task. *Diagonal GP* denotes the exact GP predictive, but with correlations removed. *Trivial* denotes a model that predicts a Gaussian distribution with the empirical means and standard deviation of the context outputs. Significantly best models in bold. Note that the FullConvGNP cannot be run on tasks where $d_x > 1$.

Model	Int. (S)	For. (S)	Rec. (S)	Int. (R)	For. (R)	Rec. (R)
ConvCNP	−3.47 ±0.02	−4.06 ±0.02	−4.85 ±0.02	−4.17 ±0.04	−4.70 ±0.06	−4.97 ±0.01
ConvCNP (AR)	**−3.30** ±0.02	**−3.47** ±0.02	**−3.60** ±0.02	**−4.10** ±0.03	**−4.27** ±0.03	**−4.32** ±0.01
ConvGNP	−3.47 ±0.02	−3.65 ±0.02	−4.15 ±0.02	−4.21 ±0.05	−4.82 ±0.13	−4.61 ±0.01
FullConvGNP	**−3.29** ±0.02	**−3.46** ±0.02	−3.79 ±0.02	−4.16 ±0.04	**−4.28** ±0.04	−4.45 ±0.00
ConvLNP (ML)	−3.41 ±0.02	−3.84 ±0.02	−4.44 ±0.02	**−4.13** ±0.04	−4.45 ±0.05	−4.54 ±0.01
ConvLNP (ELBO)	−3.77 ±0.02	−3.83 ±0.02	−4.12 ±0.02	−5.45 ±0.05	−5.47 ±0.07	−6.39 ±0.05

Table 6.3 Normalised log-likelihoods in the predator–prey experiments, showing interpolation (int.), forecasting (for.), and reconstruction (rec.) on simulated (S) and real (R) data. Significantly best results in bold.

6.4.3 Electroencephalogram experiments

Electroencephalogram data. We next trained various NPs on real time series data consisting of electroencephalogram (EEG) measurements (Zhang et al., 1995), as in Section 5.5.3. Each time series consists of 256 regularly spaced measurements across 7 correlated channels. For each channel, we randomly select a number of the 256 points uniformly at random to be target points, and use the remaining ones as context points, independently across the channels.

Results. After training, we test the models on this interpolation task and also on a reconstruction task, where we set a part of a channel as target and the remainder as context. In Table 6.4, we observe that the AR ConvCNP is competitive with the FullConvGNP, despite having significantly shorter training times and fewer parameters.

	ConvCNP	ConvCNP (AR)	ConvGNP	FullConvGNP	ConvLNP (ML)	ConvLNP (ELBO)
Int.	−1.02±0.01	**−0.34**±0.01	−0.93±0.01	**−0.35**±0.01	−1.04±0.01	−1.20±0.01
Rec.	−2.07±0.03	−0.63±0.01	−1.45±0.03	**−0.57**±0.01	−1.53±0.02	−2.00±0.06

Table 6.4 Per-datapoint predictive log-likelihoods on the EEG experiments. Significantly best results in bold.

Both the AR ConvCNP and the FullConvGNP outperform the ConvCNP and the ConvLNP. Full experimental detail are in Section D.10.

6.4.4 Environmental Modelling

Data fusion. A ubiquitous challenge in environmental sciences is fusing spatio-temporal data from disparate sources (Chang and Bai, 2018; Lahat et al., 2015). This challenge, which we refer to as *data fusion*, arises in diverse applications including climate monitoring (Gettelman et al., 2022), air quality estimation (Ferrer-Cid et al., 2020), land cover mapping (Robinson et al., 2021), population estimation (Lu et al., 2010), and hydrology (Hosseini and Kerachian, 2017). NPs naturally handle multiple sources of both on-grid and off-grid data, and can make predictions at arbitrary locations (Markou et al., 2022; Vaughan et al., 2022), making them particularly well suited to data fusion.

Joint and non-Gaussian statistics. Another challenge in environmental sciences is estimating the joint probability of events that span space and time. For example, a renewable energy provider may want to know the compound risk of *both* low wind speeds at their offshore wind farm *and* high cloud coverage over their solar panel farm, possibly at different locations. For such tasks, it is essential to model statistical dependencies to obtain well-calibrated uncertainty estimates. Furthermore, many environmental variables, such as cloud cover, are bounded in output space and are poorly modelled by Gaussian predictives. CNPs are most commonly applied with Gaussian likelihoods, but can be easily used with arbitrary likelihood functions. While GAN-based approaches (Ravuri et al., 2021) can capture both joint as well as non-Gaussian statistics, they are unsuitable for integrating off-grid with on-grid data, as may be required for data fusion. The AR ConvCNP thus fills a gap in the environmental modelling toolbox by enabling data fusion while capturing statistical dependencies and working with non-Gaussian likelihoods. Here, we assess the AR ConvCNP on two common environmental modelling tasks, namely *data assimilation* and *statistical downscaling*.

Data assimilation. Data assimilation is the task of combining observations of the Earth system to produce predictions on a regular grid, called a *reanalysis*. Reanalyses are typically generated by fitting the trajectories of physics-based climate models to observations (Gettelman et al., 2022; Hersbach et al., 2020). AR ConvCNPs can not only provide a substantial speedup over conventional data assimilation systems, but also permit rigorous uncertainty estimation. To explore the ability of the AR ConvCNP to emulate a data assimilation system for a non-Gaussian variable, we train convolutional NP models to predict simulated daily-average cloud cover fraction over Antarctica.

We use reanalysis data from ECMWF ERA5 (Hersbach et al., 2020) as ground truth. Cloud cover ranges in $[0, 1]$, with observations frequently taking values of 0 or 1 (Figure D.8). We evaluate the performance of NPs using either a Gaussian likelihood or a more appropriate beta-categorical mixture model with three components. Two of these are discrete components that capture values of exactly 0 or 1, while the last is a beta distribution which captures continuous values in $(0, 1)$. This provides a robust way of handling 0 and 1 values, unlike the existing Copula GNP model (Markou et al., 2022) which can have its output constrained in $(0, 1)$ but places zero density at the endpoints.

Results. In Table 6.5 we see that the AR ConvCNP significantly outperforms competing NPs for both the Gaussian and beta-categorical likelihoods. Figure 6.6 shows samples drawn from the models, after observing context points on half of the space. The AR ConvCNP displays remarkable ability to extrapolate rich, non-stationary, multi-scale structure, such as sudden changes in cloud cover over the Ross Ice Shelf coastline at the bottom of the figure. By comparison, the ConvLNP and ConvGNP produce blurry, lower frequency samples. Unlike GPs, convolutional NP models have a fixed receptive field induced by the CNN architecture used for the encoder, which is computationally expensive to increase. Away from the context points on the left, samples from the non-AR models will be independent of the observations, reverting to some mean representation of the data (Fig. 6.6c-e). This highlights a further benefit of AR CNPs: successive AR applications increase the receptive field, enabling rich, conditional sample structure to extrapolate far away from observed data. Further commentary and model samples are provided in Section D.11.3. See Section D.11 for full details.

Environmental downscaling. The spatial resolutions of physics-based reanalyses are limited by their vast computational demands, making them unsuitable for capturing local and extreme events (Maraun et al., 2017; Stocker et al., 2013). *Statistical downscaling* addresses this issue by leveraging additional information to produce fine-grained predictions (Maraun and Widmann, 2018b). Recently, NPs have been shown to

	Gaussian				Beta-Categorical		
	ConvGNP	ConvLNP (ML)	ConvCNP	ConvCNP (AR)	ConvLNP (ML)	ConvCNP	ConvCNP (AR)
Log-lik.	0.60±0.02	0.62±0.02	0.58±0.02	**0.88**±0.02	1.06±0.02	1.03±0.02	**1.27**±0.02
MAE (%)	13.05±0.17	12.98±0.16	13.01±0.16	13.01±0.16	12.99±0.16	13.13±0.16	13.13±0.16

Table 6.5 Log-likelihoods (log-lik.) and mean absolute errors (MAE, in units of cloud cover %), over the 2019-2019 test period for the cloud cover task. Note that log-likelihoods cannot be compared directly across the Gaussian and beta-categorical models. Errors indicate standard errors. Significantly best results in bold.

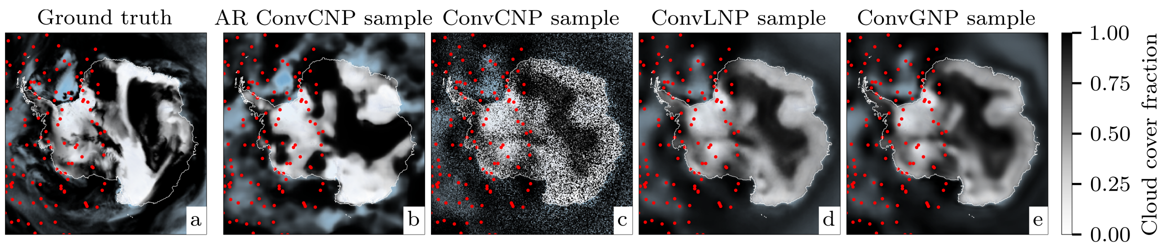


Fig. 6.6 (a) Ground truth simulated cloud cover fraction on 25/06/2018. (b-e), Sample draws from the AR ConvCNP, ConvCNP, ConvLNP and ConvGNP with context points denoted by red dots. Context points were removed from the right hand side of the 2D space to test the models' abilities to extrapolate coherent function samples far away from observations. The ConvCNP and ConvLNP models used a beta-categorical likelihood while the ConvGNP uses a low-rank Gaussian likelihood.

outperform a large ensemble of existing climate downscaling approaches (Vaughan et al., 2022). We compare the AR ConvCNP to the MLP ConvCNP of Vaughan et al. and the MLP ConvGNP from Chapter 5 in a temperature downscaling task over Germany. In this task, the context data consist of low-resolution ERA-Interim reanalysis data and high-resolution topography, and the target data consist of weather station observations from the ECA&D dataset. We also consider a second setup where we reveal some station observations to aid the downscaling process. As Section D.12.2 explains, the MLP ConvCNP and MLP ConvGNP cannot be naively extended to include these station observations. We therefore introduce a novel multiscale architecture, which we use to run the ConvCNP in AR mode. See Section D.12 for full experimental details.

Results. In Table 6.6 we observe that the AR ConvCNP matches the performance of the ConvGNP, which is remarkable as the latter has been previously demonstrated to outperform a range of state-of-the-art downscaling approaches (Markou et al., 2022; Vaughan et al., 2022). When additional observations from weather stations are revealed, the AR ConvCNP significantly outperforms the MLP ConvGNP in both metrics.

Downscaling	Log-lik.	MAE (°C)
ConvCNP (MLP)	−1.55 ±0.01	**0.94** ±0.03
ConvGNP (MLP)	−**1.36** ±0.01	1.09 ±0.09
ConvCNP (AR)	−**1.36** ±0.01	1.04 ±0.04

Down. + stations	Log-lik.	MAE (°C)
ConvCNP* (MLP)	−1.55 ±0.01	0.94 ±0.03
ConvGNP* (MLP)	−1.38 ±0.01	1.09 ±0.09
ConvCNP (AR)	−**1.31** ±0.01	**0.85** ±0.05

Table 6.6 Log-likelihoods and mean absolute errors (MAEs) in the downscaling experiments, without (*left*) and with (*right*) assisting weather station observations. Significantly best results in bold. *Cannot use extra weather station observations.

6.5 Conclusion and Discussion

In this chapter we investigate the AR CNP, a simple change to how standard CNPs are deployed at test time, *without any modifications to the model or training procedure.* Instead of making predictions independently for every target point as with standard CNPs, AR CNPs define a joint predictive distribution using the chain rule of probability, taking inspiration from the neural autoregressive density estimator (NADE) literature. We have shown that applying this AR procedure generally improves the performance of CNPs, producing coherent samples and dramatically improved likelihoods. Surprisingly, in an extensive range of experiments, this simple approach often outperforms more complicated methods which rely on latent variables or that explicitly model correlations. In this chapter, we demonstrated the effectiveness of this approach on the climate data fusion tasks, modelling $[0, 1]$-constrained data with a beta-categorical likelihood and introducing a novel multiscale architecture. Notably, AR CNPs fill a gap in the climate modelling toolbox by enabling joint, non-Gaussian predictives, which could be used to better estimate the magnitude of compound risks. We also position AR CNPs within the larger neural density estimator literature, showing the fruitfulness of combining NPs with other modelling paradigms.

More generally, AR CNPs equip the NP framework with a new knob where modelling complexity and computational expense at training time can be traded for computational expense at test time. In particular, the higher quality samples and better likelihoods obtained by applying NPs autoregressively come with the additional cost of performing a forward pass for every element in the target set. This can be prohibitively expensive for large target sets, and constitutes the primary practical drawback of using AR CNPs. In addition, since AR CNPs do not define a consistent stochastic process, design choices for the AR procedure may affect the quality of the results. Thus practitioners need to avoid choosing target sets that lead to pathological behaviour, such as when the spatial density of the target inputs is too high. However, the flexibility of this design space also presents an opportunity: as an example, in Section D.13 we show that auxiliary target points can be used to further improve predictions. Finally, promising avenues for future work include applying the AR procedure to other NPs besides CNPs, and investigating the efficacy of the block sampling scheme discussed in Section 6.2.

Chapter 7

Conclusion and Discussion

In the final chapter, we summarize the main contributions presented in this thesis. We conclude with a discussion on some open questions and future research directions in the neural process family of models.

7.1 Summary of Contributions

The main motivation for this thesis is to develop methods to address two shortcomings of the neural process family of models with the aim of making them more applicable to real-world problems. First, previous to the work in this thesis, neural processes had to learn translation equivariance directly from data as this inductive bias was not an assumption built directly into the models. Second, parametrizing a rich predictive distribution was challenging as CNPs only produced mean-field predictions modelling each target location independently. Although LNPs do not have this restrictive modelling assumption, they are more expensive and difficult to train requiring approximate inference.

In chapter 3, we introduced the convolutional conditional neural process. The ConvCNP incorporated translation equivariance into its modelling assumptions through the use of the ConvDeepSet, an important building block for neural process architectures moving forward. Using convolutional neural networks (Cohen and Welling, 2016; LeCun et al., 1998), ConvDeepSets extend standard DeepSets to be translation equivariant and we provide a representation theorem for translation-equivariant functions on sets, extending a key result of Zaheer et al. (2017b) to functional embeddings, including sets of varying size. We demonstrate that in cases where data are approximately translation equivariant the ConvCNP outperforms contemporary neural processes models on synthetic and real-world tasks.

Building on the results of the chapter 3, in chapter 4 we introduce the convolutional latent neural process, a latent variable counterpart to the ConvCNP that is able to model epistemic uncertainty and produce coherent function samples, important for many downstream estimation tasks. Unlike previous LNPs which use a finite-dimensional latent vector, the ConvLNP uses a latent function to model task specific information and capture epistemic uncertainty. To overcome some of the challenges with training latent variable models, we propose an approximate maximum likelihood alternative to the standard variational inference approach used for LNPs. We show that we achieve improved performance using our objective. We demonstrate the usefulness of ConvLNPs on synthetic time-series experiments, image-based sampling and extrapolation, and real-world environmental data sets.

In chapter 5, we introduce the Gaussian neural process which can model joint output-space dependencies, like the ConvLNP, but can avoid the issues associated with using the latent variable present in LNPs. GNPs can be trained using the same maximum likelihood technique used by standard CNPs. This chapter also introduces the idea of composing GNPs with invertible marginal transformations to go beyond the Gaussian predictive distribution assumption. We demonstrate that modelling correlations with the GNP improves performance on experiments with both Gaussian and non-Gaussian synthetic and real-world data, including a downstream estimation task that mean-field models cannot solve. We also show that GNPs outperform a standard ensemble of widely used methods in a statistical downscaling climate science application.

Lastly, in chapter 6 we take inspiration from the neural distribution estimation literature (Uria et al., 2016) and, rather than designing a new neural process architecture for modelling dependencies in output variables as we did in previous chapters, we propose evaluating existing models autoregressively via the product rule of probability. This method allows us to use existing, potentially already trained NPs such as standard CNPs to model non-Gaussian predictive distributions and produce coherent samples. This is despite the fact that the CNPs used have Gaussian likelihoods, are not trained to model joint dependencies, and are significantly cheaper to train than LNPs and FullConvGNPs. We prove that given sufficient data and model capacity, the performance of AR CNPs is at least as good as that of GNPs. In this chapter, we propose that neural processes should be viewed as a type of neural density estimator (Uria et al., 2016) highlighting their connections to a range of existing methods in the deep generative modelling literature. We deploy AR CNPs on a range of tasks involving real-world climate data and show that AR CNPs are consistently competitive with, and often significantly outperform, all other neural process models in terms of predictive log-likelihood.

As should be evident from our experimentation, there is no one neural process model that is universally better than all others for every application. As seen in Table 6.1, each member has its own set of tradeoffs. Instead, the contributions contained within this thesis should be viewed as set of tools. They should be used by practitioners when best suited to their applications. The tools that were introduced are

- incorporating translation equvariance and/or the use of infinite latent variables through the use of ConvDeepSets;
- using the Gaussian neural process construction to parametrize joint Gaussian predictive distributions via mean and kernel NPs;
- applying invertible marginal transformations to go beyond the Gaussian predictive distribution assumption;
- and an autoregressive application of standard neural process models for modelling dependencies "out of the box."

These tools can be added to the already existing set of tools in the neural process family such as conditional vs latent variable NPs, attention mechanisms and the array of neural network architecture decisions present in all deep learning frameworks. These tools can also be modularly composed to take advantage of particular aspects of each tool, as was the case when combining ConvCNPs with GNPs to yield the ConvGNP to achieve a translation equivariant neural process capable of modelling full covariance structures. Figure 7.1 shows visualization of the members of the neural process family considered in this thesis including those proposed.

7.2 Future Work

For real-world applications as well as more generally, the bespoke software used by NP researchers can be difficult for practitioners to use out-of-the-box. The creation and refinement of NP software where models can be easily and modularly constructed from simple components encourage practitioners to use CNPs for their applications. We have made strides towards developing some of these tools in the code presented in this thesis, but there is still a lot of work to be done in this area as using it still requires a level of expertise in neural processes.

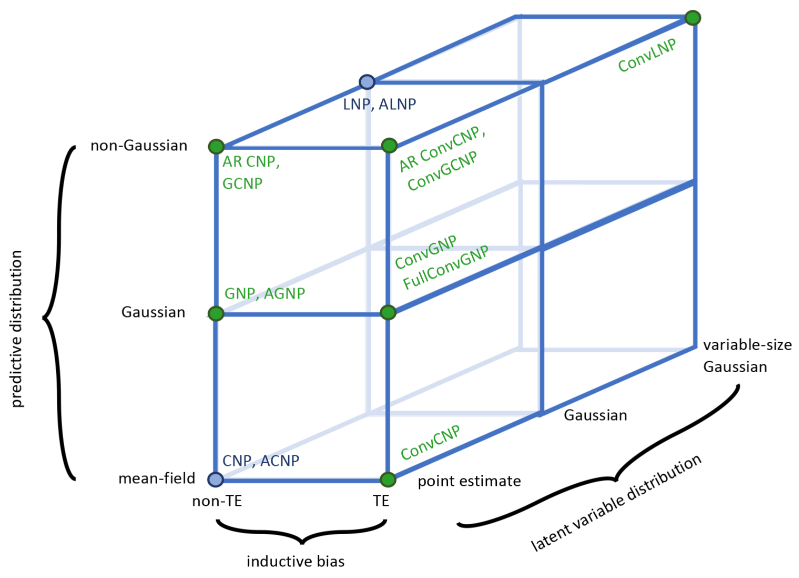


Fig. 7.1 A visualization of the members of the neural process family considered in this thesis. The vertical axis indicates whether the NP produces a mean-field distribution or is able to model a Gaussian or non-Gaussian predictive distribution. The horizontal axis indicates whether TE is incorporated into the NP. The axis going into the page indicates whether the NP models the latent variable distribution $p(z \mid D^{(c)})$ using a point estimate, a Gaussian distribution, or a variable-sized Gaussian via a ConvDeepSet. Green nodes indicate models presented in this thesis.

Unfortunately, as observed in our experimentation, there is no one best neural processes model for all applications. This puts the burden of choice back onto the user, something machine learning and meta-learning are aiming to alleviate by automating decision making. Development of an AutoML framework for neural processes such as Auto-Sklearn (Feurer et al., 2020) or AutoPyTorch (Zimmer et al., 2021) to help automate model selection would go a long way toward making to making neural processes more accessible to non-machine learning researchers.

There are many longer term research directions in the area of AR CNPs such as using AR CNPs as inference models in VAEs, allowing for infinite-dimensional latent spaces, developing architectures that guarantee consistency in AR mode (Korshunova et al., 2020), and developing new connections to the growing literature in diffusion models (Ho et al., 2020). Section D.13 proposed a promising procedure using autoregressive sampling with auxiliary data to generate more expressive marginal predictive distributions. These results also suggests a Gibbs sampling procedure where targets and auxiliary points are sampled repeatedly may produce better results. As we saw in Chapter 6, AR CNP sampling is slow. There is still work to be done investigating the efficacy of the block sampling scheme discussed in Section 6.2. Techniques proposed in the broader AR modelling literature for speeding up sampling (Ramachandran et al., 2017) could potentially be applied to AR CNPs as well.

Lastly, Figure 6.5 illustrated connections between neural processes and other models in the neural distribution estimator literature (Uria et al., 2016) and there are regions on this landscape that are still unexplored, such as a connections between GANs and NPs.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, volume 12, pages 265–283.

Alcorn, M. A. and Nguyen, A. (2021). The deformer: An order-agnostic distribution estimating transformer. *arXiv preprint arXiv:2106.06989.*

Alet, F., Schneider, M. F., Lozano-Perez, T., and Kaelbling, L. P. (2020). Meta-learning curiosity algorithms. *arXiv preprint arXiv:2003.05325.*

Allam Jr, T., Bahmanyar, A., Biswas, R., Dai, M., Galbany, L., Hložek, R., Ishida, E. E., Jha, S. W., Jones, D. O., Kessler, R., et al. (2018). The photometric lsst astronomical time-series classification challenge (plasticc): Data set. *arXiv preprint arXiv:1810.00001.*

Allen, S., Boschung, J., Nauels, A., Xia, Y., Bex, V., and Midgley, P. (2016). Climate change 2013: the physical science basis. contribution of working group i to the fifth assessment report of the intergovernmental panel on climate change.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

Arnold, V. I. (1992). *Ordinary Differential Equations.* Springer-Verlag Berlin Heidelberg.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404.

Ashman, M., So, J., Tebbutt, W., Fortuin, V., Pearce, M., and Turner, R. E. (2020). Sparse gaussian process variational autoencoders. *arXiv preprint arXiv:2010.10177.*

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450.*

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Baño-Medina, J., Manzanas, R., and Gutiérrez, J. M. (2020). Configuration and intercomparison of deep learning neural models for statistical downscaling. *Geoscientific Model Development*, 13(4):2109–2124.

Begleiter, H. (2022). EEG database data set.

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. (2018). Invertible residual networks. *arXiv preprint arXiv:1811.00995*.

Bhardwaj, A., Misra, V., Mishra, A., Wootten, A., Boyles, R., Bowden, J., and Terando, A. J. (2018). Downscaling future climate change projections over puerto rico using a non-hydrostatic atmospheric model. *Climatic Change*, 147(1-2):133–147.

Biggs, J. B. (1985). The role of metalearning in study processes. *British journal of educational psychology*, 55(3):185–212.

Bloem-Reddy, B. and Teh, Y. W. (2020). Probabilistic symmetries and invariant neural networks. *Journal of Machine Learning Research*, 21(90):1–61.

Boone, K. (2019). Avocado: Photometric classification of astronomical transients with gaussian process augmentation. *arXiv preprint arXiv:1907.04690*.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brock, A., Donahue, J., and Simonyan, K. (2018). Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.

Bruinsma, W., Perim, E., Tebbutt, W., Hosking, S., Solin, A., and Turner, R. (2020). Scalable exact inference in multi-output gaussian processes. In *International Conference on Machine Learning*, pages 1190–1201. PMLR.

Bruinsma, W. P., Requeima, J., Foong, A. Y. K., Gordon, J., and Turner, R. E. (2021). The Gaussian neural process. In *Proceedings of the 3rd Symposium on Advances in Approximate Bayesian Inference*.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2016). Manifold gaussian processes for regression.

Chang, N.-B. and Bai, K. (2018). *Multisensor data fusion and machine learning for environmental remote sensing*. CRC Press.

Chen, R. T., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016a). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. (2016b). Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*.

Chen, X., Mishra, N., Rohaninejad, M., and Abbeel, P. (2018). Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 864–872. PMLR.

Child, R. (2020). Very deep vaes generalize autoregressive models and can outperform them on images. *arXiv preprint arXiv:2011.10650*.

Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Cohen, T., Weiler, M., Kicanaoglu, B., and Welling, M. (2019). Gauge equivariant convolutional networks and the icosahedral CNN. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1321–1330, Long Beach, California, USA. PMLR.

Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA. PMLR.

Cremer, C., Li, X., and Duvenaud, D. (2018). Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*.

Cressie, N. (1990). The origins of kriging. *Mathematical geology*, 22(3):239–252.

De Finetti, B. (1937). La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68.

Dee, D. P., Uppala, S. M., Simmons, A. J., Berrisford, P., Poli, P., Kobayashi, S., Andrae, U., Balmaseda, M. A., Balsamo, G., Bauer, P., Bechtold, P., Beljaars, A. C. M., van de Berg, L., Bidlot, J., Bormann, N., Delsol, C., Dragani, R., Fuentes, M., Geer, A. J., Haimberger, L., Healy, S. B., Hersbach, H., Holm, E. V., Isaksen, L., Kållberg, P., Kohler, M., Matricardi, M., McNally, A. P., Monge-Sanz, B. M., Morcrette, J.-J., Park, B.-K., Peubey, C., de Rosnay, P., Tavolato, C., Thepaut, J.-N., and Vitart, F. (2011). The ERA-interim reanalysis: Configuration and performance of the data assimilation system. *Quarterly Journal of the Royal Meteorological Society*, 137(656):553–597.

DeGeneres, E. (2014). If only Bradley's arm was longer. Best photo ever. Oscars pic.twitter.com/c9u5notgap.

Delhomme, J. P. (1978). Kriging in the hydrosciences. *Advances in water resources*, 1:251–266.

Dieng, A. B., Ruiz, F. J., Blei, D. M., and Titsias, M. K. (2019). Prescribed generative adversarial networks. *arXiv preprint arXiv:1910.04302*.

Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

Dugundji, J. et al. (1951). An extension of tietze's theorem. *Pacific Journal of Mathematics*, 1(3):353–367.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural spline flows.

Durrett, R. (2010). *Probability: Theory and Examples*. Cambridge University Press, 4 edition.

Earth Resources Observation and Science Center, U.S. Geological Survey, U.S. Department of the Interior (1997). USGS 30 arc-second global elevation data, GTOPO30.

Elidan, G. (2013). Copulas in machine learning. In Jaworski, P., Durante, F., and Härdle, W. K., editors, *Copulae in Mathematical and Quantitative Finance*, pages 39–60, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ellison, A. M. (1987). Effect of seed dimorphism on the density-dependent dynamics of experimental populations of atriplex triangularis (chenopodiaceae). *American Journal of Botany*, 74(8):1280–1288.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., and Taylor, K. E. (2016). Overview of the coupled model intercomparison project phase 6 (cmip6) experimental design and organization. *Geoscientific Model Development*, 9(5):1937–1958.

Ferrer-Cid, P., Barcelo-Ordinas, J. M., Garcia-Vidal, J., Ripoll, A., and Viana, M. (2020). Multisensor data fusion calibration in iot air pollution platforms. *IEEE Internet of Things Journal*, 7(4):3124–3132.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2020). Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

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. In *Advances in Neural Information Processing Systems 33*. Curran Associates, Inc.

Fortuin, V., Baranchuk, D., Rätsch, G., and Mandt, S. (2020). Gp-vae: Deep probabilistic time series imputation. In *International conference on artificial intelligence and statistics*, pages 1651–1661. PMLR.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR.

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. In *Proceedings of 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*. PMLR.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. (2018b). Neural processes. In *Proceedings of 35th International Conference on Machine Learning*.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR.

Gettelman, A., Geer, A. J., Forbes, R. M., Carmichael, G. R., Feingold, G., Posselt, D. J., Stephens, G. L., van den Heever, S. C., Varble, A. C., and Zuidema, P. (2022). The future of earth system prediction: Advances in model-data fusion. *Science Advances*, 8(14):eabn3488.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361.

Gneiting, T. and Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I. J., Abadie, J. P., Mirza, M., Xu, B., Farley, D. W., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

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

Gordon, J., Bruinsma, W. P., Foong, A. Y. K., Requeima, J., Dubois, Y., and Turner, R. E. (2020). Convolutional conditional neural processes. In *Proceedings of the 8th International Conference on Learning Representations*.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2018). Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*.

Grover, A., Dhar, M., and Ermon, S. (2018). Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2016). Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*.

Harlow, H. F. (1949). The formation of learning sets. *Psychological review*, 56(1):51.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., Simmons, A., Soci, C., Abdalla, S., Abellan, X., Balsamo, G., Bechtold, P., Biavati, G., Bidlot, J., Bonavita, M., De Chiara, G., Dahlgren, P., Dee, D., Diamantakis, M., Dragani, R., Flemming, J., Forbes, R., Fuentes, M., Geer, A., Haimberger, L., Healy, S., Hogan, R. J., Holm, E., Janiskova, M., Keeley, S., Laloyaux, P., Lopez, P., Lupu, C., Radnoti, G., de Rosnay, P., Rozum, I., Vamborg, F., Villaume, S., and Thépaut, J.-N. (2020). The ERA5 global reanalysis. 146(730):1999–2049.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730. PMLR.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.

Hoogeboom, E., Gritsenko, A. A., Bastings, J., Poole, B., Berg, R. v. d., and Salimans, T. (2021). Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2021). Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169.

Hosseini, M. and Kerachian, R. (2017). A data fusion-based methodology for optimal redesign of groundwater monitoring networks. *Journal of Hydrology*, 552:267–282.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087. PMLR.

Hundley, D. R. (2022). Introduction to mathematical modelling.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Jaimungal, S. and Ng, E. K. (2009). Kernel-based copula processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 628–643. Springer.

Jetchev, N., Bergmann, U., and Vollgraf, R. (2016). Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*.

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2019). Attentive neural processes. In *Proceedings of the 7th International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2015). ADAM: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Klein Tank, A., Wijngaard, J., Können, G., Böhm, R., Demarée, G., Gocheva, A., Mileta, M., Pashiardis, S., Hejkrlik, L., Kern-Hansen, C., et al. (2002). Daily dataset of 20th-century surface air temperature and precipitation series for the european climate assessment. *International Journal of Climatology: A Journal of the Royal Meteorological Society*, 22(12):1441–1453.

Knutsson, H. and Westin, C.-F. (1993). Normalized and differential convolution. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 515–523. IEEE.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kondor, I. R. (2008). *Group theoretical methods in machine learning*. Columbia University.

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. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2747–2755, Stockholmsmässan, Stockholm Sweden. PMLR.

Korshunova, I., Gal, Y., Gretton, A., and Dambre, J. (2020). Conditional bruno: A neural process for exchangeable labelled data. *Neurocomputing*, 416:305–309.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105.

Lahat, D., Adali, T., and Jutten, C. (2015). Multimodal data fusion: an overview of methods, challenges, and prospects. *Proceedings of the IEEE*, 103(9):1449–1477.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Le, T. A., Kim, H., Garnelo, M., Rosenbaum, D., Schwarz, J., and Teh, Y. W. (2018). Empirical evaluation of neural process objectives. In *NeurIPS workshop on Bayesian Deep Learning*.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Leigh, E. R. (1968). The ecological role of volterra's equations. *Some mathematical problems in biology*.

Liu, H., Simonyan, K., and Yang, Y. (2018a). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Liu, Y., Ganguly, A. R., and Dy, J. (2020). Climate downscaling using ynet: A deep convolutional network with skip connections and fusion. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3145–3153.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2018b). Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15:2018.

Lotka, A. J. (1910). Contribution to the theory of periodic reactions. *The Journal of Physical Chemistry*, 14(3):271–274.

Louizos, C., Shi, X., Schutte, K., and Welling, M. (2019). The functional neural process. *Advances in Neural Information Processing Systems*, 32.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Lu, C., Turner, R. E., Li, Y., and Kushman, N. (2020). Interpreting spatially infinite generative models. *arXiv preprint arXiv:2007.12411*.

Lu, Z., Im, J., Quackenbush, L., and Halligan, K. (2010). Population estimation based on multi-sensor data fusion. *International Journal of Remote Sensing*, 31(21):5587–5604.

MacLulich, D. A. (1937). *Fluctuations in the Numbers of the Varying Hare (Lepus Americanus)*. University of Toronto Press.

Maraun, D., Shepherd, T. G., Widmann, M., Zappa, G., Walton, D., Gutiérrez, J. M., Hagemann, S., Richter, I., Soares, P. M. M., Hall, A., and Mearns, L. O. (2017). Towards process-informed bias correction of climate change simulations. *Nature Climate Change*, 7(11):764–773.

Maraun, D. and Widmann, M. (2018a). *Statistical downscaling and bias correction for climate research*. Cambridge University Press.

Maraun, D. and Widmann, M. (2018b). *Statistical Downscaling and Bias Correction for Climate Research*. Cambridge Uiversity Press.

Maraun, D., Widmann, M., Gutiérrez, J. M., Kotlarski, S., Chandler, R. E., Hertig, E., Wibig, J., Huth, R., and Wilcke, R. A. I. (2015). VALUE: A framework to validate downscaling approaches for climate change studies. *Earth's Future*, 3(1):1–14.

Markou, S., Requeima, J., Bruinsma, W. P., and Turner, R. E. (2021). Efficient Gaussian neural processes for regression. In *Proceedings of 39th International Conference on Machine Learning*.

Markou, S., Requeima, J., Bruinsma, W. P., Vaughan, A., and Turner, R. E. (2022). Practical conditional neural processes via tractable dependent predictions. In *Proceedings of the 10th International Conference on Learning Representations*.

Maroñas, J., Hamelijnck, O., Knoblauch, J., and Damoulas, T. (2021). Transforming gaussian processes with normalizing flows. In *International Conference on Artificial Intelligence and Statistics*, pages 1081–1089. PMLR.

Matthews, A. G. d. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the Kullback-Leibler divergence between stochastic processes. In *Artificial Intelligence and Statistics*, pages 231–239.

Menick, J. and Kalchbrenner, N. (2018). Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *arXiv preprint arXiv:1812.01608*.

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. (2018). Meta-learning update rules for unsupervised representation learning. *arXiv preprint arXiv:1804.00222*.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. volume 17, pages 362–369. Morgan Kaufmann Publishers Inc.

Misra, S., Sarkar, S., and Mitra, P. (2018). Statistical downscaling of precipitation using long short-term memory recurrent neural networks. *Theoretical and applied climatology*, 134(3):1179–1196.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*.

Morlighem, M. (2020). Measures bedmachine antarctica, version 2.

Munkres, J. (1974). *Topology; a First Course*. Prentice-Hall.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Nguyen, T. and Grover, A. (2022). Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In *International Conference on Machine Learning*, pages 16569–16594. PMLR.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and Checkerboard Artifacts. *Distill*, 1(10):e3.

Oksendal, B. (2013). *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Oord, A. v. d., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016b). Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29.

Oord, A. v. d., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Pan, B., Hsu, K., AghaKouchak, A., and Sorooshian, S. (2019). Improving precipitation estimation using convolutional neural network. *Water Resources Research*, 55(3):2301–2321.

Papamakarios, G. and Murray, I. (2016). Fast ϵ-free inference of simulation models with bayesian conditional density estimation. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 1028–1036.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image transformer. 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 4055–4064, Stockholmsmässan, Stockholm Sweden. PMLR.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Patacchiola, M., Turner, J., Crowley, E. J., and Storkey, A. J. (2019). Deep kernel transfer in gaussian processes for few-shot learning. *CoRR*, abs/1910.05199.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). FiLM: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations (ICLR-16)*.

Ramachandran, P., Paine, T. L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., Hasegawa-Johnson, M. A., Campbell, R. H., and Huang, T. S. (2017). Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *Proceedings of the 5th International Conference on Learning Representations*.

Ravuri, S., Lenc, K., Willson, M., Kangin, D., Lam, R., Mirowski, P., Fitzsimons, M., Athanassiadou, M., Kashem, S., Madge, S., Prudden, R., Mandhane, A., Clark, A., Brock, A., Simonyan, K., Hadsell, R., Robinson, N., Clancy, E., Arribas, A., and Mohamed, S. (2021). Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677. Number: 7878 Publisher: Nature Publishing Group.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. (2019). Fast and flexible multi-task classification using conditional neural adaptive processes. *Advances in Neural Information Processing Systems*, 32.

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.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. (2013). Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20110550.

Robinson, C., Malkin, K., Jojic, N., Chen, H., Qin, R., Xiao, C., Schmitt, M., Ghamisi, P., Hänsch, R., and Yokoya, N. (2021). Global land-cover mapping with weak supervision: Outcome of the 2020 ieee grss data fusion contest. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:3185–3199.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Ross, S. M., Kelly, J. J., Sullivan, R. J., Perry, W. J., Mercer, D., Davis, R. M., Washburn, T. D., Sager, E. V., Boyce, J. B., and Bristow, V. L. (1996). *Stochastic processes*, volume 2. Wiley New York.

Rudner, T. G., Fortuin, V., Teh, Y. W., and Gal, Y. (2018). On the connection between neural processes and gaussian processes with deep kernels. In *Workshop on Bayesian Deep Learning, NeurIPS*, page 14.

Sachindra, D., Ahmed, K., Rashid, M. M., Shahid, S., and Perera, B. (2018). Statistical downscaling of precipitation using machine learning techniques. *Atmospheric research*, 212:240–258.

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017). Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*.

Salimans, T., Kingma, D., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226. PMLR.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning*. PhD thesis, Technische Universität München.

Schrier, A. M. (1984). Learning how to learn: The significance and current status of learning set formation. *Primates*, 25(1):95–102.

Service, C. C. C. (2020). Copernicus Climate Change Service (C3S) (2019): C3S ERA5-Land reanalysis. (accessed: 15.05.2020).

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4080–4090.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR.

Stocker, T. F., Qin, D., Plattner, G.-K., Tignor, M. M., Allen, S. K., Boschung, J., Nauels, A., Xia, Y., Bex, V., and Midgley, P. M. (2013). Climate change 2013: The physical science basis. Technical report, Cambridge University Press.

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional variational Bayesian neural networks. In *International Conference on Learning Representations*.

Tank, A. M. G. K., Wijngaard, J. B., Können, G. P., Böhm, R., Demarée, G., Gocheva, A., Mileta, M., Pashiardis, S., Hejkrlik, L., Kern-Hansen, C., Heino, R., Bessemoulin, P., Müller-Westermeier, G., Tzanakou, M., Szalai, S., Pálsdóttir, T., Fitzgerald, D., Rubin, S., Capaldo, M., Maugeri, M., Leitass, A., Bukantis, A., Aberfeld, R., van Engelen, A. F. V., Forland, E., Mietus, M., Coelho, F., Mares, C., Razuvaev, V., Nieplova, E., Cegnar, T., López, J. A., Dahlström, B., Moberg, A., Kirchhofer, W., Ceylan, A., Pachaliuk, O., Alexander, L. V., and Petrovic, P. (2002). Daily dataset of 20th-century surface air temperature and precipitation series for the european climate assessment. *International Journal of Climatology*, 22(12):1441–1453.

Tao, T. (2011). *An introduction to measure theory*, volume 126. American Mathematical Society Providence, RI.

Theobald, D. M., Harrison-Atlas, D., Monahan, W. B., and Albano, C. M. (2015). Ecologically-relevant maps of landforms and physiographic diversity for climate adaptation planning. *PLoS One*, 10(12):e0143619.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

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

Turner, R. E. and Sahani, M. (2011). Two problems with variational expectation maximisation for time-series models. In Barber, D., Cemgil, T., and Chiappa, S., editors, *Bayesian Time series models*, chapter 5, pages 109–130. Cambridge University Press.

Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37.

Uria, B., Murray, I., and Larochelle, H. (2013). Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, 26.

Uria, B., Murray, I., and Larochelle, H. (2014). A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475. PMLR.

Vandal, T., Kodra, E., Dy, J., Ganguly, S., Nemani, R., and Ganguly, A. R. (2018). Quantifying uncertainty in discrete-continuous and skewed data with bayesian deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2377–2386.

Vandal, T., Kodra, E., and Ganguly, A. R. (2019). Intercomparison of machine learning methods for statistical downscaling: the case of daily and extreme precipitation. *Theoretical and Applied Climatology*, 137(1):557–570.

Vandal, T., Kodra, E., Ganguly, S., Michaelis, A., Nemani, R., and Ganguly, A. R. (2017). Deepsd: Generating high resolution climate change projections through single image super-resolution. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining*, pages 1663–1672.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Vaughan, A., Tebbutt, W., Hosking, J. S., and Turner, R. E. (2022). Convolutional conditional neural processes for local climate downscaling. *Geoscientific Model Development*, 15(1):251–268.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc.

Volpp, M., Flürenbrock, F., Grossberger, L., Daniel, C., and Neumann, G. (2021). Bayesian context aggregation for neural processes. In *International Conference on Learning Representations*.

Volterra, V. (1926). Variazioni e fluttuazioni del bumero d'ondividui in specie animali conviventi. *Memoria della Reale Accademia Nazionale dei Lincei*, 2:31–113.

Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. (2019). On the limitations of representing functions on sets. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6487–6494, Long Beach, California, USA. PMLR.

Wang, S., Suo, S., Ma, W.-C., Pokrovsky, A., and Urtasun, R. (2018). Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597.

White, B., Singh, A., and Albert, A. (2019). Downscaling numerical weather models with gans. In *AGU Fall Meeting Abstracts*, volume 2019, pages GC43D–1357.

Wilkinson, D. J. (2011). *Stochastic modelling for systems biology*. CRC press.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Wilson, A. G. and Ghahramani, Z. (2010). Copula processes. *Advances in Neural Information Processing Systems*, 23:2460–2468.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2015). Deep kernel learning.

Wu, W., Qi, Z., and Fuxin, L. (2019). PointConv: Deep convolutional networks on 3d point clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wu, Y., Burda, Y., Salakhutdinov, R., and Grosse, R. (2016). On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017a). Deep sets. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017b). Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404.

Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. (2019). Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR.

Zhang, X. L., Begleiter, H., Porjesz, B., Wang, W., and Litke, A. (1995). Event related potentials during object recognition tasks. *Brain Research Bulletin*, 38(6):531–538.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079–3090.

Appendix A

Chapter 3 Supplementary Material

A.1 Theoretical Results and Proofs

In this section, we provide the proof of Thm 1. Our proof strategy is as follows. We first define an appropriate topology for fixed-sized sets (Section A.1.1). With this topology in place, we demonstrate that our proposed embedding into function space is homeomorphic (Lems 1 and 2). We then show that the embeddings of fixed-sized sets can be extended to varying-sized sets by "pasting" the embeddings together while maintaining their homeomorphic properties (Lem 3). Following this, we demonstrate that the resulting embedding may be composed with a continuous mapping to our desired target space, resulting in a continuous mapping between two metric spaces (Lem 4). Finally, in Section A.1.3 we combine the above-mentioned results to prove Thm 1.

We begin with definitions that we will use throughout the section and then present our results. Let $\mathcal{X} = \mathbb{R}^d$ and let $\mathcal{Y} \subseteq \mathbb{R}$ be compact. Let ψ be a symmetric, positive-definite kernel on $\mathcal{X}$. By the Moore–Aronszajn Theorem, there is a unique Hilbert space $(\mathcal{H}, \langle \bullet, \bullet \rangle_{\mathcal{H}})$ of real-valued functions on $\mathcal{X}$ for which ψ is a reproducing kernel. This means that (i) $\psi(\bullet, \boldsymbol{x}) \in \mathcal{H}$ for all $\boldsymbol{x} \in \mathcal{X}$ and (ii) $\langle f, \psi(\bullet, \boldsymbol{x}) \rangle_{\mathcal{H}} = f(\boldsymbol{x})$ for all $f \in \mathcal{H}$ and $\boldsymbol{x} \in \mathcal{X}$ (reproducing property). For $\psi \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathcal{X}^n$, and $\boldsymbol{X}' = (\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_n) \in \mathcal{X}^n$, we denote

$$\psi(\boldsymbol{X}, \boldsymbol{X}') = \begin{bmatrix} \psi(\boldsymbol{x}_1, \boldsymbol{x}'_1) & \cdots & \psi(\boldsymbol{x}_1, \boldsymbol{x}'_n) \\ \vdots & \ddots & \vdots \\ \psi(\boldsymbol{x}_n, \boldsymbol{x}'_1) & \cdots & \psi(\boldsymbol{x}_n, \boldsymbol{x}'_n) \end{bmatrix}.$$

Definition 4 (Interpolating RKHS)**.** Call $\mathcal{H}$ *interpolating* if it interpolates any finite number of points: for every $((\boldsymbol{x}_i, y_i))_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$ with $(\boldsymbol{x}_i)_{i=1}^n$ all distinct, there is an $f \in \mathcal{H}$ such that $f(\boldsymbol{x}_1) = y_1, \dots, f(\boldsymbol{x}_n) = y_n$.

For example, the RKHS induced by any strictly positive-definite kernel, e.g. the exponentiated quadratic (EQ) kernel $\psi(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp(-\frac{1}{2\ell^2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2)$, is interpolating: Let $\boldsymbol{c} = \psi(\boldsymbol{X}, \boldsymbol{X})^{-1}\boldsymbol{y}$ and consider $f = \sum_{i=1}^n c_i \psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i) \in \mathcal{H}$. Then $f(\boldsymbol{X}) = \psi(\boldsymbol{X}, \boldsymbol{X})\boldsymbol{c} = \boldsymbol{y}$.

A.1.1 The Quotient Space $\mathcal{A}^n / \mathbb{S}_n$

Let $\mathcal{A}$ be a Banach space. For $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathcal{A}^n$ and $\boldsymbol{y} = (y_1, \dots, y_n) \in \mathcal{A}^n$, let $\boldsymbol{x} \sim \boldsymbol{y}$ if $\boldsymbol{x}$ is a permutation of $\boldsymbol{y}$; that is, $\boldsymbol{x} \sim \boldsymbol{y}$ if and only if $\boldsymbol{x} = \pi\boldsymbol{y}$ for some $\pi \in \mathbb{S}_n$ where

$$\pi\boldsymbol{y} = (y_{\pi(1)}, \dots, y_{\pi(n)}).$$

Let $\mathcal{A}^n / \mathbb{S}_n$ be the collection of equivalence classes of $\sim$. Denote the equivalence class of $\boldsymbol{x}$ by $[\boldsymbol{x}]$; for $A \subseteq \mathcal{A}^n$, denote $[A] = \{[\boldsymbol{a}] : \boldsymbol{a} \in A\}$. Call the map $\boldsymbol{x} \mapsto [\boldsymbol{x}] \colon \mathcal{A}^n \to \mathcal{A}^n / \mathbb{S}_n$ the canonical map. The natural topology on $\mathcal{A}^n / \mathbb{S}_n$ is the quotient topology, in which a subset of $\mathcal{A}^n / \mathbb{S}_n$ is open if and only if its preimage under the canonical map is open in $\mathcal{A}^n$. In what follows, we show that the quotient topology is metrizable.

On $\mathcal{A}^n$, since all norms on finite-dimensional vector spaces are equivalent, without loss of generality consider

$$\|\boldsymbol{x}\|_{\mathcal{A}^n}^2 = \textstyle\sum_{i=1}^n \|x_i\|_{\mathcal{A}}^2.$$

Note that $\|\,\boldsymbol{\cdot}\,\|_{\mathcal{A}^n}$ is permutation invariant: $\|\pi\,\boldsymbol{\cdot}\,\|_{\mathcal{A}^n} = \|\,\boldsymbol{\cdot}\,\|_{\mathcal{A}^n}$ for all $\pi \in \mathbb{S}_n$. On $\mathcal{A}^n / \mathbb{S}_n$, define

$$d \colon \mathcal{A}^n / \mathbb{S}_n \times \mathcal{A}^n / \mathbb{S}_n \to [0, \infty), \quad d([\boldsymbol{x}], [\boldsymbol{y}]) = \min\nolimits_{\pi \in \mathbb{S}_n} \|\boldsymbol{x} - \pi\boldsymbol{y}\|_{\mathcal{A}^n}.$$

Call a set $[A] \subseteq \mathcal{A}^n / \mathbb{S}_n$ bounded if $\{d([\boldsymbol{x}], [0]) : [\boldsymbol{x}] \in [A]\}$ is bounded.

Proposition 4. The function d is a metric.

Proof. We first show that d is well defined on $\mathcal{A}^n / \mathbb{S}_n$. Assume $\boldsymbol{x} \sim \boldsymbol{x}'$ and $\boldsymbol{y} \sim \boldsymbol{y}'$. Then, $\boldsymbol{x}' = \pi_{\boldsymbol{x}}\boldsymbol{x}$ and $\boldsymbol{y}' = \pi_{\boldsymbol{y}}\boldsymbol{y}$. Using the group properties of $\mathbb{S}_n$ and the permutation invariance of $\|\bullet\|_{\mathcal{A}^n}$:

$$\begin{aligned} d([\boldsymbol{x}'],[\boldsymbol{y}']) &= \min_{\pi\in\mathbb{S}_n} \|\pi_{\boldsymbol{x}}\boldsymbol{x} - \pi\pi_{\boldsymbol{y}}\boldsymbol{y}\|_{\mathcal{A}^n} \\ &= \min_{\pi\in\mathbb{S}_n} \|\pi_{\boldsymbol{x}}\boldsymbol{x} - \pi\boldsymbol{y}\|_{\mathcal{A}^n} \\ &= \min_{\pi\in\mathbb{S}_n} \|\boldsymbol{x} - \pi_{\boldsymbol{x}}^{-1}\pi\boldsymbol{y}\|_{\mathcal{A}^n} \\ &= \min_{\pi\in\mathbb{S}_n} \|\boldsymbol{x} - \pi\boldsymbol{y}\|_{\mathcal{A}^n} \\ &= d([\boldsymbol{x}],[\boldsymbol{y}]). \end{aligned}$$

It is clear that $d([\boldsymbol{x}],[\boldsymbol{y}]) = d([\boldsymbol{y}],[\boldsymbol{x}])$ and that $d([\boldsymbol{x}],[\boldsymbol{y}]) = 0$ if and only if $[\boldsymbol{x}] = [\boldsymbol{y}]$. To show the triangle inequality, note that

$$\|\boldsymbol{x} - \pi_1\pi_2\boldsymbol{y}\|_{\mathcal{A}^n} \le \|\boldsymbol{x} - \pi_1\boldsymbol{z}\|_{\mathcal{A}^n} + \|\pi_1\boldsymbol{z} - \pi_1\pi_2\boldsymbol{y}\|_{\mathcal{A}^n} = \|\boldsymbol{x} - \pi_1\boldsymbol{z}\|_{\mathcal{A}^n} + \|\boldsymbol{z} - \pi_2\boldsymbol{y}\|_{\mathcal{A}^n},$$

using permutation invariance of $\|\bullet\|_{\mathcal{A}^n}$. Hence, taking the minimum over π_1,

$$d([\boldsymbol{x}],[\boldsymbol{y}]) \le d([\boldsymbol{x}],[\boldsymbol{z}]) + \|\boldsymbol{z} - \pi_2\boldsymbol{y}\|_{\mathcal{A}^n},$$

so taking the minimum over π_2 gives the triangle inequality for d. □

Proposition 5. The canonical map $\mathcal{A}^n \to \mathcal{A}^n/\mathbb{S}_n$ is continuous under the metric topology induced by d.

Proof. Follows directly from $d([\boldsymbol{x}],[\boldsymbol{y}]) \le \|\boldsymbol{x} - \boldsymbol{y}\|_{\mathcal{A}^n}$. □

Proposition 6. Let $A \subseteq \mathcal{A}^n$ be topologically closed and closed under permutations. Then $[A]$ is topologically closed in $\mathcal{A}^n/\mathbb{S}_n$ under the metric topology.

Proof. Recall that a subset $[A]$ of a metric space is closed iff every limit point of $[A]$ is also in $[A]$. Consider a sequence $([\boldsymbol{a}_n])_{n=1}^{\infty} \subseteq [A]$ converging to some $[\boldsymbol{x}] \in \mathcal{A}^n/\mathbb{S}_n$. Then there are permutations $(\pi_n)_{n=1}^{\infty} \subseteq \mathbb{S}_n$ such that $\pi_n\boldsymbol{a}_n \to \boldsymbol{x}$. Here $\pi_n\boldsymbol{a}_n \in A$, because A is closed under permutations. Thus $\boldsymbol{x} \in A$, as A is also topologically closed. We conclude that $[\boldsymbol{x}] \in [A]$. □

Proposition 7. Let $A \subseteq \mathcal{A}^n$ be open. Then $[A]$ is open in $\mathcal{A}^n/\mathbb{S}_n$ under the metric topology. In other words, the canonical map is open under the metric topology.

Proof. Let $[\boldsymbol{x}] \in [A]$. Because A is open, there is some ball $B_\varepsilon(\boldsymbol{y})$ with $\varepsilon > 0$ and $\boldsymbol{y} \in A$ such that $\boldsymbol{x} \in B_\varepsilon(\boldsymbol{y}) \subseteq A$. Then $[\boldsymbol{x}] \in B_\varepsilon([\boldsymbol{y}])$, since $d([\boldsymbol{x}], [\boldsymbol{y}]) \le \|\boldsymbol{x} - \boldsymbol{y}\|_{\mathcal{A}^n} < \varepsilon$, and we claim that $B_\varepsilon([\boldsymbol{y}]) \subseteq [A]$. Hence $[\boldsymbol{x}] \in B_\varepsilon([\boldsymbol{y}]) \subseteq [A]$, so $[A]$ is open.

To show the claim, let $[\boldsymbol{z}] \in B_\varepsilon([\boldsymbol{y}])$. Then $d(\pi\boldsymbol{z}, \boldsymbol{y}) < \varepsilon$ for some $\pi \in \mathbb{S}_n$. Hence $\pi\boldsymbol{z} \in B_\varepsilon(\boldsymbol{y}) \subseteq A$, so $\pi\boldsymbol{z} \in A$. Therefore, $[\boldsymbol{z}] = [\pi\boldsymbol{z}] \in [A]$. □

Proposition 8. The quotient topology on $\mathcal{A}^n/\,\mathbb{S}_n$ induced by the canonical map is metrizable with the metric d.

Proof. Since the canonical map is surjective, there exists exactly one topology on $\mathcal{A}^n/\,\mathbb{S}_n$ relative to which the canonical map is a quotient map: the quotient topology (Munkres, 1974).

Let $p\colon \mathcal{A}^n \to \mathcal{A}^n/\,\mathbb{S}_n$ denote the canonical map. It remains to show that p is a quotient map under the metric topology induced by d; that is, we show that $U \subset \mathcal{A}^n/\,\mathbb{S}_n$ is open in $\mathcal{A}^n/\,\mathbb{S}_n$ under the metric topology if and only if $p^{-1}(U)$ is open in $\mathcal{A}^n$.

Let $p^{-1}(U)$ be open in $\mathcal{A}^n$. We have that $U = p(p^{-1}(U))$, so U is open in $\mathcal{A}^n/\,\mathbb{S}_n$ under the metric topology by Proposition 7. Conversely, if U is open in $\mathcal{A}^n/\,\mathbb{S}_n$ under the metric topology, then $p^{-1}(U)$ is open in $\mathcal{A}^n$ by continuity of the canonical map under the metric topology. □

A.1.2 Embeddings of Sets Into an RKHS

Whereas $\mathcal{A}$ previously denoted an arbitrary Banach space, in this section we specialize to $\mathcal{A} = \mathcal{X} \times \mathcal{Y}$. We denote an element in $\mathcal{A}$ by $(\boldsymbol{x}, y)$ and an element in $\mathcal{Z}_M = \mathcal{A}^M$ by $((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_M, y_M))$. Alternatively, we denote $((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_M, y_M))$ by $(\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_M) \in \mathcal{X}^M$ and $\boldsymbol{y} = (y_1, \dots, y_M) \in \mathcal{Y}^M$. We clarify that an element in $\mathcal{Z}_M = \mathcal{A}^M$ is permuted as follows: for $\pi \in \mathbb{S}_M$,

$$\pi(\boldsymbol{X}, \boldsymbol{y}) = \pi((\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_M, y_M)) = ((\boldsymbol{x}_{\pi(1)}, y_{\pi(1)}), \dots, (\boldsymbol{x}_{\pi(n)}, y_{\pi(n)})) = (\pi\boldsymbol{X}, \pi\boldsymbol{y}).$$

Note that permutation-invariant functions on $\mathcal{Z}_M$ are in correspondence to functions on the quotient space induced by the equivalence class of permutations, $\mathcal{Z}_M/\,\mathbb{S}_m$ The latter is a more natural representation.

Lem 3 states that it is possible to homeomorphically embed sets into an RKHS. This result is key to proving our main result. Before proving Lem 3, we provide several useful results. We begin by demonstrating that an embedding of sets of a fixed size into a RKHS is continuous and injective.

Lemmma 1. *Consider a collection $\mathcal{Z}'_M \subseteq \mathcal{Z}_M$ that has multiplicity K. Set*

$$\phi : \mathcal{Y} \to \mathbb{R}^{K+1}, \quad \phi(y) = (y^0, y^1, \cdots, y^K)$$

and let ψ be an interpolating, continuous positive-definite kernel. Define

$$\mathcal{H}_M = \left\{ \sum_{i=1}^{M} \phi(y_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i) : (\boldsymbol{x}_i, y_i)_{i=1}^M \subseteq \mathcal{Z}'_M \right\} \subseteq \mathcal{H}^{K+1}, \tag{A.1}$$

where $\mathcal{H}^{K+1} = \mathcal{H} \times \cdots \times \mathcal{H}$ is the $(K+1)$-dimensional-vector–valued–function Hilbert space constructed from the RKHS $\mathcal{H}$ for which ψ is a reproducing kernel and endowed with the inner product $\langle f, g\rangle_{\mathcal{H}^{K+1}} = \sum_{i=1}^{K+1}\langle f_i, g_i\rangle_{\mathcal{H}}$. Then the embedding

$$E_M \colon [\mathcal{Z}'_M] \to \mathcal{H}_M, \quad E_M([(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_M, y_M)]) = \sum_{i=1}^{M} \phi(y_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i)$$

is injective, hence invertible, and continuous.

Proof. First, we show that E_M is injective. Suppose that

$$\sum_{i=1}^{M} \phi(y_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i) = \sum_{i=1}^{M} \phi(y'_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}'_i).$$

Denote $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M)$ and $\boldsymbol{y} = (y_1, \ldots, y_M)$, and denote $\boldsymbol{X}'$ and $\boldsymbol{y}'$ similarly. Taking the inner product with any $f \in \mathcal{H}$ on both sides and using the reproducing property of ψ, this implies that

$$\sum_{i=1}^{M} \phi(y_i) f(\boldsymbol{x}_i) = \sum_{i=1}^{M} \phi(y'_i) f(\boldsymbol{x}'_i)$$

for all $f \in \mathcal{H}$. In particular, since by construction $\phi_1(\,\boldsymbol{\cdot}\,) = 1$,

$$\sum_{i=1}^{M} f(\boldsymbol{x}_i) = \sum_{i=1}^{M} f(\boldsymbol{x}'_i)$$

for all $f \in \mathcal{H}$. Using that $\mathcal{H}$ is interpolating, choose a particular $\hat{\boldsymbol{x}} \in \boldsymbol{X} \cup \boldsymbol{X}'$, and let $f \in \mathcal{H}$ be such that $f(\hat{\boldsymbol{x}}) = 1$ and $f(\,\boldsymbol{\cdot}\,) = 0$ at all other $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$. Then

$$\sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} 1 = \sum_{i:\boldsymbol{x}'_i=\hat{\boldsymbol{x}}} 1,$$

so the number of such $\hat{\boldsymbol{x}}$ in $\boldsymbol{X}$ and the number of such $\hat{\boldsymbol{x}}$ in $\boldsymbol{X}'$ are the same. Since this holds for every $\hat{\boldsymbol{x}}$, $\boldsymbol{X}$ is a permutation of $\boldsymbol{X}'$: $\boldsymbol{X} = \pi(\boldsymbol{X}')$ for some permutation $\pi \in \mathbb{S}_M$. Plugging in the permutation, we can write

$$\sum_{i=1}^{M} \phi(y_i) f(\boldsymbol{x}_i) = \sum_{i=1}^{M} \phi(y'_i) f(\boldsymbol{x}'_i) \overset{(\boldsymbol{X}'=\pi^{-1}(\boldsymbol{X}))}{=} \sum_{i=1}^{M} \phi(y'_i) f(\boldsymbol{x}_{\pi^{-1}(i)}) \overset{(i\leftarrow\pi^{-1}(i))}{=} \sum_{i=1}^{M} \phi(y'_{\pi(i)}) f(\boldsymbol{x}_i).$$

Then, by a similar argument, for any particular $\hat{\boldsymbol{x}}$,

$$\sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} \phi(y_i) = \sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} \phi(y'_{\pi(i)}).$$

Let the number of terms in each sum equal S. Since $\mathcal{Z}'_M$ has multiplicity K, $S \le K$. By Lemma 4 from Zaheer et al. (2017b), the 'sum-of-power mapping' from $\{y_i : \boldsymbol{x}_i = \hat{\boldsymbol{x}}\}$ to the first $S+1$ elements of $\sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} \phi(y_i)$, i.e. $\left(\sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} y_i^0, \ldots, \sum_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} y_i^S\right)$, is injective. Therefore,

$$(y_i)_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}} \quad \text{is a permutation of} \quad (y'_{\pi(i)})_{i:\boldsymbol{x}_i=\hat{\boldsymbol{x}}}.$$

Note that $\boldsymbol{x}_i = \hat{\boldsymbol{x}}$ for all above y_i. Furthermore, note that also $\boldsymbol{x}'_{\pi(i)} = \boldsymbol{x}_i = \hat{\boldsymbol{x}}$ for all above $y'_{\pi(i)}$. We may therefore adjust the permutation π such that $y_i = y'_{\pi(i)}$ for all i such that $\boldsymbol{x}_i = \hat{\boldsymbol{x}}$ whilst retaining that $\boldsymbol{x} = \pi(\boldsymbol{x}')$. Performing this adjustment for all $\hat{\boldsymbol{x}}$, we find that $y = \pi(y')$ and $\boldsymbol{x} = \pi(\boldsymbol{x}')$.

Second, we show that E_M is continuous. Compute

$$\begin{aligned}
&\left\| \sum_{i=1}^{M} \phi(y_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i) - \sum_{j=1}^{M} \phi(y'_j)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}'_j) \right\|^2_{\mathcal{H}^{K+1}} \\
&\quad = \sum_{i=1}^{K+1} \left(\phi_i^\top(\boldsymbol{y})\psi(\boldsymbol{X}, \boldsymbol{X})\phi_i(\boldsymbol{y}) - 2\phi_i^\top(\boldsymbol{y})\psi(\boldsymbol{X}, \boldsymbol{X}')\phi_i(\boldsymbol{y}') + \phi_i^\top(\boldsymbol{y}')\psi(\boldsymbol{X}', \boldsymbol{X}')\phi_i(\boldsymbol{y}') \right),
\end{aligned}$$

which goes to zero if $[\boldsymbol{X}', \boldsymbol{y}'] \to [\boldsymbol{X}, \boldsymbol{y}]$ by continuity of ψ. □

Having established the injection, we now show that this mapping is a homeomorphism, i.e. that the inverse is continuous. This is formalized in the following lemma.

Lemmma 2. *Consider Lem 1. Suppose that $\mathcal{Z}'_M$ is also topologically closed in $\mathcal{A}^M$ and closed under permutations, and that ψ also satisfies (i) $\psi(\boldsymbol{x}, \boldsymbol{x}') \geq 0$, (ii) $\psi(\boldsymbol{x}, \boldsymbol{x}) = \sigma^2 > 0$, and (iii) $\psi(\boldsymbol{x}, \boldsymbol{x}') \to 0$ as $\|\boldsymbol{x}\| \to \infty$. Then $\mathcal{H}_M$ is closed in $\mathcal{H}^{K+1}$ and E_M^{-1} is continuous.*

Remark 1. To define $\mathcal{Z}'_2$ with multiplicity one, one might be tempted to define

$$\mathcal{Z}'_2 = \{((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2)) \in \mathcal{Z}_2 : \boldsymbol{x}_1 \neq \boldsymbol{x}_2\},$$

which indeed has multiplicity one. Unfortunately, $\mathcal{Z}'_2$ is not closed: if $[0, 1] \subseteq \mathcal{X}$ and $[0, 2] \subseteq \mathcal{Y}$, then $((0, 1), (1/n, 2))_{n=1}^{\infty} \subseteq \mathcal{Z}'_2$, but $((0, 1), (1/n, 2)) \to ((0, 1), (0, 2)) \notin \mathcal{Z}'_2$, because 0 then has two observations 1 and 2. To get around this issue, one can require an arbitrarily small, but non-zero spacing $\epsilon > 0$ between input locations:

$$\mathcal{Z}'_{2,\epsilon} = \{((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2)) \in \mathcal{Z}_2 : \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| \geq \epsilon\}.$$

This construction can be generalized to higher numbers of observations and multiplicities as follows:

$$\mathcal{Z}'_{M,K,\epsilon} = \{(\boldsymbol{x}_{\pi(i)}, y_{\pi(i)})_{i=1}^{M} \in \mathcal{Z}_M : \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq \epsilon \text{ for } i, j \in [K],\, \pi \in \mathbb{S}_M\}.$$

Remark 2. Before moving on to the proof of Lem 2, we remark that Lem 2 would directly follow if $\mathcal{Z}'_M$ were bounded: then $\mathcal{Z}'_M$ is compact, so E_M is a continuous, invertible map between a compact space and a Hausdorff space, which means that E_M^{-1} must be continuous. The intuition that the result must hold for unbounded $\mathcal{Z}'_M$ is as follows. Since $\phi_1(\,\boldsymbol{\cdot}\,) = 1$, for every $f \in \mathcal{H}_M$, f_1 is a summation of M "bumps" (imagine the EQ kernel) of the form $\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i)$ placed throughout $\mathcal{X}$. If one of these bumps goes off to infinity, then the function cannot uniformly converge pointwise, which means that the function cannot converge in $\mathcal{H}$ (if ψ is sufficiently nice). Therefore, if the function does converge in $\mathcal{H}$, $(\boldsymbol{x}_i)_{i=1}^{M}$ must be bounded, which brings us to the compact case. What makes this work is the *density channel* $\phi_1(\,\boldsymbol{\cdot}\,) = 1$, which forces $(\boldsymbol{x}_i)_{i=1}^{M}$ to be well behaved. The above argument is formalized in the proof of Lem 2.

Proof. Define

$$\mathcal{Z}_J = ([-J, J]^d \times \mathcal{Y})^M \cap \mathcal{Z}'_M,$$

which is compact in $\mathcal{A}^M$ as a closed subset of the compact set $([-J, J]^d \times \mathcal{Y})^M$. We aim to show that $\mathcal{H}_M$ is closed in $\mathcal{H}^{K+1}$ and E^{-1} is continuous. To this end, consider a convergent sequence

$$f^{(n)} = \textstyle\sum_{i=1}^M \phi(y_i^{(n)})\psi(\,\bullet\,, \boldsymbol{x}_i^{(n)}) \to f \in \mathcal{H}^{K+1}.$$

Denote $\boldsymbol{X}^{(n)} = (\boldsymbol{x}_1^{(n)}, \dots, \boldsymbol{x}_M^{(n)})$ and $\boldsymbol{y}^{(n)} = (y_1^{(n)}, \dots, y_M^{(n)})$. Claim: $(\boldsymbol{X}^{(n)})_{n=1}^\infty$ is a bounded sequence, so $(\boldsymbol{X}^{(n)})_{n=1}^\infty \subseteq [-J, J]^{dM}$ for J large enough, which means that $(\boldsymbol{X}^{(n)}, \boldsymbol{y}^{(n)})_{n=1}^\infty \subseteq \mathcal{Z}_J$ where $\mathcal{Z}_J$ is compact. Note that $[\mathcal{Z}_J]$ is compact in $\mathcal{A}^M/\,\mathbb{S}_M$ by continuity of the canonical map.

First, we demonstrate that, assuming the claim, $\mathcal{H}_M$ is closed. Note that by boundedness of $(\boldsymbol{X}^{(n)}, \boldsymbol{y}^{(n)})_{n=1}^\infty$, $(f^{(n)})_{n=1}^\infty$ is in the image of $E_M|_{[\mathcal{Z}_J]} \colon [\mathcal{Z}_J] \to \mathcal{H}_M$. By continuity of $E_M|_{[\mathcal{Z}_J]}$ and compactness of $[\mathcal{Z}_J]$, the image of $E_M|_{[\mathcal{Z}_J]}$ is compact and therefore closed, since every compact subset of a metric space is closed. Therefore, the image of $E_M|_{[\mathcal{Z}_J]}$ contains the limit f. Since the image of $E_M|_{[\mathcal{Z}_J]}$ is included in $\mathcal{H}_M$, we have that $f \in \mathcal{H}_M$, which shows that $\mathcal{H}_M$ is closed.

Next, we prove that, assuming the claim, E_M^{-1} is continuous. Consider $E_M|_{[\mathcal{Z}_J]} \colon [\mathcal{Z}_J] \to E_M([\mathcal{Z}_J])$ restricted to its image. Then $(E_M|_{[\mathcal{Z}_J]})^{-1}$ is continuous, because a continuous bijection from a compact space to a metric space is a homeomorphism. Therefore

$$E_M^{-1}(f^{(n)}) = (\boldsymbol{X}^{(n)}, \boldsymbol{y}^{(n)}) = (E_M|_{[\mathcal{Z}_J]})^{-1}(f^{(n)}) \to (E_M|_{[\mathcal{Z}_J]})^{-1}(f) = (\boldsymbol{X}, \boldsymbol{y}).$$

By continuity and invertibility of E_M, then $f^{(n)} \to E_M(\boldsymbol{X}, \boldsymbol{y})$, so $E_M(\boldsymbol{X}, \boldsymbol{y}) = f$ by uniqueness of limits. We conclude that $E_M^{-1}(f^{(n)}) \to E_M^{-1}(f)$, which means that E_M^{-1} is continuous.

It remains to show the claim. Let f_1 denote the first element of f, i.e. the density channel. Using the reproducing property of ψ,

$$|f_1^{(n)}(\boldsymbol{x}) - f_1(\boldsymbol{x})| = |\langle \psi(\boldsymbol{x}, \,\bullet\,), f_1^{(n)} - f_1 \rangle| \le \|\psi(\boldsymbol{x}, \,\bullet\,)\|_{\mathcal{H}} \|f_1^{(n)} - f_1\|_{\mathcal{H}} = \sigma \|f_1^{(n)} - f_1\|_{\mathcal{H}},$$

so $f_1^{(n)} \to f_1$ in $\mathcal{H}$ means that it does so uniformly pointwise (over $\boldsymbol{x}$). Hence, we can let $N \in \mathbb{N}$ be such that $n \ge N$ implies that $|f_1^{(n)}(\boldsymbol{x}) - f_1(\boldsymbol{x})| < \frac{1}{3}\sigma^2$ for all $\boldsymbol{x}$. Let R be such that $|\psi(\boldsymbol{x}, \boldsymbol{x}_i^{(N)})| < \frac{1}{3}\sigma^2/M$ for $\|\boldsymbol{x}\| \ge R$ and all $i \in [M]$. Then, for $\|\boldsymbol{x}\| \ge R$,

$$|f_1^{(N)}(\boldsymbol{x})| \le \textstyle\sum_{i=1}^M |\psi(\boldsymbol{x}, \boldsymbol{x}_i^{(N)})| < \frac{1}{3}\sigma^2 \implies |f_1(\boldsymbol{x})| \le |f_1^{(N)}(\boldsymbol{x})| + |f_1^{(N)}(\boldsymbol{x}) - f_1(\boldsymbol{x})| < \frac{2}{3}\sigma^2.$$

At the same time, by pointwise non-negativity of ψ, we have that

$$f_1^{(n)}(\boldsymbol{x}_i^{(n)}) = \textstyle\sum_{j=1}^M \psi(\boldsymbol{x}_j^{(n)}, \boldsymbol{x}_i^{(n)}) \geq \psi(\boldsymbol{x}_i^{(n)}, \boldsymbol{x}_i^{(n)}) = \sigma^2.$$

Towards contradiction, suppose that $(\boldsymbol{X}^{(n)})_{n=1}^\infty$ is unbounded. Then $(\boldsymbol{x}_i^{(n)})_{n=1}^\infty$ is unbounded for some $i \in [M]$. Therefore, $\|\boldsymbol{x}_i^{(n)}\| \geq R$ for some $n \geq N$, so

$$\tfrac{2}{3}\sigma^2 > |f_1(\boldsymbol{x}_i^{(n)})| \geq |f_1^{(n)}(\boldsymbol{x}_i^{(n)})| - |f_1^{(n)}(\boldsymbol{x}_i^{(n)}) - f_1(\boldsymbol{x}_i^{(n)})| \geq \sigma^2 - \tfrac{1}{3}\sigma^2 = \tfrac{2}{3}\sigma^2,$$

which is a contradiction. □

The following lemma states that we may construct an encoding for sets containing no more than M elements into a function space, where the encoding is injective and every restriction to a fixed set size is a homeomorphism.

Lemmma 3. *For every $m \in [M]$, consider a collection $\mathcal{Z}'_m \subseteq \mathcal{Z}_m$ that (i) has multiplicity K, (ii) is topologically closed, and (iii) is closed under permutations. Set*

$$\phi : \mathcal{Y} \to \mathbb{R}^{K+1}, \quad \phi(y) = (y^0, y^1, \cdots, y^K)$$

and let ψ be an interpolating, continuous positive-definite kernel that satisfies (i) $\psi(\boldsymbol{x}, \boldsymbol{x}') \geq 0$, (ii) $\psi(\boldsymbol{x}, \boldsymbol{x}) = \sigma^2 > 0$, and (iii) $\psi(\boldsymbol{x}, \boldsymbol{x}') \to 0$ as $\|\boldsymbol{x}\| \to \infty$. Define

$$\mathcal{H}_m = \left\{ \sum_{i=1}^m \phi(y_i)\psi(\,\bullet\,, \boldsymbol{x}_i) : (\boldsymbol{x}_i, y_i)_{i=1}^m \subseteq \mathcal{Z}'_m \right\} \subseteq \mathcal{H}^{K+1}, \tag{A.2}$$

where $\mathcal{H}^{K+1} = \mathcal{H} \times \cdots \times \mathcal{H}$ is the $(K+1)$-dimensional-vector–valued–function Hilbert space constructed from the RKHS $\mathcal{H}$ for which ψ is a reproducing kernel and endowed with the inner product $\langle f, g\rangle_{\mathcal{H}^{K+1}} = \sum_{i=1}^{K+1} \langle f_i, g_i\rangle_{\mathcal{H}}$. Denote

$$[\mathcal{Z}'_{\leq M}] = \bigcup_{m=1}^M [\mathcal{Z}'_m] \quad \textit{and} \quad \mathcal{H}_{\leq M} = \bigcup_{m=1}^M \mathcal{H}_m.$$

Then $(\mathcal{H}_m)_{m=1}^M$ are pairwise disjoint. It follows that the embedding E

$$E \colon [\mathcal{Z}'_{\leq M}] \to \mathcal{H}_{\leq M}, \quad E([Z]) = E_m([Z]) \quad \textit{if} \quad [Z] \in [\mathcal{Z}'_m]$$

is injective, hence invertible. Denote this inverse by E^{-1}, where $E^{-1}(f) = E_m^{-1}(f)$ if $f \in \mathcal{H}_m$.

Proof. Recall that E_m is injective for every $m \in [M]$. Hence, to demonstrate that E is injective it remains to show that $(\mathcal{H}_m)_{m=1}^M$ are pairwise disjoint. To this end, suppose that

$$\sum_{i=1}^{m} \phi(y_i)\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i) = \sum_{i=1}^{m'} \phi(y_i')\psi(\,\boldsymbol{\cdot}\,, \boldsymbol{x}_i')$$

for $m \neq m'$. Then, by arguments like in the proof of Lem 1,

$$\sum_{i=1}^{m} \phi(y_i) = \sum_{i=1}^{m'} \phi(y_i').$$

Since $\phi_1(\,\boldsymbol{\cdot}\,) = 1$, this gives $m = m'$, which is a contradiction. Finally, by repeated application of Lem 2, E_m^{-1} is continuous for every $m \in [M]$. □

Lemmma 4. *Let $\Phi\colon [\mathcal{Z}'_{\le M}] \to C_b(\mathcal{X}, \mathcal{Y})$ be a map from $[\mathcal{Z}'_{\le M}]$ to $C_b(\mathcal{X}, \mathcal{Y})$, the space of continuous bounded functions from $\mathcal{X}$ to $\mathcal{Y}$, such that every restriction $\Phi|_{[\mathcal{Z}'_m]}$ is continuous, and let E be from Lem 3. Then*

$$\Phi \circ E^{-1}\colon \mathcal{H}_{\le M} \to C_b(\mathcal{X}, \mathcal{Y})$$

is continuous.

Proof. Recall that, due to Lem 1, for every $m \in [M]$, E_m^{-1} is continuous and has image $[\mathcal{Z}'_m]$. By the continuity of $\Phi|_{[\mathcal{Z}'_m]}$, then $\Phi|_{[\mathcal{Z}'_m]} \circ E_m^{-1}$ is continuous for every $m \in [M]$. Since $\Phi \circ E^{-1}|_{\mathcal{H}_m} = \Phi|_{[\mathcal{Z}'_m]} \circ E_m^{-1}$ for all $m \in [M]$, we have that $\Phi \circ E^{-1}|_{\mathcal{H}_m}$ is continuous for all $m \in [M]$. Therefore, as $\mathcal{H}_m$ is closed in $\mathcal{H}_{\le M}$ for every $m \in [M]$, the pasting lemma (Munkres, 1974) yields that $\Phi \circ E^{-1}$ is continuous. □

From here on, we let ψ be a stationary kernel, which means that it only depends on the difference of its arguments and can be seen as a function $\mathcal{X} \to \mathbb{R}$.

A.1.3 Proof of Thm 1

With the above results in place, we are finally ready to prove our central result, Thm 1.

Theorem 4. For every $m \in [M]$, consider a collection $\mathcal{Z}'_m \subseteq \mathcal{Z}_m$ that (i) has multiplicity K, (ii) is topologically closed, (iii) is closed under permutations, and (iv) is closed under translations. Set

$$\phi : \mathcal{Y} \to \mathbb{R}^{K+1}, \quad \phi(y) = (y^0, y^1, \cdots, y^K)$$

and let ψ be an interpolating, continuous positive-definite kernel that satisfies (i) $\psi(\boldsymbol{x}, \boldsymbol{x}') \geq 0$, (ii) $\psi(\boldsymbol{x}, \boldsymbol{x}) = \sigma^2 > 0$, and (iii) $\psi(\boldsymbol{x}, \boldsymbol{x}') \to 0$ as $\|\boldsymbol{x}\| \to \infty$. Define

$$\mathcal{H}_m = \left\{ \sum_{i=1}^{m} \phi(y_i)\psi(\,\bullet\,, \boldsymbol{x}_i) : (\boldsymbol{x}_i, y_i)_{i=1}^m \subseteq \mathcal{Z}'_m \right\} \subseteq \mathcal{H}^{K+1}, \tag{A.3}$$

where $\mathcal{H}^{K+1} = \mathcal{H} \times \cdots \times \mathcal{H}$ is the $(K+1)$-dimensional-vector–valued–function Hilbert space constructed from the RKHS $\mathcal{H}$ for which ψ is a reproducing kernel and endowed with the inner product $\langle f, g \rangle_{\mathcal{H}^{K+1}} = \sum_{i=1}^{K+1} \langle f_i, g_i \rangle_{\mathcal{H}}$. Denote

$$\mathcal{Z}'_{\leq M} = \bigcup_{m=1}^{M} \mathcal{Z}'_m \quad \text{and} \quad \mathcal{H}_{\leq M} = \bigcup_{m=1}^{M} \mathcal{H}_m.$$

Then a function $\Phi \colon \mathcal{Z}'_{\leq M} \to C_b(\mathcal{X}, \mathcal{Y})$ satisfies (i) continuity of the restriction $\Phi|_{\mathcal{Z}_m}$ for every $m \in [M]$, (ii) permutation invariance (Property 1), and (iii) translation equivariance (Property 2) if and only if it has a representation of the form

$$\Phi(Z) = \rho\left(E(Z)\right), \quad E((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) = \textstyle\sum_{i=1}^{m} \phi(y_i)\psi(\,\bullet - \boldsymbol{x}_i)$$

where $\rho \colon \mathcal{H}_{\leq M} \to C_b(\mathcal{X}, \mathcal{Y})$ is continuous and translation equivariant.

Proof of sufficiency. To begin with, note that permutation invariance (Property 1) and translation equivariance (Property 2) for Φ are well defined, because $\mathcal{Z}'_{\leq M}$ is closed under permutations and translations by assumption. First, Φ is permutation invariant, because addition is commutative and associative. Second, that Φ is translation equivariant (Property 2) follows from a direct verification and that ρ is also translation equivariant:

$$\begin{aligned}
\Phi(T_{\boldsymbol{\tau}} Z) &= \rho\left(\sum_{i=1}^{M} \phi(y_i)\psi(\,\bullet - (\boldsymbol{x}_i + \boldsymbol{\tau})) \right) \\
&= \rho\left(\sum_{i=1}^{M} \phi(y_i)\psi((\,\bullet - \boldsymbol{\tau}) - \boldsymbol{x}_i) \right) \\
&= \rho\left(\sum_{i=1}^{M} \phi(y_i)\psi(\,\bullet - \boldsymbol{x}_i) \right)(\cdot - \boldsymbol{\tau}) \\
&= \Phi(Z)(\cdot - \boldsymbol{\tau}) \\
&= T'_{\boldsymbol{\tau}} \Phi(Z).
\end{aligned}$$

□

Proof of necessity. Our proof follows the strategy used by Wagstaff et al. (2019); Zaheer et al. (2017b). To begin with, since Φ is permutation invariant (Property 1), we may define

$$\Phi\colon \bigcup_{m=1}^{M} [\mathcal{Z}'_m] \to C_b(\mathcal{X}, \mathcal{Y}), \quad \Phi(Z) = \Phi([Z]),$$

for which we verify that every restriction $\Phi|_{[\mathcal{Z}'_m]}$ is continuous. By invertibility of E from Lem 3, we have $[Z] = E^{-1}(E([Z]))$. Therefore,

$$\Phi(Z) = \Phi([Z]) = \Phi(E^{-1}(E([Z]))) = (\Phi \circ E^{-1}) \left(\sum_{i=1}^{M} \phi(y_i)\psi(\bullet - \boldsymbol{x}_i) \right).$$

Define $\rho\colon \mathcal{H}_{\le M} \to C_b(\mathcal{X}, \mathcal{Y})$ by $\rho = \Phi \circ E^{-1}$. First, ρ is continuous by Lem 4. Second, E^{-1} is translation equivariant, because ψ is stationary. Also, by assumption Φ is translation equivariant (Property 2). Thus, their composition ρ is also translation equivariant. □

Remark 3. The function $\rho\colon \mathcal{H}_{\le M} \to C_b(\mathcal{X}, \mathcal{Y})$ may be continuously extended to the entirety of $\mathcal{H}^{K+1}$ using a generalisation of the Tietze Extension Theorem by Dugundji et al. (1951). There are variants of Dugundji's Theorem that also preserve translation equivariance.

A.2 Baseline Neural Process Models

In both our 1d and image experiments, our main comparison is to conditional neural process models. In particular, we compare to a vanilla CNP (1d only; Garnelo et al. (2018a)) and an ACNP (Kim et al., 2019). Our architectures largely follow the details given in the relevant publications.

CNP baseline. Our baseline CNP follows the implementation provided by the authors.[1] The encoder is a 3-layer MLP with 128 hidden units in each layer, and ReLU non-linearities. The encoder embeds every context point into a representation, and the representations are then averaged across each context set. Target inputs are then concatenated with the latent representations, and passed to the decoder. The decoder follows the same architecture, outputting mean and standard deviation channels for each input.

[1]https://github.com/deepmind/neural-processes

Attentive CNP baseline. The ACNP we use corresponds to the deterministic path of the model described by Kim et al. (2019) for image experiments. Namely, an encoder first embeds each context point c to a latent representation $(\mathbf{x}^{(c)}, \mathbf{y}^{(c)}) \mapsto \mathbf{r}_{xy}^{(c)} \in \mathbb{R}^{128}$. For the image experiments, this is achieved using a 2-hidden layer MLP of hidden dimensions 128. For the 1d experiments, we use the same encoder as the CNP above. Every context point then goes through two stacked self-attention layers. Each self-attention layer is implemented with an 8-headed attention, a skip connection, and two layer normalizations (as described in Parmar et al. (2018), modulo the dropout layer). To predict values at each target point t, we embed $\mathbf{x}^{(t)} \mapsto \mathbf{r}_x^{(t)}$ and $\mathbf{x}^{(c)} \mapsto \mathbf{r}_x^{(c)}$ using the same single hidden layer MLP of dimensions 128. A target representation $\mathbf{r}_{xy}^{(t)}$ is then estimated by applying cross-attention (using an 8-headed attention described above) with keys $\mathrm{K} := \{\mathbf{r}_x^{(c)}\}_{c=1}^{C}$, values $\mathrm{V} := \{\mathbf{r}_{xy}^{(c)}\}_{c=1}^{C}$, and query $\mathbf{q} := \mathbf{r}_x^{(t)}$. Given the estimated target representation $\hat{\mathbf{r}}_{xy}^{(t)}$, the conditional predictive posterior is given by a Gaussian pdf with diagonal covariance parametrised by $(\boldsymbol{\mu}^{(t)}, \boldsymbol{\sigma}_{\mathrm{pre}}^{(t)}) = \mathrm{decoder}(\mathbf{r}_{xy}^{(t)})$ where $\boldsymbol{\mu}^{(t)}, \boldsymbol{\sigma}_{\mathrm{pre}}^{(t)} \in \mathbb{R}^3$ and decoder is a 4 hidden layer MLP with 64 hidden units per layer for the images, and the same decoder as the CNP for the 1d experiments.

Following Le et al. (2018), we enforce we set a minimum standard deviation $\boldsymbol{\sigma}_{\mathrm{min}}^{(t)} = [0.1; 0.1; 0.1]$ to avoid infinite log-likelihoods by using the following post-processed standard deviation:

$$\boldsymbol{\sigma}_{\mathrm{post}}^{(t)} = 0.1\boldsymbol{\sigma}_{\mathrm{min}}^{(t)} + (1 - 0.1)\log(1 + \exp(\boldsymbol{\sigma}_{\mathrm{pre}}^{(t)})) \tag{A.4}$$

A.3 1-Dimensional Experiments

In this section, we give details regarding our experiments for the 1d data. We begin by detailing model architectures, and then provide details for the data generating processes and training procedures. The density at which we evaluate the grid differs from experiment to experiment, and so the values are given in the relevant subsections. In all experiments, the weights are optimized using Adam (Kingma and Ba, 2015) and weight decay of 10^{-5} is applied to all model parameters. The learning rates are specified in the following subsections.

A.3.1 CNN Architectures

Throughout the experiments (Sections 3.5.1 to 3.5.3), we consider two models: ConvCNP (which utilizes a smaller architecture), and ConvCNPXL (with a larger architecture). For all architectures, the input kernel ψ was an EQ (exponentiated quadratic) kernel with a learnable length scale parameter, as detailed in Section 3.4, as was the kernel for the final output layer ψ_ρ. When dividing by the density channel, we add $\varepsilon = 10^{-8}$ to avoid numerical issues. The length scales for the EQ kernels are initialized to twice the spacing $1/\gamma^{1/d}$ between the discretization points $(\boldsymbol{t}_i)_{i=1}^{T}$, where γ is the density of these points and d is the dimensionality of the input space $\mathcal{X}$.

Moreover, we emphasize that the size of the receptive field is a product of the width of the CNN filters and the spacing between the discretization points. Consequently, for a fixed width kernel of the CNN, as the number of discretization points increases, the receptive field size decreases. One potential improvement that was not employed in our experiments, is the use of depthwise-separable convolutions (Chollet, 2017). These dramatically reduce the number of parameters in a convolutional layer, and can be used to increase the CNN filter widths, thus allowing one to increase the number of discretization points without reducing the receptive field.

The architectures for ConvCNP and ConvCNPXL are described below.

ConvCNP. For the 1d experiments, we use a simple, 4-layer convolutional architecture, with ReLU nonlinearities. The kernel size of the convolutional layers was chosen to be 5, and all employed a stride of length 1 and zero padding of 2 units. The number of channels per layer was set to $[16, 32, 16, 2]$, where the final channels where then processed by the final, EQ-based layer of ρ as mean and standard deviation channels. We employ a softplus nonlinearity on the standard deviation channel to enforce positivity. This model has 6,537 parameters.

ConvCNPXL. Our large architecture takes inspiration from UNet (Ronneberger et al., 2015). We employ a 12-layer architecture with skip connections. The number of channels is doubled every layer for the first 6 layers, and halved every layer for the final 6 layers. We use concatenation for the skip connections. The following describes which layers are concatenated, where $L_i \leftarrow [L_j, L_k]$ means that the input to layer i is the concatenation of the activations of layers j and k:

- $L_8 \leftarrow [L_5, L_7]$,
- $L_9 \leftarrow [L_4, L_8]$,
- $L_{10} \leftarrow [L_3, L_9]$,
- $L_{11} \leftarrow [L_2, L_{10}]$,
- $L_{12} \leftarrow [L_1, L_{11}]$.

Like for the smaller architecture, we use ReLU nonlinearities, kernels of size 5, stride 1, and zero padding for two units on all layers.

A.3.2 Synthetic 1d Experimental Details and Additional Results

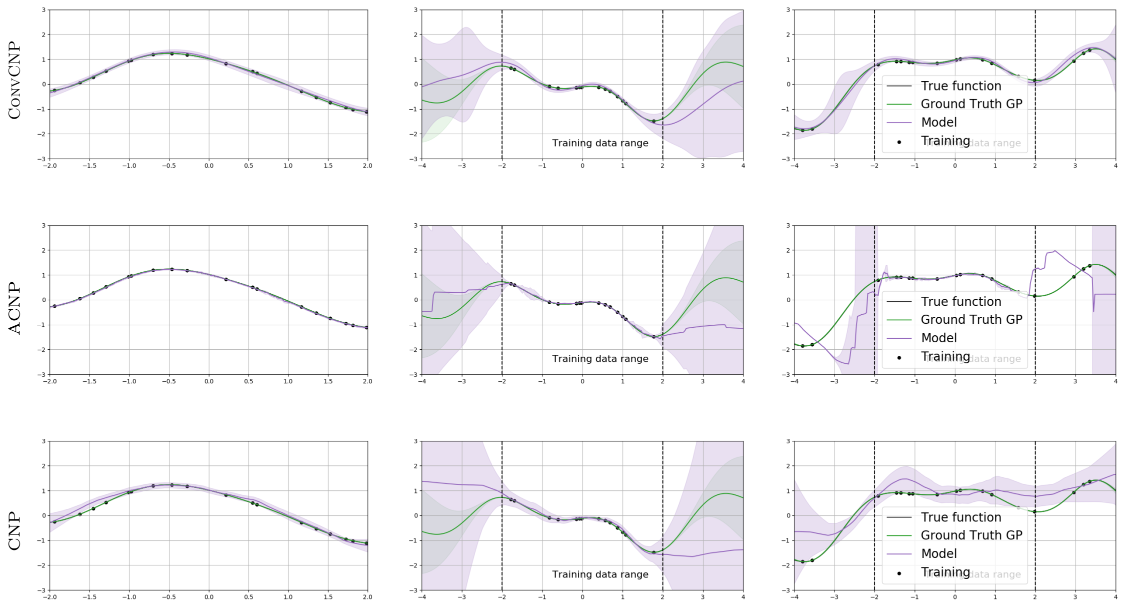


Fig. A.1 Example functions learned by the (top) ConvCNP, (center) ACNP, and (bottom) CNP when trained on an EQ kernel (with length scale parameter 1). “True function” refers to the sample from the GP prior from which the context and target sets were sub-sampled. “Ground Truth GP” refers to the GP posterior distribution when using the exact kernel and performing posterior inference based on the context set. The left column shows the predictive posterior of the models when data is presented in same range as training. The centre column shows the model predicting outside the training data range when no data is observed there. The right-most column shows the model predictive posteriors when presented with data outside the training data range.

The kernels used for the Gaussian Processes which generate the data in this experiment are defined as follows:

- EQ:

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

- weakly periodic:

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

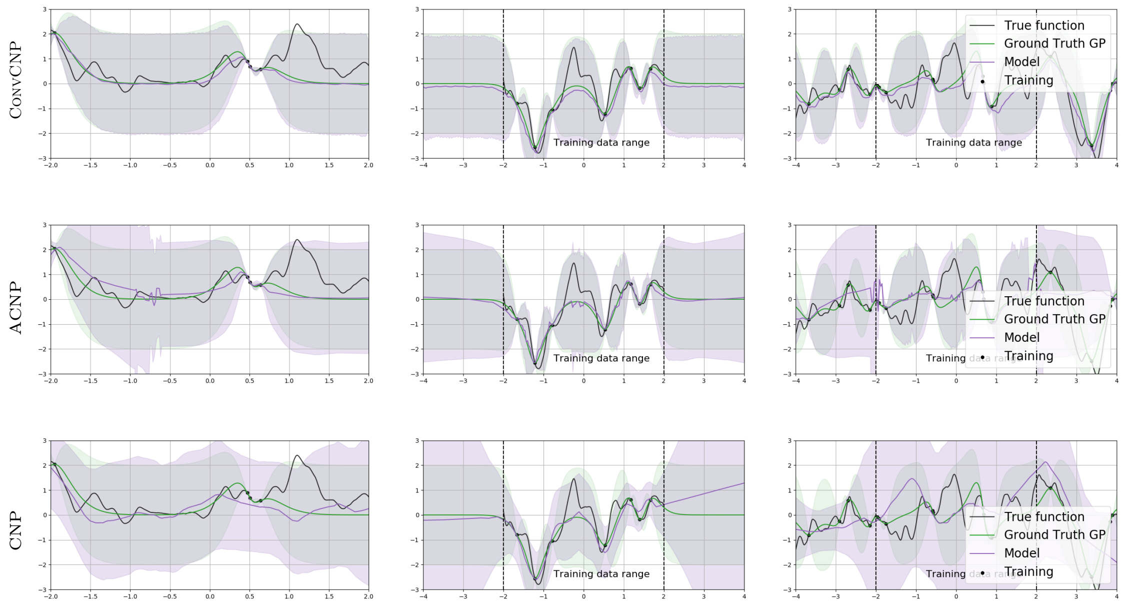


Fig. A.2 Example functions learned by the (top) ConvCNP, (center) ACNP, and (bottom) CNP when trained on a Matérn-5/2 kernel (with length scale parameter 0.25). "True function" refers to the sample from the GP prior from which the context and target sets were subsampled. "Ground Truth GP" refers to the GP posterior distribution when using the exact kernel and performing posterior inference based on the context set. The left column shows the predictive posterior of the models when data is presented in same range as training. The centre column shows the model predicting outside the training data range when no data is observed there. The right-most column shows the model predictive posteriors when presented with data outside the training data range.

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

- Matern–$\frac{5}{2}$:

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

with $d = 4|x - x'|$.

During the training procedure, the number of context points and target points for a training batch are each selected randomly from a uniform distribution over the integers between 3 and 50. This number of context and target points are randomly sampled from a function sampled from the process (a Gaussian process with one of the above kernels or the sawtooth process), where input locations are uniformly sampled from the interval $[-2, 2]$. All models in this experiment were trained for 200 epochs using 256 batches per epoch of batch size 16. We discretize $E(Z)$ by evaluating 64 points per unit in this setting. We use a learning rate of 3e−4 for all models, except for ConvCNPXL

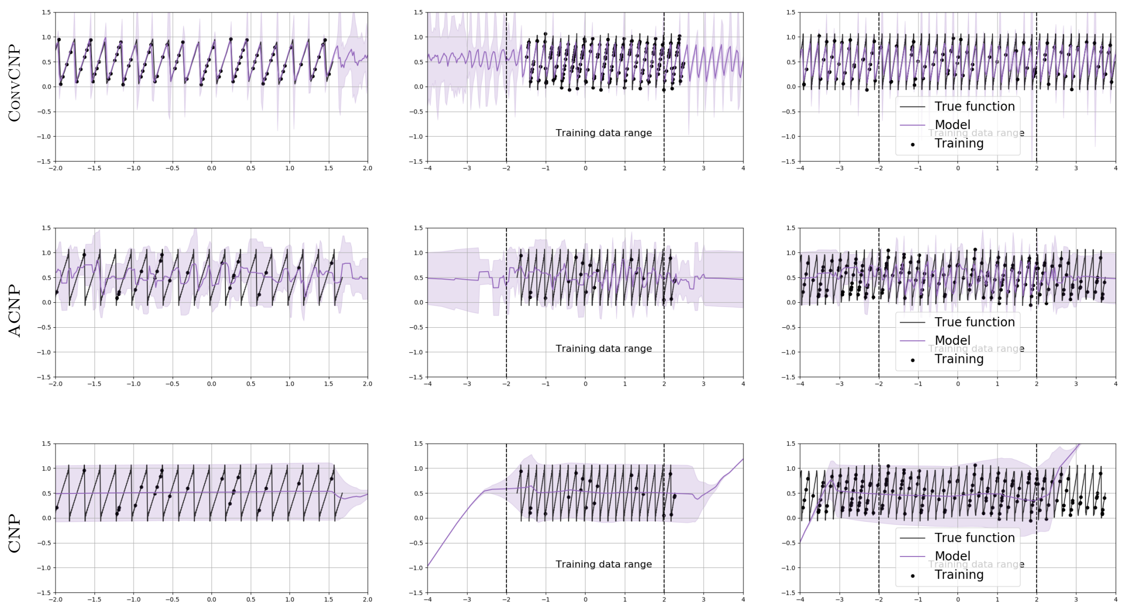


Fig. A.3 Example functions learned by the (top) ConvCNP, (center) ACNP, and (bottom) CNP when trained on a random sawtooth sample. The left column shows the predictive posterior of the models when data is presented in the same range as training. The centre column shows the model predicting outside the training data range when no data is observed there. The right-most column shows the model predictive posteriors when presented with data outside the training data range.

on the sawtooth data, where we use a learning rate of 1e−3 (this learning rate was too large for the other models).

The random sawtooth samples are generated from the following function:

$$y_{\text{sawtooth}}(t) = \frac{A}{2} - \frac{A}{\pi}\sum_{k=1}^{\infty}(-1)^k \frac{\sin(2\pi k f t)}{k}, \tag{A.5}$$

where A is the amplitude, f is the frequency, and t is "time". Throughout training, we fix the amplitude to be one. We truncate the series at an integer K. At every iteration, we sample a frequency uniformly in $[3, 5]$, K in $[10, 20]$, and a random shift in $[-5, 5]$. As the task is much harder, we sample context and target set sizes over $[3, 100]$. Here the CNP and ACNP employ learning rates of 10^{-3}. All other hyperparameters remain unchanged.

We include additional figures showing the performance of ConvCNPs, ACNPs and CNPs on GP and sawtooth function regression tasks in Figures A.1 to A.3.

Variable	m	s
`time`	5.94×10^4	8.74×10^2
`lsstu`	1.26	1.63×10^2
`lsstg`	-0.13	3.84×10^2
`lsstr`	3.73	3.41×10^2
`lssti`	5.53	2.85×10^2
`lsstz`	6.43	2.69×10^2
`lssty`	6.27	2.93×10^2

Table A.1 Values used to normalise the data in the PLAsTiCC experiments.

A.3.3 PLAsTiCC Experimental Details

The ConvCNP was trained for 200 epochs using 1024 batches of batch size 4 per epoch. For training and testing, the number of context points for a batch are each selected randomly from a uniform distribution over the integers between 1 and the number of points available in the series (usually between 10–30 per bandwidth). The remaining points in the series are used as the target set. For testing, a batch size of 1 was used and statistics were computed over 1000 evaluations. We compare ConvCNP to the GP models used in (Boone, 2019) using the implementation in https://github.com/kboone/avocado. The data used for training and testing is normalized according to $t(v) = (v - m)/s$ with the values in Table A.1. These values are estimated from a batch sampled from the training data. To remove outliers in the GP results, log-likelihood values less than -10 are removed from the evaluation. These same datapoints were removed from the ConvCNP results as well.

For this dataset, we only used the ConvCNPXL, as we found the ConvCNP to underfit. The learning rate was set to 10^{-3}, and we discretize $E(Z)$ by evaluating 256 points per unit.

A.3.4 Predator–Prey Experimental Details

We describe the way simulated training data for the experiment in Section 3.5.3 was generated from the Lotka–Volterra model. The description is borrowed from (Wilkinson, 2011).

Let X be the number of predators and Y the number of prey at any point in our simulation. According to the model, one of the following four events can occur:

A: A single predator is born according to rate $\theta_1 XY$, increasing X by one.

B: A single predator dies according to rate $\theta_2 X$, decreasing X by one.

C: A single prey is born according to rate $\theta_3 Y$, increasing Y by one.

D: A single prey dies (is eaten) according to rate $\theta_4 XY$, decreasing Y by one.

The parameter values θ_1, θ_2, θ_3, and θ_4, as well as the initial values of X and Y govern the behavior of the simulation. We choose $\theta_1 = 0.01$, $\theta_2 = 0.5$, $\theta_3 = 1$, and $\theta_4 = 0.01$, which are also used in (Papamakarios and Murray, 2016) and generate reasonable time series. Note that these are likely not the parameter values that would be estimated from the Hudson's Bay lynx–hare data set (Leigh, 1968), but they are used because they yield reasonably oscillating time series. Obtaining oscillating time series from the simulation is sensitive to the choice of parameters and many parametrizations result in populations that simply die out.

Time series are simulated using Gillespie's algorithm (Gillespie, 1977):

1. Draw the time to the next event from an exponential distribution with rate equal to the total rate $\theta_1 XY + \theta_2 X + \theta_3 Y + \theta_4 XY$.
2. Select one of the above events A, B, C, or D at random with probability proportional to its rate.
3. Adjust the appropriate population according to the selected event, and go to 1.

The simulations using these parameter settings can yield a maximum population of approximately 300 while the context set in the lynx–hare data set has an approximate maximum population of about 80 so we scaled our simulation population by a factor of 2/7. We also remove time series which are longer than 100 units of time, which have more than 10000 events, or where one of the populations is entirely zero. The number of context points n for a training batch are each selected randomly from a uniform distribution between 3 and 80, and the number of target points is $150 - n$. These target and context points are then sampled from the simulated series. The Hudson's Bay lynx–hare data set has time values that range from 1845 to 1935. However, the values supplied to the model range from 0 to 90 to remain consistent with the simulated data.

For evaluation, an interval of 18 points is removed from the the Hudson's Bay lynx–hare data set to act as a target set, while the remaining 72 points act as the context set. This construction highlights the model's interpolation as well as its uncertainty in the presence of missing data.

Models in this setting were trained for 200 epochs with 256 batches per epoch, each batch containing 50 tasks. For this data set, we only used the ConvCNP, as we found the ConvCNPXL to overfit. The learning rate was set to 10^{-3}, and we discretize $E(Z)$ by evaluating 100 points per unit.

A.4 Image Experimental Details and Additional Results

A.4.1 Experimental Details

Training details. In all experiments, we sample the number of context points uniformly from $\mathcal{U}(\frac{n_{\text{total}}}{100}, \frac{n_{\text{total}}}{2})$, and the number of target points is set to n_{total}. The context and target points are sampled randomly from each of the 16 images per batch. The weights are optimised using Adam (Kingma and Ba, 2015) with learning rate 5×10^{-4}. We use a maximum of 100 epochs, with early stopping of 15 epochs patience. All pixel values are divided by 255 to rescale them to the $[0, 1]$ range. In the following discussion, we assume that images are RGB, but very similar models can be used for greyscale images or other gridded inputs (e.g. 1d time series sampled at uniform intervals).

Proposed convolutional CNP. Unlike ACNP and off-the-grid ConvCNP, on-the-grid ConvCNP takes advantage of the gridded structure. Namely, the target and context points can be specified in terms of the image, a context mask M_c, and a target mask M_t instead of sets of input–value pairs. Although this is an equivalent formulation, it makes it more natural and simpler to implement in standard deep learning libraries. In the following, we dissect the architecture and algorithmic steps succinctly summarized in Section 3.4. Note that all the convolutional layers are actually depthwise separable (Chollet, 2017); this enables a large kernel size (i.e. receptive fields) while being parameter and computationally efficient.

1. Let I denote the image. Select all context points $\text{signal} := \mathrm{M}_c \odot \mathrm{I}$ and append a density channel $\text{density} := \mathrm{M}_c$, which intuitively says that "there is a point at this position": $[\text{signal}, \text{density}]^\top$. Each pixel value will now have 4 channels: 3 RGB channels and 1 density channel M_c. Note that the mask will set the pixel value to 0 at a location where the density channel is 0, indicating there are no points at this position (a missing value).

2. Apply a convolution to the density channel $\text{density}' = \textsc{conv}_{\boldsymbol{\theta}}(\text{density})$ and a normalized convolution to the signal $\text{signal}' := \textsc{conv}_{\boldsymbol{\theta}}(\text{signal})/\text{density}'$. The normalized convolution makes sure that the output mostly depends on the scale of the signal rather than the number of observed points. The output channel size is 128 dimensional. The kernel size of $\textsc{conv}_{\boldsymbol{\theta}}$ depends on the image shape and model used (Table A.2). We also enforce element-wise positivity of the trainable

filter by taking the absolute value of the kernel weights $\boldsymbol{\theta}$ before applying the convolution. As discussed in Section A.4.4, the normalization and positivity constraints do not empirically lead to improvements for on-the-grid data. Note that in this setting, $E(Z)$ is $[\text{signal}', \text{density}']^\top$.

3. Now we describe the on-the-grid version of $\rho(\cdot)$, which we decompose into two stages. In the first stage, we apply a CNN to $[\text{signal}', \text{density}']^\top$. This CNN is composed of residual blocks (He et al., 2016), each consisting of 1 or 2 (Table A.2) convolutional layers with ReLU activations and no batch normalization. The number of output channels in each layer is 128. The kernel size is the same across the whole network, but depends on the image shape and model used (Table A.2).

4. In the second stage of $\rho(\cdot)$, we apply a shared pointwise MLP : $\mathbb{R}^{128} \to \mathbb{R}^{2C}$ (we use the same architecture as used for the ACNP decoder) to the output of the first stage at each pixel location in the target set. Here C denotes the number of channels in the image. The first C outputs of the MLP are treated as the means of a Gaussian predictive distribution, and the last C outputs are treated as the standard deviations. These then pass through the positivity-enforcing function shown in Equation (A.4).

Model	Input Shape	CONV$_{\boldsymbol{\theta}}$ Kernel Size	CNN Kernel Size	CNN Num. Res. Blocks	Conv. Layers per Block
ConvCNP	$<$ 50 pixels	9	5	4	1
	$>$ 50 pixels	7	3	4	1
ConvCNP XL	any	9	11	6	2

Table A.2 CNN architecture for the image experiments.

A.4.2 Zero Shot Multi MNIST (ZSMM) data

In the real world, it is very common to have multiple objects in our field of view which do not interact with each other. Yet, many image data sets in machine learning contain only a single, well-centered object. To evaluate the translation equivariance and generalization capabilities of our model, we introduce the zero-shot multi-MNIST setting.

The training set contains all 60000 28×28 MNIST training digits centered on a black 56×56 background. (Figure A.4a). For the test set, we randomly sample with

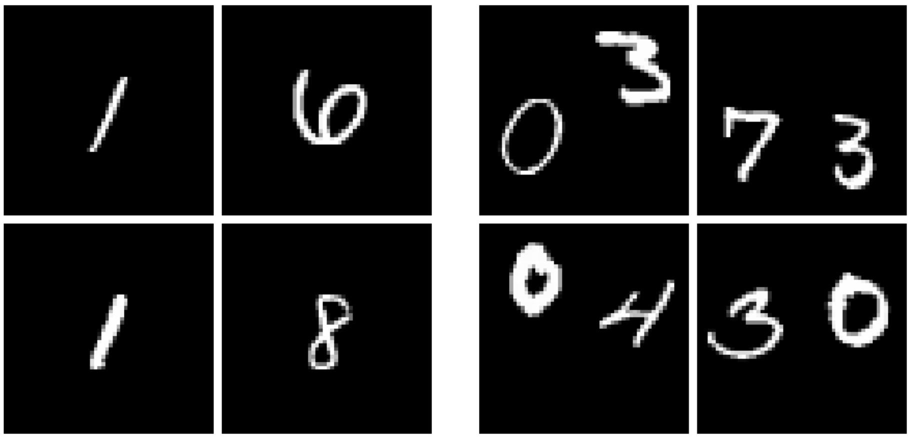

(a) Train (b) Test

Fig. A.4 Samples from our generated Zero Shot Multi MNIST (ZSMM) data set.

replacement 10000 pairs of digits from the MNIST test set, place them on a black 56×56 background, and translate the digits in such a way that the digits can be arbitrarily close but cannot overlap (Figure A.4b). Importantly, the scale of the digits and the image size are the same during training and testing.

A.4.3 ACNP and ConvCNP Qualitative Comparison

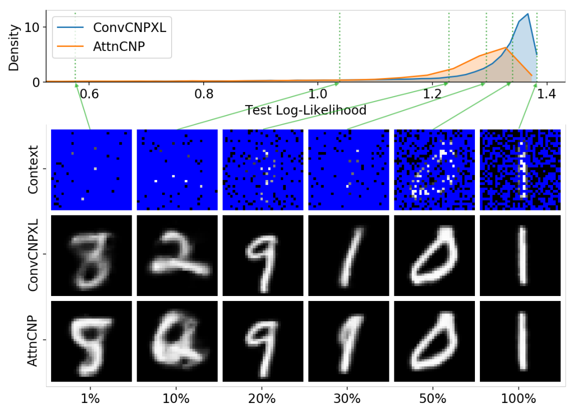
Figure A.5 shows the test log-likelihood distributions of an ACNP and ConvCNP model as well as some qualitative comparisons between the two.

Although most mean predictions of both models look relatively similar for SVHN and CelebA32, the real advantage of ConvCNP becomes apparent when testing the generalization capacity of both models. Figure A.6 shows ConvCNP and ACNP trained on CelebA32 and tested on a downscaled version of Ellen's famous Oscar selfie. We see that ConvCNP generalizes better in this setting. [2]

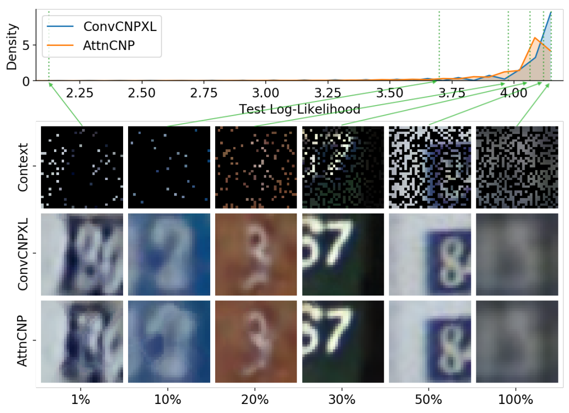
A.4.4 Ablation Study: First Layer

To understand the importance of the different components of the first layer, we performed an ablation study by removing the density normalization (ConvCNP no norm.), removing the density channel (ConvCNP no dens.), removing the positivity

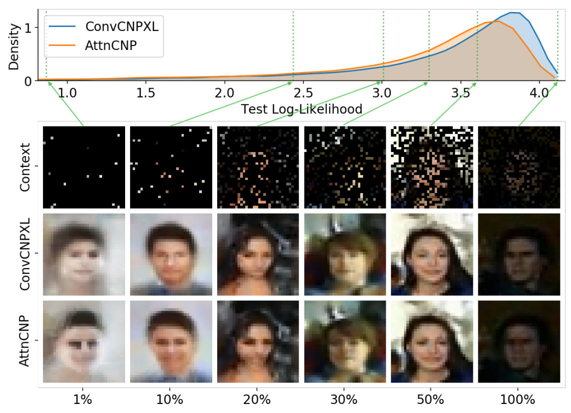
[2]The reconstruction looks worse than Figure 3.5b despite the larger context set, because the test image has been downscaled and the models are trained on a low resolution CelebA32. These constraints come from ACNP's large memory footprint.



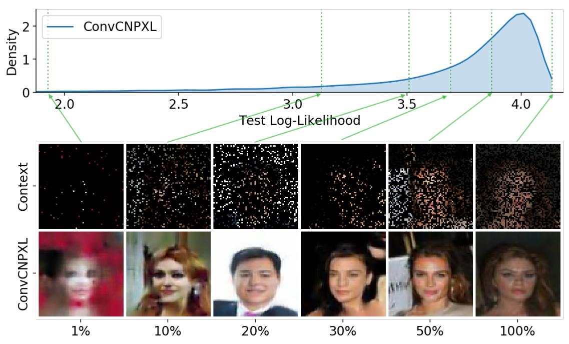
(a) MNIST



(b) SVHN



(c) CelebA 32 × 32



(d) CelebA 64 × 64

Fig. A.5 Log-likelihood and qualitative comparisons between ACNP and ConvCNP on four standard benchmarks. The top row shows the log-likelihood distribution for both models. The images below correspond to the context points (top), ConvCNP target predictions (middle), and ACNP target predictions (bottom). Each column corresponds to a given percentile of the ConvCNP distribution. ACNP could not be trained on CelebA64 due to its memory inefficiency.

Model	MNIST	SVHN	CelebA32	CelebA64	ZSMM
ConvCNP	1.19 ±0.01	3.89 ±0.01	3.19 ±0.02	3.64 ±0.01	1.21 ±0.00
. . . no density	1.15 ±0.01	3.88 ±0.01	3.15 ±0.02	3.62 ±0.01	1.13 ±0.08
. . . no norm.	1.19 ±0.01	3.86 ±0.03	3.16 ±0.03	3.62 ±0.01	1.20 ±0.01
. . . no abs.	1.15 ±0.02	3.83 ±0.02	3.08 ±0.03	3.56 ±0.01	1.15 ±0.01
. . . no abs. norm.	1.19 ±0.01	3.86 ±0.03	3.16 ±0.03	3.62 ±0.01	1.20 ±0.01
. . . EQ	1.18 ±0.00	3.89 ±0.01	3.18 ±0.02	3.63 ±0.01	1.21 ±0.00

Table A.3 Log-likelihood from image ablation experiments (6 runs).

constraints (ConvCNP no abs.), removing the positivity constraints and the normal-

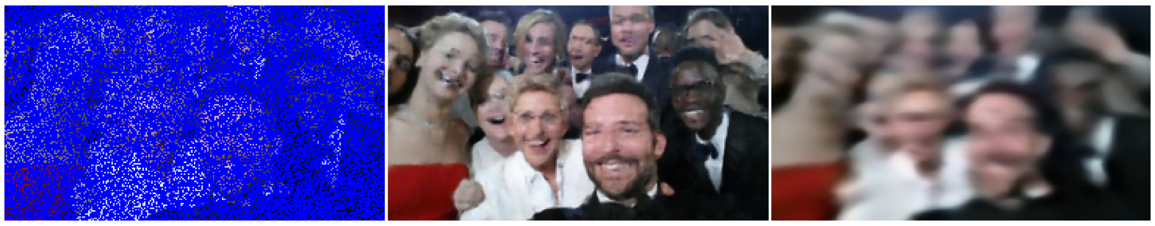


Fig. A.6 Qualitative evaluation of a ConvCNP (center) and ACNP (right) trained on CelebA32 and tested on a downscaled version (146×259) of Ellen's Oscar selfie (DeGeneres, 2014) with 20% of the pixels as context (left).

ization (ConvCNP no abs. norm.), and replacing the fully trainable first layer by an EQ kernel similar to the continuous case (ConvCNP EQ).

Table A.3 shows the following: (i) Appending a density channel helps. (ii) Enforcing the positivity constraint is only important when using a normalized convolution. (iii) Using a less expressive EQ filter does not significantly decrease performance, suggesting that the model might be learning similar filters (Section A.4.5).

A.4.5 Qualitative Analysis of the First Filter

As discussed in Section A.4.4, using a less expressive EQ filter does not significantly decrease performance. Figure A.7 shows that this happens because the fully trainable kernel learns to approximate the EQ filter.

A.4.6 Effect of Receptive Field on Translation Equivariance

As seen in Table 3.3, a ConvCNPXL with large receptive field performs significantly worse on the ZSMM task than ConvCNP, which has a smaller receptive field. Figure A.8 shows a more detailed comparison of the models, and suggests that ConvCNPXL learns to model non-stationary behaviour, namely that digits in the training set are centred. We hypothesize that this issue stems from the the treatment of the image boundaries. Indeed, if the receptive field is large enough and the padding values are significantly different than the inputs to each convolutional layer, the model can learn position-dependent behaviour by "looking" at the distance from the padded boundaries.

For ZSMM, Figure A.9 suggests that "circular" padding, where the padding is implied by tiling the image, helps prevent the model from learning non-stationarities, even as the size of the receptive field becomes larger. We hypothesize that this is due to the fact that "circularly" padded values are harder to distinguish from actual values

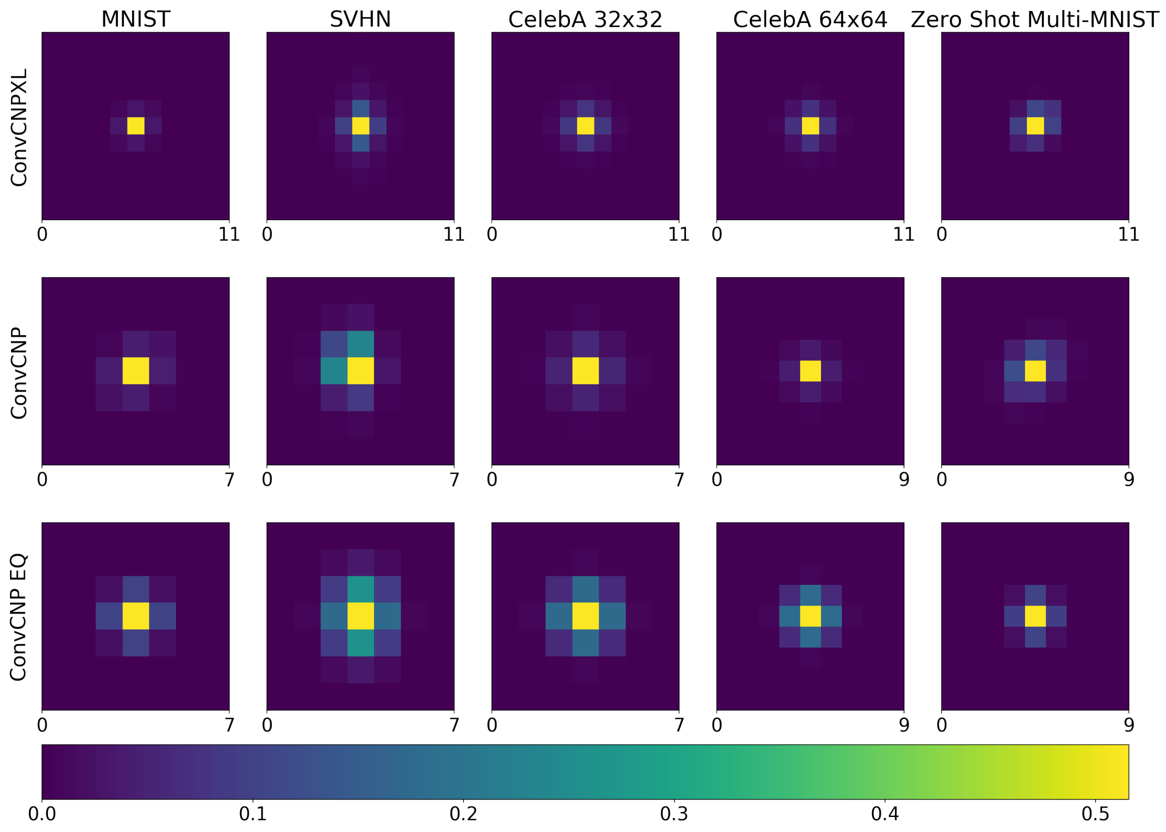


Fig. A.7 First filter learned by ConvCNPXL, ConvCNP, and ConvCNP EQ for all our datasets. In the case of RGB images, the plotted filters are for the first channel (red). Note that not all filters are of the same size.

than zeros. We have not tested the effect of padding on other datasets, and note that "circular" padding could result in other issues.

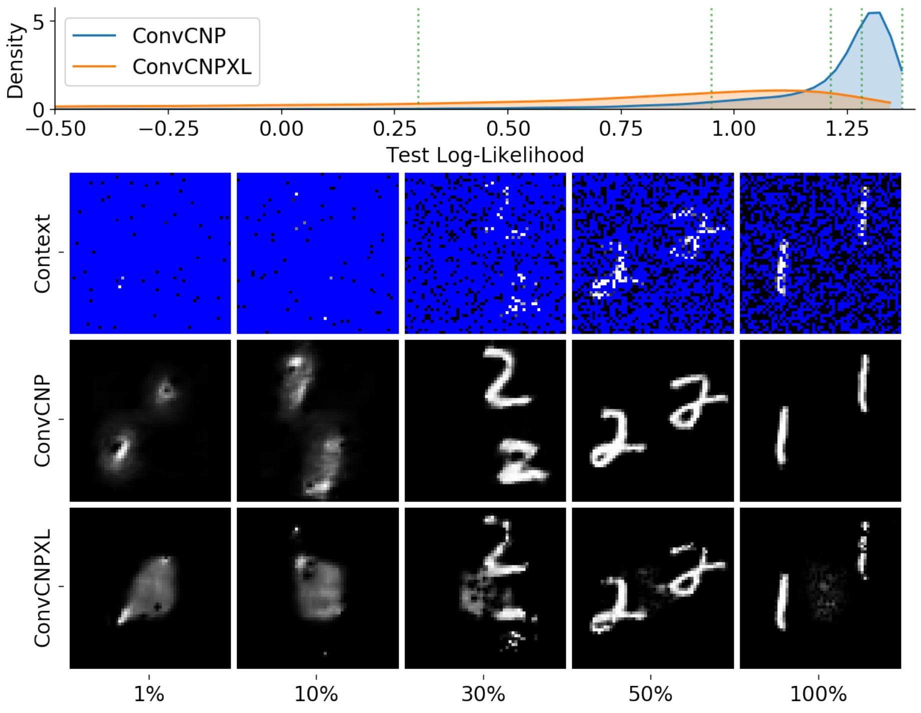


Fig. A.8 Log-likelihood and qualitative results on ZSMM. The top row shows the log-likelihood distribution for both models. The images below correspond to the context points (top), ConvCNP target predictions (middle), and ConvCNPXL target predictions (bottom). Each column corresponds to a given percentile of the ConvCNP distribution.

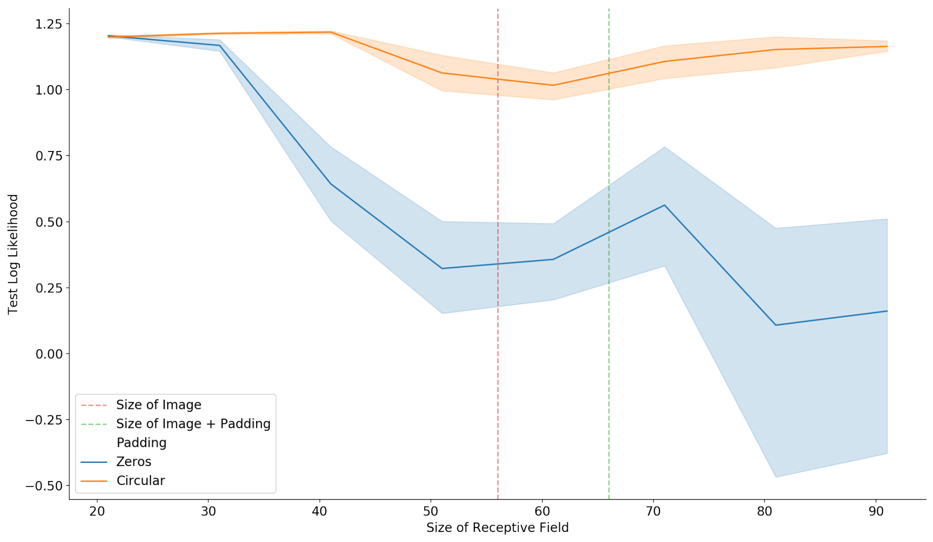


Fig. A.9 Effect of the receptive field size on ZSMM's log-likelihood. The line plot shows the mean and standard deviation over 6 runs. The blue curve corresponds to a model with zero padding, while the orange one corresponds to “circular” padding.

Appendix B

Chapter 4 Supplementary Material

B.1 Formal Definitions and Set-up

Notation. We first review the notation introduced in the main body for convenience. Let $\mathcal{X} = \mathbb{R}^{d_{\text{in}}}$ and $\mathcal{Y} = \mathbb{R}$ denote the input and output spaces respectively, and let $(\boldsymbol{x}, y)$ denote a generic input-output pair (higher-dimensional outputs can be treated easily). Define $\mathcal{S}_N = (\mathcal{X} \times \mathcal{Y})^N$ to be the collection of all data sets of size N, and let $\mathcal{S} := \bigcup_{N=1}^{\infty} \mathcal{S}_N$. Let $D^{(c)}, D^{(t)} \in \mathcal{S}$ denote a *context* and *target* set respectively. Later, as is common in recent meta-learning approaches, we will consider predicting the target set from the context set Garnelo et al. (2018a,b). Let $\boldsymbol{x}^{(c)} = (x_1, \ldots, x_N)$ denote a collection (matrix) of context set inputs, with $\boldsymbol{y}^{(c)} = (y_1, \ldots, y_N)$ the corresponding outputs; $\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}$ are defined analogously. We denote a single *task* as $\xi = (D^{(c)}, D^{(t)}) = (D^{(c)}, (\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}))$.

Stochastic processes. For our purposes, a stochastic process on $\mathcal{X}$ will be defined as a probability measure on the set of functions from $\mathcal{X} \to \mathbb{R}$, i.e. $\mathbb{R}^{\mathcal{X}}$, equipped with the product σ-algebra of the Borel σ-algebra over each index point (Tao, 2011), denoted Σ. The measurable sets of Σ are those which can be specified by the values of the function at a countable subset $I \subset \mathcal{X}$ of its input locations. Since in practice we only ever observe data at a finite number of points, this is sufficient for our purposes. We denote the set of all such measures as $\mathcal{P}(\mathcal{X})$. We model the world as having a ground truth stochastic process $P \in \mathcal{P}(\mathcal{X})$. Consider a Kolmogorov-consistent collection of distributions on finite index sets $I \subset \mathcal{X}$. By the Kolmogorov extension theorem, there exists a unique measure on $(\mathbb{R}^{\mathcal{X}}, \Sigma)$ that has these distributions as its finite marginals. Hence we may think of these stochastic processes as defined by their finite-dimensional marginals.

B.2 Stationary Processes and Translation Equivariance

Definition 5 (Translating data sets and SPs)**.** We define the action of the translation operator $T_{\boldsymbol{\tau}}$ on data sets and SPs, where $\boldsymbol{\tau} \in \mathcal{X}$ denotes the shift vector of the translation.[1]

1. Let $(x_n, y_n)_{n=1}^N = S \in \mathcal{S}$. For the index set $\boldsymbol{x} = (x_1, \ldots, x_n)$, the translation by $\boldsymbol{\tau}$ is defined as $T_{\boldsymbol{\tau}}\boldsymbol{x} = (x_1 + \boldsymbol{\tau}, \ldots, x_n + \boldsymbol{\tau})$. Similarly, $T_{\boldsymbol{\tau}}S \coloneqq (x_n + \boldsymbol{\tau}, y_n)_{n=1}^N$.
2. For a function $f \in \mathbb{R}^{\mathcal{X}}$, define $T_{\boldsymbol{\tau}}f(x) \coloneqq f(x - \boldsymbol{\tau})$ for all $x \in \mathcal{X}$. Let $F \in \Sigma$ be a measurable set of functions. Then $T_{\boldsymbol{\tau}}F \coloneqq \{T_{\boldsymbol{\tau}}f : f \in F\}$.
3. For any SP $P \in \mathcal{P}(\mathcal{X})$, we now define $T_{\boldsymbol{\tau}}P$ by setting[2] $T_{\boldsymbol{\tau}}P(F) \coloneqq P(T_{-\boldsymbol{\tau}}F)$ for all $F \in \Sigma$.

Definition 6 (Stationary SP)**.** We say a stochastic process is (strictly) *stationary* if the densities of its finite marginals satisfy

$$p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}) = p(\boldsymbol{y}^{(t)}|T_{\boldsymbol{\tau}}\boldsymbol{x}^{(t)}) \tag{B.1}$$

for all $\boldsymbol{y}^{(t)}$, $\boldsymbol{x}^{(t)}$ and $\boldsymbol{\tau}$.

Definition 7 (Translation equivariant prediction maps)**.** We say that $\Psi \colon \mathcal{S} \to \mathcal{P}(\mathcal{X})$ is *translation equivariant* if $\Psi(T_{\boldsymbol{\tau}}S) = T_{\boldsymbol{\tau}}\Psi(S)$ for any data set $S \in \mathcal{S}$ and shift $\boldsymbol{\tau} \in \mathcal{X}$.

The following simple statement highlights the link between stationarity and translation equivariance:

Proposition 9. *Let P be a stationary SP. Then the prediction map π_P is translation equivariant.*[3]

Proof. Let $p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, D^{(c)})$ denote the finite dimensional density of $\pi_P(D^{(c)})$ at index set $\boldsymbol{x}^{(t)}$. To show that $\pi_P(T_{\boldsymbol{\tau}}D^{(c)}) = T_{\boldsymbol{\tau}}\pi_P(D^{(c)})$ it suffices to show that $p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, T_{\boldsymbol{\tau}}D^{(c)}) =$

[1]To prevent notational clutter, the same symbol, $T_{\boldsymbol{\tau}}$, will denote translations on multiple kinds of objects.

[2]This is well-defined since Σ is closed under translations. Equivalently, we could define $T_{\boldsymbol{\tau}}P$ as the push-forward of P under the the translation map on functions, $T_{\boldsymbol{\tau}} : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^{\mathcal{X}}$.

[3]We exclude conditioning on observations that have zero density, so that the prediction map is well defined.

$p(\boldsymbol{y}^{(t)}|T_{-\boldsymbol{\tau}}\boldsymbol{x}^{(t)}, D^{(c)})$. We have

$$\begin{aligned}
p(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(t)}, T_{\boldsymbol{\tau}}D^{(c)}) &= \frac{p(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)}|\boldsymbol{X}^{(t)}, T_{\boldsymbol{\tau}}\boldsymbol{x}^{(c)})}{p(\boldsymbol{y}^{(c)}|T_{\boldsymbol{\tau}}\boldsymbol{x}^{(c)})} \\
&= \frac{p(\boldsymbol{y}^{(t)}, \boldsymbol{y}^{(c)}|T_{-\boldsymbol{\tau}}\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(c)})}{p(\boldsymbol{y}^{(c)}|\boldsymbol{x}^{(c)})} \\
&= p(\boldsymbol{y}^{(t)}|T_{-\boldsymbol{\tau}}\boldsymbol{x}^{(t)}, D^{(c)}),
\end{aligned}$$

where we used the stationarity assumption in the second line. □

B.3 Translation Equivariance of the ConvLNP

We prove that the ConvLNP is a translation equivariant map from data sets to stochastic processes, by proving that the decoder and encoder are separately translation equivariant. In this section we suppress the dependence on parameters $(\boldsymbol{\phi}, \boldsymbol{\theta})$.

Lemma 1. Let d be a measurable, translation equivariant map from $(\mathbb{R}^{\mathcal{X}}, \Sigma)$ to $(\mathbb{R}^{\mathcal{X}}, \Sigma)$. The ConvLNP decoder $\mathrm{D} : \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{X})$, defined by $\mathrm{D}(P) = d_*(P)$, where $d_*(P)$ is the pushforward measure under d, is translation equivariant.

Proof. Let $F \in \Sigma$ be measurable. Then:

$$\begin{aligned}
\mathrm{D}(T_{\boldsymbol{\tau}}P)(F) &\overset{\text{(a)}}{=} T_{\boldsymbol{\tau}}P(d^{-1}(F)) \\
&= P(T_{-\boldsymbol{\tau}}d^{-1}(F)) \\
&\overset{\text{(b)}}{=} P(d^{-1}(T_{-\boldsymbol{\tau}}F)) \\
&= \mathrm{D}(P)(T_{-\boldsymbol{\tau}}F) \\
&= T_{\boldsymbol{\tau}}\mathrm{D}(P)(F).
\end{aligned}$$

Here (a) follows from definition of the pushforward, and (b) follows because

$$\begin{aligned}
T_{-\boldsymbol{\tau}} d^{-1}(F) &= T_{-\boldsymbol{\tau}}\{f : d(f) \in F\} \\
&= \{T_{-\boldsymbol{\tau}} f : d(f) \in F\} \\
&= \{f : d(T_{\boldsymbol{\tau}} f) \in F\} \\
&= \{f : T_{\boldsymbol{\tau}} d(f) \in F\} \\
&= \{f : d(f) \in T_{-\boldsymbol{\tau}} F\} \\
&= d^{-1}(T_{-\boldsymbol{\tau}} F).
\end{aligned}$$

□

Lemma 2. The ConvLNP encoder E (a ConvCNP), is a translation equivariant map from data sets to stochastic processes.

Proof. Recall that the mean and variance $\mu(\cdot, S), \sigma^2(\cdot, S)$ (viewed as maps from $\mathcal{S} \to C_b(\mathcal{X})$) of the encoder E are both given by ConvDeepSets. Due to the translation equivariance of ConvDeepSets (Gordon et al., 2020, Theorem 1), $\mu(\cdot, T_{\boldsymbol{\tau}} S) = T_{\boldsymbol{\tau}} \mu(\cdot, S)$ for all $S, \boldsymbol{\tau}$, and similarly for σ^2. Let $F \in \Sigma$. Then since the measure $\mathrm{E}(S) \in \mathcal{P}_{\mathrm{N}}(\mathcal{X})$ is defined entirely by its mean and variance function, $\mathrm{E}(T_{\boldsymbol{\tau}} S)(F) = \mathrm{E}(S)(T_{-\boldsymbol{\tau}} F) = T_{\boldsymbol{\tau}} \mathrm{E}(S)(F)$. □

Noting that a composition of translation equivariant maps is itself translation equivariant, we obtain the following proposition:

Proposition 10. Define ConvLNP = D∘E. Then ConvLNP is a translation equivariant map from data sets to stochastic processes.

B.4 Experimental Details on 1D Regression

For the full results of the 1D regression tasks, see Section B.5. Code to reproduce the 1D regression experiments can be found at https://github.com/wesselb/NeuralProcesses.jl.

In the 1D regression experiments, we consider the following generative processes:

EQ: samples from a Gaussian process with the following exponentiated-quadratic kernel:

$$k(t, t') = \exp\left(-\frac{1}{8}(t - t')^2\right);$$

Matérn–$\frac{5}{2}$: samples from a Gaussian process with the following Matérn–$\frac{5}{2}$ kernel:

$$k(t,t') = \left(1 + 4\sqrt{5}d + \frac{5}{3}d^2\right)\exp\left(-\sqrt{5}d\right)$$

with $d = 4|x - x'|$;

noisy mixture: samples from a Gaussian process with the following noisy mixture kernel:

$$k(t,t') = \exp\left(-\frac{1}{8}(t-t')^2\right) + \exp\left(-\frac{1}{2}(t-t')^2\right) + 10^{-3}\delta[t-t'];$$

weakly periodic: samples from a Gaussian process with the following weakly-periodic kernel:

$$k(t,t') = \exp\left(-\frac{1}{2}(f_1(t) - f_1(t'))^2 - \frac{1}{2}(f_2(t) - f_2(t'))^2 - \frac{1}{8}(t-t')^2\right)$$

with $f_1(t) = \cos(8\pi t)$ and $f_2(t) = \sin(8\pi t)$; and

sawtooth: samples from the following sawtooth process:

$$f(t) = \frac{A}{2} - \frac{A}{\pi}\sum_{k=1}^{K}(-1)^k \frac{\sin(2\pi k f(t-s))}{k}$$

with $A = 1$, $f \sim \mathcal{U}[3,5]$, $s \sim \mathcal{U}[-5,5]$, and $K \in \{10,\ldots,20\}$ chosen uniformly.

We compare the following models, where all activation functions are leaky ReLUs with leak 0.1:

ConvCNP: The first model is the ConvCNP. The architecture of the ConvCNP is equal to that of the encoder in the ConvLNP, described next.

ConvLNP: The second model is the ConvLNP as described in the main body. The functional embedding uses separate length scales for the data channel and density channel (Figure 4.2), which are initialized to twice the inter-point spacing of the discretization and learned during training. The discretization uniformly ranges over $[\min(x) - 1, \max(x) + 1]$ at density $\rho = 64$ points per unit, where $\min(x)$ is the minimum x value occurring in the union of the context and target sets in the current batch and $\max(x)$ is corresponding maximum x

value. The discretization is passed through a 10-layer (excluding an initial and final point-wise linear layer) CNN with 64 channels and depthwise-separable convolutions. The width of the filters depends on the data set and is chosen such that the receptive field sizes are as follows:

EQ:	2,
Matérn–$\frac{5}{2}$:	2,
noisy mixture:	4,
weakly periodic:	4,
sawtooth:	16.

The discretized functional representation consists of 16 channels. The smoothing at the end of the encoder also has separate length scales for the mean and variance which are initialized similarly and learned. The encoder parametrizes the standard deviations by passing the output of the CNN through a softplus. The decoder has the same architecture as the encoder.

ALNP: The third model is the Attentive NP with latent dimensionality $d = 128$ and 8-head dot-product attention (Vaswani et al., 2017). In the attentive deterministic encoder, the keys (t), queries (t), and values (concatenation of t and y) are transformed by a three-layer MLP of constant width d. The dot products are normalised by $\sqrt{d}$. The output of the attention mechanism is passed through a constant-width linear layer, which is then passed through two layers of layer normalization (Ba et al., 2016) to normalise the latent representation. In the first of these two layers, first the transformed queries are passed through a constant-width linear layer and added to the input. In the second of these two layers, the output of the first layer is first passed through a two-layer constant-width MLP and added to itself, making a residual layer. In the stochastic encoder, the inputs and outputs are concatenated and passed though a three-layer MLP of constant width d. The result is mean-pooled and passed through a two-layer constant-width MLP. The decoder consists of a three-layer MLP of constant width d.

	EQ	Matérn–$\frac{5}{2}$	Noisy Mixt.	Weakly Per.	Sawtooth
ConvCNP	42 822	42 822	51 014	51 014	100 166
ConvLNP	88 486	88 486	104 870	104 870	203 174
ALNP	530 178	530 178	530 178	530 178	530 178
LNP	479 874	479 874	479 874	479 874	479 874

Table B.1 Parameter counts for the ConvCNP, ConvLNP, ALNP, and LNP in the 1D regression tasks

NP: The fourth model is the original NP (Garnelo et al., 2018b). The architecture is similar to that of the ALNP, where the architecture of the deterministic encoder is replaced by that of the stochastic encoder.

For all models, positivity of the observation noise is enforced with a softplus function. Parameter counts of the ConvCNP, ConvLNP, ALNP, and NP are listed in Table B.1.

The models are trained with $\mathcal{L}_{\text{ML}}$ ($L = 20$) and $\mathcal{L}_{\text{LNP}}$ ($L = 5$). For $\mathcal{L}_{\text{LNP}}$, the context set is appended to the target set when evaluating the objective. The models are optimised using ADAM with learning rate $5 \cdot 10^{-3}$ for 100 epochs. One epoch consists of 2^{14} tasks divided into batches of size 16. For training, the inputs of the context and target sets are sampled uniformly from $[-2, 2]$. The size of the context set is sampled uniformly from $\{0, \dots, 50\}$ and the size of the target set is fixed to 50. To encourage the NP-based models—not the CNP-based models—to fit and not revert to their conditional variants, the observation noise standard deviation σ is held fixed to 10^{-2} for the first 20 epochs.

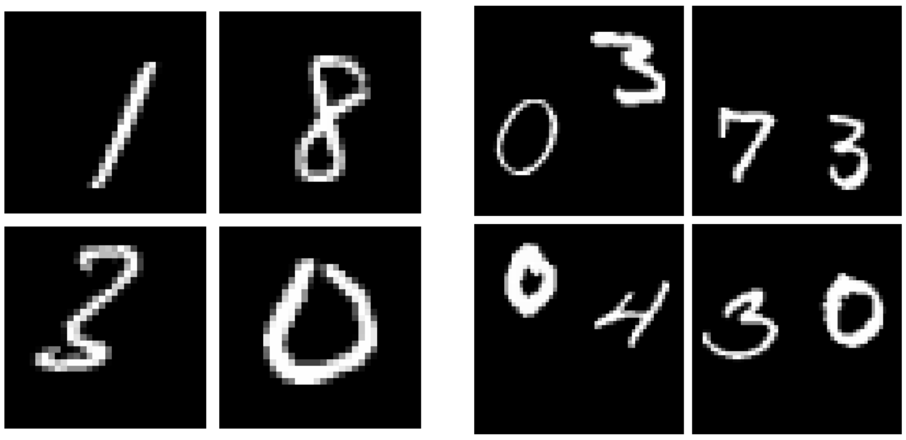
For evaluation, the size of the context set is sampled uniformly from $\{0, \dots, 10\}$, and the losses are evaluated with $L = 5000$ and batch size one. To test interpolation within the training range, the inputs of the context and target sets are, like training, sampled uniformly from $[-2, 2]$. To test interpolation beyond the training range, the inputs of the context and target sets are sampled uniformly from $[2, 6]$. To test extrapolation beyond the training range, the inputs of the context sets are sampled uniformly from $[-2, 2]$ and the inputs of the target sets are sampled uniformly from $[-4, -2] \cup [2, 4]$. As described in Section 4.4.4, models trained with $\mathcal{L}_{\text{LNP}}$ are evaluated using importance weighting to obtain a better estimate of the evaluation loss.

B.5 Additional Results on 1D Regression

Table B.2 presents results for all models with all losses on all data sets described in Section B.4 according to the evaluation protocol described in Sections B.4 and 4.4.4.

B.6 Experimental Details on Image Completion

B.6.1 Data Details



(a) Train (32×32) (b) Test (56×56)

Fig. B.1 Samples from our generated Zero Shot Multi MNIST (ZSMM) data set.

We use three standard data sets throughout our image experiments: SVHN (Netzer et al., 2011), MNIST LeCun et al. (1989), and 32×32 CelebA Netzer et al. (2011). The aforementioned standard data sets all contain only a single, well-centered object. To evaluate the translation equivariance and generalization capabilities of our model we evaluate on a Zero Shot Multi-MNIST (ZSMM) task, which is similar to ZSMM described in Appendix D.2 of Gordon et al. (2020). Namely, we generate a test set by randomly sampling with replacement 10000 pairs of digits from the MNIST test set, place them on a black 56×56 background, and translate the digits in such a way that the digits can be arbitrarily close but cannot overlap (Figure B.1b). The difference with the dataset from Gordon et al. (2020), is that the training set consists of the standard MNIST digits (instead of a single digit placed in the center of 56×56 canvas), augmented by up to 4 pixel shifts (Figure B.1a). The model thus has to generalize both to a larger canvas size as well as to seeing multiple digits.

		EQ	Matérn–$\frac{5}{2}$	Noisy Mixt.	Weakly Per.	Sawtooth
Interpolation inside training range						
GP (full)		5.80 ± 0.02	$1.22 \pm 6.3\mathrm{E}{-3}$	$1.00 \pm 4.1\mathrm{E}{-3}$	$-0.06 \pm 4.6\mathrm{E}{-3}$	N/A
GP (diag)		-0.59 ± 0.01	$-0.84 \pm 9.0\mathrm{E}{-3}$	-0.89 ± 0.01	$-1.17 \pm 5.2\mathrm{E}{-3}$	N/A
ConvCNP		-0.70 ± 0.02	-0.88 ± 0.01	-0.92 ± 0.02	$-1.19 \pm 7.0\mathrm{E}{-3}$	1.15 ± 0.04
ConvLNP	$\mathcal{L}_{\text{ML}}$	-0.30 ± 0.02	-0.58 ± 0.01	-0.55 ± 0.01	$-1.02 \pm 6.0\mathrm{E}{-3}$	2.30 ± 0.01
ALNP	$\mathcal{L}_{\text{ML}}$	-0.52 ± 0.01	-0.73 ± 0.01	-0.69 ± 0.01	$-1.14 \pm 6.0\mathrm{E}{-3}$	$0.09 \pm 3.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{ML}}$	$-0.84 \pm 9.0\mathrm{E}{-3}$	$-0.96 \pm 7.0\mathrm{E}{-3}$	$-0.93 \pm 9.0\mathrm{E}{-3}$	$-1.23 \pm 5.0\mathrm{E}{-3}$	$-0.02 \pm 2.0\mathrm{E}{-3}$
ConvLNP	$\mathcal{L}_{\text{NP}}$	-0.50 ± 0.02	-0.77 ± 0.01	-0.48 ± 0.02	$-1.03 \pm 8.0\mathrm{E}{-3}$	$2.47 \pm 8.0\mathrm{E}{-3}$
ALNP	$\mathcal{L}_{\text{NP}}$	-0.82 ± 0.01	-0.96 ± 0.01	-1.04 ± 0.01	$-1.37 \pm 6.0\mathrm{E}{-3}$	$0.20 \pm 9.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{NP}}$	$-0.58 \pm 9.0\mathrm{E}{-3}$	$-1.00 \pm 9.0\mathrm{E}{-3}$	-0.72 ± 0.01	$-1.22 \pm 5.0\mathrm{E}{-3}$	$-0.16 \pm 2.0\mathrm{E}{-3}$
Interpolation beyond training range						
GP (full)		5.80 ± 0.02	$1.22 \pm 6.3\mathrm{E}{-3}$	$1.00 \pm 4.1\mathrm{E}{-3}$	$-0.06 \pm 4.6\mathrm{E}{-3}$	N/A
GP (diag)		-0.59 ± 0.01	$-0.84 \pm 9.0\mathrm{E}{-3}$	-0.89 ± 0.01	$-1.17 \pm 5.2\mathrm{E}{-3}$	N/A
ConvCNP		-0.69 ± 0.02	-0.87 ± 0.01	-0.94 ± 0.02	$-1.19 \pm 7.0\mathrm{E}{-3}$	1.11 ± 0.04
ConvLNP	$\mathcal{L}_{\text{ML}}$	-0.30 ± 0.02	-0.58 ± 0.01	-0.56 ± 0.01	$-1.03 \pm 6.0\mathrm{E}{-3}$	2.29 ± 0.02
ALNP	$\mathcal{L}_{\text{ML}}$	$-1.35 \pm 6.0\mathrm{E}{-3}$	$-1.39 \pm 7.0\mathrm{E}{-3}$	$-1.65 \pm 5.0\mathrm{E}{-3}$	$-1.35 \pm 4.0\mathrm{E}{-3}$	$-0.17 \pm 1.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{ML}}$	$-2.70 \pm 3.0\mathrm{E}{-3}$	$-2.60 \pm 3.0\mathrm{E}{-3}$	$-2.82 \pm 3.0\mathrm{E}{-3}$	-	$-0.03 \pm 2.0\mathrm{E}{-3}$
ConvLNP	$\mathcal{L}_{\text{NP}}$	-0.48 ± 0.02	-0.79 ± 0.01	-0.48 ± 0.02	$-1.04 \pm 8.0\mathrm{E}{-3}$	$2.47 \pm 8.0\mathrm{E}{-3}$
ALNP	$\mathcal{L}_{\text{NP}}$	-1.91 ± 0.03	$-1.48 \pm 4.0\mathrm{E}{-3}$	$-1.85 \pm 7.0\mathrm{E}{-3}$	-1.66 ± 0.01	$-0.30 \pm 4.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{NP}}$	-13.7 ± 0.82	-3.96 ± 0.04	-3.80 ± 0.02	-	-4.98 ± 0.02
Extrapolation beyond training range						
GP (full)		$4.29 \pm 6.2\mathrm{E}{-3}$	$0.82 \pm 4.3\mathrm{E}{-3}$	$0.66 \pm 2.2\mathrm{E}{-3}$	$-0.33 \pm 3.4\mathrm{E}{-3}$	N/A
GP (diag)		$-1.40 \pm 5.0\mathrm{E}{-3}$	$-1.41 \pm 4.8\mathrm{E}{-3}$	$-1.72 \pm 6.2\mathrm{E}{-3}$	$-1.40 \pm 4.0\mathrm{E}{-3}$	N/A
ConvCNP		$-1.41 \pm 6.0\mathrm{E}{-3}$	$-1.41 \pm 7.0\mathrm{E}{-3}$	$-1.73 \pm 8.0\mathrm{E}{-3}$	$-1.41 \pm 6.0\mathrm{E}{-3}$	0.27 ± 0.02
ConvLNP	$\mathcal{L}_{\text{ML}}$	$-1.09 \pm 5.0\mathrm{E}{-3}$	$-1.11 \pm 5.0\mathrm{E}{-3}$	$-1.30 \pm 4.0\mathrm{E}{-3}$	$-1.24 \pm 4.0\mathrm{E}{-3}$	1.61 ± 0.02
ALNP	$\mathcal{L}_{\text{ML}}$	$-1.29 \pm 6.0\mathrm{E}{-3}$	$-1.29 \pm 5.0\mathrm{E}{-3}$	$-1.55 \pm 5.0\mathrm{E}{-3}$	$-1.34 \pm 5.0\mathrm{E}{-3}$	$-0.25 \pm 2.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{ML}}$	$-2.23 \pm 4.0\mathrm{E}{-3}$	$-2.08 \pm 3.0\mathrm{E}{-3}$	$-2.50 \pm 4.0\mathrm{E}{-3}$	$-1.39 \pm 4.0\mathrm{E}{-3}$	$-0.06 \pm 2.0\mathrm{E}{-3}$
ConvLNP	$\mathcal{L}_{\text{NP}}$	-1.21 ± 0.01	-1.31 ± 0.01	-1.19 ± 0.01	$-1.51 \pm 8.0\mathrm{E}{-3}$	$2.10 \pm 7.0\mathrm{E}{-3}$
ALNP	$\mathcal{L}_{\text{NP}}$	$-1.44 \pm 6.0\mathrm{E}{-3}$	$-1.45 \pm 6.0\mathrm{E}{-3}$	$-1.77 \pm 7.0\mathrm{E}{-3}$	$-1.46 \pm 6.0\mathrm{E}{-3}$	$-0.20 \pm 2.0\mathrm{E}{-3}$
NP	$\mathcal{L}_{\text{NP}}$	-5.85 ± 0.05	$-2.65 \pm 3.0\mathrm{E}{-3}$	-4.06 ± 0.04	$-1.49 \pm 5.0\mathrm{E}{-3}$	$-1.99 \pm 6.0\mathrm{E}{-3}$

Table B.2 Log-likelihood for ConvCNP, ConvLNP, ALNP, and NP. Each of the stochastic models was trained on each data set with $\mathcal{L}_{\text{ML}}$ and $\mathcal{L}_{\text{NP}}$, separately.

For all data sets, pixel values are divided by 255 to rescale them to the $[0, 1]$ range. We evaluate on predefined test splits when available (MNIST, SVHN, ZSMM) and make our own test set for CelebA by randomly selecting 10% of the data. For each dataset we also set aside 10% of the training set as validation.

B.6.2 Training Details

In all experiments, we sample the number of context pixels uniformly from $\mathcal{U}(0, \frac{n_{\text{total}}}{2})$, and the number of target points is set to n_{total}. The weights are optimized using Adam (Kingma and Ba, 2015) with learning rate 5×10^{-4}. We use a maximum of 100 epochs, with early stopping — based on log likelihood on the validation set — of 10 epochs patience. Unless stated otherwise, we use $L = 16$ samples from the latent function during training, and $L = 128$ at test time. We clip the $L2$ norm of all gradients to 1, which was particularly important for ConvLNP. We use a batch size of 32 for all models besides ALNP trained on ZSMM which used a batch size of 8 due to memory constraints.

B.6.3 Architecture Details

General architecture details. For all models, we follow Le et al. (2018) and process the predicted standard deviation of the latent function $\boldsymbol{\sigma}_z$ using a sigmoid and the standard deviation $\boldsymbol{\sigma}$ of the predictive distribution using lower-bounded softplus:

$$\boldsymbol{\sigma}_z = 0.001 + (1 - 0.001)\frac{1}{1 + \exp(f_{\sigma,z})} \tag{B.2}$$

$$\boldsymbol{\sigma} = 0.001 + (1 - 0.001)\ln(1 + \exp(f_{\sigma})) \tag{B.3}$$

As the pixels are rescaled to $[0, 1]$, we also process the mean of the posterior predictive (conditioned on a single sample) to be in $[0, 1]$ using a logistic function

$$\boldsymbol{\mu} = \frac{1}{1 + \exp(-f_{\mu})} \tag{B.4}$$

In the following, we describe the architecture of ALNP and ConvLNP. Unless stated otherwise, all vectors in the following paragraphs are in $\mathbb{R}^{128}$ and all MLPs have 128 hidden units.

ALNP details. We provide details for the ALNP trained with $\mathcal{L}_{\text{ML}}$. As the ALNP cannot take advantage of the fact that images are on the grid, we preprocess each pixel so that $\mathbf{x} \in [-1, 1]^2$. The only exception being for the test set of ZSMM, where

$\mathbf{x} \in [-\frac{56}{32}, \frac{56}{32}]^2$ as the model is trained on 32×32 but evaluated on 56×56 images. Each context feature is first encoded $\mathbf{x}^{(c)} \mapsto \mathbf{r}_x^{(c)}$ by a single hidden layer MLP, while a second single hidden layer MLP encodes values $\mathbf{y}^{(c)} \mapsto \mathbf{r}_y^{(c)}$. We produce a representation $\mathbf{r}_{xy}^{(c)}$ by summing both representations $\mathbf{r}_x^{(c)} + \mathbf{r}_y^{(c)}$ and passing them through two self-attention layers (Vaswani et al., 2017). Following Parmar et al. (2018), each self-attention layer is implemented as 8-headed attention, a skip connection, and two layer normalizations (Ba et al., 2016). To predict values at each target point t, we embed $\mathbf{x}^{(t)} \mapsto \mathbf{r}_x^{(t)}$ using the hidden layer MLP used for $\mathbf{r}_x^{(c)}$. A deterministic target representation $\mathbf{r}_{xy}^{(t)}$ is then computed by applying cross-attention (using an 8-headed attention described above) with keys $\mathrm{K} := \{\mathbf{r}_x^{(c)}\}_{c=1}^{C}$, values $\mathrm{V} := \{\mathbf{r}_{xy}^{(c)}\}_{c=1}^{C}$, and query $\mathbf{q} := \mathbf{r}_x^{(t)}$. For the latent path, we average over context representations $\mathbf{r}_{xy}^{(c)}$, and pass the resulting representation through a single hidden layer MLP that outputs $(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z) \in \mathbb{R}^{256}$. $\boldsymbol{\sigma}_z$ is made positive by post-processing it using Equation (B.2). We then sample (with reparametrization (Kingma and Welling, 2014)) L latent representation $\boldsymbol{z}_\ell \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z^2)$.

We describe the remainder of the forward pass for a single $\boldsymbol{z}_\ell$, though in practice multiple samples may be processed in parallel. The deterministic and latent representations of the context set are concatenated, and the resulting representation is passed through a linear layer $[\mathbf{r}_{xy}^{(t)}; \boldsymbol{z}_\ell] \to \mathbf{r}_{xyz}^{(t)} \in \mathbb{R}^{128}$. Given the target and context-set representations, the predictive posterior is given by a Gaussian pdf with diagonal covariance parametrised by $(\boldsymbol{\mu}^{(t)}, \boldsymbol{\sigma}_{\text{pre}}^{(t)}) = \text{decoder}([\mathbf{r}_x^{(t)}; \mathbf{r}_{xyz}^{(t)}])$ where $\boldsymbol{\mu}^{(t)}, \boldsymbol{\sigma}_{\text{pre}}^{(t)} \in \mathbb{R}^3$ and decoder is a 4 hidden layer MLP. Finally, the $\boldsymbol{\sigma}^{(t)}$ is processed by Equation (B.3) using Equation (B.4). In the case of MNIST and ZSMM, $\boldsymbol{\sigma}^{(t)}$ is also spatially mean pooled, which corresponds to using homoskedastic noise. This improves the qualitative performance by forcing ALNP and ConvLNP to model the digit instead of focusing on predicting the black background with high confidence. Kim et al. (2019) did not suffer from that issue as they used a much larger lower bound for Equation (B.3).

ConvLNP details. The core algorithm of on-the-grid ConvLNP is outlined in Algorithm 5 as well as Algorithm 3. Here we discuss the parametrizations used for each step of the algorithm. All convolutional layers are depthwise separable (Chollet, 2017). $\textsc{conv}_{\boldsymbol{\theta}}$ is a convolutional layer with kernel size of 11 (no bias). Following Gordon et al. (2020), we enforce positivity on the weights in the first convolutional layer by only convolving their absolute value with the signal.

The CNNs are ResNets (He et al., 2016) with 9 blocks, where each convolution has a kernel size of 3. Each residual block consists of two convolutional layers, pre-activation batch normalization layers Ioffe and Szegedy (2015), and ReLU activations. The output of the pre-latent CNN (CNN in Algorithm 3) goes through a single hidden layer MLP that outputs $(\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z) \in \mathbb{R}^{256}$. As with ALNP, $f_{\sigma,z}$ is processed by Equation (B.2) and then used to sample (with reparametrization (Kingma and Welling, 2014)) L latent functions $\mathbf{Z}_\ell$. Importantly, we found that the coherence of samples improves if the model uses a *global representation* in addition to the the pixel dependent representation. We achieve this by mean-pooling half of the functional representation. Namely, we replace $\boldsymbol{z}_\ell$ by the channel-wise concatenation of $\boldsymbol{z}_\ell^{(1:64)}$ and $\textsc{mean}(\boldsymbol{z}_\ell^{(65:128)})$, where the mean is taken over the spatial dimensions. This latent function then goes through the post-latent CNN (CNN in Algorithm 5), as well as a linear layer to output $(f_\mu, f_\sigma) \in \mathbb{R}^{256}$. As for ALNP f_μ is processed by Equation (B.4) and f_σ is re-scaled with Equation (B.3) and is spatially pooled in the case of MNIST and ZSMM to obtain homoskedastic noise.

B.7 Additional results on image completion.

We provide additional qualitative samples and quantitative analyses for the ConvLNP and ALNP.

Additional ConvLNP samples. Figure B.2 provides further samples from a ConvLNP trained with $\mathcal{L}_{\text{ML}}$. We observe that the ConvLNP produces reasonably diverse yet coherent samples when evaluated in a regime that resembles the training regime (in the first four sub-columns of MNIST, SVHN, and CelebA). However, Figure B.2 also demonstrates that the ConvLNP struggles with context sets that are significantly different from those seen during training.

Further comparisons of ALNP and ConvLNP. We provide further qualitative comparisons of ConvLNPs, ALNPs trained with $\mathcal{L}_{\text{ML}}$, and ALNPs trained with $\mathcal{L}_{\text{LNP}}$. We omit ConvLNPs trained with $\mathcal{L}_{\text{LNP}}$ as these are significantly outperformed by ConvLNPs trained with $\mathcal{L}_{\text{ML}}$ (see e.g. Table 4.2).

Figure B.3 shows that all models perform relatively well when context sets are drawn from a similar distribution as employed during training (first four sub-columns of MNIST, SVHN, and CelebA). Furthermore, we observe that samples from the ConvLNP prior tend to be closer to samples from the underlying data distribution (e.g. for CelebA).

The qualitative advantage of ConvLNP is most significant in settings that require translation equivariance for generalization. Figure B.3 row 2 (ZSMM) clearly demonstrates that ConvLNP generalizes to larger canvas sizes and multiple digits, while ALNP attempts to reconstruct a single digit regardless of the context set. Finally, Figure B.4 provides the test log-likelihood distributions of ALNP and ConvLNP as well as some qualitative comparisons between the two.

B.8 Experimental Details on Environmental Data

B.8.1 Data Details

	Central (train)	Western (test)	Eastern (test)	Southern (test)
Latitudes	$(52, 46)$	$(50, 46)$	$(52, 49)$	$(46, 42)$
Longitudes	$(08, 28)$	$(01, 08)$	$(28, 35)$	$(19, 26)$

Table B.3 Coordinates for boxes defining the train and test regions. Latitidues are given as (north, south), and longitudes as (west, east).

ERA5-Land (Service, 2020) contains high resolution information on environmental variables at a 9 km spacing across the globe.[4] The data we use contains daily measurements of accumulated precipitation at 11pm and temperature at 11pm at every location, between 1981 and 2020, yielding a total of 14,304 temporal measurements across the spatial grid. In addition, we provide orography (elevation) values for each location. We normalize the data such that the precipitation values in the train set have zero mean and unit standard deviation.

We consider the task of predicting daily precipitation y, with latitude and longitude as $\boldsymbol{x}$. In addition, at each context and target location, we provide the model with access to side information in the form of orography (elevation) and temperature values. We also normalize the orography and temperature values to have zero mean and unit standard deviation. We choose a large region of central Europe as our train set, and use regions East, West and South of the train set as held out test sets (see Figure B.5 and Table B.3). At train time, to sample a task, we first sample a random date between 1981 and 2020. We then sample a square subregion of grid of values from within the train region (which has size 61×201). We consider two models, one trained on 28×28 subregions, and another trained on 40×40 subregions. During training, each subregion is then split into context and target sets. Context points are randomly chosen with a keep rate p_{keep} with $p_{\text{keep}} \sim \mathcal{U}[0, 0.3]$. In this section, we train only on the $\mathcal{L}_{\text{ML}}$ objective.

[4]URL: https://www.ecmwf.int/en/era5-land. Neither the European Commission nor ECMWF is responsible for any use that may be made of the Copernicus Information or data it contains.

B.8.2 Gaussian Process Baseline

We mean-centre the data for each task for the GP before training, and add the mean offset back for evaluation and sampling. We use an Automatic Relevance Determination (ARD) kernel, with separate factors for latitude/longitude, temperature and orography. In detail, let $\boldsymbol{x} = (x_{\text{lat}}, x_{\text{lon}})$ denote position, and let ω, t denote orography and precipitation respectively, and let $\boldsymbol{r} \coloneqq (\boldsymbol{x}, \omega, t)$. Then the kernel is given by

$$k(\boldsymbol{r}, \boldsymbol{r}') = \sigma_v^2 k_\ell(\boldsymbol{x}, \boldsymbol{x}') k_\omega(\omega, \omega') k_t(t, t') + \sigma_n^2 \delta(\boldsymbol{r}, \boldsymbol{r}').$$

Here each of k_ℓ, k_ω and k_t are Matérn–$\frac{5}{2}$ kernels with separate learnable lengthscales; $\delta(\boldsymbol{r}, \boldsymbol{r}') = 1$ if $\boldsymbol{r} = \boldsymbol{r}'$ and 0 otherwise; and σ_v^2, σ_n^2 are learnable signal and noise variances respectively. We learn all hyperparameters by maximising the log-marginal likelihood using Scipy's implementation of L-BFGS.

Transforming the data. As the data is non-negative, we considered applying the transform $y \mapsto \log(\epsilon + y)$ for the GP to model. If $\epsilon = 0$, this would guarantee that the GP would only yield positive samples, which would be physically sensible as precipitation is non-negative. However, this cannot be done as precipitation often takes the value $y = 0$, which would lead to the transform being undefined. On the other hand, if $\epsilon > 0$, the GP samples after performing the inverse transform could still predict a precipitation value as low as $-\epsilon$, which is still unphysical. Further, a small value of ϵ leads to large distortion of the y values in transformed space. In the end, we run all experiments for the GP and NP without log-transforming the data; hence the models have to learn non-negativity.

B.8.3 ConvLNP Architecture and Training Details

As the ERA5-Land dataset is regularly spaced, we use the on-the-grid version of the architecture, without the need for an RBF smoothing layer at the input. All experiments used a convolutional architecture with 3 residual blocks (He et al., 2016) for the encoder and 3 residual blocks for the decoder. Each residual block is defined with two layers of ReLU activations followed by convolutions, each with kernel size 5. The first convolution in each block is a standard convolution layer, whereas the second is depthwise separable (Chollet, 2017). All intermediate convolutional layers have 128 channels, and the latent function $\boldsymbol{z}$ has 16 channels. The networks were trained using ADAM with a learning rate of 10^{-4}. We used 16 channels for the latent function $\boldsymbol{z}$, and estimated $\mathcal{L}_{\text{ML}}$ using 16-32 samples at train time, with batches of 8-16 images.

We train the models for between 400 and 500 epochs, where each epoch is defined as a single pass through each day in the training set, where at each day, a random subregion of the full 61×201 central Europe region is cropped. We estimated the predictive density using 2500 samples of $\boldsymbol{z}$ during test time.

B.8.4 Prediction and Sampling

To create Table 4.3, at test time we sample 28×28 subregions from each of the train and test regions. This is done 1000 times. For the GP, we randomly restart optimisation 5 times per task and use the best hyper-parameters found. In order to remove outliers where the GP has very poor likelihood, we set a log-likelihood threshold for the GP. If the GP has a log-likelihood of less than 0 nats on a particular task, then that task is removed from the evaluation.

We find that to produce high quality samples, we need to train the model on subregions that are roughly as large as the lengthscale of the precipitation process. Hence we sample from the model trained on 40×40 subregions in Figure 4.7 in the main body. We show samples from the model trained on both 28×28 subregions and 40×40 subregions in Section B.9. We also compare to samples from GPs trained on each context set (no random restarts were used for sampling).

B.8.5 Bayesian Optimization

We use the models described in Section B.8.3, trained on random 28×28 subregions of the train region, and compare to the GP baselines described in Section B.8.2. For the Bayesian optimization experiments in Figure 4.8 in the main body, we do not perform random restarts as this was too time-consuming. We carry out the Bayesian optimization (BayesOpt) experiments in each of the four regions: Central (train), West (test), East (test), and South (test). Each Bayesian optimization “episode” is defined by randomly sub-sampling a day (uniformly at random between 1981 and 2020), then sampling a sub-region from the tested region. To test the models’ spatial generalization capacity (where possible), we sub-sample episodes from each of the four regions with the following sizes: (i) Central: 42x42, (ii) West: 40x40, (iii) East: 28x28, and (iv) South: 36x36.

Episodes begin from empty sets $D_c^{(0)} =$, and models sequentially query locations for $t = 1, \ldots, 50$. Denoting $(\boldsymbol{x}^{(t)}, y^{(t)})$ the query location and queried value at iteration t, the context set is then updated as $D_c^{(t)} = D_c^{(t-1)} \cup \{(\boldsymbol{x}^{(t)}, y^{(t)})\}$. Denoting $\boldsymbol{y}$ as the complete set of rainfall values in the sub-region, and $\boldsymbol{y}^{(t)}$ as the set of queried values at iteration t, we can define the *instantaneous regret* as $r_t = \max(\boldsymbol{y}) - \max(\boldsymbol{y}_c^{(t)})$, and compute the average regret (plotted in Figure 4.8 in the main text) at the t^{th} iteration as $\bar{r}_t = \frac{1}{t}\sum_{i=1}^{t} r_i$.

B.9 Additional Figures for Environmental Data

B.9.1 Predictive density

Figure B.6 displays the predictive densities for precipitation at different locations, conditioned on a context set used for testing. The density of the ConvLNP is estimated using 2500 samples of $\boldsymbol{z}$. To examine why the ConvLNP outperforms the GP in terms of log-likelihood, we plot cases where the ConvLNP likelihood is significantly better than the GP likelihood. We see that this is due to the GP occasionally making very overconfident predictions compared to the ConvLNP. We also see that the ConvLNP in a small proportion of cases exhibits very non-Gaussian, asymmetric predictive distribtuions.

B.9.2 Additional Samples

In this section we show additional samples from the model trained on 28×28 images (Figures B.7 and B.8) and also on 40×40 images (Figures B.9 and B.10). Training on larger images reduces the occurence of blocky artefacts. Figure 4.7 in the main body was trained on 40×40 images. Note that samples shown here are 61×201, i.e. the size of the entire central Europe train region.

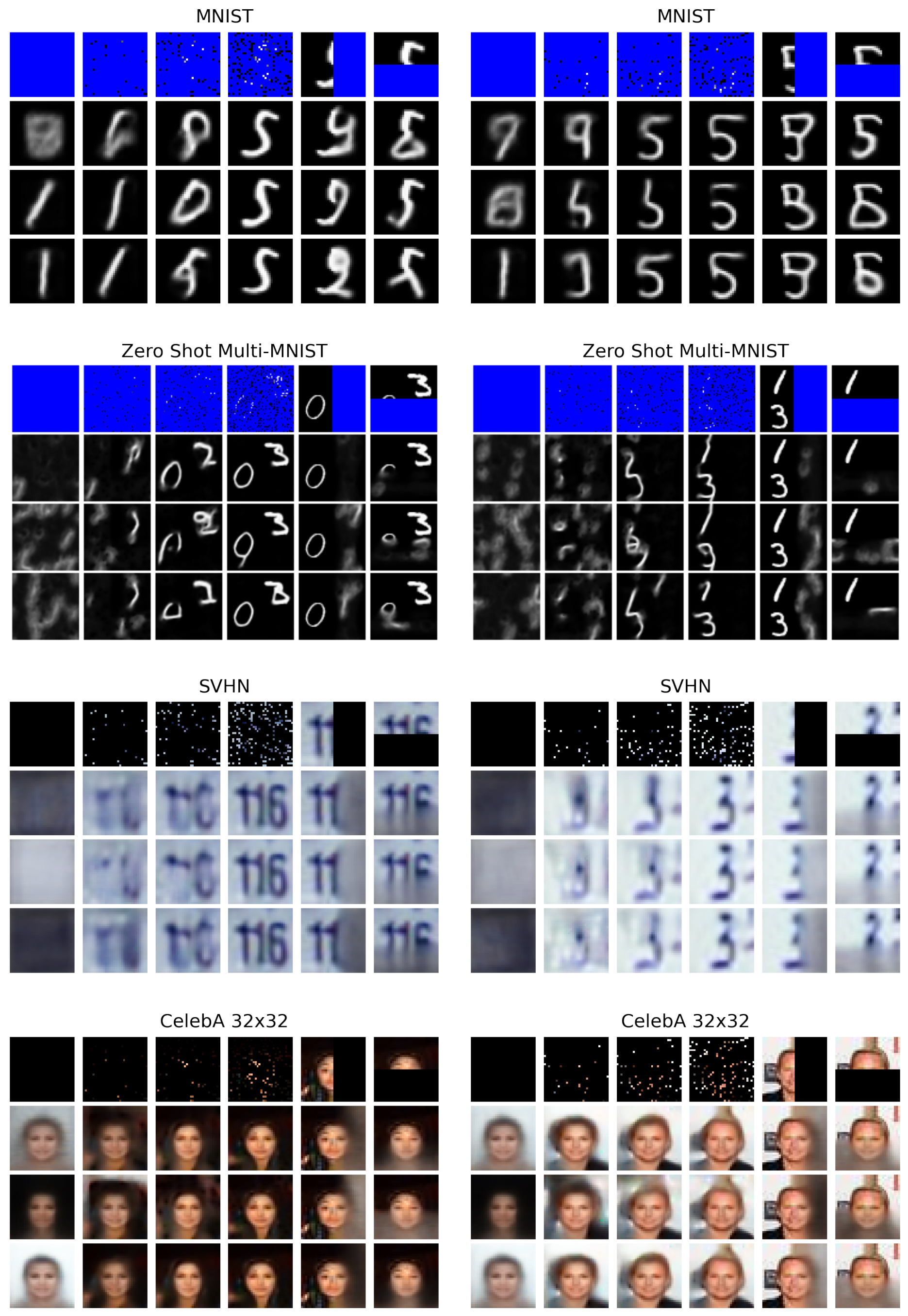
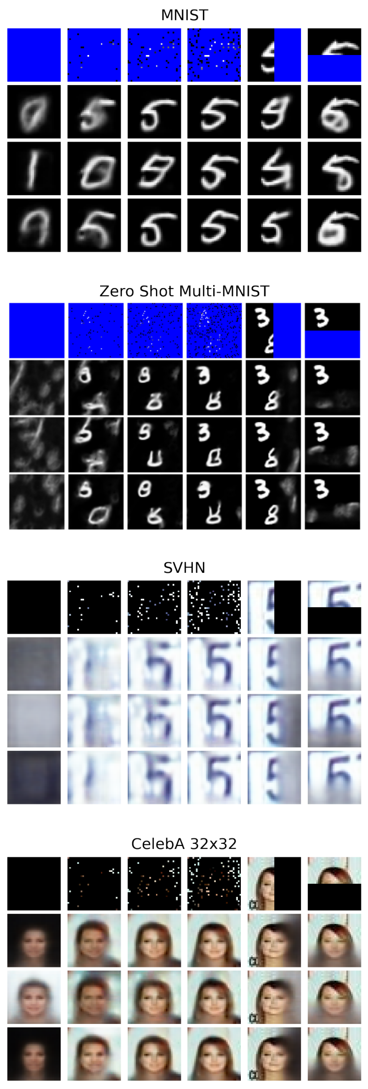
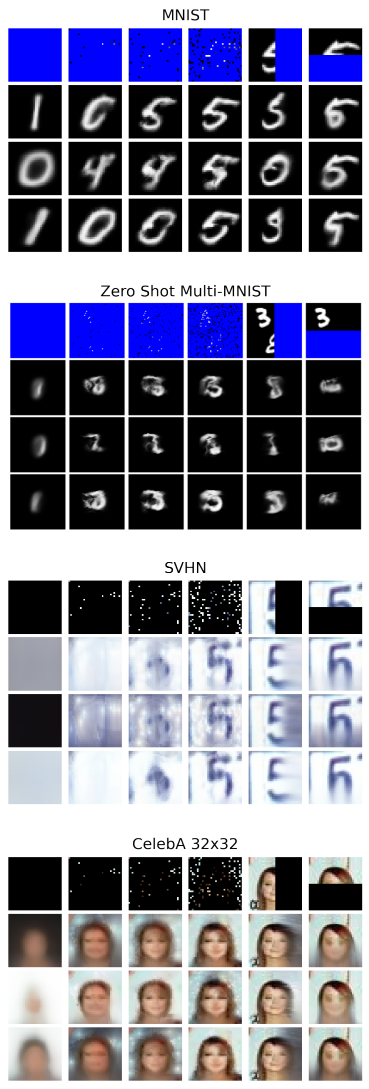


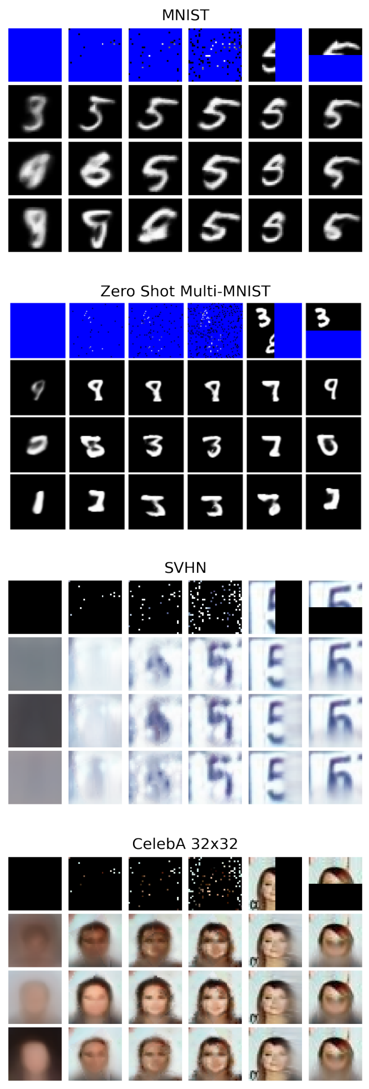
Fig. B.2 Qualitative samples for one of the ConvLNP trained with $\mathcal{L}_{\mathrm{ML}}$ in Table 4.2. From top to bottom the four major rows correspond to MNIST, ZSMM, SVHN, CelebA32 datasets. For each dataset and each of the two major columns, a different image is randomly sampled; the first sub-row shows the given context points (missing pixels are in blue for MNIST and ZSMM but in black for SVHN and CelebA), while the next three sub-rows show the mean of the posterior predictive corresponding to different samples of the latent function. To show diverse samples we select three samples that maximize the average Euclidean distance between pixels of the samples. From left to right the first four sub-columns correspond to a context set with 0%, 1%, 3%, 10% randomly sampled context points. In the last two sub-columns, the context sets respectively contain all the pixels in the left and top half of the image.



(a) ConvNP $\mathcal{L}_{\mathrm{ML}}$

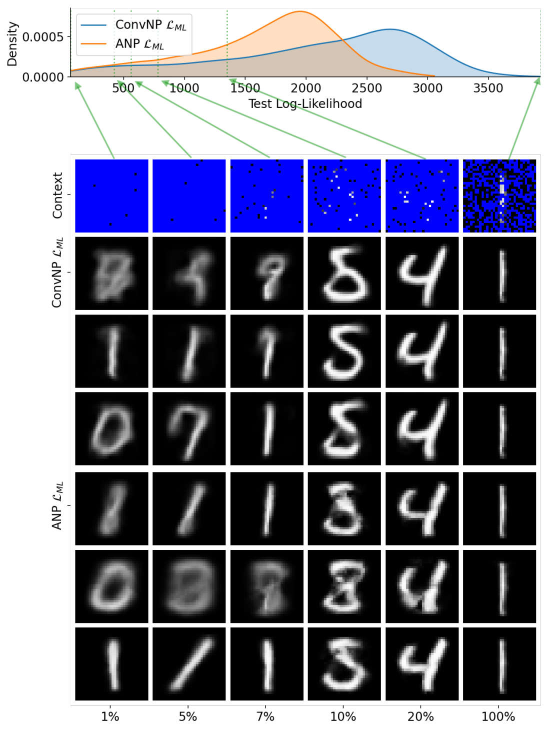


(b) ANP $\mathcal{L}_{\mathrm{ML}}$

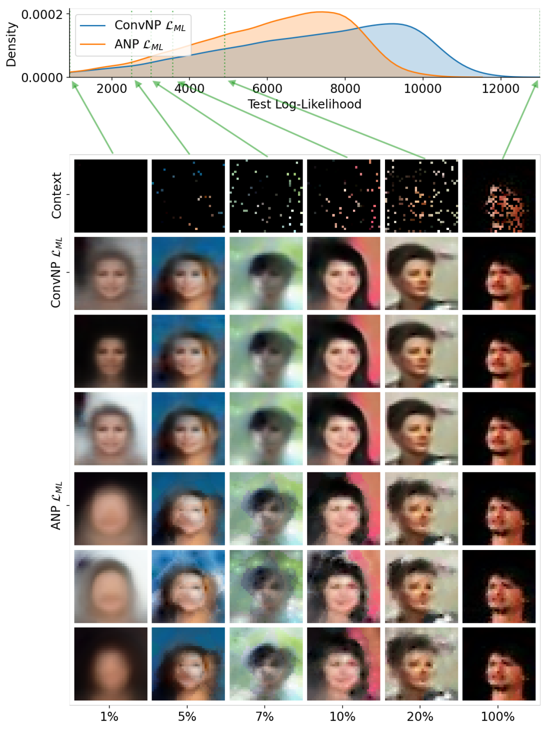


(c) ANP $\mathcal{L}_{\mathrm{NP}}$

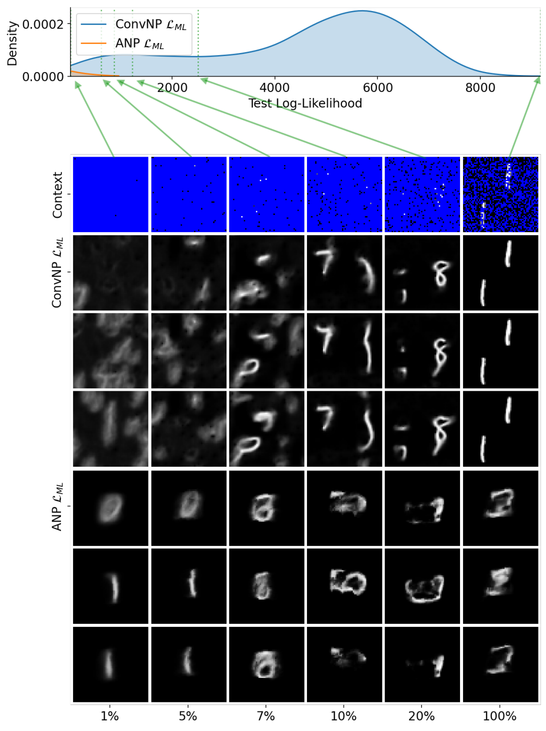
Fig. B.3 Qualitative samples between (a) ConvLNP trained with $\mathcal{L}_{\mathrm{ML}}$; (b) ANLP trained with $\mathcal{L}_{\mathrm{ML}}$; (c) ANLP trained with $\mathcal{L}_{\mathrm{LNP}}$. For each model the figure shows the same as Figure B.2.



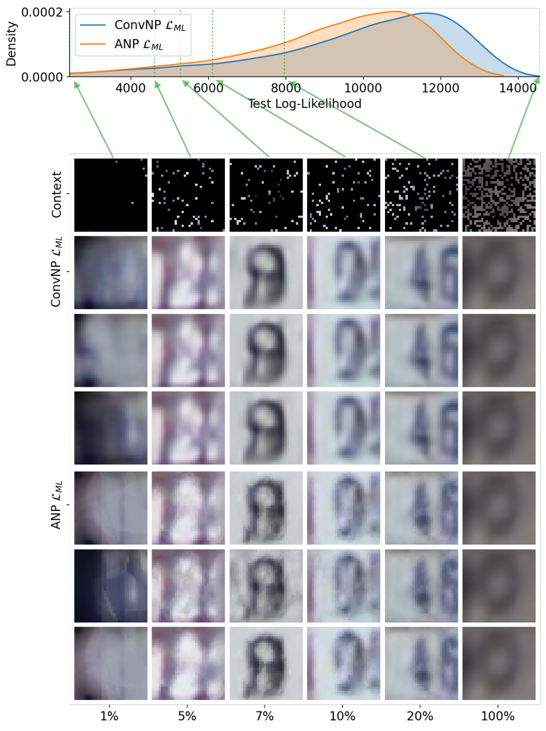
(a) MNIST



(b) CelebA32



(c) Zero Shot Multi-MNIST



(d) SVHN

Fig. B.4 Log-likelihood and qualitative samples comparing ConvLNP and ALNP trained with $\mathcal{L}_{\text{ML}}$ on (a) MNIST; (b) CelebA; (c) ZSMM; (d) SVHN. For each sub-figure, the top row shows the log-likelihood distribution for both models. The images below correspond to the context points (top), followed by three samples form ConvLNP (mean of the posterior predictive corresponding to different samples from the latent function), and three samples from ALNP. Each column corresponds to a given percentile of the ConvLNP test log likelihood (as shown by green arrows).

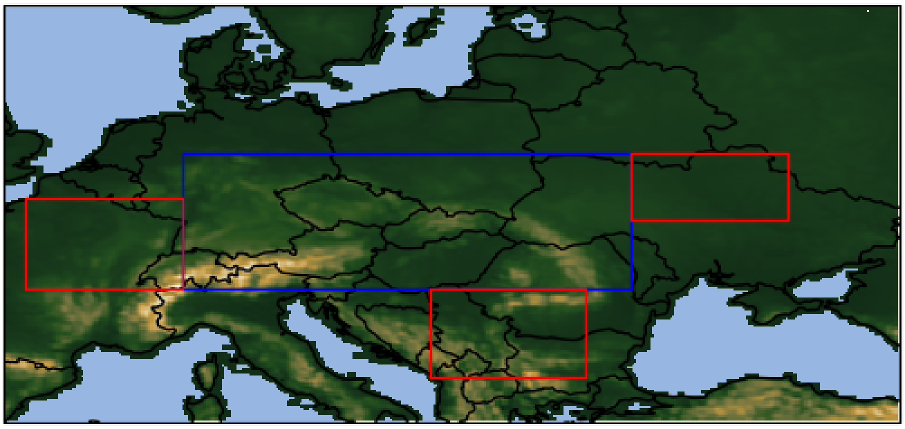
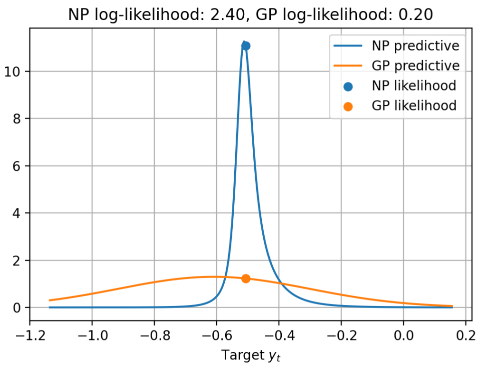
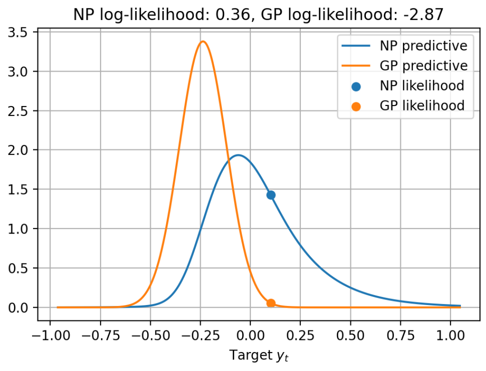


Fig. B.5 Training (blue) and test (red) regions in Europe, along with orography data from ERA5Land.



(a)



(b)

Fig. B.6 Predictive density at two target points, where the ConvLNP significantly outperforms the GP. The orange and blue circles show the likelihood of the ground truth target value under the GP and ConvLNP. Note that as the precipitation values are normalized to zero mean and unit standard deviation, $y_t = -0.53$ corresponds to no rain. In Figure B.6a, we see the ConvLNP sometimes produces predictions heavily centered on this value, showing it has learned the sparsity of precipitation values. In Figure B.6b we see the ConvLNP predictive distribution is sometimes asymmetric with a heavier positive tail, reflecting the non-negativity of precipitation.

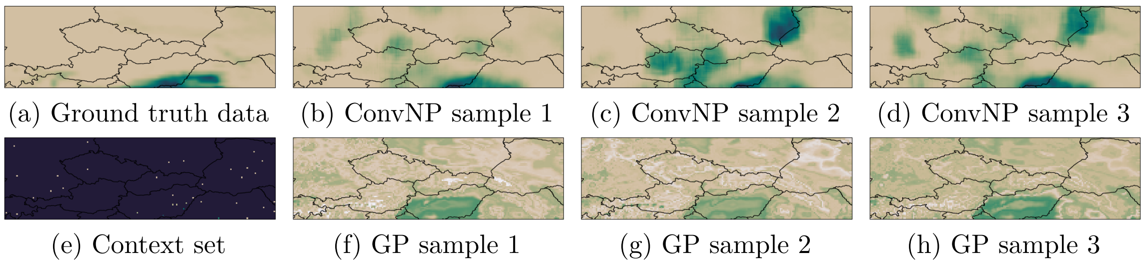


Fig. B.7 Samples from the predictive processes overlaid on central Europe, for a model trained on random 28×28 subregions of the full 61×201 central Europe region. Note some blocky artefacts in the ConvNP samples due to training on small subregions. Here the GP has overfit to the orography data, with samples that resemble the orography rather than precipitation.

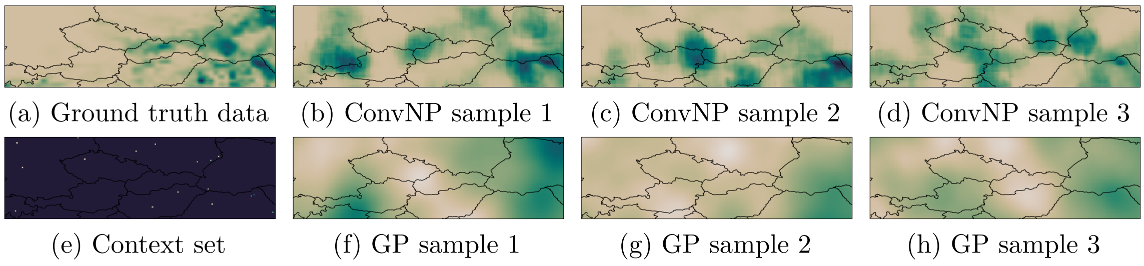


Fig. B.8 Samples from the predictive processes overlaid on central Europe, for a model trained on random 28×28 subregions of the full 61×201 central Europe region. Here the GP has learned a lengthscale that is too large.

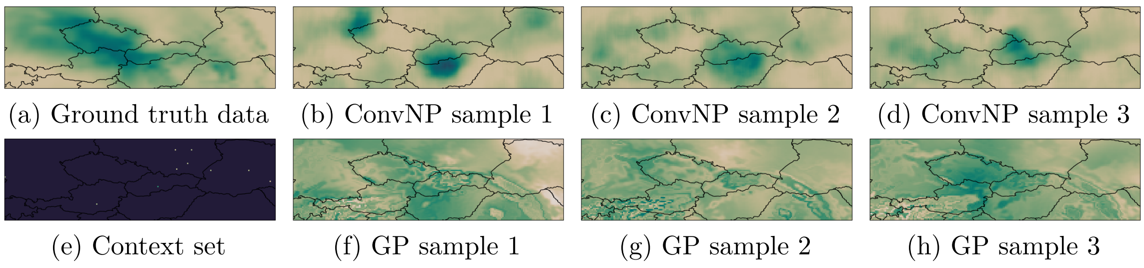


Fig. B.9 Samples from the predictive processes overlaid on central Europe, for a model trained on random 40×40 subregions of the full 61×201 central Europe region. Here the GP has overfit to the orography data, with samples that resemble the orography rather than precipitation.

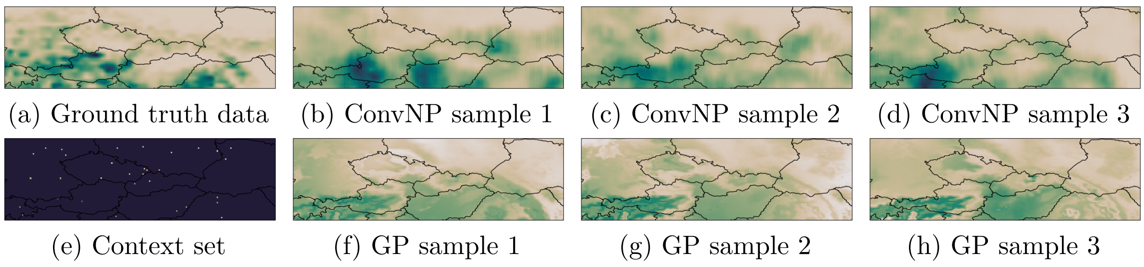
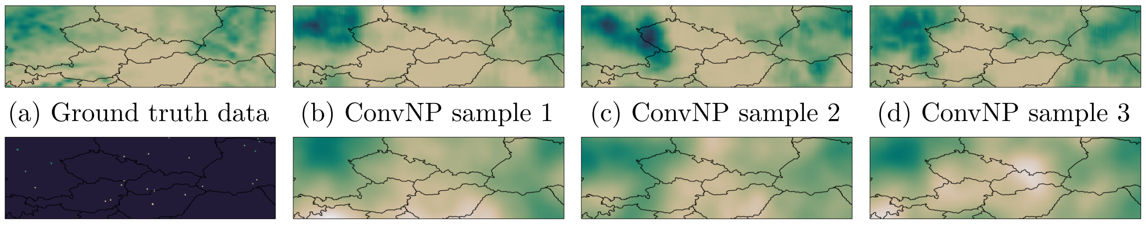


Fig. B.10 Samples from the predictive processes overlaid on central Europe, for a model trained on random 40×40 subregions of the full 61×201 central Europe region. The GP has again overfit to the orography data.



(a) Ground truth data (b) ConvNP sample 1 (c) ConvNP sample 2 (d) ConvNP sample 3

(e) Context set (f) GP sample 1 (g) GP sample 2 (h) GP sample 3

Fig. B.11 Samples from the predictive processes overlaid on central Europe, for a model trained on random 40×40 subregions of the full 61×201 central Europe region.

Appendix C

Chapter 5 Supplementary Material

C.1 Additional theoretical considerations

In this appendix we provide some additional discussion on theoretical aspects of our models. In Section C.1.1 we provide a proof for the translation equivariance of the ConvGNP, which follows from the fact that the ConvCNP is translation equivariant. In Section C.1.2 we provide some additional discussion on composing arbitrary invertible maps such as Normalising Flows (NFs) with neural processes.

C.1.1 Translation equivariance of the ConvGNP

In this section of the appendix we provide a proof for the translation equivariance of the ConvGNP model. The mean function of a ConvGNP has the same form as a ConvCNP model (Gordon et al., 2020) and is thus translation equivariant. It therefore remains to show that the covariance function of the ConvGNP is also translation equivariant.

As explained in the main text, the covariance of the ConvGNP is computed as follows. First, a feature function g is applied to the context set and target inputs. This feature function consists of the following sequence of computations

$$(\mathbf{x}^{(c)}, \mathbf{y}^{(c)}) \xrightarrow{①} (\tilde{\mathbf{x}}, \mathbf{h}) \xrightarrow{②} \mathbf{r} = \mathrm{CNN}_D(\mathbf{h}) \xrightarrow{③} g(x_i^{(t)}, \mathbf{r}) = \sum_{\ell=1}^{L} \psi(x_i^{(t)}, x_{r,\ell})\, r_\ell, \tag{C.1}$$

where step ① maps $(\mathbf{x}^{(c)}, \mathbf{y}^{(c)})$ to a D-dimensional grid of values $\mathbf{h}$ with corresponding locations at $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_L)$, $\tilde{x}_\ell \in \mathbb{R}^D$, using a SetConv layer (Gordon et al., 2020), step ② maps $\mathbf{h}$ to $\mathbf{r}$ through a CNN with D-dimensional convolutions, and ③ aggregates $\mathbf{r}$ using an RBF ψ. The grid locations $\tilde{\mathbf{x}}$ are set using the context and target inputs (see Gordon et al., 2020), according to

$$\tilde{\mathbf{x}} = \texttt{grid}(x_{min}, x_{max}), \text{ where } x_{min} = \min\{\mathbf{x}^{(c)}, \mathbf{x}^{(t)}\}, x_{max} = \max\{\mathbf{x}^{(c)}, \mathbf{x}^{(t)}\}. \tag{C.2}$$

Lastly, the features outputted by g are passed through a positive definite function k to produce the entries of the covariance

$$\mathbf{K}_{ij} = k(g(x_i^{(t)}, \mathbf{r}), g(x_j^{(t)}, \mathbf{r})). \tag{C.3}$$

To show the translation equivariance of the covariance function, it suffices to show that if a translation is applied to both the context and target inputs, then the resulting covariance matrix remains unchanged. In particular, consider applying a translation u to the context and target inputs

$$\mathbf{x}^{(c)\prime} = (x_1'^{(c)}, \dots, x_N'^{(c)}) = (x_1^{(c)} + u, \dots, x_N^{(c)} + u) \tag{C.4}$$
$$\mathbf{x}^{(t)\prime} = (x_1'^{(t)}, \dots, x_M'^{(t)}) = (x_1^{(t)} + u, \dots, x_M^{(c)} + u) \tag{C.5}$$

and applying g to the translated inputs, according to

$$(\mathbf{x}^{(c)\prime}, \mathbf{y}^{(c)}) \xrightarrow{①} (\tilde{\mathbf{x}}', \mathbf{h}') \xrightarrow{②} \mathbf{r}' = \mathrm{CNN}_D(\mathbf{h}') \xrightarrow{③} g(x_i^{(t)\prime}, \mathbf{r}') = \sum_{\ell=1}^{L} \psi(x_i^{(t)\prime}, x_{r,\ell}')\, r_\ell'. \tag{C.6}$$

where the grid locations are now $\tilde{\mathbf{x}}' = (\tilde{x}_1', \dots, \tilde{x}_L')$, $\tilde{x}_\ell \in \mathbb{R}^D$, where by substituting $\mathbf{x}^{(c)\prime}, \mathbf{x}^{(t)\prime}$ into Equation (C.2) we have

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{x}} + u. \tag{C.7}$$

Now, noting that $\mathbf{h}' = \mathbf{h}$, by the translation equivariance of the SetConv layer (Gordon et al., 2020) and $\mathbf{r}' = \mathbf{r}$, by the equivariance of the CNN, and substituting these together with Equations (C.4), (C.5) and (C.7) into Equation (C.6) we obtain

$$g(x_i^{(t)\prime}, \mathbf{r}') = \sum_{\ell=1}^{L} \psi(x_i^{(t)\prime}, x_{r,\ell}')\, r_\ell' = \sum_{\ell=1}^{L} \psi(x_i^{(t)} + u, x_{r,\ell} + u)\, r_\ell = \sum_{\ell=1}^{L} \psi(x_i^{(t)}, x_{r,\ell})\, r_\ell = g(x_i^{(t)}, \mathbf{r})$$

where we have used the stationarity of the RBF ψ. This shows that g is invariant to translations of the context and target inputs. The covariance function of the ConvGNP is therefore equivariant, as required.

C.1.2 Normalising flows and general invertible maps

Composing Normalising Flows with Neural Processes: In Section 5.3 of the main text we explained that general arbitrary maps could be composed with neural processes in an attempt to model joint dependencies and non-Gaussian marginals. However, requiring a model that is consistent under marginalisation places theoretical limitations on the form of Θ which we can use.

Theoretical limitations: Suppose we have a Neural Process with prediction map $\pi(\cdot\,; x^{(c)}, y^{(c)}, x^{(t)})$, which we wish to compose with an invertible map Θ. Thus, to draw a sample from the predictive distribution, we first draw

$$(u_1^{(t)}, \dots, u_M^{(t)}) \sim \pi(u^{(t)}; x^{(c)}, y^{(c)}, x^{(t)}), \tag{C.8}$$

and then apply the marginal transformation

$$(y_1^{(t)}, \dots, y_M^{(t)}) = \Theta(u_1^{(t)}, \dots, u_M^{(t)}). \tag{C.9}$$

The role of Θ could be either modelling the marginals of the data, representing joint dependencies in the output variable, or both. Now, Θ must be able to handle variable-length tuples as arguments, to handle arbitrary target points. Thus, we should be able to query Θ with any number M of inputs. In addition, the model should be consistent under marginalisations: making a joint prediction for target variables $y_1^{(t)}$ and $y_2^{(t)}$, and then marginalising over $y_2^{(t)}$, should be equivalent to making a prediction over $y_1^{(t)}$ directly. Put concretely, marginalisation consistency requires that the following equality in distribution holds

$$\Theta(u_1^{(t)}, u_2^{(t)})_i \overset{d}{=} \Theta(u_i^{(t)}). \tag{C.10}$$

For a careful choice of Θ and distribution of $(u_1^{(t)}, u_2^{(t)})$ this equality can be true, but in general will not be true. If we want to construct an invertible Θ such that it gives a consistent model for *all* underlying Neural Processes, then that requires the the stronger condition that

$$\Theta(u_1^{(t)}, u_2^{(t)})_i = \Theta(u_i^{(t)}) \quad \text{almost surely.} \tag{C.11}$$

From this condition, we see that Θ can only be a marginal transformation because

$$\Theta(u_1^{(t)}, u_2^{(t)}) = (\Theta(u_1^{(t)}), \Theta(u_2^{(t)})), \tag{C.12}$$

and similarly for other M. Though for a given Neural Process there may exist an appropriate Θ which satisfies marginalisation consistency, constraining Θ to achieve this is a challenging research problem, which we found to be beyond the scope of this paper.

Marginal maps cannot model joint dependencies: We now note that if π is mean-field and Θ is a marginal transformation, then it follows that the $y^{(t)}$ variables are independent. Therefore, it is not possible to compose a mean-field CNP with an NF to model joint dependencies in this way. Instead, we must rely on π for modelling dependencies, and Θ for learning the marginals.

C.2 Gaussian synthetic experiments

Data generation process: Each synthetic task consists of a collection of datasets sampled from the same generative process. To generate each of these datasets, we first determine the number of context and target points. We use a random number between 3 and 50 of context points and a fixed number of 100 target points. For each dataset we sample the inputs of both the context and target points, that is $\mathbf{x}_c, \mathbf{x}_t$ uniformly at random in the region $[-2, 2]$ for the 1D tasks and in $[-2, 2] \times [-2, 2]$ for the 2D tasks. We then sample the corresponding outputs $\mathbf{y}_c, \mathbf{y}_t$ as follows.

Exponentiated Quadratic (EQ): We sample $\mathbf{y}_c, \mathbf{y}_t$ from a GP with an EQ covariance

$$k_{\text{EQ}}(x, x') = \sigma_v^2 \exp\left(-\frac{1}{2\ell^2}(x - x')^2\right),$$

with parameters $(\sigma_v^2, \ell) = (1.00, 1.00)$.

Matern 5/2: We sample $\mathbf{y}_c, \mathbf{y}_t$ from a GP with a covariance

$$k_{\text{M}}(x, x') = \sigma_v^2 \left(1 + \frac{r}{\ell} + \frac{r^2}{3\ell^2}\right) \exp\left(-\frac{r}{\ell}\right), \tag{C.13}$$

where $r = |x - x'|$, with parameters $(\sigma_v^2, \ell) = (1.00, 1.00)$.

Noisy mixture: We sample $\mathbf{y}_c, \mathbf{y}_t$ from a GP which is a sum of two EQ kernels

$$k_{\text{NM}}(x, x') = k_{EQ,1}(x, x') + k_{EQ,2}(x, x'),$$

with the following parameters $(\sigma_{v,1}^2, \ell_1) = (1.00, 1.00)$ and $(\sigma_{v,2}^2, \ell_2) = (1.00, 0.25)$.

Weakly periodic: We sample $\mathbf{y}_c, \mathbf{y}_t$ from a GP which is the product of an EQ and a periodic covariance

$$k_{\mathrm{WP}}(x, x') = k_{EQ}(x, x') \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell_p^2}\right),$$

with EQ parameters $(\sigma_{v,EQ}^2, \ell_p) = (1.00, 1.00)$ and periodic parameters $(p, \ell_{EQ}) = (0.25, 1.00)$.

Lastly, for all tasks we add iid Gaussian noise with zero mean and variance $\sigma_n^2 = 0.05^2$. This noise level was not given to the models, which in every case learned a noise level from the data. We generate training data for 100 epochs, each consisting of 1024 iterations, each of which contains 8 different tasks. For testing, we use a single epoch of 1024 iterations, each of 8 different tasks.

Neural architectures for 1D tasks (GNP, AGNP, ANP): For the GNP model we use a fully connected neural network consisting of an encoder with six hidden layers of 128 units each, a mean aggregation layer and a decoder with a single hidden layer, also with 128 units. For the AGNP model, we use the same architecture as for the GNP model, except the aggregation layer consists of a dot product self-attention layer (Vaswani et al., 2017). For the ANP model, we follow the same architecture which was used in Kim et al. (2019), which uses a *deterministic* and a *stochastic* path in the encoder. The deterministic path consists of the same encoder used in the GNP and AGNP models, while the stochastic path consists of a fully connected network with two hidden layers of 128 units each. The outputs of the deterministic and the stochastic paths are concatenated and passed through the same decoder architecture as for the GNP and the AGNP.

Neural architectures for 1D Gaussian tasks (ConvGNP, ConvNP): We use the same architecture for all the ConvGNP models. This consists of a SetConv layer (Gordon et al., 2020), followed by a pointwise linear transformation with a nonlinearity, a CNN with a UNet architecture (Ronneberger et al., 2015), another linear transformation, and lastly a SetConv layer for producing the features required by the mean-field, `linear` or `kvv` covariance. The first SetConv maps the context set to a discretised grid with 64 points per unit, producing two channels at each point on the grid, referred to as the data and the density channels in Gordon et al. (2020). The pointwise linear transformation maps the two features outputted by the first SetConv to eight features which are fed to the UNet network. The UNet itself consists of six regular convolution layers with a kernel size of 5 and a stride of 2, and channel sizes

$$(c_{\text{in}}, c_{\text{out}}) = (8, 8), (8, 16), (16, 16), (16, 32), (32, 64),$$

followed by six layers of transpose convolutions, again with a kernel size of 5, a stride of 2 and channel sizes

$$(c_{\text{in}}, c_{\text{out}}) = (64, 32), (64, 32), (64, 16), (32, 16), (32, 8), (16, 8).$$

Note the numbers of channels in the transpose convolutions are of the above dimensions because of the additional connections of the UNet architecture. For the ConvNP model, we use a similar architecture as for the ConvGNP, except the model contains two stacked UNet networks. The first SetConv of the ConvNP uses the same discretisation of 64 points per unit, as the ConvGNP. Then, the first UNet of the ConvNP is the same as the UNet of the ConvGNP, except it has twice the number of output channels. Half of these are used as the mean and the other half as the log-variance of 64 independent Gaussian variables, for each position in the convolution grid. This is followed by another UNet with the same architecture as the ConvGNP UNet, followed by a linear transformation and a SetConv layer. The SetConv maps the outputs of the second UNet and a target input to a mean and a log-variance.

Neural architecture for 1D tasks (FullConvGNP): For the FullConvGNP we follow Bruinsma et al. (2021), who use a one-dimensional ConvCNP-like for the predictive mean and another two-dimensional convolutional architecture for the predictive covariance. For the precise algorithm of the FullConvGNP, we refer the reader to Appendix E.2 of Bruinsma et al.. Here we give the specific details of the CNNs used in our implementation. Unlike Bruinsma et al., we use UNet-style architectures for both these CNNs, as opposed to depthwise-separable CNNs, because we find the former to train much quicker both in terms of wall-clock time as well as number of epochs. For the mean parametrisation, we use the same architecture as for the ConvGNP, except this outputs a single feature representing the predictive mean. For the covariance architecture we use a discretisation of 30 points per unit use a UNet consisting of six layer of two-dimensional convolutions, with a kernel size of 5×5, a stride of 2 and the same numbers of channels that are used in the mean parameterisation, namely

$$(c_{\text{in}}, c_{\text{out}}) = (8, 8), (8, 16), (16, 16), (16, 32), (32, 64),$$

followed by six layers of transpose convolutions, also with a kernel size of 5×5 and stride of 2, and channel sizes

$$(c_{\text{in}}, c_{\text{out}}) = (64, 32), (64, 32), (64, 16), (32, 16), (32, 8), (16, 8).$$

Neural architectures for 2D tasks (ConvGNP, ConvNP): We use the same UNet architectures for the 2D tasks as those for the 1D tasks, except the convolutions are now two-dimensional and the discretisation resolution of the SetConv layer is set to 32 points per unit.

General notes: We use the same number of features in the output layer of the GNP, AGNP and ConvGNP models. The mean-field covariance uses two features, one for the mean and one for the marginal variance of the predictive, while the `linear` and `kvv` models both use the same number of $D_g = 512$ features for computing the predictive covariance. We use a learnable homoscedastic noise variable for all models, and ReLU activation functions for all hidden layers. We optimise all models with Adam (Kingma and Ba, 2015), using a learning rate of 5×10^{-4}. For the ConvNP we use 10 latent samples to evaluate the loss during training, and 512 samples during testing. We do not use any weight regularisation.

C.3 Predator-prey synthetic experiments

Data generation process: We broadly follow the method specified in Appendix C.4 of Gordon et al. (2020) for generating data from the Lotka-Volterra model, which uses the algorithm specified in Gillespie (1977). For each time series, we first sample the predator birth and death parameters θ_1, θ_2 and the θ_3, θ_4 from the distributions

$$\theta_1 \sim 1.00 \times 10^{-2} \times \text{Uniform}(1 - \epsilon, 1 + \epsilon), \tag{C.14}$$

$$\theta_2 \sim 5.00 \times 10^{-1} \times \text{Uniform}(1 - \epsilon, 1 + \epsilon), \tag{C.15}$$

$$\theta_3 \sim 5.00 \times 10^{-1} \times \text{Uniform}(1 - \epsilon, 1 + \epsilon), \tag{C.16}$$

$$\theta_4 \sim 1.00 \times 10^{-2} \times \text{Uniform}(1 - \epsilon, 1 + \epsilon), \tag{C.17}$$

where $\epsilon = 0.1$. We choose these parameters following Gordon et al. (2020), because they result in plausible oscillatory as well as transient behaviours of the predator and prey populations. We then initialise the predator and prey populations at $X = 50$ and $Y = 100$ respectively at time $t = 0$, and perform a sequence of discrete steps. At each step one of the following events occur:

1. We sample $\Delta t \sim \text{Exponential}(R^{-1})$, where $R = \theta_1 XY + \theta_2 X + \theta_3 Y + \theta_4 XY$.
2. We sample one of the following events:
   (a) A predator is born with probability $\theta_1 XY/R$, increasing X by 1.
   (b) A predator dies with probability $\theta_2 X/R$, decreasing X by 1.
   (c) A prey is born with probability $\theta_3 Y/R$, increasing Y by 1.
   (d) A prey dies with probability $\theta_4 XY/R$, decreasing Y by 1.
3. Increment $t \leftarrow t + \Delta t$ and repeat until $t = 100$ is reached, or when 10^4 events occur.

Lastly, we linearly interpolate the predator and prey time series and randomly choose context and target points from the range $[0, t]$. We choose a random number of between 1 and 50 context and a fixed number of 100 target points, to produce a dataset. The linear interpolation is performed because otherwise the algorithm yields many more context and target points at regions where R is large, however we would like the input locations of the data to be independent of the event rate R. Lastly we scale the target outputs by a factor of 10^2 and add a positive constant of 10^{-2} to it. The reason for this last positive constant is that strictly speaking the marginal transformations used in the ConvGCNP models are not differentiable at 0, and we circumvent this pathology by adding this constant. We generate training data for 100 epochs, each consisting of 1024 iterations, each of which contains 16 different time series. For testing, we use a single epoch of 1024 iterations, each of 16 different time series.

Neural architectures for the predator-prey tasks (ConvGNP, ConvNP, FullConvGNP): We use the same model architectures for the ConvGNP, ConvNP and FullConvGNP in the predator-prey tasks as those which we used for the 1D synethtic Gaussian experiments, except we modify the discretisation resolution used by the various SetConv layers. More specifically, for the SetConvs of the ConvGNP, the ConvNP and that used in the mean parameterisation of the FullConvGNP, we use a discretisation of 16 points per time unit. For the SetConv used in the covariance architecture of the FullConvGNP we use 8 points per unit.

Neural architectures for the ConvGCNPs: For the ConvGCNPs we use identical architectures as for their ConvGNP counterparts, except we also compose the model with a marginal transformation. More specifically, we use a marginal transformation $\Phi_M^{-1}(\Phi_G(\cdot), \boldsymbol{\psi})$, where Φ_M is the CDF of the exponential distribution

$$\Phi_M(u) = 1 - e^{u/\psi},$$

where $\psi(\cdot) = \psi(\mathbf{x}^{(c)}, \mathbf{y}^{(c)}, \cdot)$ is an additional feature outputted by the ConvGNP. To avoid numerical instabilities during training, we limit ψ to the range $[1, \infty)$, by passing the corresponding raw feature outputted by the ConvGNP through a SoftPlus function and adding 1 to the result.

General notes: We use the same number of features in the output layer of the ConvGNP models. The mean-field covariance uses two features, one for the mean and one for the marginal variance of the predictive, while the `linear` and `kvv` models both use the same number of $D_g = 32$ features for computing the predictive covariance. We use a learnable heteroscedastic noise variable for all models, and ReLU activation functions for all hidden layers. We optimise all models with Adam, using a learning rate of 5×10^{-4}. For the ConvNP we use 16 latent samples to evaluate the loss during training, and 512 samples during testing. We do not use any weight regularisation.

C.4 Electroencephalogram experiments

Details on datasets: For the EEG experiments we use the publicly available EEG dataset, available on the UCI Datasets[1] website. We preprocess the data to remove invalid time series, leaving us with 7632 time series across 106 subjects. We pool all 106 subjects together (both control and alcoholic subjects) and sample 86 of the subjects for training, 10 for validation and 10 for testing. Each time series consists of 256 equispaced measurements over 64 EEG channels, from which we keep the seven channels with names `FZ`, `F1`, `F2`, `F3`, `F4`, `F5`, `F6`, following Bruinsma et al. (2020).

Details on datasets for training the meta-models (ConvGNP, ConvNP): We train each of our meta-models for 1000 epochs, each consisting of 256 iterations, at each of which the model is presented with a batch 16 different tasks. To generate each task, we first select a window size W between 1 and 50. We then choose a window of size W from the 256-long time series, and set the channels `FZ`, `F1`, `F2` within this window as context points. All other channels in this window, as well as all seven channels outside this window are used as context points.

Details on datasets for testing the models (ConvGNP, ConvNP, MOGP): During the testing phase of the meta-models, and the training phase of the MOGP, we use the same procedure for generating datasets as that used during the training phase of the meta-models, except we set the window size to a constant $W = 50$. We evaluate the models on the held-out test set. For the meta-models, we use 1 testing epoch of 256 iterations, each of which consists of presenting 16 tasks to the model. The

[1]https://kdd.ics.uci.edu/databases/eeg/eeg.data.html

MOGP model is trained and tested on individual time series without a meta-learning component, and is used as a non meta-learnt baseline to illustrate the benefits of meta-learning. We use 500 randomly sampled tasks, selected in the same way as the datasets used to test the meta-models. For each task, we fit the hyperparameters and the mixing matrix parameters on the context set of each dataset individually, and report the normalised predictive log-likelihood on the target set.

Neural architectures for the EEG tasks (ConvGNP, ConvNP): In these EEG experiments we use an on-the-grid ConvCNP architecture for the ConvGNP model, as specified in Gordon et al. (2020). More specifically, an on-the grid architecture does not require an input SetConv architecture, but instead uses a convolutional layer with positive-constrained weights. Following this convolution, a linear pointwise transformation is applied, followed by a UNet CNN, with six convolution and six transpose convolution layers. All layers use a kernel size of 5 and a stride of 2. The numbers of channels for the six convolution layers are

$$(c_{in}, c_{out}) = (16, 32), (32, 64), (64, 128), (128, 256), (256, 512), (512, 1024)$$

while for the transpose convolutions, the channels are

$$(c_{in}, c_{out}) = (1024, 512), (1024, 256), (512, 128), (256, 64), (128, 32), (64, 32)$$

followed by a last linear transformation mapping the output of a CNN to the dimensionality expected by the ConvGNP model. An output SetConv is used to interpolate the features to off-the-grid locations for purposes such as sampling. For the ConvNP, we follow the same approach as for the synthetic experiments, and stack two UNets with a latent variable layer in between. These are then followed by a SetConv for making off-the-grid predictions.

General notes for the meta-models (ConvGNP, ConvNP): We optimise both the ConvGNPs and the ConvNP using Adam and a learning rate of 2×10^{-4}. We use early stopping, selecting the model snapshot with the best performance on the validation set, through all of the 1000 training epochs, as the final model. In no case did we observe overfitting to the training set, even though we used no weight decay. In our experiments, we used $D_g = 512$ features for the `linear` and `kvv` models.

Details of the MOGP model: To demonstrate the benefits of the meta-learning approach, we also compare with a baseline multi-output Gaussian process (MOGP) model which, for every task separately, is trained from scratch, without a meta-learning component. To accelerate the training of many MOGP models, we use the approach by Bruinsma et al. (2020); an implementation is openly available on GitHub (link[2]). The MOGP models have $p = 7$ outputs and $m = 3$ latent processes with EQ kernels with initial length scales 10^{-2}. All noises are initialised to 10^{-2}. The mixing matrix and all hyperparameters are optimised using `scipy`'s implementation of L-BFGS-B.

C.5 Environmental experiments

C.5.1 Experimental design

Experiments are conducted within the VALUE framework (Maraun et al., 2015) to facilitate comparison to existing downscaling methods. VALUE provides a suite of experiments for temperature and precipitation downscaling in an idealised framework mapping from two-degree resolution gridded reanalysis observations to station observations for daily maximum temperature at 86 locations around Europe. Models are trained on data from 1979-2003 and evaluated on data from 2003-2008. This mapping is then able to be applied to low-resolution climate model output to generate high resolution future projections for downstream tasks. Formally, we predict temperature y at target (longitude, latitude) location $\mathbf{x}$ given low resolution predictors at two degrees resolution and orographic data at $\mathbf{x}$. We consider three experiments:

1. Europe value only: models are trained on context data from ERA-Interim reanalysis from 1979-2002 and the 86 VALUE stations, and evaluated at these same locations for 2003-2008. The purpose of this experiment is to exactly reproduce the VALUE experiment protocol to facilitate comparison to state of the art baselines.

2. Europe all: In practice, a key limitation of current downscaling models is that they can only make predictions at a discrete set of locations determined at training time. Neural process models offer a significant advantage as the posterior stochastic process can be queried at an arbitrary location at inference time, regardless of the availability of training data. To evaluate how well the GNP models perform making predictions at new locations in the validation period we design a second experiment where models are trained on context data from ERA-Interim reanalysis from 1979-2002 and 3043 target stations around Europe, and evaluated on context data from 2003-2008 at the 86 VALUE stations.

[2]https://github.com/wesselb/oilmm

3. Germany only: Germany has the highest density of stations of any country in Europe. As this is the scenario where modelling correlations between targets is most likely to improve performance we conduct a final experiment limiting the domain to Germany. Training is on ERA-Interim reanalysis context data from 1979-2002 and station data from 689 stations around Germany, with evaluation from 2003-2008 on 250 held out stations.

C.5.2 Data

Consistent with the VALUE protocol, we use the following context and target data:
Context data. Context data are taken from ERA-Interim reanalysis (Dee et al., 2011) from 1979-2008 with daily temporal resolution and spatial resolution interpolated to two degrees using bilinear interpolation. We consider 25 variables:

- Surface: maximum temperature, mean temperature, northward and eastward wind.
- Upper atmosphere (850/700/500 hPa): specific humidity, temperature, northward and eastward wind.
- Invariant: angle of sub-grid scale orography, anisotropy of sub-grid scale orography, standard deviation of filtered subgrid orography, standard deviation of orography, geopotential, longitude, latitude and day of year transformed to $(\cos(t), \sin(t))$.

To account for sub-grid-scale topography, the context set also includes topographic variables at each target location. These include the true elevation, difference between reanalysis gridscale and true elevation and mTPI (Theobald et al., 2015).

Target data. Target data are taken from weather station observations from 3129 weather stations around Europe reported as part of the European Climate Assessment Dataset (Klein Tank et al., 2002). This dataset includes daily observations of maximum temperature from 1979-2008.

C.5.3 Neural architectures and training

For the architectures we follow (Vaughan et al., 2022). The encoder consists of a CNN with six residual blocks, each consisting of two layers of depth-separable convolutions (Chollet, 2017) with a kernel size of 3 and 128 channels followed by ReLU activations. The encoder is followed by a SetConv layer with RBF kernel. Finally, a MLP is used to update predictions given the elevation of the target locations. This MLP consist of four hidden layers each with 64 units and ReLU activations. For the ConvGNP-`linear` and ConvGNP-`kvv` models we use $D_g = 128$ features for computing the covariance. For the ConvNP we use 32 channels for the latent function and 24 samples to calculate the neural process likelihood at training time. Each model is trained for 500 epochs.

C.5.4 Additional samples

This section shows additional examples similar to Figure 5.10 comparing the ConvCNP and ConvGNP samples for different days.

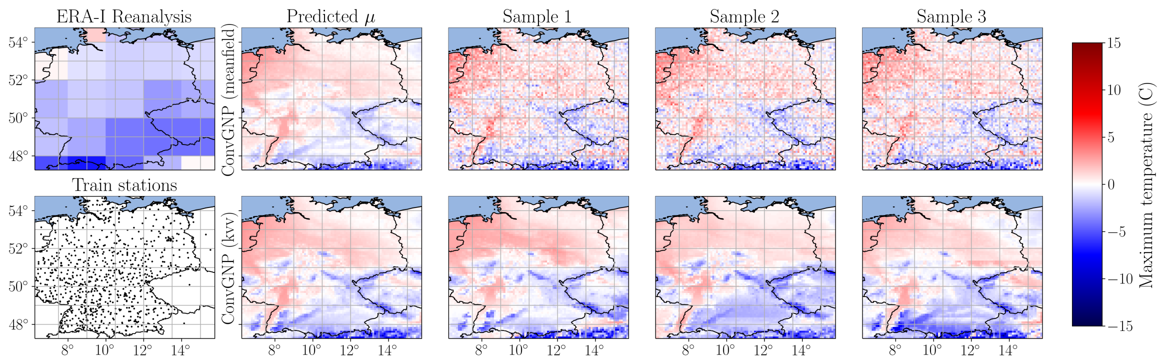


Fig. C.1 As for Figure 9, but for 12/01/2003.

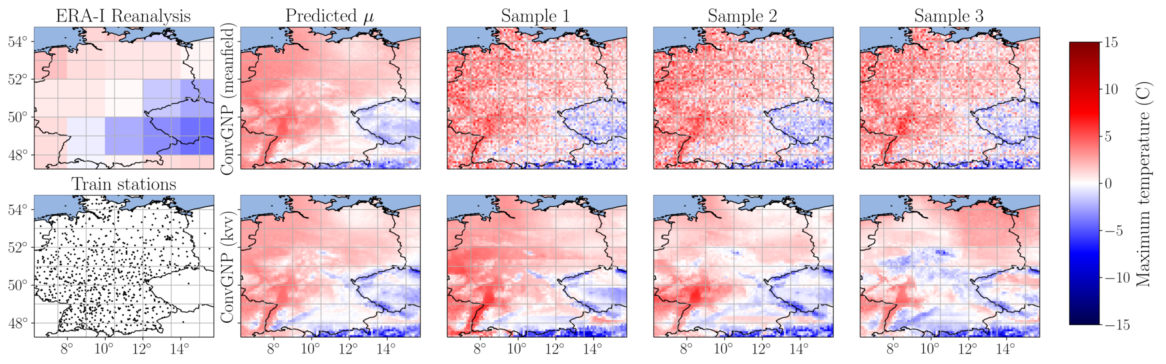


Fig. C.2 As for Figure 9, but for 24/02/2003.

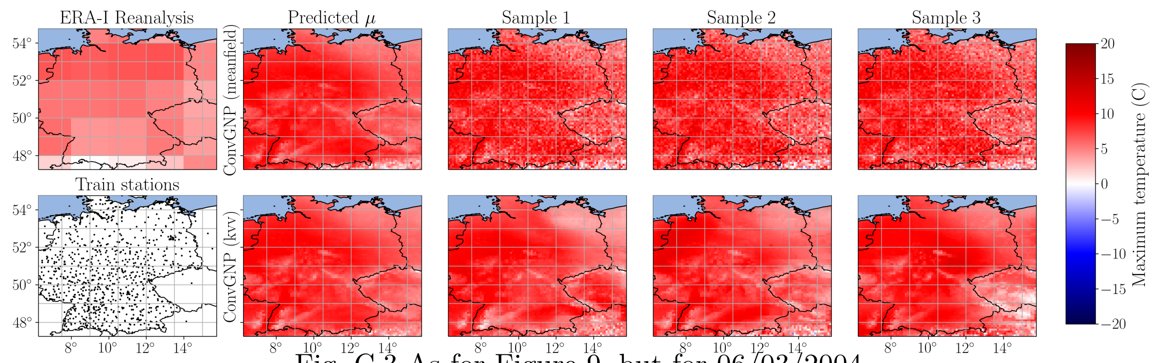


Fig. C.3 As for Figure 9, but for 06/03/2004.

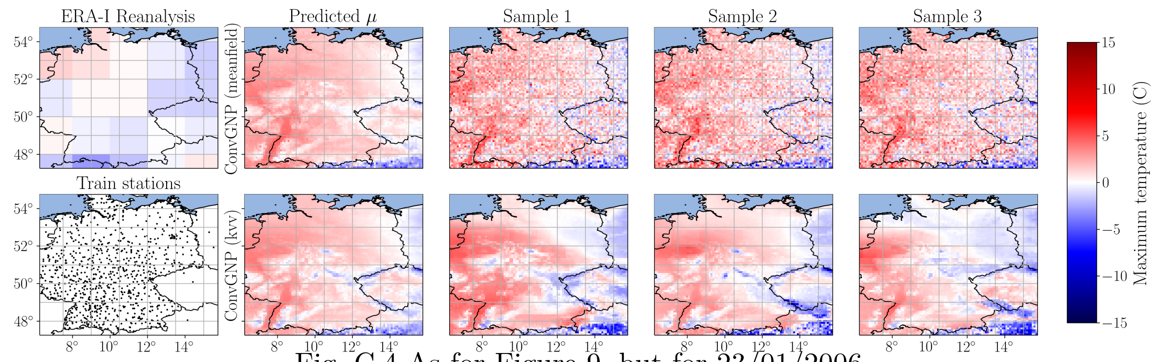


Fig. C.4 As for Figure 9, but for 23/01/2006.

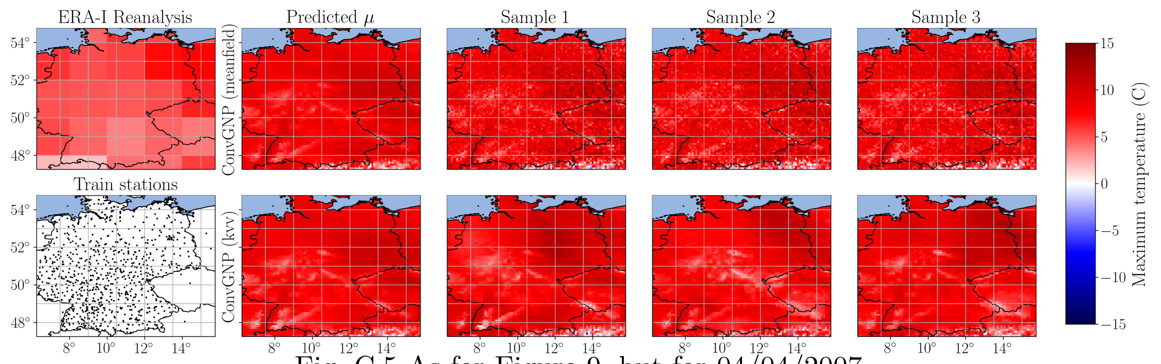


Fig. C.5 As for Figure 9, but for 04/04/2007.

Appendix D

Chapter 6 Supplementary Material

D.1 Proof of Proposition 2

Additional notation. If $\boldsymbol{y}_1 \oplus \boldsymbol{y}_2 \sim P_{\boldsymbol{x}_1 \oplus \boldsymbol{x}_2}\pi(D)$, then denote the distribution of $\boldsymbol{y}_1 \,|\, \boldsymbol{y}_2$ by $P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi(D)$. Note that $P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi(D)$ depends on $\boldsymbol{y}_2$, because it is the distribution of $\boldsymbol{y}_1 \,|\, \boldsymbol{y}_2$, even though the notation does not make this dependence explicit.

The "appropriate regularity conditions". Let $\mathcal{P}^N_\lambda$ be the collection of distributions on $\mathbb{R}^N$ that (a) have a density with respect to the Lebesgue measure and (b) have a covariance matrix which is strictly positive definite. Let $\mathcal{P}^N_{\lambda,\mathrm{G}} \subseteq \mathcal{P}^N_\lambda$ be the subcollection of distributions which are Gaussian. Then, by Corollary B.1 by Bruinsma et al. (2021), for all $\mu \in \mathcal{P}^N_\lambda$ such that $\inf_{\nu \in \mathcal{P}^N_{\lambda,\mathrm{G}}} \mathrm{KL}(\mu, \nu) < \infty$,

$$\arg\min\nolimits_{\nu \in \mathcal{P}^N_{\lambda,\mathrm{G}}} \mathrm{KL}(\mu, \nu) = \mathcal{N}(\mu) \tag{D.1}$$

where $\mathcal{N}(\mu)$ denotes the Gaussian distribution with mean vector and covariance matrix equal to those of μ.

In the proposition, by appropriate regularity conditions on y, we mean the assumption that, for all inputs $\boldsymbol{x}$ and $D \in \mathcal{D}$, $P_{\boldsymbol{x}}\pi_y(D)$ is in $\mathcal{P}^{|\boldsymbol{x}|}_\lambda$ and such that $\inf_{\nu \in \mathcal{P}^{|\boldsymbol{x}|}_{\lambda,\mathrm{G}}} \mathrm{KL}(P_{\boldsymbol{x}}\pi_y(D), \nu) < \infty$.

Assume the appropriate regularity conditions on y. We now list three technical observations.

1. Note that $P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi_y(D)$ is the distribution of $y(\boldsymbol{x}_1) \,|\, D, (\boldsymbol{x}_2, \boldsymbol{y}_2)$, so we have the identity $P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi_y(D) = P_{\boldsymbol{x}_1}\pi_y(D \oplus (\boldsymbol{x}_2, \boldsymbol{y}_2))$. Therefore, for all inputs $\boldsymbol{x}_1$, inputs $\boldsymbol{x}_2$, and $D \in \mathcal{D}$, $P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi_y(D)$ is in $\mathcal{P}^{|\boldsymbol{x}_1|}_\lambda$ and such that $\inf_\nu \mathrm{KL}(P_{\boldsymbol{x}_1 \,|\, \boldsymbol{x}_2}\pi_y(D), \nu) < \infty$.

2. The ideal CNP π_{C} matches the means and marginal variances of the true posterior predictives (Section 6.2). Hence, for all $x \in \mathcal{X}$ and $D \in \mathcal{D}$, $P_x\pi_{\mathrm{C}}(D)$ is in $\mathcal{P}^1_{\lambda,\mathrm{G}}$.

3. The ideal GNP π_{G} matches the mean vectors and covariance matrices of the true posterior predictives (Section 6.2). Hence, for all inputs $\boldsymbol{x}$ and $D \in \mathcal{D}$, $P_{\boldsymbol{x}}\pi_{\mathrm{G}}(D)$ is in $\mathcal{P}^{|\boldsymbol{x}|}_{\lambda,\mathrm{G}}$; which means that, for all $x_1 \in \mathcal{X}$, inputs $\boldsymbol{x}_2$, and $D \in \mathcal{D}$, $P_{x_1\,|\,\boldsymbol{x}_2}\pi_{\mathrm{G}}(D)$ is in $\mathcal{P}^1_{\lambda,\mathrm{G}}$.

In the proof, to apply and use equation D.1, we implicitly use these observations.

Proof of Proposition 2. Let $\boldsymbol{x}$ be some inputs and let $D \in \mathcal{D}$ be some data set. We will argue that, for all $n = 1, \ldots, |\boldsymbol{x}|$,

$$\begin{aligned}&\mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), P_{x_n}\pi_{\mathrm{C}}(D \oplus (\boldsymbol{x}_{1:(n-1)}, \boldsymbol{y}_{1:(n-1)})))\\ &\qquad\le \mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_{\mathrm{G}}(D)).\end{aligned} \tag{D.2}$$

Assuming this inequality, the result follows directly from the chain rule for the KL divergence in combination with the definition of $\mathrm{AR}_{\boldsymbol{x}}$ (Procedure 6.2.1):

$$\begin{aligned}&\mathrm{KL}(P_{\boldsymbol{x}}\pi_y(D), \mathrm{AR}_{\boldsymbol{x}}(\pi_{\mathrm{C}}, D))\\ &\quad= \textstyle\sum_{n=1}^{|\boldsymbol{x}|} \mathbb{E}_{\boldsymbol{y}_{1:(n-1)}}[\mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), P_{x_n}\pi_{\mathrm{C}}(D \oplus (\boldsymbol{x}_{1:(n-1)}, \boldsymbol{y}_{1:(n-1)})))] && \text{(D.3)}\\ &\quad\le \textstyle\sum_{n=1}^{|\boldsymbol{x}|} \mathbb{E}_{\boldsymbol{y}_{1:(n-1)}}[\mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_{\mathrm{G}}(D))] && \text{(D.4)}\\ &\quad= \mathrm{KL}(P_{\boldsymbol{x}}\pi_y(D), P_{\boldsymbol{x}}\pi_{\mathrm{G}}(D)) && \text{(D.5)}\end{aligned}$$

where the expectations are over $\boldsymbol{y}_{1:(n-1)} \sim P_{\boldsymbol{x}_{1:(n-1)}}\pi_y(D)$. To prove the inequality, note that, conditional on $\boldsymbol{y}_{1:(n-1)}$, using equation D.1,

$$\arg\min\nolimits_{\nu\in\mathcal{P}^1_{\lambda,\mathrm{G}}} \mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), \nu) = \mathcal{N}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D)). \tag{D.6}$$

By the property of π_{C} that it matches the mean and marginal variance of the true posterior (Section 6.2),

$$\begin{aligned}\mathcal{N}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D)) &= \mathcal{N}(P_{x_n}\pi_y(D \oplus (\boldsymbol{x}_{1:(n-1)}, \boldsymbol{y}_{1:(n-1)}))) && \text{(D.7)}\\ &= P_{x_n}\pi_{\mathrm{C}}(D \oplus (\boldsymbol{x}_{1:(n-1)}, \boldsymbol{y}_{1:(n-1)})). && \text{(D.8)}\end{aligned}$$

Therefore,

$$\underset{\nu\in\mathcal{P}^1_{\lambda,\mathrm{G}}}{\arg\min}\ \mathrm{KL}(P_{x_n\,|\,\boldsymbol{x}_{1:(n-1)}}\pi_y(D), \nu) = P_{x_n}\pi_{\mathrm{C}}(D \oplus (\boldsymbol{x}_{1:(n-1)}, \boldsymbol{y}_{1:(n-1)})). \tag{D.9}$$

Noting that $P_{x_n \,|\, \boldsymbol{x}_{1:(n-1)}} \pi_{\mathrm{G}}(D) \in \mathcal{P}^1_{\lambda,\mathrm{G}}$, we obtain the desired inequality. $\square$

D.2 Proof of Proposition 3

Proof of Proposition 3. Consider the increasing filtration $\mathcal{F}_n = \sigma(y_1, \ldots, y_n)$ with limit $\mathcal{F}_\infty = \sigma(\bigcup_{n=1}^\infty \mathcal{F}_n)$. Also let $\mathcal{T}_n = \sigma(\varepsilon_{n+1}, \varepsilon_{n+2}, \ldots)$ and consider the tail σ-algebra $\mathcal{T} = \bigcap_{n=1}^\infty \mathcal{T}_n$. Let $(x_{n_i})_{i=1}^\infty$ be a subsequence of $(x_n)_{n=1}^\infty$ such that $x_{n_i} \to x^*$. Let $g_n = \frac{1}{n} \sum_{i=1}^n y_i$. Since g_n is a function of $y_1, \ldots, y_n$, it is $\mathcal{F}_n$–measurable and therefore $\mathcal{F}_\infty$–measurable. Note that

$$g_n = \frac{1}{n} \sum_{i=1}^n f(x_{n_i}) + \frac{1}{n} \sum_{i=1}^n \varepsilon_i. \tag{D.10}$$

By sure continuity of f, the first term converges to $f(x^*)$ surely. By the strong law of large numbers (Example 5.6.1; Durrett, 2010), the second term converges to zero on a tail event $A \in \mathcal{T}$ of probability one. We conclude that $\mathbb{1}_A f(x^*)$ is $\sigma(\mathcal{F}_\infty, \mathcal{T})$–measurable. Therefore, by almost sure convergence of L^2–bounded martingales (Theorem 5.4.5; Durrett, 2010),

$$\begin{aligned}
\lim_{n\to\infty} \mathbb{E}[y(x^*) \,|\, y_1, \ldots, y_n] &= \lim_{n\to\infty} \mathbb{E}[f(x^*) \,|\, y_1, \ldots, y_n] && (\mathbb{E}[\varepsilon_0] = 0) && \text{(D.11)} \\
&= \lim_{n\to\infty} \mathbb{E}[f(x^*) \,|\, \mathcal{F}_n] && (\text{definition of } \mathcal{F}_n) && \text{(D.12)} \\
&= \lim_{n\to\infty} \mathbb{E}[f(x^*) \,|\, \mathcal{F}_n, \mathcal{T}] && (\sigma(f(x^*), \mathcal{F}_n) \perp \mathcal{T}) && \text{(D.13)} \\
&= \lim_{n\to\infty} \mathbb{E}[\mathbb{1}_A f(x^*) \,|\, \mathcal{F}_n, \mathcal{T}] && (\mathbb{P}(A) = 1) && \text{(D.14)} \\
&= \mathbb{E}[\mathbb{1}_A f(x^*) \,|\, \mathcal{F}_\infty, \mathcal{T}] && (L^2\text{–mart. convergence}) && \text{(D.15)} \\
&= \mathbb{1}_A f(x^*) && (\mathbb{1}_A f(x^*) \in \sigma(\mathcal{F}_\infty, \mathcal{T})) && \text{(D.16)} \\
&= f(x^*), && (\mathbb{P}(A) = 1) && \text{(D.17)}
\end{aligned}$$

where all equalities hold almost surely. $\square$

D.3 Illustration of the AR procedure

Figure D.1 depicts the AR sampling procedure (Procedure 6.2.1) and procedure to produce smooth samples (Proposition 3) using the ConvCNP trained on the EQ data process from Section 6.4.1.

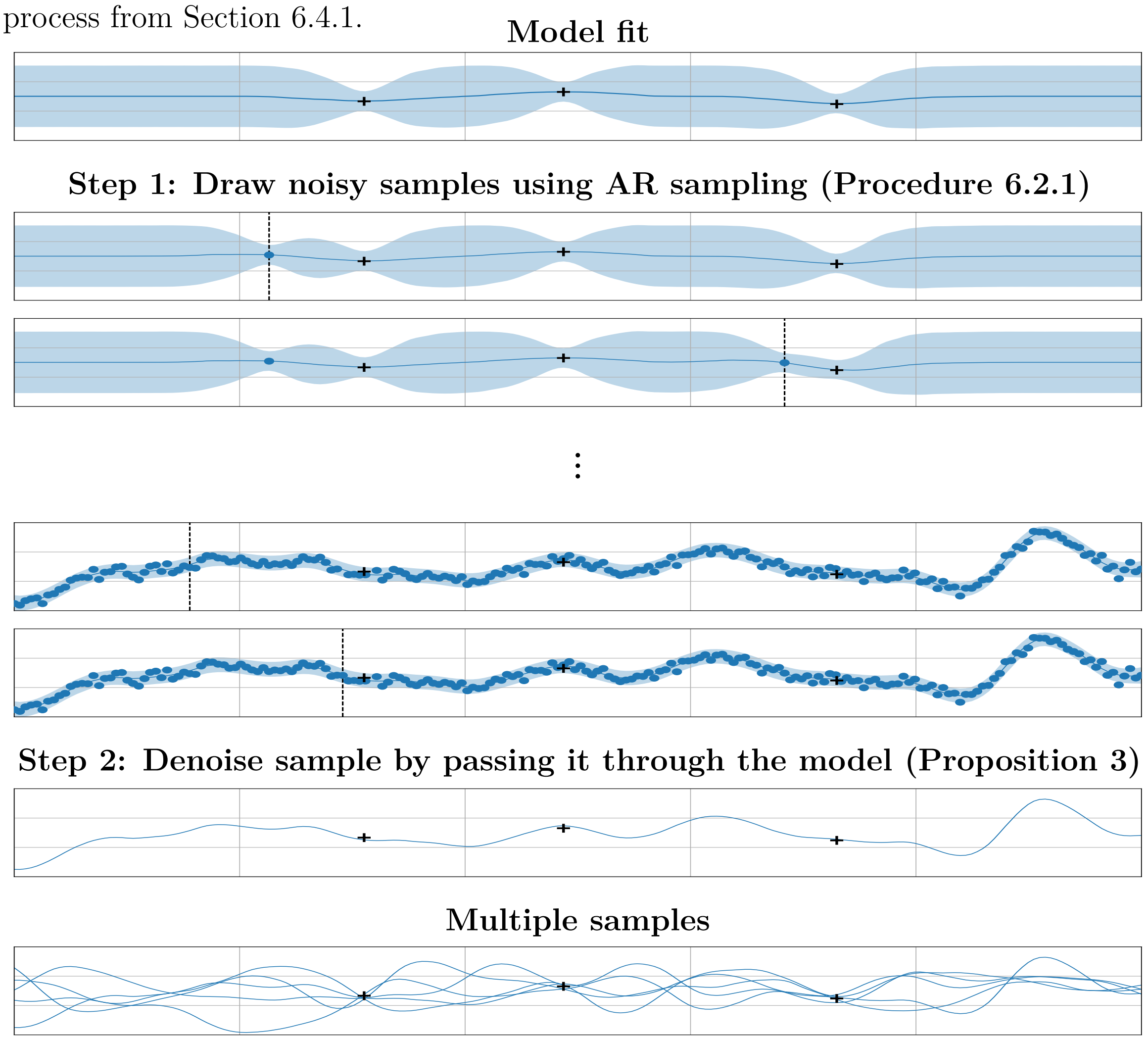


Fig. D.1 Illustration of the AR procedure with a random AR ordering and the de-noising step (Procedure 6.2.1 and Proposition 3), to produce smooth samples. Given a context set (black crosses), we can use the CNP to get marginal predictions at arbitrary input locations (first figure). We choose a randomly sampled input location, draw a corresponding output sample from the model's predictive (blue dot in the second plot), append this to the context set, and pass the augmented context set through the model again. We repeat this step a number of times (third and fourth figures), until all function (epistemic) uncertainty has been removed and all that remains is irreducible noise (aleatoric) uncertainty (fifth figure). This procedure yields noisy function samples (blue dots in the sixth plot), which we pass one last time through the model to obtained a denoised sample, treating the mean prediction as an approximate noiseless sample (seventh figure). Repeating this procedure yields high-quality samples from the model predictive (eighth figure).

D.4 Number and Order of Target Points

When deploying a conditional neural process (CNP) autoregressively (AR; Procedure 6.2.1), the number and ordering of the target points matters. In this appendix, we describe our observations of the effects of the number and ordering of the target points on the quality of the predictions. In short, our recommendation is to choose a different random ordering for every sample, and to not let the number/density of target points exceed that at training time.

D.4.1 Effects of the Number of Target Points

During the AR sampling procedure, the AR CNP is evaluated at context sets of increasing size. Our experience is that, as long as the sizes of these context sets do not exceed the sizes seen at training time, the predictions should not be significantly affected by changes in the number of target points. However, if the AR sampling procedure evaluates the model at context sets of larger sizes than seen during training time, then that presents the model with an out-of-distribution situation. What happens then comes down to how well the neural networks generalise. Our experience is that the predictions quickly start to break down.

A notable exception of this rule of thumb are convolutional-deep-set–based models, such as the Convolutional Conditional Neural Process (ConvCNP; Gordon et al., 2020). For these models, the *magnitude of the density channel* is what determines whether the models generalises or not. This means that it is not the total number of points that matters, but rather the *density* of the points. Therefore, the AR ConvCNP can be evaluated at arbitrarily many target points, as long as the density of these points does not significantly exceed the density of context points seen at training time. Once the density exceeds the density of the training data, the model is presented with an out-of-distribution situation, and what happens then again comes down to how well neural networks generalise.

Figure Section D.4.1 illustrates this observation. When the density target points does not exceed the training data (50 and 100 points), the predictions look calibrated. However, once the density of target points comes close or exceed the training data (200, 500, and 1000 points), bias starts to creep into the predictions.

Although the number/density of points in the AR sampling procedure should not exceed that at training time, AR CNPs can still produce high-quality samples at arbitrarily many target points by following the trick outlined at the end of the two-step procedure below Proposition 3.

D.4.2 Effects of the Ordering of Target Points

Our experience is that, as long as the number of target points (or density) does not exceed that at training time, the ordering of the target point does not really matter. Section D.4.1 also demonstrates this. When the density of the target points does not exceed the training data (50 and 100 points), sampling randomly or left to right does not really matter. However, once the density of the target points comes close to or exceeds the training data (200, 500, and 1000 points), we observe a difference in performance between sampling randomly and sampling left to right. Across all numbers of target points, a random ordering seems to perform most robustly. Our recommendation is therefore to choose a different random ordering of the target points for every sample.

D.4.3 Analysis of AR CNPs for CNPs with Gaussian Marginals

In this subsection, we argue that, for CNPs with Gaussian marginals, predictions in the first few AR steps might be poor, but predictions in later AR steps tend to be more accurate. Choosing a different random ordering for every sample therefore "averages out" the effects from these first few AR steps.

When evaluating a CNP with Gaussian marginals in AR mode, every conditional prediction in the AR process is Gaussian. Let us consider the process of producing an AR sample. For the first target input x_1, we run the CNP forward to obtain a distribution for the corresponding target output y_1. In reality, the true posterior most likely is non-Gaussian, which means that the prediction for the first target point may be poor. Nevertheless, we sample this Gaussian, append the sample (x_1, y_1) to the context set, and run the CNP forward again. Because we now feed the earlier sample y_1 through the non-linear network, the marginal predictive for the next target output y_2 (having integrated out y_1) is non-Gaussian. As we perform more AR steps, the marginal predictions of later points become increasingly non-Gaussian, increasing the model's flexibility.

We see that, for a given ordering of the target inputs, the prediction for the first target input is likely poor (because it is Gaussian), and (in the best case) the predictions become more and more accurate as we take more AR steps (because they become more and more non-Gaussian). This is exactly what is happening in Figure 6.3: the left prediction is Gaussian and therefore a poor approximation, and, as we go to the right and take more and more AR steps, the prediction becomes more and more non-Gaussian and therefore more accurate. If we were to feed the target inputs in right to left, then the same phenomenon would happen. The right prediction would be a Gaussian and a very poor approximation, and, as we go to the left and take more AR steps, the prediction would become more non-Gaussian and therefore more accurate.

More generally, for a given ordering of the target points, the ordering will produce high quality predictions if the conditional distributions of the AR factorisation match the corresponding conditional distributions of the true posterior. Since the conditionals of the AR CNP are typically Gaussian by design, this means that the ordering is "good" if the corresponding conditionals of the true posterior are close to Gaussian.

So when is a conditional of the posterior close to Gaussian? Let us assume that the true underlying process is a sum of a non-Gaussian process (constituting epistemic uncertainty) and independent Gaussian noise (constituting aleatoric uncertainty). Generally, a conditional will have both epistemic and aleatoric uncertainty, so a Gaussian will be a bad fit. *However*, as we condition the conditionals of the true generative process on more and more data, the underlying function will be pinned down more and more accurately, meaning that the conditional will consist mostly of aleatoric uncertainty, which is Gaussian. Therefore, as we condition on more and more data, we expect the conditionals to become more and more Gaussian. This again suggests that the samples in the first few AR steps might be a poor fit (because the corresponding conditionals of the true posterior are not yet Gaussian), but that samples in later AR steps should be a better fit (because the corresponding conditionals are then close to Gaussian).

To summarise, an ordering of the target points is "good" if the corresponding conditionals of the true posterior are also close to Gaussian. Under the assumption that the ground-truth process is a non-Gaussian process with additive Gaussian noise, conditionals tend to be close to Gaussian if they are conditioned on many data points. As a consequence, the earlier conditionals in the AR factorisation tend to be poor fits to the ground-truth posterior, whereas later conditionals tend to produce better fits. Choosing a different random ordering for every sample therefore "averages out" the effects from the first few AR steps.

D.4.4 Effect of the random ordering on the spread of the log-likelihood

We have thus far argued for the benefit of using random ordering in AR, due to the robustness it provides. However, one issue with random orderings is that, since different random orderings do not in general give rise to the same predictive distribution, we may obtain different predictive log-likelihoods in practice, depending on the exact random ordering that we sample. Ideally, we would like not only the mean predictive log-likelihood (averaged out over orderings) to be high, but also the standard deviation of the log-likelihood (due to, again, different random orderings) to be small. In other words, we would like the model to perform well regardless of the random ordering which we happen to sample.

At this point, note that if the true underlying process is Gaussian, then a sufficiently well-trained AR CNP with Gaussian marginals would have a small such spread in the log-likelihood, because all conditional predictions of the model will be close to the ground truth conditional predictions. Consequently the order with which we make predictions will have a small effect on the log-likelihood, resulting in a small spread of predictive log-likelihood values. Consider for example the case where the conditionals of the CNP exactly match the conditionals of the true process. In this case, there will be zero variance in the predictive log-likelihood of the process under different orderings. However, the situation is different when the ground truth is non-Gaussian. In this case, as we explained in the previous section, the conditionals of the first few target points may be highly non-Gaussian under the true process, while those of the AR CNP are Gaussian. In this case, we may get different log-likelihoods depending on the random order that we happen to sample.

Figure D.3 provides a quantitative illustration of the above point. In this figure, we show the standard deviation in the per datapoint predictive log-likelihood of an AR CNP (due to different random orderings) on two variants of a task with sawtooth data. On the first variant, we always pass an empty context set to the model (blue), and on the other task, we pass non-emmpty context sets with randomly sampled number of context points, uniformly distributed between 0 and 100 (red). We observe that for empty contexts (blue), we get a relatively large standard deviation in predictive log-likelihood for the first few target points. This likely happens because, initially, the model may randomly pick a target input where the conditional of the true process is highly non-Gaussian (making a poor prediction), or it might pick a target input where the true conditional is Gaussian (making a good prediction). This results in a larger variance in performance for the first few target points. However, as more target points are introduced, the standard deviation shrinks. This is because the conditionals of the true process become increasingly Gaussian, which means that no matter which target input is picked next, the model will approximate the true conditional accurately using a Gaussian, thereby reducing the impact of the ordering of subsequent points on the variance of the log likelihood. Further, introducing a relatively modest number of initial context points (red) in a second variant of the task, substantially reduces the spread in the predictive log-likelihoods. This is again because conditioning on a context set means that the conditionals of the true process are better approximated by Gaussians, reducing the impact that different random orderings have on the spread of the log-likelihood. In practice, in our experiments, we have found the variance in the log-likelihood to be near-zero for Gaussian or Gaussian-like ground truth processes, and larger, but acceptable, for non-Gaussian tasks.

D.5 Details for Figure 6.3

The generative process visualised in the top panel of figure 6.3 is defined by the following mixture distribution:

$$p_{\text{true}}(y \,|\, x) = a_1 \cdot \mathcal{N}(f_1(x), 1) + a_2 \cdot \mathcal{N}(f_2(x), 1) + a_3 \cdot \mathcal{N}(f_3(x), 1). \tag{D.18}$$

Given this mixture distribution, the (Gaussian) ideal CNP can be computed in closed form by computing the first two moments of p_{true}:

$$p_{\text{CNP}}(y \,|\, x) = \mathcal{N}(\mu(x), \sigma(x)^2) \tag{D.19}$$

where

$$\mu(x) = \sum_{i=1}^{3} a_i f_i(x) \tag{D.20}$$

$$\sigma(x)^2 = \sum_{i=1}^{3} a_i \left(1 + f_i(x)^2\right) - \left(\sum_{i=1}^{3} a_i f_i(x)\right)^2. \tag{D.21}$$

The updated mixture weights for the posterior distribution $p_{\text{true}}(y \,|\, x, D^{(c)})$ given a context set $D^{(c)}$ can be computed via Bayes rule and $p_{\text{CNP}}(y \,|\, x, D^{(c)})$ can be computed given the updated mixture weights. Note that in figure 6.3 the prior mixture weights are $a_1 = a_3 = 0.25$ and $a_2 = 0.5$, means are given by

$$f_1(x) = x^2 + 1, \tag{D.22}$$

$$f_2(x) = x, \tag{D.23}$$

$$f_3(x) = -x, \tag{D.24}$$

and the target locations are $x = 1, 2, 4$, and 6. The bottom four panels of figure 6.3 show squared exponential kernel density estimates of 30 000 samples drawn from the generative distribution $p_{\text{true}}(y_1, y_2, y_4, y_6)$, the ideal CNP $p_{\text{CNP}}(y_1, y_2, y_4, y_6)$, and the ideal CNP applied in AR mode from left to right

$$p_{\text{AR CNP}}(y_1, y_2, y_4, y_6) = p_{\text{CNP}}(y_1) \cdot p_{\text{CNP}}(y_2 \,|\, y_1) \cdot p_{\text{CNP}}(y_4 \,|\, y_1, y_2) \cdot p_{\text{CNP}}(y_6 \,|\, y_1, y_2, y_4). \tag{D.25}$$

D.6 Description of Models

The architectures follow the descriptions from the respective papers they are introduced. Although these descriptions are for one-dimensional inputs and outputs, the architectures are readily generalised to multidimensional inputs and outputs; we will explicitly mention wherever that generalisation requires extra care. All architectures use ReLU activation functions. All GNPs, in addition to a covariance matrix over the target points, also output heterogeneous observation noise along the marginal means; the total covariance over the target points is thus the sum of the covariance by the model and a diagonal matrix formed from these observation noises.

Conditional neural process (CNP; Garnelo et al., 2018a). Set the dimensionality of the encoding to $K = 256$. Parametrise the encoder with a three-hidden-layer multi-layer perceptron (MLP) of width 256; and parametrise the decoder with a six-hidden-layer MLP of width 256. For multidimensional outputs, let the decoder have width 512. For multidimensional outputs where outputs can have context points at different inputs, produce a separate encoding for every output and concatenate these into one big encoding. These encoders may or may not share parameters. In our experiments, for two-dimensional outputs, parametrise separate encoders; for higher-dimensional outputs, apply the same encoder.

Gaussian neural process (GNP; Markou et al., 2022). Use the same choices for K, the encoder, and the decoder as the CNP. Set the rank of the kernel map to $R = 64$. As mentioned in the introduction, let the decoder produce one extra dimension which forms heterogeneous observation noise. For multidimensional outputs, the same caveats as for the CNP apply.

Latent neural process (LNP; Garnelo et al., 2018b). The LNP builds off the CNP. Call the existing encoder the *deterministic encoder*. The NP adds one more encoder called the *stochastic encoder*. The stochastic encoder mimics the deterministic encoder, but outputs a K-dimensional vector of means and a K-dimensional vector of marginal variances. These are used to sample a K-dimensional Gaussian latent variable (the *stochastic encoding*). The decoder now additionally takes in the stochastic encoding. For multidimensional outputs, the same caveats as for the CNP apply.

Attentive conditional neural process (ACNP; Kim et al., 2019). The ACNP builds off the CNP. It replaces the deterministic encoder $\mathrm{enc}_\theta \colon \mathcal{D} \to \mathbb{R}^K$ with an eight-head attentive encoder $\mathrm{enc}_\theta^{(\mathrm{att})} \colon \mathcal{D} \times \mathcal{X} \to \mathbb{R}^K$ (Vaswani et al., 2017). Unlike the original deterministic encoder enc_θ, the new attentive encoder $\mathrm{enc}_\theta^{(\mathrm{att})}$ also takes in the target input. Let $D^{(\mathrm{c})} = (\boldsymbol{x}^{(\mathrm{c})}, \boldsymbol{y}^{(\mathrm{c})}) \in \mathcal{D}$ be a context set of size N and let $x^{(\mathrm{t})} \in \mathcal{X}$ be a target input. We now descibe the computation of $\mathrm{enc}_\theta^{(\mathrm{att})}(D^{(\mathrm{c})}, x^{(\mathrm{t})})$.

Parametrise $\phi_x \colon \mathcal{X} \to (\mathbb{R}^{32})^8$ and $\phi_{xy} \colon \mathcal{X} \times \mathcal{Y} \to (\mathbb{R}^{32})^8$ both with three-hidden-layer MLPs of width 256. Compute

$$\begin{aligned} \text{the } \textit{keys}\text{:} \quad & (\boldsymbol{k}_{h,n})_{h=1}^8 = \phi_x(x_n^{(\mathrm{c})}) && \text{for } n = 1, \ldots, N, && \text{(D.26)} \\ \text{the } \textit{values}\text{:} \quad & (\boldsymbol{v}_{h,n})_{h=1}^8 = \phi_{xy}(x_n^{(\mathrm{c})}, y_n^{(\mathrm{c})}) && \text{for } n = 1, \ldots, N, && \text{(D.27)} \\ \text{the } \textit{query}\text{:} \quad & (\boldsymbol{q}_h)_{h=1}^8 = \phi_x(x^{(\mathrm{t})}). && && \text{(D.28)} \end{aligned}$$

Then compute

$$\boldsymbol{v}_h^{(\mathrm{q})} = \sum_{n=1}^N \frac{e^{\langle \boldsymbol{q}_h, \boldsymbol{k}_{h,n} \rangle}}{\sum_{n'=1}^N e^{\langle \boldsymbol{q}_h, \boldsymbol{k}_{h,n'} \rangle}} \boldsymbol{v}_{h,n} \in \mathbb{R}^{256} \tag{D.29}$$

Concatenate $\boldsymbol{v}^{(\mathrm{q})} = (\boldsymbol{v}_1^{(\mathrm{q})}, \ldots, \boldsymbol{v}_8^{(\mathrm{q})}) \in \mathbb{R}^{256}$ and $\boldsymbol{q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_8) \in \mathbb{R}^{256}$. Let $\boldsymbol{L} \colon \mathbb{R}^{256} \to \mathbb{R}^{256}$ be a linear layer; let $\phi^{(\mathrm{res})} \colon \mathbb{R}^{256} \to \mathbb{R}^{256}$ be a one-hidden-layer MLP of width 256; and let norm_1 and norm_2 be two layer normalisation layers with learned pointwise transformations (Ba et al., 2016). Then

$$\mathrm{enc}_\theta^{(\mathrm{att})}(D^{(\mathrm{c})}, x^{(\mathrm{t})}) = \mathrm{norm}_2(\boldsymbol{z} + \phi^{(\mathrm{res})}(\boldsymbol{z})) \quad \text{where} \quad \boldsymbol{z} = \mathrm{norm}_1(\boldsymbol{v}^{(\mathrm{q})} + \boldsymbol{L}\boldsymbol{q}). \tag{D.30}$$

For multidimensional outputs, the same caveats as for the CNP apply.

Attentive Gaussian neural process (AGNP). The AGNP build off the GNP. It replaces the deterministic encoder with the same eight-head attentive deterministic encoder of the ACNP.

Attentive neural process (ALNP; Kim et al., 2019). The ALNP build off the LNP. It replaces the deterministic encoder with the same eight-head attentive deterministic encoder of the ACNP.

Convolutional Conditional Neural Process (ConvCNP; Gordon et al., 2020). Set the discretisation to an evenly spaced grid at a certain density (the *points per unit*) spanning a bit more (the *margin*) than the most extremal context and target inputs. The points per unit and margin are specified separately for every experiment. Initialise the length scales of all Gaussian kernels to twice the interpoint spacing of the discretisation. Divide the data channel by the density channel. Parametrise dec_θ with a U-Net (Ronneberger et al., 2015). Before the U-turn, let the U-Net have six convolutional layers with kernel size five, stride two, and 64 output channels; and six more such layers, but using transposed convolutions, after the U-turn. The layers after the U-turn additionally take in the outputs of the layers before the U-turn in reversed order; this is the U-net structure (Figure 1; Ronneberger et al., 2015). For multidimensional outputs where outputs can have context points at different inputs, produce a separate data and density channel for every output and concatenate these into one big encoding; use separate length scales for every application of enc_θ.

Convolutional Gaussian neural process (ConvGNP; Markou et al., 2022). Use the same choices for the discretisation, length scales, and CNN architecture as for the ConvCNP. Set the rank of the kernel map to $R = 64$. As mentioned in the introduction, let the decoder produce one extra channel which forms heterogeneous observation noise. For multidimensional outputs, the same caveat as for the ConvCNP applies.

Fully convolutional Gaussian neural process (FullConvGNP; Bruinsma et al., 2021). For the mean architecture and the kernel architecture, use the same choices for the discretisation, length scales, and CNN architecture as for the ConvCNP. Implement the source channel with the identity matrix and apply the matrix transform $\boldsymbol{Z} \mapsto \boldsymbol{Z}\boldsymbol{Z}^\mathsf{T}$ to ensure positive definiteness. Let the decoder produce one extra channel which forms heterogeneous observation noise. For multidimensional outputs, in addition to the caveat for the ConvCNP, two additional caveats apply. First, for D_o-dimensional outputs, let the decoder produce D_o^2 channels rather than just one. These channels should be interpreted as all covariance and cross-covariance matrices between all outputs. Second, when applying the matrix transform $\boldsymbol{Z} \mapsto \boldsymbol{Z}\boldsymbol{Z}^\mathsf{T}$, these channels should first be assembled into one total covariance matrix.

Convolutional latent neural process (ConvLNP; Foong et al., 2020). The ConvLNP builds off the ConvCNP. The ConvLNP replaces the CNN architecture by two copies of this architecture placed in sequence. In between the two architectures, there is a sampling step: the first architecture outputs 32 channels, comprising 16 means and 16 marginal variances, which are used to sample a 16-dimensional Gaussian latent variable; and the second architecture then takes in this sample.

Autoregressive Conditional Neural Processes (AR CNPs). The AR CNP, AR ACNP, and AR ConvCNP use the architectures described above. Rolling out an AR CNP according to Procedure 6.2.1 requires an ordering of the target points. In all experiments, we choose a random ordering of the target points.

D.7 Training, Cross-Validation, and Evaluation Protocols

The following description applies to the synthetic experiments (Section 6.4.1), the predator–prey experiments (Section 6.4.2), the EEG experiments (Section 6.4.3), and the environmental downscaling experiments (Section 6.4.4). For the environmental data assimilation experiments, a different protocol was used; we refer the reader to Section D.11 for full details of the environmental data assimilation experiments.

A *task* consists of a context set and target set. How precisely the context and target sets are generated is specific to an experiment. To train a model, we consider batches of 16 tasks at a time, compute an objective function value, and update the model parameters using ADAM (Kingma and Ba, 2015). The learning rate is specified separately for every experiment. We define an epoch to consist of $2^{14} \approx 16\,\text{k}$ tasks. We typically train a model for between 100 and 1000 epochs.

For an experiment, we split up the meta–data set into a *training set*, a *cross-validation set*, and an *evaluation set*. The model is trained on the training set. During training, after every epoch, the model is cross-validated on the cross-validation set. Cross-validation uses 2^{12} fixed tasks. These 2^{12} are fixed, which means that cross-validation always happens with exactly the same data. The cross-validation objective is a confidence bound computed from the model objective. Suppose that model objective over all 2^{12} cross-validation tasks has empirical mean $\hat{\mu}$ and empirical variance $\hat{\sigma}^2$. If a higher model objective is better, then the cross-validation objective is given by $\hat{\mu} - 1.96 \cdot \hat{\sigma}/\sqrt{2^{12}}$. The model with the best cross-validation objective is selected and used for evaluation. Evaluation is performed with the evaluation set and also uses 2^{12} tasks.

Conditional neural processes and Gaussian neural processes are trained, cross-validated, and evaluated with the neural process ELBO objective proposed by Garnelo et al. (2018a). We normalise the terms in the neural process objective by the target set sizes. Latent-variable neural processes (LNPs) are trained, cross-validated, and evaluated with the ELBO objective proposed by Garnelo et al. (2018b) using five samples, also normalised by the target set size. When training LNPs with the ELBO objective, but not when cross-validating and evaluating, the context set is subsumed in the target set. Additionally, LNPs are trained, cross-validated, and evaluated with the ML objective proposed by Foong et al. (2020), again normalised by the target set size. When training and cross-validating LNPs with the ML objective, we use twenty samples; and when evaluating, we use 512 samples. For completeness, LNPs trained with the ELBO objective are also evaluated with the ML objective using 512 samples.

To stabilise the numerics for GNPs, we increase the regularisation of covariance matrices for one epoch. To encourage LNPs to fit, we fix the variance of the observation noise of the decoder to 10^{-4} for the first three epochs.

D.8 Details of Synthetic Experiments

D.8.1 Description of Experiments

We synthetically generate data sets by randomly sampling from five different choices for the ground-truth stochastic process f. Let the inputs be d_x-dimensional. Then define the following stochastic processes:

EQ: a Gaussian process with an exponentiated quadratic (EQ) kernel:

$$f \sim \mathcal{GP}(0, \exp(-\tfrac{1}{2\ell^2}\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2)) \tag{D.31}$$

where $\ell > 0$ is a length scale;

Matérn–$\frac{5}{2}$: a Gaussian process with a Matérn–$\frac{5}{2}$ kernel:

$$f \sim \mathcal{GP}(0, k(\tfrac{1}{\ell}\|\boldsymbol{x}-\boldsymbol{x}'\|_2)) \tag{D.32}$$

where $k(r) = (1+\sqrt{5}r+\frac{5}{3}r^2)e^{-r}$ and $\ell > 0$ is a length scale;

weakly periodic: a Gaussian process with a weakly periodic kernel:

$$f \sim \mathcal{GP}(0, \exp(-\tfrac{1}{2\ell_{\mathrm{d}}^2}\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2 - \tfrac{2}{\ell_{\mathrm{p}}^2}\|\sin(\tfrac{\pi}{p}(\boldsymbol{x}-\boldsymbol{x}'))\|_2^2)) \tag{D.33}$$

where $\ell_{\mathrm{d}} > 0$ is a length scale specifying how quickly the periodic pattern changes, $\ell_{\mathrm{p}} > 0$ a length scale of the periodic pattern, and $p > 0$ the period; and where the application of sin is elementwise;

sawtooth: a sawtooth process with a random frequency, direction, and phase:

$$f = \omega\langle \boldsymbol{x}, \boldsymbol{u}\rangle_2 + \phi \mod 1 \tag{D.34}$$

where $\omega \sim \mathrm{Unif}(\Omega)$ is the frequency of the sawtooth wave, $\boldsymbol{u} \sim \mathrm{Unif}(\{\boldsymbol{x} \in \mathbb{R}^{d_x} : \|\boldsymbol{x}\|_2 = 1\})$ the direction, and $\phi \sim \mathrm{Unif}([0,1])$ the phase;

mixture: with equal probability, sample f from the EQ process, Matérn–$\frac{5}{2}$ process, weakly periodic process, or sawtooth process.

We will call these stochastic processes the *data processes*. The data processes are stochastic processes with d_x-dimensional inputs and one-dimensional outputs. We will turn them into processes with d_y-dimensional outputs according to the following procedure: sample from the one-dimensional-output prior d_y times; and, for these d_y samples, take d_y different linear combinations.

We choose the parameters of the data processes based on the input dimensionality d_x:

$$\ell = c \cdot \tfrac{1}{4}, \qquad \ell_{\mathrm{d}} = c \cdot \tfrac{1}{2}, \qquad \ell_{\mathrm{s}} = c, \qquad p = c \cdot \tfrac{1}{4}, \qquad \Omega = [c^{-1} \cdot 2, c^{-1} \cdot 4] \tag{D.35}$$

with $c = \sqrt{d_x}$. Scaling with the input dimensionality aims to roughly ensure that data with one-dimensional inputs and data with two-dimensional inputs are equally difficult. Figure D.4 illustrates the sawtooth data process in all four configurations.

We will construct data sets by sampling inputs uniformly at random from $\mathcal{X} = [-2, 2]^{d_x}$ and then sampling outputs from one of the data processes. We will colloquially call $\mathcal{X}$ the *training range*. For the EQ, Matérn–$\frac{5}{2}$, and weakly periodic process, but not for the sawtooth process[1], we also add independent Gaussian noise with variance 0.05. The numbers of context and target points are as follows. For the EQ, Matérn–$\frac{5}{2}$, and weakly periodic process, the number of context points is chosen uniformly at random from $\{0, \ldots, 30 \cdot d_x\}$ and the number of targets points is fixed to $50 \cdot d_x$. For the sawtooth and mixture process, the number of context points is chosen uniformly at random from $\{0, \ldots, 30\}$ if $d_x = 1$ and $\{0, \ldots, 75 \cdot d_x\}$ otherwise; and the number of targets points is fixed to $100 \cdot d_x$. In the case of a multidimensional-output data process, we separately sample the number and positions of the context and target inputs for every output dimension.

For every data process and each of the four configurations, we evaluate every model in three different ways. First, we evaluate the model on data generated exactly like the training data. This task is called *interpolation* and abbreviated "int." in the tables of results. The interpolation task measures how well a model fits the data and is the primary measure of performance. Second, we evaluate the model on data with inputs sampled from $[2, 6]^{d_x}$. This task is called *out-of-input-distribution (OOID) interpolation* and abbreviated "OOID" in the tables of results. OOID interpolation measures how well a model generalises to data sampled from other regions of the input space. Third, we evaluate the model on data with context inputs sampled from $[-2, 2]^{d_x}$ and target inputs sampled from $[2, 6]^{d_x}$. This task is called *extrapolation* and abbreviated "ext." in the tables of results. The extrapolation task measures how well predictions based on data in the training range generalise to other regions of the input space.

[1]The sawtooth process is already challenging enough.

For this experiment, the learning rate is $3 \cdot 10^{-4}$, the margin is 0.1, and the points per unit is 64. We trained the models for 100 epochs. Due to an error in the cross-validation procedure, we did not use cross-validation, but used the model at epoch 100.

For the kernel architecture of the FullConvGNP, we reduce the points per unit and the number of channels in the U-Net by a factor two. For the ConvLNP with two-dimensional inputs, we reduce the number of outputs channels in the U-Net by a factor $\sqrt{2}$; and, for training and cross-validation, we reduce the number of samples of the ELBO objective to one and the number of samples for the ML objective to five.

D.8.2 Multi-Modality of Predictions by AR ConvCNP

Figure D.5 demonstrates multi-modality of predictions by the AR ConvCNP trained on the sawtooth process. Note that the prediction is bimodal for one and two observations, and collapses to a single mode upon observing the third observation.

D.8.3 Full Results

We the show the full results for all data sets and tasks in Tables D.1 to D.12. The AR ConvCNP consistently shows very strong performance compared to other NP models. Note that the FullConvGNP takes much longer to train than the ConvCNP (Figure 6.2), and cannot be applied to tasks with 2-dimensional input spaces.

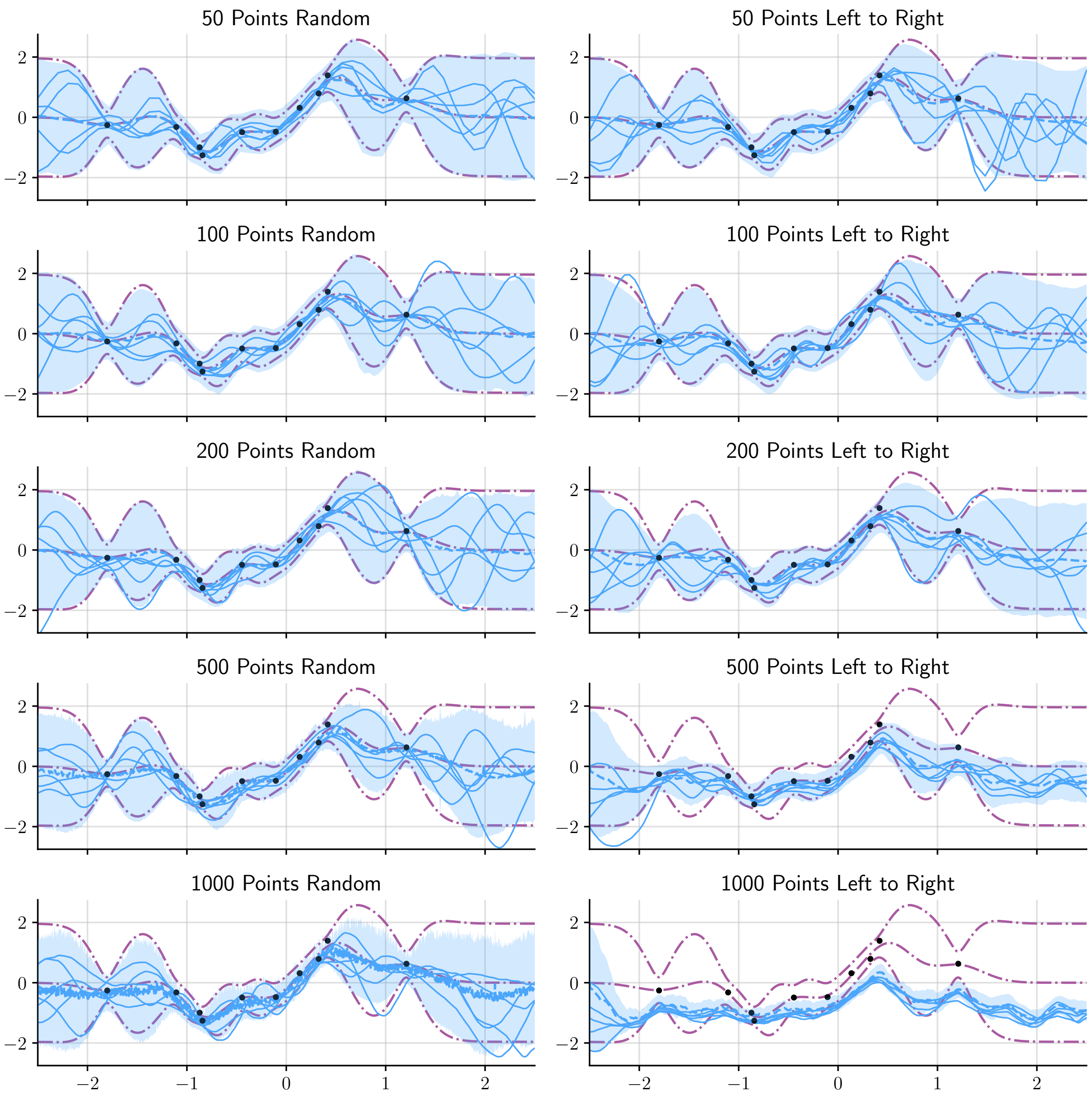


Fig. D.2 Samples and predictions for an AR ConvCNP with various numbers of target points ordered randomly (*left column*) and ordered left to right (*right column*). When the density of the target points does not exceed the training data (50 and 100 points), ordering the target points randomly or left to right does not matter. When the density of the target points comes close to the training data or exceeds it (200, 500, and 1000 points), bias creeps into the predictions. The random ordering appears to perform more robustly than left to right. The data is sampled from the EQ data process from the synthetic experiments (Section 6.4.1), and the trained model is also taken from the synthetic experiments. The predictions by the model are shown in solid blue and the marginals by the ground-truth EQ process are shown in dot-dashed purple.

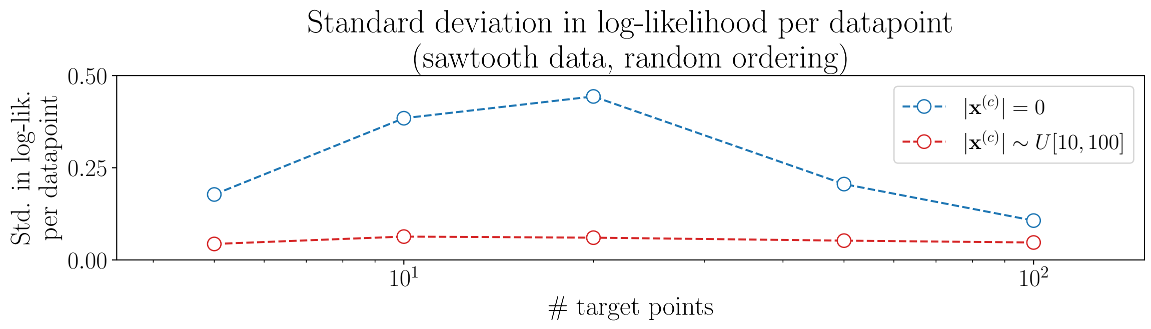


Fig. D.3 Plot of the standard deviation, due to different random orderings, of the per datapoint predictive log-likelihood (in nats) of an AR ConvCNP on one-dimensional sawtooth data, as a function of the number of target set size. For each point in the plot, we have used 2^{10} randomly sampled and fixed tasks, on each of which we apply the AR ConvCNP with 100 different randomly sampled orderings.

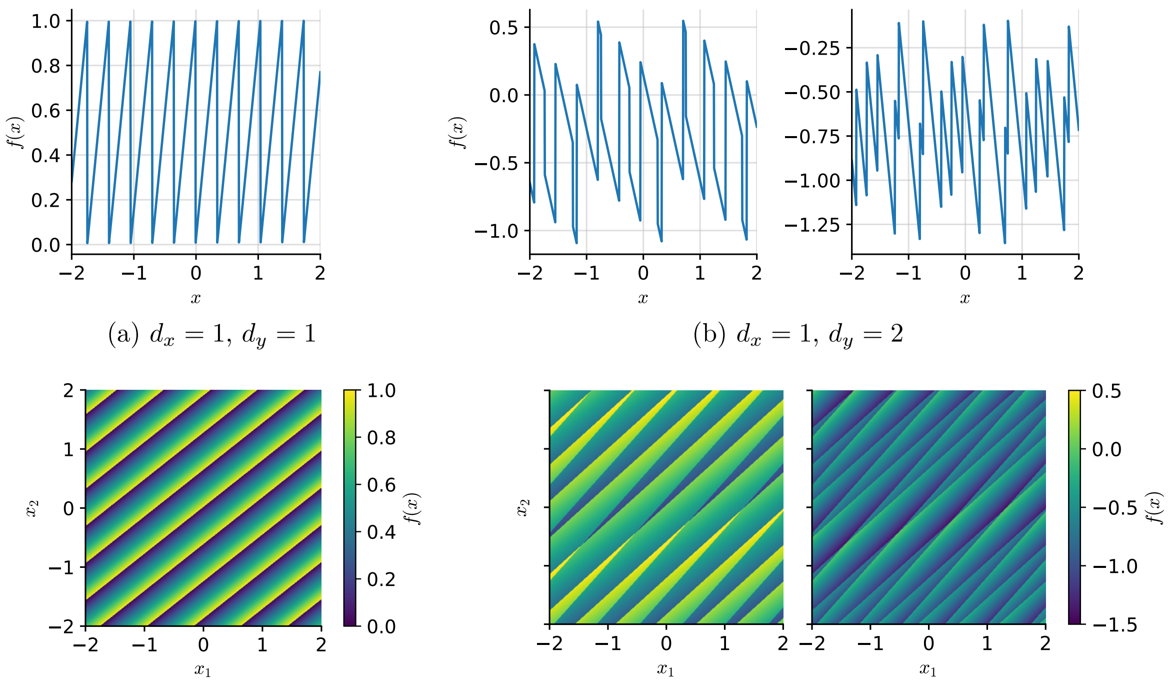


Fig. D.4 Samples from the sawtooth data process with one and two-dimensional inputs ($d_x = 1$ and $d_x = 2$) and one and two-dimensional outputs ($d_y = 1$ and $d_y = 2$)

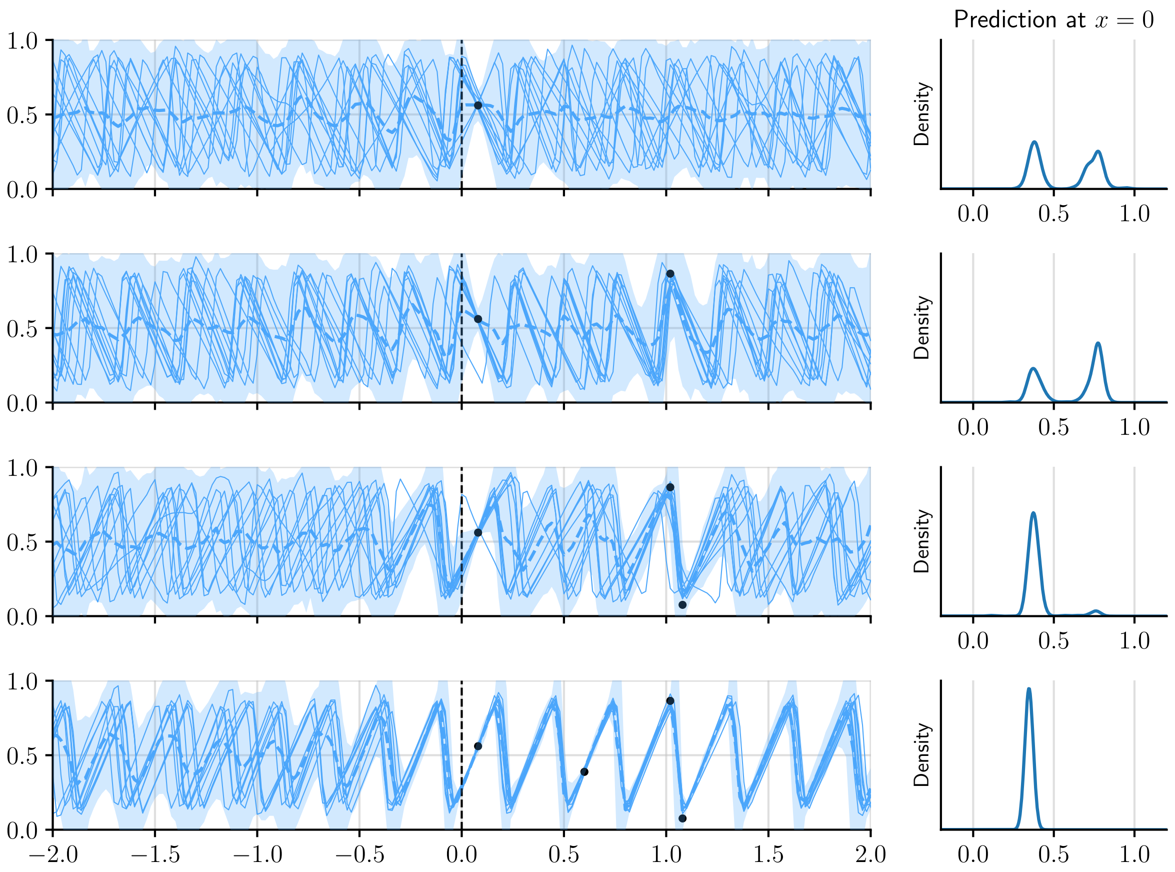


Fig. D.5 Multi-modality of predictions by the AR ConvCNP. Shows four observations sampled from the sawtooth process. In the four rows, these four observations are revealed one data point at a time. Every row also shows a kernel density estimate of the prediction at $x = 0$. Filled regions are central 95%-credible regions.

Model	Int. (1D)	OOID (1D)	Ext. (1D)	Int. (2D)	OOID (2D)	Ext. (2D)
FullConvGNP	**0.01** $\pm$0.00	**0.01** $\pm$0.00	**0.00** $\pm$0.00			
ConvCNP (AR)	0.03 $\pm$0.00	0.03 $\pm$0.00	0.02 $\pm$0.00	**0.03** $\pm$0.00	**0.03** $\pm$0.00	**0.02** $\pm$0.00
ConvGNP	0.04 $\pm$0.00	0.04 $\pm$0.00	1.75 $\pm$0.12	0.12 $\pm$0.00	0.12 $\pm$0.00	0.71 $\pm$0.03
AGNP	0.10 $\pm$0.00	4.34 $\pm$0.17	5.45 $\pm$0.23	0.17 $\pm$0.00	0.62 $\pm$0.01	0.39 $\pm$0.01
ConvLNP (E)	0.19 $\pm$0.01	0.19 $\pm$0.01	0.29 $\pm$0.03	0.39 $\pm$0.01	0.39 $\pm$0.01	0.36 $\pm$0.01
ACNP (AR)	0.24 $\pm$0.01	1.08 $\pm$0.02	0.86 $\pm$0.01	0.13 $\pm$0.00	0.57 $\pm$0.01	0.40 $\pm$0.01
GNP	0.25 $\pm$0.01	F	F	0.25 $\pm$0.01	0.75 $\pm$0.01	0.57 $\pm$0.00
ConvLNP (M)	0.31 $\pm$0.01	0.31 $\pm$0.01	0.64 $\pm$0.01	0.28 $\pm$0.01	0.28 $\pm$0.01	0.36 $\pm$0.01
Diagonal GP	0.42 $\pm$0.02	0.42 $\pm$0.02	0.84 $\pm$0.01	0.29 $\pm$0.01	0.29 $\pm$0.01	0.40 $\pm$0.01
ALNP (M)	0.43 $\pm$0.01	1.03 $\pm$0.02	0.78 $\pm$0.01	0.31 $\pm$0.01	0.55 $\pm$0.01	0.39 $\pm$0.01
ConvCNP	0.43 $\pm$0.02	0.43 $\pm$0.02	0.84 $\pm$0.01	0.30 $\pm$0.01	0.30 $\pm$0.01	0.40 $\pm$0.01
CNP (AR)	0.46 $\pm$0.01	F	F	0.36 $\pm$0.01	F	F
LNP (E)	0.51 $\pm$0.01	F	4.34 $\pm$0.76	0.40 $\pm$0.01	3.03 $\pm$1.68	0.60 $\pm$0.01
LNP (E–M)	0.52 $\pm$0.02	F	2.39 $\pm$0.33	0.39 $\pm$0.01	2.35 $\pm$1.06	0.57 $\pm$0.01
ALNP (E–M)	0.53 $\pm$0.01	1.12 $\pm$0.03	0.85 $\pm$0.02	0.42 $\pm$0.01	0.78 $\pm$1.72	0.41 $\pm$0.01
ACNP	0.54 $\pm$0.02	1.11 $\pm$0.02	0.84 $\pm$0.01	0.34 $\pm$0.01	0.57 $\pm$0.01	0.40 $\pm$0.01
ALNP (E)	0.54 $\pm$0.01	1.60 $\pm$0.06	1.25 $\pm$0.03	0.43 $\pm$0.01	1.06 $\pm$3.04	0.42 $\pm$0.01
LNP (M)	0.59 $\pm$0.01	F	1.13 $\pm$0.01	0.41 $\pm$0.01	0.88 $\pm$0.03	0.52 $\pm$0.01
CNP	0.63 $\pm$0.01	F	1.08 $\pm$0.02	0.43 $\pm$0.01	1.16 $\pm$0.45	0.52 $\pm$0.01
Trivial	1.08 $\pm$0.01	1.08 $\pm$0.01	0.85 $\pm$0.01	0.57 $\pm$0.01	0.57 $\pm$0.01	0.40 $\pm$0.00
ConvLNP (E–M)	2.01 $\pm$0.11	2.01 $\pm$0.11	5.95 $\pm$0.16	0.44 $\pm$0.01	0.44 $\pm$0.01	0.47 $\pm$0.01

Table D.1 For the Gaussian experiments, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional inputs (1D; $d_x = 1$) and two-dimensional inputs (2D; $d_y = 2$) the performance for interpolation within the range $[-2, 2]^{d_x}$ where the models were trained ("Int."); interpolation within the range $[2, 6]^{d_x}$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^{d_x}$ to the range $[2, 6]^{d_x}$ ("Ext."). Models are ordered by interpolation performance for one-dimensional inputs. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Model	Int. (1D)	OOID (1D)	Ext. (1D)	Int. (2D)	OOID (2D)	Ext. (2D)
ConvCNP (AR)	**1.52** ±0.04	**1.53** ±0.04	**1.32** ±0.04	**0.56** ±0.03	**0.56** ±0.03	**0.29** ±0.03
ConvLNP (E)	1.40 ±0.05	1.40 ±0.05	0.82 ±0.05	0.06 ±0.03	0.06 ±0.03	−0.62 ±0.04
ConvLNP (M)	1.08 ±0.06	1.08 ±0.06	−0.36 ±0.03	0.26 ±0.04	0.26 ±0.04	−0.70 ±0.02
ConvGNP	0.79 ±0.06	0.79 ±0.06	−1.03 ±0.07	0.23 ±0.04	0.23 ±0.04	−0.79 ±0.02
FullConvGNP	0.71 ±0.08	0.72 ±0.06	−0.20 ±0.02			
ConvCNP	0.57 ±0.07	0.57 ±0.07	−0.73 ±0.02	0.18 ±0.05	0.18 ±0.05	−0.86 ±0.03
ACNP (AR)	0.07 ±0.03	−0.85 ±0.03	−0.84 ±0.02	−0.53 ±0.02	−1.52 ±0.10	−1.51 ±0.09
AGNP	−0.31 ±0.03	−1.22 ±0.07	−1.58 ±0.11	−0.55 ±0.02	−0.79 ±0.02	−0.76 ±0.03
ALNP (E–M)	−0.33 ±0.03	−0.91 ±0.03	−0.80 ±0.04	−0.67 ±0.03	−1.06 ±0.62	−0.70 ±0.03
ALNP (E)	−0.35 ±0.03	−5.00 ±0.17	−3.37 ±0.06	−0.68 ±0.03	−2.59 ±7.64	−0.75 ±0.03
ALNP (M)	−0.36 ±0.02	−0.68 ±0.02	−0.68 ±0.02	−0.53 ±0.02	−0.74 ±0.04	−0.69 ±0.02
GNP	−0.38 ±0.02	F	F	−0.69 ±0.02	−0.74 ±0.04	−0.70 ±0.03
LNP (E–M)	−0.43 ±0.02	F	−3.34 ±0.53	−0.66 ±0.02	F	−0.96 ±0.03
LNP (E)	−0.44 ±0.02	F	F	−0.66 ±0.02	F	F
ACNP	−0.50 ±0.03	−0.83 ±0.03	−0.85 ±0.03	−0.60 ±0.02	−1.50 ±0.10	−0.73 ±0.03
LNP (M)	−0.53 ±0.02	−1.28 ±0.05	−0.80 ±0.03	−0.62 ±0.02	−1.50 ±0.11	−0.75 ±0.03
CNP (AR)	−0.65 ±0.02	−1.14 ±0.19	−0.98 ±0.06	−0.69 ±0.02	−1.05 ±0.07	−0.72 ±0.03
CNP	−0.68 ±0.02	−0.79 ±0.04	−0.73 ±0.03	−0.69 ±0.02	−1.05 ±0.08	−0.71 ±0.03
Trivial	−0.82 ±0.00	−0.82 ±0.00	−0.82 ±0.00	−0.82 ±0.00	−0.82 ±0.00	−0.82 ±0.00
ConvLNP (E–M)	F	F	F	−0.04 ±0.05	−0.04 ±0.05	−1.47 ±0.87

Table D.2 For the non-Gaussian experiments, average log-likelihoods normalised by the number of target points. Shows for one-dimensional inputs (1D; $d_x = 1$) and two-dimensional inputs (2D; $d_y = 2$) the performance for interpolation within the range $[-2, 2]^{d_x}$ where the models were trained ("Int."); interpolation within the range $[2, 6]^{d_x}$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^{d_x}$ to the range $[2, 6]^{d_x}$ ("Ext."). Models are ordered by interpolation performance for one-dimensional inputs. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

EQ $d_x = 1,\ d_y = 1$	Int.	OOID	Ext.
FullConvGNP	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00
ConvGNP	0.01 ±0.00	0.01 ±0.00	3.46 ±0.08
ConvCNP (AR)	0.01 ±0.00	0.01 ±0.00	0.01 ±0.00
AGNP	0.03 ±0.00	4.28 ±0.08	7.38 ±0.13
ConvLNP (E)	0.06 ±0.00	0.06 ±0.00	0.11 ±0.01
ACNP (AR)	0.07 ±0.00	1.19 ±0.01	0.98 ±0.01
GNP	0.08 ±0.00	F	F
ConvLNP (M)	0.25 ±0.01	0.25 ±0.01	0.67 ±0.01
CNP (AR)	0.28 ±0.00	F	F
ALNP (M)	0.31 ±0.01	1.04 ±0.01	0.84 ±0.01
LNP (E)	0.34 ±0.01	F	1.34 ±0.01
LNP (E–M)	0.37 ±0.01	F	1.27 ±0.01
Diagonal GP	0.40 ±0.01	0.40 ±0.01	0.95 ±0.01
ConvCNP	0.41 ±0.01	0.41 ±0.01	0.95 ±0.01
ANP (E–M)	0.42 ±0.01	1.18 ±0.01	0.94 ±0.01
ANP (E)	0.44 ±0.01	1.32 ±0.01	1.25 ±0.01
ACNP	0.45 ±0.01	1.22 ±0.01	0.95 ±0.01
LNP (M)	0.49 ±0.01	1.54 ±0.01	1.45 ±0.01
CNP	0.54 ±0.01	F	1.41 ±0.01
ConvLNP (E–M)	0.90 ±0.04	0.90 ±0.04	4.05 ±0.06
Trivial	1.19 ±0.00	1.19 ±0.00	0.96 ±0.00

EQ $d_x = 1,\ d_y = 2$	Int.	OOID	Ext.
FullConvGNP	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00
ConvGNP	0.01 ±0.00	0.01 ±0.00	1.73 ±0.05
ConvCNP (AR)	0.01 ±0.00	0.01 ±0.00	0.01 ±0.00
AGNP	0.04 ±0.00	7.71 ±0.10	7.87 ±0.10
ACNP (AR)	0.07 ±0.00	1.31 ±0.01	1.09 ±0.01
ConvLNP (E)	0.08 ±0.00	0.08 ±0.00	0.13 ±0.00
GNP	0.13 ±0.00	F	F
ConvLNP (M)	0.36 ±0.01	0.36 ±0.01	0.88 ±0.00
ALNP (M)	0.41 ±0.01	1.23 ±0.01	0.99 ±0.00
CNP (AR)	0.42 ±0.00	F	F
Diagonal GP	0.47 ±0.01	0.47 ±0.01	1.06 ±0.00
ConvCNP	0.48 ±0.01	0.48 ±0.01	1.06 ±0.00
ACNP	0.51 ±0.01	1.38 ±0.01	1.06 ±0.01
ALNP (E–M)	0.52 ±0.01	1.47 ±0.02	1.07 ±0.01
ALNP (E)	0.53 ±0.01	3.79 ±0.05	2.84 ±0.02
LNP (E)	0.54 ±0.01	F	1.47 ±0.01
LNP (E–M)	0.56 ±0.01	F	1.42 ±0.01
LNP (M)	0.64 ±0.00	F	1.52 ±0.00
CNP	0.66 ±0.01	F	1.28 ±0.00
Trivial	1.31 ±0.00	1.31 ±0.00	1.07 ±0.00
ConvLNP (E–M)	2.14 ±0.06	2.14 ±0.06	9.30 ±0.08

Table D.3 For the EQ synthetic experiments with one-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]$ where the models where trained ("Int."); interpolation within the range $[2, 6]$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]$ to the range $[2, 6]$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

EQ $d_x=2,\ d_y=1$	Int.	OOID	Ext.
ConvCNP (AR)	**0.01** ±0.00	**0.01** ±0.00	**0.01** ±0.00
ConvGNP	0.08 ±0.00	0.08 ±0.00	1.92 ±0.02
AGNP	0.09 ±0.00	0.70 ±0.00	0.50 ±0.00
ACNP (AR)	0.09 ±0.00	0.72 ±0.00	0.51 ±0.00
GNP	0.19 ±0.00	1.01 ±0.00	0.80 ±0.00
ConvLNP (M)	0.34 ±0.00	0.34 ±0.00	0.47 ±0.00
ALNP (M)	0.34 ±0.00	0.70 ±0.00	0.51 ±0.00
Diagonal GP	0.36 ±0.00	0.36 ±0.00	0.51 ±0.00
ConvCNP	0.37 ±0.00	0.37 ±0.00	0.51 ±0.00
ACNP	0.40 ±0.00	0.72 ±0.00	0.51 ±0.00
ConvLNP (E)	0.41 ±0.00	0.41 ±0.00	0.46 ±0.00
CNP (AR)	0.41 ±0.00	0.90 ±0.00	0.71 ±0.00
LNP (E–M)	0.46 ±0.00	0.99 ±0.01	0.65 ±0.00
LNP (E)	0.48 ±0.00	1.04 ±0.01	0.67 ±0.00
ConvLNP (E–M)	0.48 ±0.00	0.48 ±0.00	0.59 ±0.01
ALNP (E–M)	0.49 ±0.01	0.72 ±0.00	0.51 ±0.00
ALNP (E)	0.50 ±0.01	0.73 ±0.00	0.52 ±0.00
LNP (M)	0.51 ±0.00	0.92 ±0.00	0.72 ±0.00
CNP	0.52 ±0.00	0.92 ±0.00	0.72 ±0.00
Trivial	0.72 ±0.00	0.72 ±0.00	0.51 ±0.00
FullConvGNP			

EQ $d_x=2,\ d_y=2$	Int.	OOID	Ext.
ConvCNP (AR)	**0.03** ±0.00	**0.03** ±0.00	**0.02** ±0.00
ACNP (AR)	0.11 ±0.00	0.79 ±0.00	0.56 ±0.00
ConvGNP	0.19 ±0.00	0.19 ±0.00	0.74 ±0.01
AGNP	0.22 ±0.00	0.87 ±0.01	0.57 ±0.00
GNP	0.38 ±0.00	1.06 ±0.00	0.75 ±0.00
ConvNP (M)	0.39 ±0.00	0.39 ±0.00	0.52 ±0.00
Diagonal GP	0.40 ±0.00	0.40 ±0.00	0.56 ±0.00
ConvCNP	0.41 ±0.00	0.41 ±0.00	0.56 ±0.00
ANP (M)	0.42 ±0.00	0.79 ±0.00	0.54 ±0.00
ACNP	0.44 ±0.00	0.79 ±0.00	0.56 ±0.00
CNP (AR)	0.52 ±0.00	F	F
NP (E–M)	0.56 ±0.00	1.95 ±0.03	0.72 ±0.00
ANP (E–M)	0.56 ±0.00	1.90 ±1.72	0.55 ±0.00
NP (E)	0.57 ±0.00	1.99 ±0.03	0.72 ±0.00
ANP (E)	0.57 ±0.00	3.51 ±3.04	0.56 ±0.00
NP (M)	0.59 ±0.00	1.17 ±0.01	0.75 ±0.00
CNP	0.60 ±0.00	3.08 ±0.42	0.66 ±0.00
Trivial	0.79 ±0.00	0.79 ±0.00	0.56 ±0.00
ConvNP (E–M)	0.79 ±0.00	0.79 ±0.00	0.56 ±0.00
ConvNP (E)	0.79 ±0.00	0.79 ±0.00	0.56 ±0.00
FullConvGNP			

Table D.4 For the EQ synthetic experiments with two-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]^2$ where the models where trained ("Int."); interpolation within the range $[2, 6]^2$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^2$ to the range $[2, 6]^2$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Matérn–$\frac{5}{2}$ $d_x = 1$, $d_y = 1$	Int.	OOID	Ext.
FullConvGNP	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00
ConvCNP (AR)	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
ConvGNP	0.01 ±0.00	0.01 ±0.00	2.32 ±0.06
AGNP	0.03 ±0.00	4.53 ±0.08	7.22 ±0.12
ACNP (AR)	0.04 ±0.00	1.08 ±0.01	0.87 ±0.01
GNP	0.09 ±0.00	F	F
ConvLNP (E)	0.13 ±0.00	0.13 ±0.00	0.31 ±0.02
ConvLNP (M)	0.26 ±0.01	0.26 ±0.01	0.58 ±0.00
ALNP (M)	0.30 ±0.00	0.98 ±0.01	0.78 ±0.01
CNP (AR)	0.34 ±0.01	1.81 ±0.04	1.32 ±0.02
LNP (E)	0.36 ±0.00	F	1.31 ±0.01
LNP (E–M)	0.37 ±0.01	F	1.14 ±0.00
Diagonal GP	0.40 ±0.01	0.40 ±0.01	0.84 ±0.01
ConvCNP	0.40 ±0.01	0.40 ±0.01	0.84 ±0.01
ALNP (E–M)	0.41 ±0.01	1.13 ±0.01	0.84 ±0.01
ACNP	0.42 ±0.01	1.10 ±0.01	0.84 ±0.01
ALNP (E)	0.43 ±0.01	1.15 ±0.01	0.87 ±0.01
LNP (M)	0.51 ±0.00	1.87 ±0.02	1.30 ±0.01
CNP	0.54 ±0.01	1.47 ±0.02	1.11 ±0.01
Trivial	1.08 ±0.00	1.08 ±0.00	0.85 ±0.00
ConvLNP (E–M)	1.37 ±0.04	1.36 ±0.04	4.30 ±0.06

Matérn–$\frac{5}{2}$ $d_x = 1$, $d_y = 2$	Int.	OOID	Ext.
FullConvGNP	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00
ConvCNP (AR)	0.01 ±0.00	0.01 ±0.00	0.01 ±0.00
ConvGNP	0.02 ±0.00	0.02 ±0.00	1.71 ±0.04
AGNP	0.04 ±0.00	6.14 ±0.08	7.03 ±0.09
ACNP (AR)	0.05 ±0.00	1.18 ±0.01	0.96 ±0.01
GNP	0.13 ±0.00	F	F
ConvLNP (E)	0.16 ±0.00	0.16 ±0.00	0.29 ±0.00
ConvLNP (M)	0.36 ±0.00	0.36 ±0.00	0.76 ±0.00
ALNP (M)	0.40 ±0.00	1.10 ±0.01	0.88 ±0.00
CNP (AR)	0.45 ±0.00	F	F
Diagonal GP	0.46 ±0.01	0.46 ±0.01	0.93 ±0.00
ConvCNP	0.46 ±0.01	0.46 ±0.01	0.93 ±0.00
ACNP	0.49 ±0.01	1.23 ±0.01	0.93 ±0.00
ALNP (E–M)	0.51 ±0.01	1.28 ±0.01	0.99 ±0.01
ALNP (E)	0.51 ±0.01	1.43 ±0.02	1.10 ±0.01
LNP (E–M)	0.54 ±0.00	F	1.24 ±0.00
LNP (E)	0.54 ±0.00	F	1.79 ±0.01
LNP (M)	0.63 ±0.00	2.33 ±0.02	1.23 ±0.00
CNP	0.65 ±0.00	7.72 ±0.69	1.23 ±0.00
Trivial	1.18 ±0.00	1.18 ±0.00	0.94 ±0.00
ConvLNP (E–M)	3.07 ±0.06	3.06 ±0.06	9.83 ±0.09

Table D.5 For the Matérn–$\frac{5}{2}$ synthetic experiments with one-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]$ where the models where trained ("Int."); interpolation within the range $[2, 6]$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]$ to the range $[2, 6]$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Matérn–$\frac{5}{2}$ $d_x=2$, $d_y=1$	Int.	OOID	Ext.
ConvCNP (AR)	**0.01** ±0.00	**0.01** ±0.00	**0.00** ±0.00
ACNP (AR)	0.05 ±0.00	0.54 ±0.00	0.38 ±0.00
AGNP	0.08 ±0.00	0.83 ±0.01	0.37 ±0.00
ConvGNP	0.08 ±0.00	0.08 ±0.00	0.60 ±0.01
GNP	0.16 ±0.00	0.90 ±0.00	0.75 ±0.00
ConvLNP (M)	0.25 ±0.00	0.25 ±0.00	0.34 ±0.00
ALNP (M)	0.26 ±0.00	0.51 ±0.00	0.37 ±0.00
Diagonal GP	0.28 ±0.00	0.28 ±0.00	0.39 ±0.00
ConvCNP	0.28 ±0.00	0.28 ±0.00	0.39 ±0.00
ACNP	0.29 ±0.00	0.54 ±0.00	0.39 ±0.00
CNP (AR)	0.31 ±0.00	0.69 ±0.00	0.52 ±0.00
ConvLNP (E)	0.32 ±0.00	0.32 ±0.00	0.30 ±0.00
LNP (E–M)	0.34 ±0.00	1.07 ±0.01	0.69 ±0.00
LNP (E)	0.35 ±0.00	1.25 ±0.01	0.72 ±0.00
ConvLNP (E–M)	0.36 ±0.00	0.36 ±0.00	0.43 ±0.00
LNP (M)	0.37 ±0.00	0.75 ±0.01	0.51 ±0.00
ALNP (E–M)	0.39 ±0.00	0.65 ±0.01	0.43 ±0.00
CNP	0.39 ±0.00	0.67 ±0.00	0.54 ±0.00
ALNP (E)	0.41 ±0.01	0.67 ±0.01	0.44 ±0.00
Trivial	0.55 ±0.00	0.55 ±0.00	0.39 ±0.00
FullConvGNP			

Matérn–$\frac{5}{2}$ $d_x=2$, $d_y=2$	Int.	OOID	Ext.
ConvCNP (AR)	**0.01** ±0.00	**0.01** ±0.00	**0.01** ±0.00
ACNP (AR)	0.06 ±0.00	0.58 ±0.00	0.41 ±0.00
ConvGNP	0.14 ±0.00	0.14 ±0.00	0.64 ±0.01
AGNP	0.17 ±0.00	0.58 ±0.00	0.40 ±0.00
GNP	0.28 ±0.00	0.78 ±0.00	0.60 ±0.00
ConvLNP (M)	0.29 ±0.00	0.29 ±0.00	0.38 ±0.00
ALNP (M)	0.29 ±0.00	0.56 ±0.00	0.40 ±0.00
Diagonal GP	0.30 ±0.00	0.30 ±0.00	0.41 ±0.00
ConvCNP	0.30 ±0.00	0.30 ±0.00	0.41 ±0.00
ACNP	0.32 ±0.00	0.58 ±0.00	0.41 ±0.00
ConvLNP (E)	0.36 ±0.00	0.36 ±0.00	0.37 ±0.00
CNP (AR)	0.37 ±0.00	F	0.88 ±0.17
LNP (E–M)	0.41 ±0.00	2.29 ±0.05	0.59 ±0.00
LNP (E)	0.41 ±0.00	2.36 ±0.05	0.60 ±0.00
ALNP (E–M)	0.42 ±0.00	0.61 ±0.00	0.41 ±0.00
ALNP (E)	0.42 ±0.00	0.61 ±0.00	0.41 ±0.00
LNP (M)	0.43 ±0.00	0.68 ±0.00	0.53 ±0.00
CNP	0.44 ±0.00	0.86 ±0.17	0.59 ±0.00
ConvLNP (E–M)	0.49 ±0.00	0.49 ±0.00	0.61 ±0.00
Trivial	0.58 ±0.00	0.58 ±0.00	0.41 ±0.00
FullConvGNP			

Table D.6 For the Matérn–$\frac{5}{2}$ synthetic experiments with two-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]^2$ where the models where trained ("Int."); interpolation within the range $[2, 6]^2$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^2$ to the range $[2, 6]^2$ ("Ext."). Models are ordered by interpolation performance. Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Weakly Periodic $d_x=1$, $d_y=1$	Int.	OOID	Ext.
FullConvGNP	**0.02** ±0.00	**0.02** ±0.00	**0.00** ±0.00
ConvCNP (AR)	0.05 ±0.00	0.05 ±0.00	0.04 ±0.00
ConvGNP	0.05 ±0.00	0.05 ±0.00	0.56 ±0.02
AGNP	0.22 ±0.00	1.25 ±0.02	1.25 ±0.02
ConvLNP (M)	0.28 ±0.00	0.28 ±0.00	0.43 ±0.00
ConvLNP (E)	0.34 ±0.00	0.33 ±0.00	0.45 ±0.02
Diagonal GP	0.38 ±0.01	0.38 ±0.01	0.59 ±0.01
ConvCNP	0.40 ±0.01	0.40 ±0.01	0.60 ±0.01
ALNP (M)	0.53 ±0.00	0.77 ±0.01	0.57 ±0.01
ACNP (AR)	0.57 ±0.01	0.82 ±0.01	0.61 ±0.01
GNP	0.59 ±0.01	1.31 ±0.02	0.62 ±0.01
CNP (AR)	0.59 ±0.01	2.33 ±0.27	1.46 ±0.05
LNP (E–M)	0.60 ±0.01	F	4.09 ±0.28
ALNP (E–M)	0.60 ±0.01	0.78 ±0.01	0.59 ±0.01
LNP (M)	0.60 ±0.01	0.80 ±0.01	0.62 ±0.01
LNP (E)	0.61 ±0.01	F	9.91 ±0.70
ALNP (E)	0.62 ±0.01	1.01 ±0.01	0.71 ±0.01
ACNP	0.65 ±0.01	0.82 ±0.01	0.61 ±0.01
CNP	0.67 ±0.01	1.45 ±0.03	0.68 ±0.01
Trivial	0.82 ±0.00	0.82 ±0.00	0.61 ±0.00
ConvLNP (E–M)	1.58 ±0.03	1.57 ±0.03	2.85 ±0.04

Weakly Periodic $d_x=1$, $d_y=2$	Int.	OOID	Ext.
FullConvGNP	**0.03** ±0.00	**0.03** ±0.00	**0.00** ±0.00
ConvCNP (AR)	0.09 ±0.00	0.09 ±0.00	0.06 ±0.00
ConvGNP	0.12 ±0.00	0.12 ±0.00	0.72 ±0.01
AGNP	0.25 ±0.00	2.17 ±0.02	1.95 ±0.02
ConvLNP (M)	0.38 ±0.00	0.38 ±0.00	0.54 ±0.00
ConvLNP (E)	0.39 ±0.00	0.39 ±0.00	0.44 ±0.00
Diagonal GP	0.42 ±0.00	0.42 ±0.00	0.65 ±0.00
ConvCNP	0.46 ±0.00	0.46 ±0.00	0.65 ±0.00
GNP	0.50 ±0.00	1.02 ±0.01	0.76 ±0.00
ALNP (M)	0.62 ±0.00	1.04 ±0.01	0.64 ±0.00
ACNP (AR)	0.63 ±0.00	0.89 ±0.01	0.66 ±0.00
CNP (AR)	0.67 ±0.00	2.52 ±0.07	1.21 ±0.01
LNP (E–M)	0.68 ±0.00	F	5.18 ±0.18
LNP (M)	0.69 ±0.00	1.26 ±0.01	0.68 ±0.01
LNP (E)	0.69 ±0.00	F	F
ALNP (E–M)	0.70 ±0.00	0.85 ±0.01	0.64 ±0.00
ACNP	0.71 ±0.00	0.89 ±0.01	0.66 ±0.00
ALNP (E)	0.72 ±0.00	0.93 ±0.01	0.72 ±0.00
CNP	0.74 ±0.00	1.27 ±0.01	0.77 ±0.01
Trivial	0.89 ±0.00	0.89 ±0.00	0.67 ±0.00
ConvLNP (E–M)	3.02 ±0.03	3.02 ±0.03	5.40 ±0.04

Table D.7 For the weakly periodic synthetic experiments with one-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]$ where the models where trained ("Int."); interpolation within the range $[2, 6]$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]$ to the range $[2, 6]$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Weakly Periodic $d_x = 2$, $d_y = 1$	Int.	OOID	Ext.
ConvCNP (AR)	**0.05** ±0.00	**0.05** ±0.00	**0.03** ±0.00
ConvGNP	0.10 ±0.00	0.10 ±0.00	0.19 ±0.00
ConvLNP (M)	0.18 ±0.00	0.18 ±0.00	0.21 ±0.00
Diagonal GP	0.19 ±0.00	0.19 ±0.00	0.27 ±0.00
ConvCNP	0.20 ±0.00	0.20 ±0.00	0.27 ±0.00
ConvLNP (E)	0.22 ±0.00	0.22 ±0.00	0.22 ±0.00
AGNP	0.23 ±0.00	0.35 ±0.00	0.26 ±0.00
ACNP (AR)	0.23 ±0.00	0.37 ±0.00	0.26 ±0.00
ConvLNP (E–M)	0.23 ±0.00	0.23 ±0.00	0.29 ±0.00
GNP	0.23 ±0.00	0.35 ±0.00	0.24 ±0.00
CNP (AR)	0.25 ±0.00	0.82 ±0.01	0.72 ±0.01
ALNP (M)	0.25 ±0.00	0.36 ±0.00	0.25 ±0.00
LNP (M)	0.26 ±0.00	1.32 ±0.03	0.33 ±0.00
LNP (E–M)	0.26 ±0.00	6.28 ±1.06	0.44 ±0.01
LNP (E)	0.27 ±0.00	9.91 ±1.68	0.52 ±0.01
ACNP	0.28 ±0.00	0.37 ±0.00	0.27 ±0.00
CNP	0.29 ±0.00	0.81 ±0.01	0.29 ±0.00
ALNP (E–M)	0.31 ±0.00	0.39 ±0.00	0.26 ±0.00
ALNP (E)	0.32 ±0.00	0.42 ±0.00	0.29 ±0.00
Trivial	0.38 ±0.00	0.38 ±0.00	0.27 ±0.00
FullConvGNP			

Weakly Periodic $d_x = 2$, $d_y = 2$	Int.	OOID	Ext.
ConvCNP (AR)	**0.08** ±0.00	**0.08** ±0.00	**0.05** ±0.00
ConvGNP	0.13 ±0.00	0.13 ±0.00	0.18 ±0.00
Diagonal GP	0.20 ±0.00	0.20 ±0.00	0.28 ±0.00
ConvLNP (M)	0.21 ±0.00	0.21 ±0.00	0.25 ±0.00
ConvCNP	0.23 ±0.00	0.23 ±0.00	0.28 ±0.00
AGNP	0.24 ±0.00	0.39 ±0.00	0.27 ±0.00
ACNP (AR)	0.25 ±0.00	0.40 ±0.00	0.28 ±0.00
GNP	0.25 ±0.00	0.38 ±0.00	0.25 ±0.00
ConvLNP (E)	0.26 ±0.00	0.26 ±0.00	0.27 ±0.00
CNP (AR)	0.27 ±0.00	3.21 ±0.14	0.77 ±0.02
ALNP (M)	0.28 ±0.00	0.39 ±0.00	0.28 ±0.00
LNP (M)	0.29 ±0.00	0.42 ±0.00	0.31 ±0.00
LNP (E–M)	0.29 ±0.00	1.52 ±0.04	0.36 ±0.00
LNP (E)	0.30 ±0.00	1.60 ±0.05	0.39 ±0.00
ConvLNP (E–M)	0.30 ±0.00	0.30 ±0.00	0.34 ±0.00
ACNP	0.30 ±0.00	0.40 ±0.00	0.29 ±0.00
CNP	0.31 ±0.00	0.63 ±0.01	0.33 ±0.00
ALNP (E–M)	0.36 ±0.00	0.43 ±0.00	0.28 ±0.00
ALNP (E)	0.36 ±0.00	0.44 ±0.00	0.29 ±0.00
Trivial	0.40 ±0.00	0.40 ±0.00	0.28 ±0.00
FullConvGNP			

Table D.8 For the weakly periodic synthetic experiments with two-dimensional inputs, average Kullback–Leibler divergences of the posterior prediction map π_y with respect to the model normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]^2$ where the models where trained ("Int."); interpolation within the range $[2, 6]^2$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^2$ to the range $[2, 6]^2$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Diagonal GP refers to predictions by the ground-truth Gaussian processes without correlations. Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Sawtooth $d_x = 1$, $d_y = 1$	Int.	OOID	Ext.	Sawtooth $d_x = 1$, $d_y = 2$	Int.	OOID	Ext.
ConvCNP (AR)	**3.60**±0.01	**3.60**±0.01	**3.34**±0.01	ConvCNP (AR)	**2.22**±0.01	**2.22**±0.01	**1.91**±0.01
ConvLNP (E)	3.51±0.02	3.52±0.02	2.68±0.02	ConvLNP (E)	2.01±0.01	2.01±0.01	1.47±0.01
ConvLNP	3.06±0.04	3.06±0.04	0.64±0.01	ConvLNP	1.73±0.03	1.73±0.03	−0.12±0.01
ConvGNP	2.62±0.05	2.61±0.08	−0.04±0.01	ACNP (AR)	1.01±0.01	−0.47±0.01	−0.45±0.01
ConvCNP	2.38±0.04	2.37±0.04	−0.00±0.01	FullConvGNP	0.99±0.06	1.04±0.03	0.11±0.00
FullConvGNP	2.16±0.04	2.15±0.04	0.18±0.01	ConvCNP	0.83±0.03	0.84±0.02	−0.29±0.00
ALNP (E–M)	0.27±0.01	−0.18±0.00	−0.31±0.02	ConvGNP	0.82±0.03	0.82±0.03	−0.29±0.00
ALNP (E)	0.27±0.01	−15.96±0.17	−9.14±0.04	GNP	−0.03±0.00	F	F
ALNP	0.20±0.00	−0.18±0.00	−0.18±0.00	ALNP (E–M)	−0.06±0.01	−0.75±0.01	−0.35±0.00
LNP (E–M)	0.07±0.01	F	−8.67±0.53	ALNP (E)	−0.07±0.01	−0.78±0.02	−0.35±0.00
LNP (E)	0.06±0.01	F	F	AGNP	−0.07±0.01	−0.37±0.00	−0.48±0.07
Trivial	−0.18±0.00	−0.18±0.00	−0.18±0.00	ACNP	−0.08±0.01	−0.41±0.01	−0.42±0.01
LNP	−0.18±0.00	−0.18±0.00	−0.18±0.00	LNP (E–M)	−0.20±0.00	F	−1.33±0.00
CNP (AR)	−0.18±0.00	−0.18±0.00	−0.18±0.00	LNP (E)	−0.20±0.00	F	−1.33±0.00
CNP	−0.18±0.00	−0.18±0.00	−0.18±0.00	ALNP	−0.28±0.00	−0.33±0.00	−0.33±0.00
GNP	−0.18±0.00	−0.18±0.00	−0.18±0.00	CNP (AR)	−0.29±0.00	−1.96±0.19	−1.43±0.06
AGNP	−0.18±0.00	−0.18±0.00	−0.18±0.00	CNP	−0.30±0.00	−0.51±0.01	−0.34±0.00
ACNP	−0.18±0.00	−0.18±0.00	−0.18±0.00	LNP	−0.32±0.00	−0.32±0.00	−0.32±0.00
ACNP (AR)	−0.18±0.00	−0.18±0.00	−0.18±0.00	*Trivial*	−0.33±0.00	−0.33±0.00	−0.33±0.00
ConvLNP (E–M)	F	F	F	ConvLNP (E–M)	−2.98±0.10	−2.98±0.10	−6.74±0.04

Table D.9 For the sawtooth synthetic experiments with one-dimensional inputs, average log-likelihoods normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]$ where the models where trained ("Int."); interpolation within the range $[2, 6]$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]$ to the range $[2, 6]$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Sawtooth $d_x = 2,\ d_y = 1$	Int.	OOID	Ext.	Sawtooth $d_x = 2,\ d_y = 2$	Int.	OOID	Ext.
ConvCNP (AR)	**2.59** ±0.01	**2.59** ±0.01	**2.10** ±0.01	ConvCNP (AR)	**0.38** ±0.00	**0.38** ±0.00	**0.18** ±0.00
ConvLNP (M)	2.07 ±0.02	2.08 ±0.02	–0.17 ±0.00	ConvLNP (M)	0.31 ±0.01	0.31 ±0.01	–0.32 ±0.00
ConvCNP	1.93 ±0.04	1.94 ±0.03	–0.18 ±0.00	ConvGNP	0.26 ±0.01	0.26 ±0.01	–0.33 ±0.00
ConvGNP	1.90 ±0.04	1.91 ±0.03	–0.18 ±0.00	ConvCNP	0.12 ±0.01	0.12 ±0.01	–0.32 ±0.00
ConvLNP (E)	1.77 ±0.02	1.77 ±0.02	0.33 ±0.02	ConvLNP (E)	0.04 ±0.00	0.04 ±0.00	–0.30 ±0.00
ConvLNP (E–M)	1.71 ±0.04	1.72 ±0.04	–2.30 ±0.87	ConvLNP (E–M)	–0.07 ±0.01	–0.07 ±0.01	–0.48 ±0.00
Trivial	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	*Trivial*	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
CNP (AR)	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	ALNP (M)	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
CNP	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	CNP (AR)	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
GNP	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	CNP	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
LNP (M)	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	LNP (M)	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
AGNP	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	ACNP	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
ALNP (M)	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	ACNP (AR)	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
ACNP (AR)	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	GNP	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
ACNP	–0.18 ±0.00	–0.18 ±0.00	–0.18 ±0.00	AGNP	–0.32 ±0.00	–0.32 ±0.00	–0.32 ±0.00
LNP (E–M)	–0.19 ±0.00	F	–0.86 ±0.02	LNP (E–M)	–0.33 ±0.00	–0.54 ±0.00	–0.34 ±0.00
LNP (E)	–0.19 ±0.00	F	F	LNP (E)	–0.33 ±0.00	–0.54 ±0.00	–0.34 ±0.00
ALNP (E–M)	–0.20 ±0.01	–0.18 ±0.00	–0.18 ±0.00	ALNP (E–M)	–0.36 ±0.00	–0.33 ±0.00	–0.33 ±0.00
ALNP (E)	–0.20 ±0.01	–0.71 ±0.00	–0.33 ±0.00	ALNP (E)	–0.36 ±0.00	–0.33 ±0.00	–0.33 ±0.00
FullConvGNP				FullConvGNP			

Table D.10 For the sawtooth synthetic experiments with two-dimensional inputs, average log-likelihoods normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]^2$ where the models where trained ("Int."); interpolation within the range $[2, 6]^2$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^2$ to the range $[2, 6]^2$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Mixture $d_x = 1,\ d_y = 1$	Int.	OOID	Ext.
ConvCNP (AR)	**0.45**±0.04	**0.45**±0.04	**0.30**±0.04
ConvLNP (E)	0.12±0.04	0.12±0.04	−0.37±0.03
FullConvGNP	−0.05±0.03	−0.05±0.03	−0.49±0.01
ConvLNP	−0.06±0.03	−0.06±0.03	−0.88±0.02
ACNP (AR)	−0.19±0.02	−1.32±0.01	−1.32±0.01
ConvCNP	−0.23±0.04	−0.24±0.04	−1.23±0.01
ConvGNP	−0.24±0.02	−0.23±0.02	−1.00±0.02
AGNP	−0.41±0.02	−1.03±0.03	−2.61±0.06
ALNP	−0.61±0.02	−1.01±0.02	−1.03±0.02
ALNP (E–M)	−0.63±0.02	−1.24±0.01	−1.10±0.02
ALNP (E)	−0.67±0.02	−1.52±0.02	−2.25±0.03
LNP (E–M)	−0.68±0.01	−3.05±0.04	−1.75±0.01
GNP	−0.70±0.02	−2.22±0.05	−1.62±0.04
LNP (E)	−0.71±0.01	−3.71±0.05	−2.08±0.01
LNP	−0.72±0.01	−1.46±0.01	−1.29±0.02
ACNP	−0.79±0.02	−1.31±0.01	−1.25±0.01
CNP (AR)	−1.00±0.02	−1.15±0.02	−1.09±0.02
CNP	−1.05±0.02	−1.14±0.02	−1.15±0.02
ConvLNP (E–M)	−1.41±0.15	−1.43±0.13	−3.40±0.06
Trivial	−1.32±0.00	−1.32±0.00	−1.32±0.00

Mixture $d_x = 1,\ d_y = 2$	Int.	OOID	Ext.
ConvLNP (E)	**−0.05**±0.03	**−0.05**±0.03	**−0.50**±0.02
ConvCNP (AR)	−0.17±0.02	−0.17±0.02	−0.29±0.02
FullConvGNP	−0.27±0.01	−0.27±0.01	−0.63±0.01
ConvGNP	−0.29±0.02	−0.29±0.02	−2.59±0.06
ACNP (AR)	−0.35±0.02	−1.43±0.02	−1.43±0.02
ConvLNP	−0.39±0.03	−0.40±0.03	−1.10±0.02
AGNP	−0.57±0.02	−3.29±0.07	−3.05±0.06
GNP	−0.60±0.01	−1.70±0.03	−1.67±0.03
ConvCNP	−0.68±0.02	−0.68±0.02	−1.39±0.01
ALNP	−0.76±0.02	−1.19±0.02	−1.19±0.02
LNP	−0.89±0.01	−3.18±0.05	−1.40±0.02
LNP (E–M)	−0.89±0.01	−4.43±0.07	−1.62±0.02
ALNP (E–M)	−0.92±0.02	−1.47±0.03	−1.46±0.03
LNP (E)	−0.92±0.01	−4.71±0.07	−1.86±0.01
ACNP	−0.93±0.02	−1.42±0.02	−1.53±0.03
ALNP (E)	−0.95±0.02	−1.75±0.04	−1.73±0.04
CNP (AR)	−1.15±0.02	−1.27±0.02	−1.21±0.02
CNP	−1.18±0.02	−1.32±0.03	−1.27±0.02
ConvLNP (E–M)	−3.45±0.07	−3.40±0.07	−5.30±0.07
Trivial	−1.46±0.00	−1.46±0.00	−1.46±0.00

Table D.11 For the mixture synthetic experiments with one-dimensional inputs, average log-likelihoods normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]$ where the models where trained ("Int."); interpolation within the range $[2, 6]$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]$ to the range $[2, 6]$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

Mixture $d_x = 2$, $d_y = 1$	Int.	OOID	Ext.	Mixture $d_x = 2$, $d_y = 2$	Int.	OOID	Ext.
ConvCNP (AR)	**−0.10** ±0.03	**−0.10** ±0.03	**−0.34** ±0.03	ConvCNP (AR)	**−0.62** ±0.01	**−0.62** ±0.01	**−0.79** ±0.01
ConvCNP	−0.49 ±0.03	−0.49 ±0.03	−1.45 ±0.02	ConvGNP	−0.74 ±0.01	−0.74 ±0.01	−1.43 ±0.02
ConvGNP	−0.50 ±0.02	−0.50 ±0.02	−1.24 ±0.02	ConvLNP (M)	−0.78 ±0.02	−0.79 ±0.02	−1.25 ±0.02
ConvLNP (M)	−0.57 ±0.02	−0.57 ±0.02	−1.07 ±0.02	ConvCNP	−0.85 ±0.01	−0.85 ±0.01	−1.50 ±0.02
ConvLNP (E)	−0.63 ±0.02	−0.63 ±0.02	−1.08 ±0.02	ACNP (AR)	−0.85 ±0.01	−4.30 ±0.09	−4.24 ±0.09
ALNP (M)	−0.73 ±0.02	−1.04 ±0.02	−1.05 ±0.02	ALNP (M)	−0.88 ±0.01	−1.41 ±0.03	−1.21 ±0.02
ConvLNP (E–M)	−0.76 ±0.02	−0.76 ±0.02	−1.37 ±0.02	ConvLNP (E)	−0.92 ±0.01	−0.92 ±0.01	−1.41 ±0.02
ACNP (AR)	−0.77 ±0.01	−1.28 ±0.01	−1.30 ±0.01	AGNP	−0.93 ±0.01	−1.34 ±0.01	−1.21 ±0.02
AGNP	−0.78 ±0.01	−1.32 ±0.02	−1.32 ±0.02	ACNP	−0.99 ±0.02	−4.19 ±0.10	−1.27 ±0.02
ACNP	−0.91 ±0.02	−1.29 ±0.01	−1.17 ±0.02	ConvLNP (E–M)	−1.05 ±0.01	−1.05 ±0.01	−1.73 ±0.03
LNP (M)	−0.91 ±0.02	−1.44 ±0.04	−1.19 ±0.02	LNP (M)	−1.07 ±0.01	−4.04 ±0.11	−1.31 ±0.02
LNP (E–M)	−0.92 ±0.02	−1.51 ±0.02	−1.38 ±0.02	ALNP (E–M)	−1.11 ±0.02	−2.65 ±0.62	−1.23 ±0.02
LNP (E)	−0.93 ±0.02	−1.74 ±0.03	−1.44 ±0.02	ALNP (E)	−1.12 ±0.02	−8.22 ±7.64	−1.26 ±0.02
ALNP (E–M)	−1.00 ±0.02	−1.07 ±0.02	−1.08 ±0.02	GNP	−1.20 ±0.02	−1.39 ±0.03	−1.21 ±0.02
ALNP (E)	−1.03 ±0.02	−1.08 ±0.02	−1.09 ±0.02	CNP (AR)	−1.20 ±0.02	−2.61 ±0.07	−1.32 ±0.02
CNP (AR)	−1.06 ±0.02	−1.07 ±0.02	−1.08 ±0.02	CNP	−1.20 ±0.02	−2.60 ±0.08	−1.23 ±0.02
GNP	−1.06 ±0.02	−1.09 ±0.02	−1.08 ±0.02	LNP (E–M)	−1.20 ±0.02	−3.23 ±0.10	−1.24 ±0.02
CNP	−1.07 ±0.02	−1.09 ±0.02	−1.10 ±0.02	LNP (E)	−1.20 ±0.02	−3.25 ±0.10	−1.24 ±0.02
Trivial	−1.32 ±0.00	−1.32 ±0.00	−1.32 ±0.00	*Trivial*	−1.46 ±0.00	−1.46 ±0.00	−1.46 ±0.00
FullConvGNP				FullConvGNP			

Table D.12 For the mixture synthetic experiments with two-dimensional inputs, average log-likelihoods normalised by the number of target points. Shows for one-dimensional outputs ($d_y = 1$) and two-dimensional outputs ($d_y = 2$) the performance for interpolation within the range $[-2, 2]^2$ where the models where trained ("Int."); interpolation within the range $[2, 6]^2$ which the models have never seen before ("OOID"); and extrapolation from the range $[-2, 2]^2$ to the range $[2, 6]^2$ ("Ext."). Models are ordered by interpolation performance. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Trivial refers to predicting the empirical means and standard deviation of the test data. Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with "F". Numbers which are missing could not be run.

D.9 Details of Sim-to-Real Transfer Experiments

D.9.1 Description of Experiment

Our goal will be to make predictions for the famous hare–lynx data set. The hare–lynx data set is a time series from 1845 to 1935 recording yearly population counts of a population of Snowshoe hares and a population of Canadian lynx (MacLulich, 1937). A digital version extracted from the original graph by MacLulich (1937) is available by Hundley (2022).[2] Hundley (2022), the author of this digital source, says that other authors caution that the hare–lynx data is actually a composition of multiple time series, and presents the data with that caution. We, therefore, also present the data with this caution. Figure D.6a visualises the hare–lynx data set.

To make predictions for the hare–lynx data set, we use the Lotka–Volterra equations (Lotka, 1910; Volterra, 1926), also called the predator–prey equations. The Lotka–Volterra equations are an idealised mathematical model for the population counts of a prey population and a predator population:

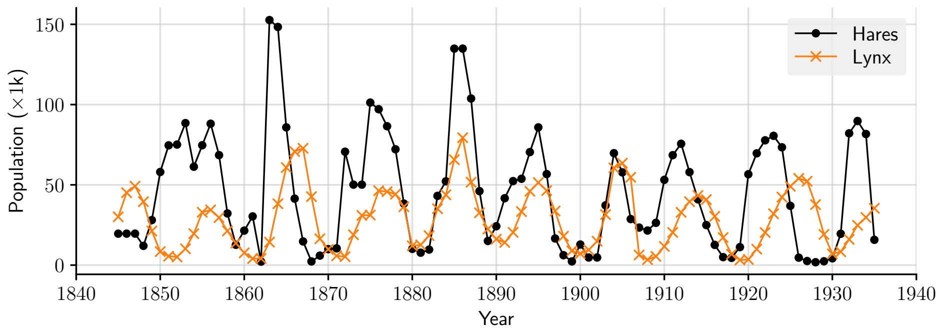
$$\begin{aligned} \text{prey population:} \qquad x'(t) &= \alpha x(t) - \beta x(t)y(t), && \text{(D.36)} \\ \text{predator population:} \qquad y'(t) &= -\delta y(t) + \gamma x(t)y(t). && \text{(D.37)} \end{aligned}$$

These differential equations say that the prey population naturally grows exponentially with rate α, and that the predator population naturally decays exponentially with rate δ. In addition, the predators hunt the prey. The resulting additional growth in the predator population and the resulting additional decrease in the prey population are both proportional to the product of the densities. In this idealised mathematical form, the population counts converge to a smooth, noiseless limit cycle and then perfectly track this limit cycle ever after. This is unlike real-world predator–prey population counts, which exhibit noisy behaviour and imperfect cycles. We therefore consider a stochastic version of the Lotka–Volterra equations, given by the following two coupled stochastic differential equations:

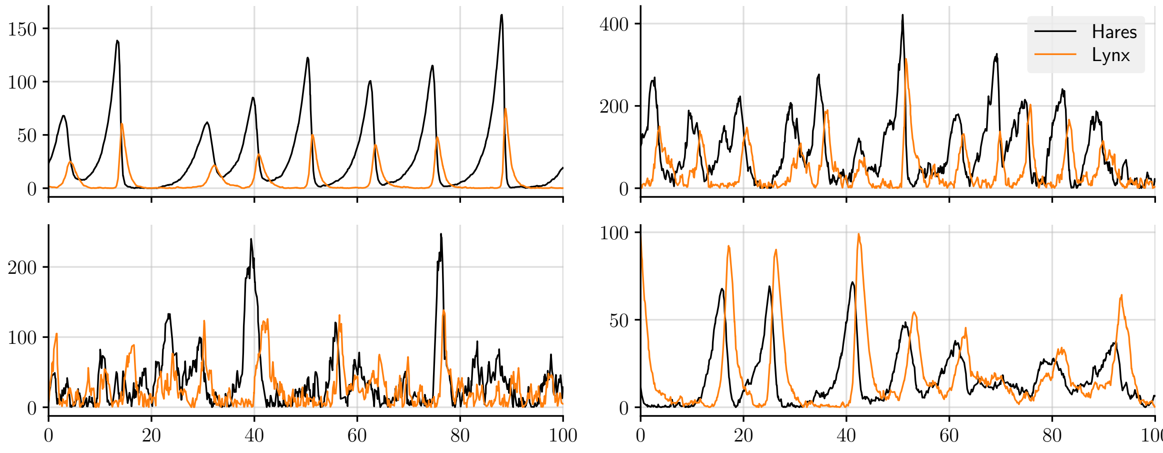
$$\begin{aligned} \mathrm{d}X_t &= \alpha X_t \,\mathrm{d}t - \beta Y_t X_t \,\mathrm{d}t + \sigma X_t^{\nu} \,\mathrm{d}W_t^{(1)}, && \text{(D.38)} \\ \mathrm{d}Y_t &= -\gamma X_t \,\mathrm{d}t + \delta Y_t X_t \,\mathrm{d}t + \sigma Y_t^{\nu} \,\mathrm{d}W_t^{(2)} && \text{(D.39)} \end{aligned}$$

where $W^{(1)}$ and $W^{(2)}$ are two independent Brownian motions. Compared to the Lotka–Volterra equations, equation D.38 and equation D.39 have two additional

[2]See http://people.whitman.edu/~hundledr/courses/M250F03/LynxHare.txt.



(a) Illustration of the hare–lynx data set



(b) Four samples from the proposed stochastic version of the Lotka–Volterra equations equation D.38 and equation D.39. The parameters of equation D.38 and equation D.39 are sampled according to Table D.13.

Fig. D.6 Hare–lynx data set and proposed stochastic simulator

terms, $\sigma X_t^\nu \, \mathrm{d}W_t^{(1)}$ and $\sigma Y_t^\nu \, \mathrm{d}W_t^{(2)}$, which introduce noisy behaviour. In these terms, multiplying by X_t^ν and Y_t^ν makes the noise go to zero when X_t and Y_t become small, ensuring that X_t and Y_t remain positive. In addition, we multiply by a parameter $\sigma > 0$ to control the magnitude of the noise, and we raise X_t and Y_t to a power $\nu > 0$ to control how quickly the noise grows as X_t and Y_t grow. Namely, X_t naturally grows exponentially, so, by adding noise of magnitude proportional to X_t, we risk large spikes in the prey population. To moderate this behaviour, we choose ν to be strictly less than one. Finally, to obtain a variety of magnitudes of population counts, we multiply the realisation with a scale η.

After simulating from equation D.38 and equation D.39 a few times, we settle on $\nu = \frac{1}{6}$. For the remainder of the parameters, we simply manually play around with equation D.38 and equation D.39, settle on parameter ranges that look reasonable, and randomly sample parameters from those intervals. Table D.13 summarises the sampling

Parameter	Distribution
Initial condition X_{-10}	$\mathrm{Unif}([5, 100])$
Initial condition Y_{-10}	$\mathrm{Unif}([5, 100])$
α	$\mathrm{Unif}([0.2, 0.8])$
β	$\mathrm{Unif}([0.04, 0.08])$
γ	$\mathrm{Unif}([0.8, 1.2])$
δ	$\mathrm{Unif}([0.04, 0.08])$
ν	Fixed to 1/6
σ	$\mathrm{Unif}([0.5, 10])$
η	$\mathrm{Unif}([1, 5])$

Table D.13 Sampling distributions for the parameters of the stochastic version of the Lotka–Volterra equations equation D.38 and equation D.39. These equations are simulated on a dense grid spanning $[-10, 100]$. The table also shows the distribution for the initial conditions at $t = -10$. To not depend too heavily on these initial conditions, the simulation results on $[-10, 0]$ are discarded.

distributions for all parameters of equation D.38 and equation D.39. Figure D.6b shows four samples from the proposed stochastic model.

To generate a meta–data set, we simulate equation D.38 and equation D.39 on a dense grid spanning 110 years, throw away the first 10 years, and retain between 150 and 250 data points for X_t and Y_t. The numbers of data points and the locations of the data points are sampled separately for X_t and Y_t. Hence, whereas the hare–lynx data is regularly spaced and the populations are always simultaneously observed, our simulator generates data at arbitrary and nonsimultaneous points in time. We split these data sets into context and target sets in three different ways. To train the models, for every batch, we randomly choose one of the interpolation, forecasting or reconstruction tasks by rolling a three-sided die. We will also perform these tasks on the real hare–lynx data; in that case, for interpolation, we let the number of target points per output be between one and fifteen. The tasks on simulated and real data are similar, but slightly differ in the number of context and target points.

To deal with the positivity of population counts, we transform the marginals of all models to distributions on $(0, \infty)$ by pushing the marginals through $x \mapsto \log(1 + x)$.

For this experiment, the learning rate is $1 \cdot 10^{-4}$, the margin is 1, and the points per unit is 4. We trained the models for 200 epochs.

The convolutional models use a U-Net architecture with seven layers instead of six where, in the first layer, the stride is one instead of two. For the kernel architecture of the FullConvGNP, we reduce the points per unit and the number of channels in the U-Net by a factor two.

Model	Int. (S)	For. (S)	Rec. (S)	Int. (R)	For. (R)	Rec. (R)
FullConvGNP	**–3.29** ±0.02	**–3.46** ±0.02	–3.79 ±0.02	–4.16 ±0.04	**–4.28** ±0.04	–4.45 ±0.00
ConvCNP (AR)	**–3.30** ±0.02	**–3.47** ±0.02	**–3.60** ±0.02	**–4.10** ±0.03	**–4.27** ±0.03	**–4.32** ±0.01
ConvNP (ML)	–3.41 ±0.02	–3.84 ±0.02	–4.44 ±0.02	**–4.13** ±0.04	–4.45 ±0.05	–4.54 ±0.01
ConvGNP	–3.47 ±0.02	–3.65 ±0.02	–4.15 ±0.02	–4.21 ±0.05	–4.82 ±0.13	–4.61 ±0.01
ConvCNP	–3.47 ±0.02	–4.06 ±0.02	–4.85 ±0.02	–4.17 ±0.04	–4.70 ±0.06	–4.97 ±0.01
ConvNP (ELBO)	–3.77 ±0.02	–3.83 ±0.02	–4.12 ±0.02	–5.45 ±0.05	–5.47 ±0.07	–6.39 ±0.05
ANP (ML)	–4.09 ±0.02	–4.32 ±0.02	–4.55 ±0.02	–4.31 ±0.03	–4.43 ±0.04	–4.49 ±0.01
ANP (ELBO–ML)	–4.22 ±0.02	–4.54 ±0.02	–4.80 ±0.02	–4.58 ±0.11	–4.58 ±0.04	–4.68 ±0.01
ACNP (AR)	–4.23 ±0.02	–4.44 ±0.02	–4.58 ±0.02	–4.40 ±0.03	–4.55 ±0.04	–4.59 ±0.02
ANP (ELBO)	–4.32 ±0.03	–4.58 ±0.02	–4.82 ±0.02	–4.71 ±0.15	–4.63 ±0.05	–4.70 ±0.01
ACNP	–4.34 ±0.02	–4.65 ±0.02	–4.88 ±0.02	–4.43 ±0.04	–4.58 ±0.04	–4.74 ±0.00
ConvNP (E.–M.)	–6.71 ±0.05	–8.44 ±0.11	F	–7.20 ±0.31	F	F

Table D.14 Normalised log-likelihoods in the predator–prey experiments. Shows the performance for interpolation (“Int.”), forecasting (“For.”), and reconstruction (“Rec.”) on simulated (“S”) and real (“R”) data. Models are ordered by interpolation performance on simulated data. The latent variable models are trained and evaluated with the ELBO objective (E); trained and evaluated with the ML objective (M); and trained with the ELBO objective and evaluated with the ML objective (E–M). Errors indicate the central 95%-confidence interval. Numbers which are significantly best ($p < 0.05$) are boldfaced. Numbers which are very large are marked as failed with “F”. Numbers which are missing could not be run.

D.9.2 Full Results

In Table D.14, we present the full results for the sim-to-real experiments.

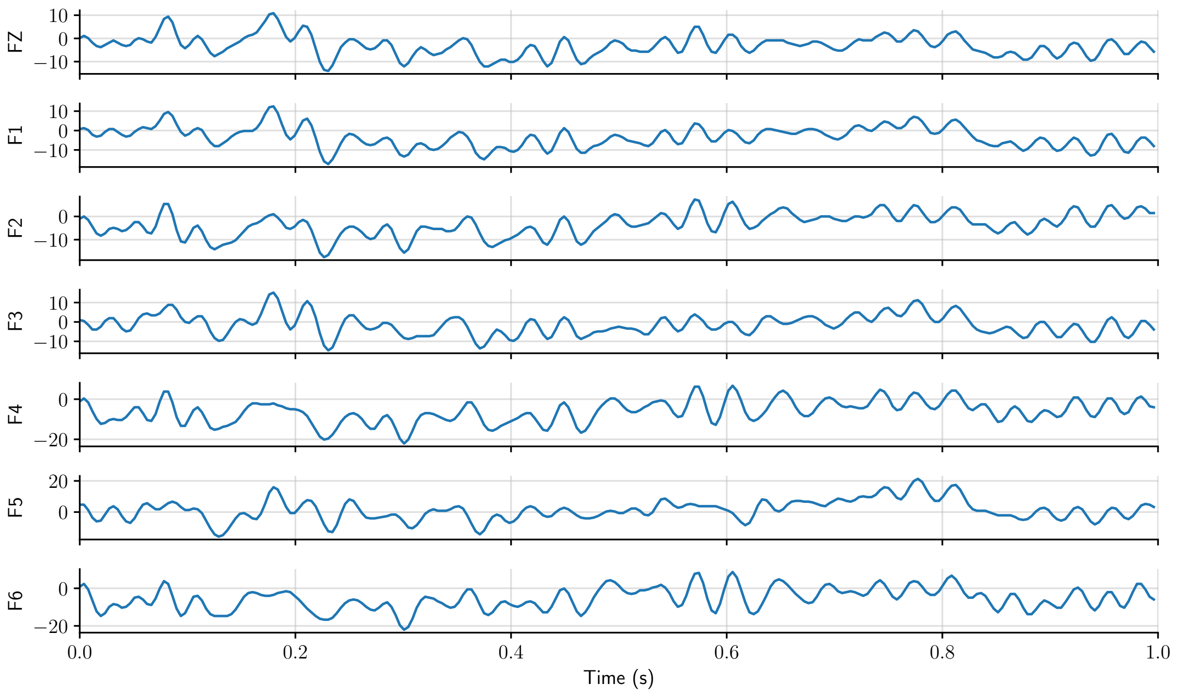


Fig. D.7 Example of trial in the EEG data set. Note that the signals for all electrodes appear correlated, but are subtly different.

D.10 Details of Electroencephalography Experiments

We explore an electroencephalography data set collected from 122 subjects (Begleiter, 2022). There are two groups of subjects: alcoholic and control. Every subject was subjected to a single stimulus or two stimuli, and their response was measured with 64 electrodes placed on a subject's scalp. These measurements are in terms of *trials*, where a trial consists of 256 samples of the electrodes spanning one second. The data sets contains up to 120 trials for each subject. The data is available at https://archive.ics.uci.edu/ml/datasets/eeg+database and the collection is described in detail by Zhang et al. (1995). In this experiment, we focus only on seven frontal electrodes: FZ, F1, F2, F3, F4, F5, and F6. Figure D.7 illustrates a trial of a subject, showing the samples for these seven electrodes.

We randomly split all subjects into three sets: an evaluation set consisting of ten subjects, a cross-validation set consisting of ten other subjects, and a training set consisting of all remaining subjects. For each of these sets, we create a meta–data set by aggregating the trials for all subjects. We split every trial into a context and target set in the same three ways as for the predator–prey experiment. First, for all seven electrodes separately, randomly designate between 50 and 256 points to be the target points and let the remainder (between 0 and 206) be the context points. This task is called *interpolation* and is the primary measure of performance. Additionally, randomly choose one of the seven electrodes and, for that choice, split the data in two exactly like for forecasting. For all other electrodes, append all data to the context set. This task is called *reconstruction* and measures the model's ability to infer a signal for one electrode from the others. We train all models on the interpolation task, and evaluate the models on the interpolation and reconstruction task.

For this experiment, the learning rate is $5 \cdot 10^{-5}$, the margin is 0.1, and the points per unit is 256. We trained the models for 1000 epochs. For the FullConvGNP, the learning rate is $2 \cdot 10^{-4}$. The training run for the FullConvGNP was terminated after 84 hours, reaching epoch 127.

The convolutional models use a U-Net architecture where, in the first layer, the stride is one instead of two. In addition, the number of channels are adjusted as follows: the ConvCNP and ConvGNP use 128 channels, the ConvLNP uses 96 channels, and the FullConvGNP uses 64 channels. The length scales of the Gaussian kernels of the convolutional model is initialised to 0.77/256. To scale to seven outputs, the deep set–based and attentive models reuse the same encoder for every output dimension.

D.11 Details of Environmental Data Assimilation Experiment

In this section we provide further details on the Antarctic cloud cover data assimilation experiment described in Section 6.4.4.

D.11.1 Data considerations

Data sources. Daily-averaged cloud cover reanalysis data was obtained from ERA5 (Hersbach et al., 2020). An Antarctic land mask and elevation field was obtained from the BedMachine dataset (Morlighem, 2020).

Figure D.8 shows an empircal density of the ERA5 cloud cover values calculated over the models' training period of 2000-2013. The spikes at 0 and 1 correspond largely to values of exactly 0 and exactly 1. This motivates the beta-categorical likelihood described in Section 6.4.4.

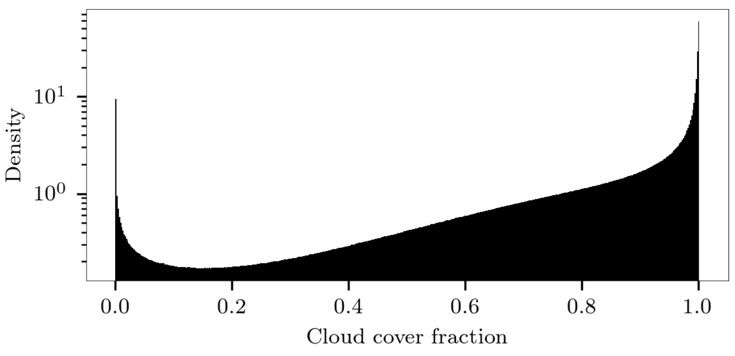


Fig. D.8 Empirical density of ERA5 cloud cover fraction computed over the period 2000-2013.

Data preprocessing. The cloud cover data and land/elevation auxiliary data were regridded from lat/lon to a Southern Hemisphere Equal Area Scalable Earth 2 (EASE2) grid at 25 km resolution and cropping to a size of 280 $\times$ 280. This centres the data on the South Pole.

Data normalisation. We normalised the data before passing it to the convolutional NP models. The cloud cover and land mask data already took appropriate normalised values in $[0, 1]$. The elevation field was normalised from metres to values in $[0, 1]$.

The input coordinates x were normalised from metres to take values in $[-1, 1]$.

D.11.2 Model considerations

Here we provide details on the training procedure and architectures for each of the convolutional NP models in the Antarctic data assimilation experiment.

Generating the training, validation, and test tasks. Following meta-learning principles, we collect data from day τ into a task D_τ. Each task D_τ was generated by first drawing the integer number of simulated cloud cover context points $N^{(\mathrm{c})} \sim \mathrm{Unif}\{1, 2, \ldots 500\}$. Letting $N^{(\mathrm{c})}$ vary encourages the model to learn to deal with both data-sparse and data-rich scenarios. The number of target points $N^{(\mathrm{t})}$ was fixed to a value of 2,000.

Next, the input locations $\boldsymbol{x}_\tau^{(\mathrm{c})}$ and $\boldsymbol{x}_\tau^{(\mathrm{t})}$ were sampled uniformly at random across the entire 280×280 input space and the corresponding $\boldsymbol{y}_\tau$ values were sampled without observation noise.

For the training dates, the random seed used for generating D_τ is updated every epoch, allowing for an infinitely growing simulated training data set. In contrast, for the validation and test dates, the random seeds were held fixed so that metrics computed over the validation and test sets are not stochastic.

Training procedure. Each model was trained for 150 epochs on 14 years of data from 2000–2013. An Adam optimiser was used with a learning rate of 5×10^{-5} and a batch size of 2. For the loss functions we use a negative log-likelihood loss function for the ConvCNP and ConvGNP. For the ConvLNP we use the ELBO objective and fix the variance of the observation noise to 0.01 for the first four epochs. Validation data from 2014–2017 was used for checkpointing the model weights using the per-datapoint predictive log-likelihood. The two year period of 2018–2019 data was reserved for the test set.

The time taken to train each model on a Tesla V100 GPU is as follows:

- ConvCNP: 25.0 hours,
- ConvGNP: 27.5 hours,
- ConvLNP: 43.6 hours.

Architectures. For each model, the U-Net component of the encoder uses 5×5 convolutional kernels with the following sequence of channel numbers (d.s. = 2×2 downsample layer, u.s. = 2×2 upsample layer): $16 \xrightarrow{\text{d.s.}} 32 \xrightarrow{\text{d.s.}} 64 \xrightarrow{\text{d.s.}} 128 \xrightarrow{\text{u.s.}} 64 \xrightarrow{\text{u.s.}} 32 \xrightarrow{\text{u.s.}} 16$. We use bilinear resize operators for the upsampling layers to fix checkerboard artifacts that we encountered when using standard zero-padding upsampling (Odena et al., 2016). We use a margin of 0.1 and 150 points per unit for the encoder's internal discretisation.

The length scales of the Gaussian kernels for both the encoder and decoder SetConv layers are set to 1/150 and held fixed during training. These architecture choices result in a receptive field of 0.433 in normalised input space, or roughly 1.500 km in raw input space, spanning around 20% of the region in Figure 6.6 in either dimension.

For the ConvGNP we use 128 basis functions for the low-rank covariance parameterisation described in Markou et al. 2021.

For the ConvLNP we use an 8-dimensional latent variable and evaluate the ML objective (Foong et al., 2020) using 8 latent samples.

The number of learnable parameters for each model is as follows:

- ConvCNP: 618 k,
- ConvGNP: 621 k,
- ConvLNP: 1.234 k (increase due to second UNet architecture after the latent variable).

The difference in parameters from switching to a beta-categorical likelihood from a Gaussian likelihood is negligible.

Input data. Each model receives two context sets as input. The first contains observations of the simulated ERA5 daily-average cloud cover. The second contains a set of 6 gridded auxiliary variables. These are elevation, land mask, $\cos(2\pi \times \text{day of year}/365)$, $\sin(2\pi \times \text{day of year}/365)$, x_1, and x_2. The elevation and land mask auxiliary fields allow the models to predict spatial non-stationarity. For example, the convolutional filters of the model's encoder could learn how cloud cover around the Antarctic coastline behaves differently to the centre of the continent. The cos and sin variables inform the model at what time of year D_τ corresponds to, helping with learning seasonal variations in the data. The x_1 and x_2 inputs again help with breaking translation equivariance in the convolutional filters by informing the model where in input space the data corresponds to.

D.11.3 Antarctic cloud cover model samples

Figure D.9 gives a detailed breakdown of sample extrapolation ability, showing seven samples from the four Antarctic cloud cover models. The AR ConvCNP samples display remarkable structure and variation while still closely interpolating the context points. The samples also provide interesting scenarios in the gaps between the context points on the left hand side.

In contrast, the ConvCNP samples are incoherent, underestimating the probability of joint events.

The ConvLNP samples were generated by sampling from the latent variable and then computing the mean of the marginal distributions. As is visible in Figure D.9, the ConvLNP displays low sample variance with respect to the latent variable. However, to the best of our knowledge we used a faithful reproduction of the original ConvLNP model, so we leave a more rigorous treatment of this undesirable behaviour to future work.

For all the non-AR models, the limited receptive field size leads to samples on the right hand side becoming independent of the context observations on the left hand side after roughly 750 km of distance from them. This results in the models defaulting to some mean representation of the data. It is interesting to see that all the non-AR models display similar marginal mean structure, with greater cloud cover towards the centre of the continent and lower cloud cover towards the coastline, followed by increased cloud levels over the Southern Ocean.

The AR samples were drawn on a sparse 70x70 grid spanning the entire input space to save compute time. The ConvCNP model was then conditioned on these AR samples and the predictive mean was computed over the dense 280x280 target space. It took 14 minutes to generate these AR ConvCNP samples on a Tesla V100 GPU.

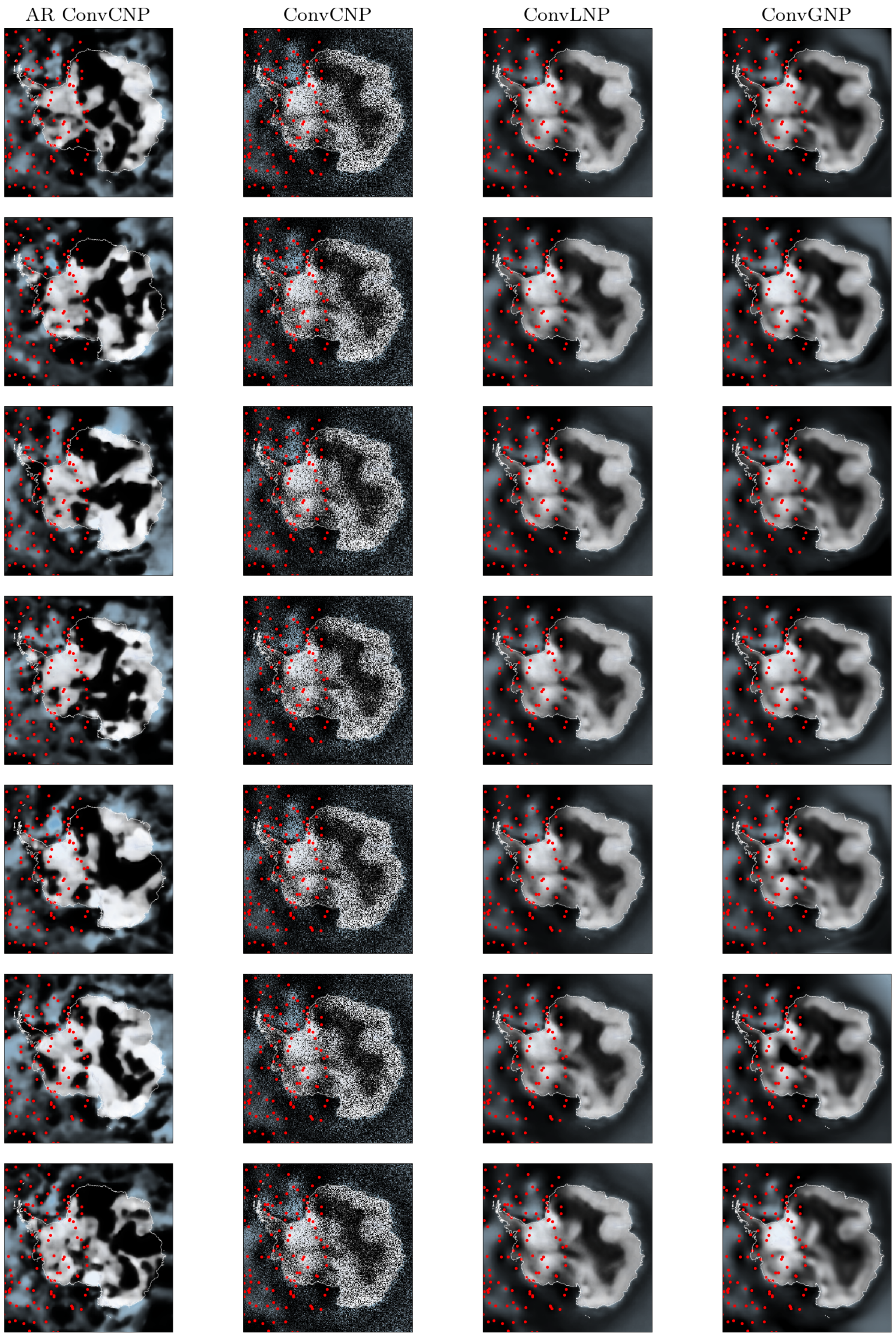


Fig. D.9 Seven samples from each model in the Antarctic cloud cover sample extrapolation task for 25/06/2018.

D.12 Details of Climate Downscaling Experiments

D.12.1 Description of Experiment

The MLP ConvGNP (Markou et al., 2022) can be used to successfully model dependencies between outputs in a statistical downscaling task, improving log-likelihoods over the MLP ConvCNP (Vaughan et al., 2022) and enabling coherent samples. In this experiment, we demonstrate that the AR ConvCNP can also be used for this purpose.

The goal of this experiment is to estimate the maximum daily temperature at 589 weather stations around Germany. To estimate these temperatures, we follow Vaughan et al. (2022) and use 25 coarse-grained ERA-Interim reanalysis variables (Dee et al., 2011) in combination with 1 km–resolution elevation data (Earth Resources Observation and Science Center, U.S. Geological Survey, U.S. Department of the Interior, 1997). We also consider a second setup where we reveal some of the weather station observations. These revealed weather station observations can be used by the models to aid downscaling performance.

The ERA-Interim reanalysis variables considered are tabulated in Table D.15. In contrast to previous downscaling work, which degrade reanalysis data to between 2° and 2.5°, we opt to use the ERA-Reanalysis data at the native 0.75° resolution, consistent with the latest high-resolution climate models with horizontal resolution ranging from 0.5° to 1.0°. All variables are spatially subset to between 6° to 16° longitude and 47° to 55° latitude, covering Germany. The weather station data are a subselection from of the European Climate Assessment & Dataset (Tank et al., 2002) and are available at https://www.ecad.eu; we use the blended data. Like for the ERA-Reanaysis variables, we take the weather stations located within the aforementioned square. The locations of the weather stations around Germany are visualised in Figure D.10. The 1 km–resolution elevation data is taken from the United States Geological Survey GTOPO30 elevation data set available at https://doi.org/10.5066/F7DF6PQS. This provides global elevation data at 30-arc second resolution, which is approximately 1 km.

Following the VALUE framework (Maraun et al., 2015), we consider all days of the years 1979–2008 and split these years into five folds. We use the first four folds (spanning 1979–2003) for training, holding out the last 1000 days for cross-validation; and use the fifth fold (spanning 2003–2008) for evaluation.

Variable	Level	Units
Surface		
Maximum temperature	2 m	degrees Celsius
Mean temperature	2 m	degrees Celsius
Northward wind	10 m	knots
Eastward wind	10 m	knots
Upper atmosphere		
Specific humidity	850, 700, and 500 hPa	degrees Celsius
Mean temperature	850, 700, and 500 hPa	degrees Celsius
Northward wind	850, 700, and 500 hPa	knots
Eastward wind	850, 700, and 500 hPa	knots
Invariant		
Angle of sub-grid-scale orography	surface	
Anisotropy of sub-grid-scale orography	surface	
Standard deviation of filtered sub-grid-scale orography	surface	
Standard deviation of orography	surface	
Geopotential	surface	metres
Longitude	surface	degrees Celsius
Latitude	surface	degrees Celsius
Temporal		
Fractional position in the year t transformed encoded with $t \mapsto (\cos(2\pi t), \sin(2\pi t))$		

Table D.15 ERA-Interim reanalysis predictors.

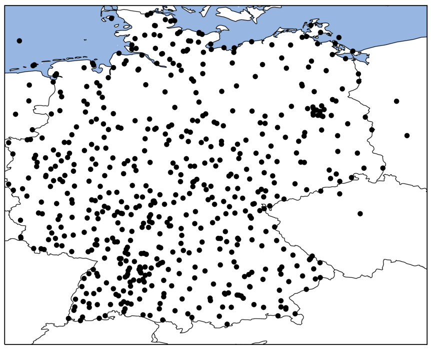


Fig. D.10 Locations of the 589 weather stations around Germany in the downscaling experiments.

D.12.2 Multiscale Convolutional Architecture

Deploying the AR ConvCNP in this downscaling experiment comes with a significant challenge. Because the elevation data has a fine resolution of 1 km, we expect that predictions by the AR ConvCNP will vary roughly also on this length scale. In the autoregressive sampling procedure (Procedure 6.2.1), samples from the model will be fed back into the model. Therefore, the AR ConvCNP must handle context data which varies on a 1 km spatial scale, which means that the discretisation of the AR ConvCNP must roughly be a 1 km–resolution grid. Unfortunately, making the discretisation this fine is prohibitively expensive and imposes prohibitive memory requirements. It is this limitation that prevents us from extending the Vaughan et al. (2022)'s MLP ConvCNP and Markou et al. (2022)'s MLP ConvGNP to include additional weather station observations. We must therefore innovate on the AR ConvCNP design to come up with a convolutional architecture that can handle such a fine discretisation at reasonable computational expense.

The architecture that we propose is a *multiscale architecture* operating on multiple spatial length scales. Let us divide the context set $D = D_{\text{lr}} \cup D_{\text{mr}} \cup D_{\text{hr}}$ into a low-resolution component D_{lr}, a medium-resolution component D_{mr}, and a high-resolution component D_{hr}. Let the low-resolution component D_{lr} consist of features of the context set that vary on a *long* spatial length scale, the medium-resolution component D_{mr} of features that vary on a *medium-range* spatial length scale, and the high-resolution component D_{hr} of features that vary on a *short* spatial length scale. The central assumption of the architecture is that predictions for target points depend on precise short-length-scale details D_{hr} nearby, but that this dependence weakens as we move away from the target point, starting to depend more on broad-stroke long-lengthscale components D_{lr}. For example, predictions might depend on detailed orographic information nearby, but more on general orographic shapes farther away.

Figure D.11 depicts the multiscale architecture. The architecture is a cascade of three convolutional deep sets, parametrised by three CNNs; please see the caption. The low-resolution CNN handles the context data D_{lr} with a long spatial length scale. Because these features have a long spatial length scale, the CNN can get away with a low-resolution discretisation. The output of the low-resolution CNN then feeds into a medium-resolution CNN. The medium-resolution CNN handles the context data D_{mr} with a medium spatial length scale and has a medium-resolution discretisation. Finally, the output of the medium-resolution CNN feeds into a high-resolution CNN. This CNN handles the context data D_{hr} with a short spatial length scale and has a high-resolution discretisation.

The key to the computational efficiency of this architecture is that we construct the high-resolution discretisation only locally to the target points: a small square covering $0.25°$ more than the most extremal target points. If the target points are confined to a small region, then the high-resolution grid will also be small, covering only $0.25°$ more than that region. Crucially, the high-resolution grid will not be constructed over all of Germany, like it would if we were to more naively apply the ConvCNP with a high-resolution discretisation, incurring prohitive computational cost. Even though the high-resolution grid is only constructed locally to the target points, the model can still capture long-range dependencies via the medium-resolution and low-resolution grids. Namely, the medium-resolution grid is a square covering $5°$ more than the most extremal target points, and the low-resolution grid covers all of Germany; see Figure D.11. To utilise this computational gain, the target points must be confined to a small region. This perfectly synergises with the autoregressive sampling procedure (Procedure 6.2.1), because this procedure evaluates the model one target point at a time. The training procedure, however, must be adapted. During training, we subsample the target points to ensure that the target set is always confined to a small square, which is described in Section D.12.4.

During the autoregressive sampling procedure, the AR ConvCNP takes in earlier AR samples from the model. In the architectures of the MLP ConvCNP and MLP ConvGNP, these is no natural context data to which these samples can be appended. Therefore, in addition to the ERA-Interim reanalysis variables and the elevation data, we also let the AR ConvCNP take in observed weather stations as context data. We will append the earlier AR samples to these weather station context data. To have the model make appropriate use of the weather station context set, we must randomly divide the weather stations observations over the context and target set, which we describe in Section D.12.4. We let the low-resolution context data D_{lr} consist of the 25 coarse-grained ERA-Interim reanalysis variables, and let the medium-resolution D_{mr} and high-resolution context data D_{hr} both consist of the weather station observations (and earlier AR samples) and the 1 km–resolution elevation data. When the 1 km–resolution data is fed to the medium-resolution CNN, the data loses some detail, because the internal discretisation of the medium-resolution CNN is coarser than the data; however, when it is fed to the high-resolution CNN, the data retains its detail. The same holds for the weather station observations (and earlier AR samples).

D.12.3 Architectures

MLP ConvCNP and MLP ConvGNP (Markou et al., 2022; Vaughan et al., 2022). The MLP ConvCNP and MLP ConvGNP are a respectively a ConvCNP and ConvGNP where the decoder $\text{dec}_\theta = \mathsf{fuse}_\theta \circ \text{dec}'_\theta$ is decomposed into a convolutional architecture dec'_θ followed by a pointwise MLP fuse_θ:

$$\mathsf{fuse}_\theta(\boldsymbol{z}(\,\boldsymbol{\cdot}\,)) = \mathsf{MLP}_\theta(\boldsymbol{z}(\,\boldsymbol{\cdot}\,), \mathsf{elevation}(\,\boldsymbol{\cdot}\,)). \tag{D.40}$$

In this architecture, the ERA-Interim variables are incorporated via the convolutional architecture, producing the encoding $\boldsymbol{z}(\,\boldsymbol{\cdot}\,)$. On the other hand, as equation D.40 shows, the 1 km–resolution elevation data is included via the pointwise MLP fuse_θ.

Parametrise dec'_θ with a seven-layer residual convolutional neural network (He et al., 2016). Every residual layer involves one depthwise-separable convolutional filter (Chollet, 2017) with kernel size three followed by a pointwise MLP. Every layer has 128 channels, and the network also outputs 128 channels. The discretisation for dec'_θ is the grid of the ERA-Interim reanalysis variables. Parametrise fuse_θ with a three-hidden-layer MLP of width 128.

AR ConvCNP. The AR ConvCNP does not use the pointwise MLP fuse_θ to incorporate the 1 km–resolution elevation data. Instead, it is a normal ConvCNP where the convolutional architecture is implemented by the multi-scale architecture described in Figure D.11.

Parametrise CNN_{lr} with a depthwise-separable residual convolutional neural network like in the MLP ConvCNP and MLP ConvGNP, but use six layers instead of seven. Let CNN_{lr} output 64 channels. The discretisation for CNN_{lr} is the grid of the ERA-Interim reanalysis variables. Parametrise CNN_{mr} with a U-Net (Ronneberger et al., 2015) using an architecture similar to what we have been using. Before the U-turn, let the U-Net have five convolutional layers with kernel size five, stride one for the first layer and stride two afterwards, 64 output channels for the first three layers and 128 output channels afterwards. After the U-turn, instead of using transposed convolutions, use regular convolutions combined with an upsampling layer using bilinear interpolation. Let $\mathsf{CNN}_{\mathrm{mr}}$ output 64 channels. The receptive field of $\mathsf{CNN}_{\mathrm{mr}}$ is approximately $10°$. The discretisation for $\mathsf{CNN}_{\mathrm{mr}}$ is centred around the target points with margin $5°$. Parametrise $\mathsf{CNN}_{\mathrm{hr}}$ with a U-Net like for $\mathsf{CNN}_{\mathrm{hr}}$, but with four convolutional layers before the U-turn. The receptive field of $\mathsf{CNN}_{\mathrm{hr}}$ is approximately $0.5°$. The discretisation for $\mathsf{CNN}_{\mathrm{hr}}$ is centred around the target points with margin $0.25°$.

D.12.4 Training Details

MLP ConvCNP and MLP ConvGNP. The MLP ConvCNP and MLP ConvGNP are trained with learning rate $2.5 \cdot 10^{-5}$ for 500 epochs. For the MLP ConvGNP, to encourage the covariance to fit, we fix the variance of the decoder to $10^{-4}\boldsymbol{I}$ for the first ten epochs.

AR ConvCNP. The AR ConvCNP is trained with learning rate $1 \cdot 10^{-5}$ for 500 epochs. During training and cross-validation, the target points are subsampled to lie in a $3° \times 3°$ square. For training, the number of target points is ensured to be at least ten; and for cross-validation, at least one. The size of the cross-validation set is increased ten fold.

Sampling of data. For the MLP ConvCNP and MLP ConvGNP, since these architectures cannot take in weather station observations, all weather stations are used as context data. For the AR ConvCNP, a data set is split into a context and target set by randomly selecting n points as context points and letting the remainder be target points. Specifically, the number of context points n is sampled from $p(n) \propto e^{-0.01n}$. This splitting is done after subsampling the $3° \times 3°$ square.

D.13 Alternate AR Procedure with Auxiliary Data

We propose an additional procedure which uses autoregressive sampling with auxiliary data to generate more expressive marginal predictives. The input points of the auxiliary data are chosen randomly, and then sampled autoregressively before sampling the target points. Finally, we discard the sampled values for the auxiliary data, but retain the samples for the target points. Adding auxiliary points in this way allows the model to roll out autoregressively with more steps, even if the target set is small (or just a single point). We describe the procedure below:

Procedure D.13.1 (Autoregressive application of neural process with auxiliary data)**.** For a neural process π_θ, context set $D^{(\mathrm{c})} = (\boldsymbol{x}^{(\mathrm{c})}, \boldsymbol{y}^{(\mathrm{c})})$, a target input $x^{(\mathrm{t})}$, a distribution r over $\mathcal{X}$, a number of auxiliary data points $R \in \mathbb{N}$, and a number of trajectories $M \in \mathbb{N}$, let $\mathrm{AuxAR}_{\boldsymbol{x}^{(\mathrm{t})}}(\pi_\theta, D^{(\mathrm{c})}, r, R, M)$ be the distribution defined as follows. We first autoregressively sample the auxiliary data trajectories at random locations sampled from r:

$$\text{for } j = 1, \dots, M \text{ and } \ell = 1, \dots, R, \qquad x_\ell^{(\mathrm{aux},j)} \sim r, \tag{D.41}$$

$$\text{for } j = 1, \dots, M, \qquad \boldsymbol{y}^{(\mathrm{aux},j)} \sim \mathrm{AR}_{\boldsymbol{x}^{(\mathrm{aux},j)}}(\pi_\theta, D^{(\mathrm{c})}). \tag{D.42}$$

Next, conditioned on the auxiliary data, we sample the target point of interest to make predictions. We then marginalise out the auxiliary data by averaging over the M trajectories:

$$y^{(\mathrm{t})} \sim \tfrac{1}{M} \textstyle\sum_{j=1}^{M} P_{x^{(\mathrm{t})}}(\boldsymbol{x}^{(\mathrm{c})} \oplus \boldsymbol{x}^{(\mathrm{aux},j)},\, \boldsymbol{y}^{(\mathrm{c})} \oplus \boldsymbol{y}^{(\mathrm{aux},j)}). \tag{D.43}$$

This procedure introduces three hyperparameters: the distribution from which to draw inputs r, the length of trajectories R, and the number of trajectories to sample M.

In the following experiments, we set the distribution r to be uniform over the training domain with no dependence on the context set or target point of interest: $r = \mathrm{Uniform}([b, h])$, where b and h are the lower and upper bounds of the training domain, respectively. One could experiment with other choices for the distribution r. The trajectory length R is chosen between 0[3] and 8, and the number of trajectories M is chosen between 1 and 128.

[3]A trajectory length of 0 is equivalent to the standard test-time procedure.

D.13.1 Generated Data

We create three data generating processes for our experiments: a mixture of functions, random sawtooth functions, and random audio-like functions. The first two experiments have multi-modal true marginals, whereas the last has heavy-tailed marginals.

Function mixture. The function mixture data are generated by choosing one of the following three functions, the first with a probability of $\frac{1}{4}$, the second with a probability of $\frac{1}{2}$, and the third with a probability of $\frac{1}{4}$:

$$y = x^2 + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 0.25), \tag{D.44}$$
$$y = x + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 0.0625), \tag{D.45}$$
$$y = -x + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 0.25). \tag{D.46}$$

Sawtooth. The sawtooth data are generated from the following function:

$$y(x) = [\omega(dx - \phi)] \bmod 1 \tag{D.47}$$

We sample the frequency $\omega \sim \mathrm{Unif}([3, 5])$, the direction d as either -1 or 1 with equal probability, and the shift as $\phi \sim \mathrm{Unif}([\frac{1}{\omega}, 1])$.

Synthetic Audio. Synthetic audio data are generated by convolving a Dirac comb[4] with a truncated decaying sum of sinusoids:

$$s(t) = \begin{cases} e^{-\frac{t}{\tau}} \left[\sin(\omega_1 t) + \sin(\omega_2 t)\right] & \text{for } 0 \leq t < T, \\ 0 & \text{otherwise,} \end{cases} \tag{D.48}$$
$$f(x) = \mathrm{Comb}_T(x) * s(x), \tag{D.49}$$
$$y = f(x) + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, 0.001) \tag{D.50}$$

where

$$\omega_1, \omega_2 \sim \mathrm{Unif}([50, 70]), \quad T \sim \mathrm{Unif}([0.75, 1.25]), \quad \tau \sim \mathrm{Unif}([0.1, 0.3]). \tag{D.51}$$

We truncate the waves up to the period length, because otherwise the convolution with the Dirac comb would lead to increasing amplitude, resulting in a non-stationary process.

[4]The Dirac comb is defined as $\mathrm{Comb}_T(t) := \sum_{k=-\infty}^{\infty} \delta(t - kT)$ for given period P.

Experiment	ConvCNP	ConvCNP (AuxAR)
Function Mixture	-0.63 ± 0.20	$-\mathbf{0.46} \pm 0.15$
Sawtooth	1.46 ± 0.30	1.64 ± 0.28
Synthetic Audio	0.12 ± 0.14	$\mathbf{0.55} \pm 0.10$

Table D.16 Using autoregressive sampling for marginal approximation improves held-out log-likelihoods on all experiments. Values are normalized by the number of target points. Values which are significantly best ($p < 0.05$) are shown in bold.

D.13.2 Training

For the function mixture experiment, no training is required because we use the analytically derived ideal CNP π_∞ as our model. See Section 6.2.

For the sawtooth and synthetic audio experiments, we train ConvCNP models. We train each model for 100 epochs using 1024 batches per epoch with a batch size of 16. We discretise the encoder by evaluating 64 points per unit. We use a margin of 0.1, and a stride length of 2 for each of the 6 layers of the U-Net. Each layer has 64 channels. The receptive field size from this combination of parameters is 6.953.

During training, we sample a number of context points between uniformly at random from $\{0,\ldots,75\}$, and we sample exactly 100 target points. The context points and target points are sampled uniformly from $[-2, 2]$. We use the Adam optimizer with a learning rate of 3×10^{-4}.

D.13.3 Results

Model	0	1	2	4	8	16
ConvCNP	-0.47 ± 0.07	-0.45 ± 0.18	-0.24 ± 0.12	-0.12 ± 0.12	0.36 ± 0.15	1.07 ± 0.18
ConvCNP (AuxAR)	$\mathbf{0.02} \pm 0.11$	$\mathbf{0.19} \pm 0.11$	$\mathbf{0.24} \pm 0.12$	$\mathbf{0.31} \pm 0.11$	$\mathbf{0.74} \pm 0.10$	1.27 ± 0.12

Table D.17 Log likelihood values for varying context sizes using ConvCNP and ConvCNP (AuxAR) using the function mixture data generator. Column headers indicate the context set size. Log-likelihoods are shown in bold when they are significantly best ($p < 0.05$). Column headers are context sizes. Errors indicate central 95% confidence interval.

In Table D.16, we see that using this procedure improves the held-out log-likelihoods for all of the experiments. We can better understand the utility of this method by observing the performance for different context sizes in Tables D.18 and D.17. For example, the sawtooth data results in Table D.18 show that, for context set sizes 16, the AR method provides no benefits. The increased flexibility of this AR method is not needed in this case — a Gaussian predictive models the marginal sufficiently well.

Model	0	1	2	4	8	16	32
ConvCNP	**−0.18**±0.00	−0.14±0.02	−0.02±0.05	0.42±0.19	1.89±0.23	3.06±0.13	3.54±0.08
ConvCNP (AuxAR)	−0.21±0.02	−0.09±0.04	0.08±0.08	**0.95**±0.16	**2.37**±0.15	3.03±0.13	3.51±0.08

Table D.18 Log likelihood values for varying context sizes using ConvCNP and ConvCNP (AuxAR) using the sawtooth data generator. Column headers indicate the context set size.. Log-likelihoods are shown in bold when they are significantly best ($p < 0.05$). Column headers are context sizes. Errors indicate central 95% confidence interval.

Model	0	1	2
ConvCNP	−1.40±0.11	−0.91±0.28	−0.36±0.20
ConvCNP (AuxAR)	**−1.05**±0.11	−0.63±0.23	−0.29±0.17

Table D.19 Log likelihood values for varying context sizes using ConvCNP and ConvCNP (AuxAR) using the function mixture data generator. Column headers indicate the context set size. Log-likelihoods are shown in bold when they are significantly best ($p < 0.05$). Column headers are context sizes. Errors indicate central 95% confidence interval.

For context set sizes of 4 and 8, on the other hand, we see significant improvements using the AR method. Similarly, for the synthetic audio data results in Table D.17, we see improvements using the ConvCNP (AuxAR) for all context set sizes except 16.

The autoregressive sampling with auxiliary data method shown here shows promise for improving modeling of processes with multi-modal and heavy-tailed marginal distributions — all with no changes to the training procedure. The scenarios where this method are most useful are highly contingent upon the context set size, because of its impact how well the marginals are modeled by Gaussians.

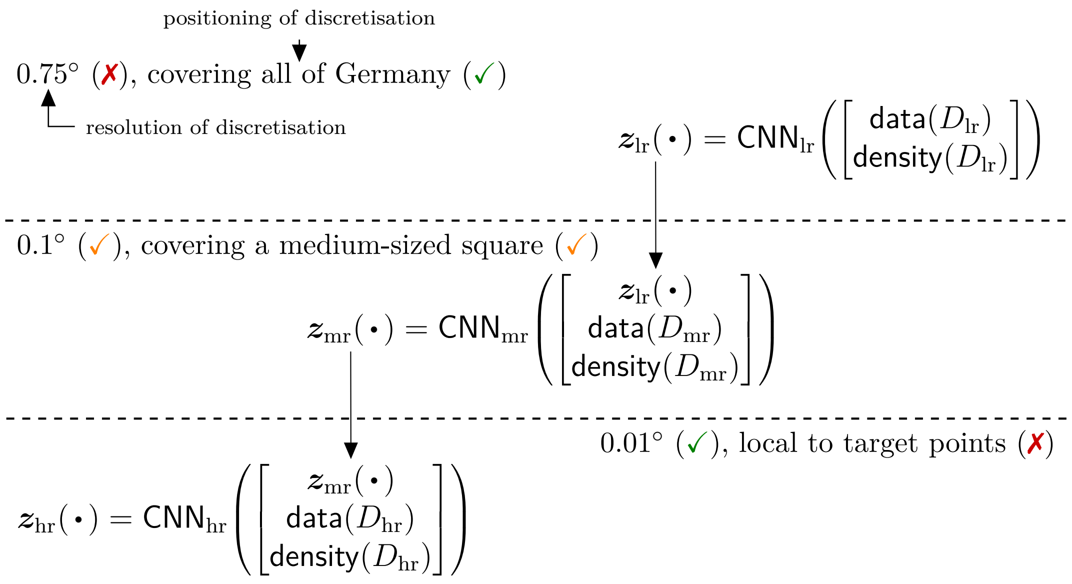

positioning of discretisation
0.75° (✗), covering all of Germany (✓)
resolution of discretisation
$z_{\mathrm{lr}}(\bullet) = \mathsf{CNN}_{\mathrm{lr}}\left(\begin{bmatrix}\mathsf{data}(D_{\mathrm{lr}})\\ \mathsf{density}(D_{\mathrm{lr}})\end{bmatrix}\right)$
0.1° (✓), covering a medium-sized square (✓)
$z_{\mathrm{mr}}(\bullet) = \mathsf{CNN}_{\mathrm{mr}}\left(\begin{bmatrix}z_{\mathrm{lr}}(\bullet)\\ \mathsf{data}(D_{\mathrm{mr}})\\ \mathsf{density}(D_{\mathrm{mr}})\end{bmatrix}\right)$
0.01° (✓), local to target points (✗)
$z_{\mathrm{hr}}(\bullet) = \mathsf{CNN}_{\mathrm{hr}}\left(\begin{bmatrix}z_{\mathrm{mr}}(\bullet)\\ \mathsf{data}(D_{\mathrm{hr}})\\ \mathsf{density}(D_{\mathrm{hr}})\end{bmatrix}\right)$

Fig. D.11 Multiscale architecture for the AR ConvCNP. A cascade of three convolutional deep sets (Gordon et al., 2020) representing a low-resolution, medium-resolution, and high-resolution component. Shows the resolution and positioning of the internal discretisation for every convolutional deep set. The context set $D = D_{\mathrm{lr}} \cup D_{\mathrm{mr}} \cup D_{\mathrm{hr}}$ is also divided into a low-resolution D_{lr}, medium-resolution D_{mr}, and high-resolution component D_{hr}. The low-resolution context data D_{lr} consists of the 25 coarse-grained ERA-Interim reanalysis variables. The medium-resolution D_{mr} and high-resolution context data D_{hr} both consist of the station observations and the 1 km–resolution elevation data. The functions $\mathsf{data}(D)$ and $\mathsf{density}(D)$ produce respectively the data channel and density channel for context data D; see Gordon et al. (2020). The variables $z_{\mathrm{lr}}(\bullet)$, $z_{\mathrm{mr}}(\bullet)$, and $z_{\mathrm{hr}}(\bullet)$ represent intermediate representations as continuous functions, and the maps $\mathsf{CNN}_{\mathrm{lr}}$, $\mathsf{CNN}_{\mathrm{mr}}$, and $\mathsf{CNN}_{\mathrm{hr}}$ are translation-equivariant maps between functions on $\mathcal{X}$. Following the construction of the ConvCNP (Gordon et al., 2020), these maps are all implemented with convolutional neural networks (CNN) using a discretisation. For $\mathsf{CNN}_{\mathrm{lr}}$, the internal discretisation is the 0.75°-resolution grid corresponding to the 25 coarse-grained ERA-Interim reanalysis variables. For $\mathsf{CNN}_{\mathrm{mr}}$, the internal discretisation is a 0.1°-resolution grid spanning 5° more than the most extremal target inputs; the discretisation does *not* depend on the context set. For $\mathsf{CNN}_{\mathrm{hr}}$, the internal discretisation is a 0.01°-resolution grid spanning 0.25° more than the most extremal target inputs; the discretisation also does *not* depend on the context set.

Index