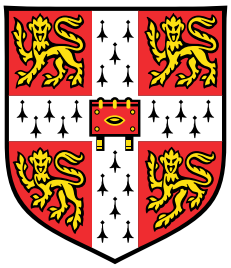
Better Encoders for Neural Process Family Models



Benedict King

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Master of Philosophy in

Machine Learning and Machine Intelligence

Clare College August 2022

Dedicated to the family and friends who have made my time at Cambridge so enjoyable.

In particular my Grandma, whose support made this master's possible for me.

Declaration

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

I further declare the software and data used for this thesis. The experimentation was carried out using Python and its standard scientific and machine learning libraries: Numpy, Scipy, Pandas, and PyTorch. All plots and visualisations included in this report were generated using Matplotlib and Seaborn. The neuralprocesses module that was used extensively and built upon during this thesis was developed by W. Bruinsma, 2022. Additional code that was added to this module to execute experiments contained in this thesis are as follows:

- The addition of adapted decoder likelihoods for the ConvCNP, ConvNP, and CorrConvNP models. Specifically the likelihoods were Bernoulli, Gamma, and composite Bernoulli-Gamma distributions.
- The addition of synthetic data generation code for (i) a sinusoidal synthetic regression experiment with bi-modal noise; (ii) the Mixture of Gaussians classification experiment in arbitrary dimensions with arbitrary number of classes (1D with 2 class was used in this thesis); (iii) the GP-cutoff classification experiments in arbitrary dimensions with 2 classes (2D was used in this thesis); (iv) the Gamma process regression experiments in arbitrary dimensions (2D was used in this thesis).
- Code to conduct the significance testing at the end of Chapter 3. Specifically this code allows for the forced diagonalisation of the CorrConvNP latent variable covariance at a set point during training in order to return a ConvNP.
- Code for generating all figures in Chapters 3, 4 and 5, except for Figure 3.3, which utilised existing code from W. Bruinsma, 2022.
- Code for reading in and processing the rainfall datasets from the Copernicus Climate Change Service, 2021, and the subsequent sim-to-real experiments using this data.

The only external dataset used was from the Copernicus Climate Change Service, 2021.

Total word count (including the appendix): 14,804

Benedict King

August 2022

Acknowledgements

I would like to thank Professor Richard Turner, who originally suggested this project and has provided constant support and direction throughout. In addition I would like to say a huge thank you to Wessel Bruinsma for answering my endless stream of questions, letting me make use of your code-base, and effortlessly fixing my code whenever I got stuck in a sea of unintelligible errors.

Abstract

As research into machine learning matures into its 8^{th} decade it has become increasingly the case that the most celebrated advances have been in areas with the greatest abundance of data. From reinforcement learning models that achieve super-human game-play performance (Silver et al., 2016) to massive language models that are able to generate convincing news stories in the style of specific authors[1] (Brown et al., 2020), the path of progress has often been to make models bigger and supply them with more information.

Whilst in many cases this is a suitable approach, we must not neglect the areas where it is unfeasible. When data is scarce or sampled from many different environments we must turn to methods which are much more efficient in the way that they extract information and do not overfit during training to the data that is available to them. The purpose of this thesis is to explore these problems, and in particular, a recent family of models that have been developed to work well in them, coined Neural Processes.

At their root, neural processes are meta-learners that provide a mapping from a dataset to the predictive posterior distribution of an underlying stochastic processes. In this way they can be thought of as both meta-learning models and stochastic process approximators.

In this thesis we start by exploring the strengths and weaknesses of the existing models in the neural process family (NPF), and build a new model called the *Correlated Convolutional Neural Process* (*CorrConvNP*) to give more robustness and flexibility to changes in task distributions. We show that the CorrConvNP exhibits strong performance over a spectrum of synthetic tasks that reflect real use-cases, and evaluation shows it to perform statistically better than the next best existing model in certain scenarios.

Finally, we show how the CorrConvNP can be extended to different data regimes, such as classifying categories and zero-inflated regression, via a simple change of likelihood function. We then test it on both synthetic classification data sets and a practical rainfall prediction scenario.

[1] See `https://newsyoucantuse.com/ to see the results!`

Table of contents

List of figures

List of tables

Nomenclature

$\boldsymbol{h}(D)$ (or $\mathbb{H}(D)$) A continuous vector (or discretised functional) embedding of a set of data $\mathscr{D}$ for use in a feed-forward (or convolutional) neural network.

$\mathscr{F}_\theta$ A neural network (possibly convolutional) with trainable parameters θ.

$\psi(x,y)$ A pre-aggregation function acting on a pair of points x,y.

$\rho(h)$ A post-aggregation function acting on an aggregate of individual embeddings $\{h_i\} = \{\psi(x_i,y_i)\}$.

D_c The context set. A set of pairs of points $\boldsymbol{x},y$.

D_t The target set. At test-time the target set need not contain y values.

ELBO Evidence Lower Bound. An objective function used to train encoder models.

LV Latent Variable. These exist in the models that have an encoder part to them, as the encoder outputs are the latent variables. They can either take a vector form (for MLP encoder networks) or a functional form (for CNN encoder networks).

MLE Maximum Likelihood Estimation. An objective function simpler than ELBO that predominantly is used to train non-encoder models.

NLL Negative Log-likelihood. A metric used to gauge the extent of training of the models. It is the negative of the log-likelihood, which describes the probability of the test set data given the model that has been trained on the training set. A lower value near to 0 therefore indicates a stronger performance.

NPF The Neural Process Family (the set of all neural process models, both convolutional and non-convolutional).

TE Translation Equivariance. The property of CNNs that when the input domain is translated the output predictions are also translated equally.

Chapter 1

Introduction

1.1 What is a Neural Process?

A neural process is an amortized meta-learning model (Ravi and Beatson, 2019) that is trained on tasks from multiple data distributions to make it directly applicable to unseen tasks at test-time. They make use of a context and target set[1] from each task to first find a good representation of the current task within a learned distribution over tasks, and then condition on that to find a good representation of that task's internal data distribution. As a consequence of meta-learning, it is required that the training tasks follow different distributions, rather than subsets of the same one as in traditional learning. Note that it is still possible to learn a single large task however, as it can be separated into multiple smaller tasks with different distributions. For example, consider breaking up the task of learning to distinguish between K classes of image into multiple smaller tasks, each distinguishing between $k < K$ image classes.

1.2 Why Neural Processes?

A strong starting point for any piece of research is to consider why it is important and what gap it is trying to fill. This is quite straightforward to lay out for NPF models: they present a means of learning quickly on tasks where there is a small amount of data to train on.

[1] Explained in more detail in Chapter 2 and similar to the support and query sets in Ravi and Beatson, 2019.

There is also a risk in low data regimes that a model with a large number of parameters will overfit such that it performs excellently on the training data but is unable to generalise to a held-out test set. In other words, it will become so confident during training that when applied to unseen data it makes predictions with high amounts of certainty, even if those predictions are wrong. NPF models therefore seek to avoid this pitfall via the process of meta-learning, where they *learn to learn*[2] rather than solely learn a dataset. A few examples of how this is useful are as follows.

1.2.1 Meta-learning for Few-Shot Learning

Meta-learning takes a different approach to conventional model training by utilising experience gained over multiple tasks to learn a new task, rather than training from scratch on each task individually and assuming all the information necessary to train is contained in that task's training data (Hospedales et al., 2021). This takes advantage of the assumption that whilst data distributions may differ across tasks there are still sufficient similarities between them that relevant information can be transferred from training in one domain to another.

The effect of this is that if there is little data for a task that we wish to train on, progress can still be made by training on multiple tasks with similar specifications. As an example consider a map where regions are classified as *it will rain tomorrow* and *it will not rain tomorrow*. By making observations of the current climate at certain geographical points and using existing forecasting methods we can make accurate predictions for tomorrow's precipitation at those observation points. This can then be used as a training set for a model that can predict whether it will rain tomorrow in the remainder of the grid of locations that we are interested in. This is considered a single task following a single data distribution: for this one day the distribution of raining tomorrow or not against geographical location is fixed and able to be learned by a model given enough observation points.

However, when tomorrow comes and we wish to repeat the task for the next day, this distribution will have changed, and consequently just using the previous day's model will result in a poor prediction now that the model has fitted to a distribution that is no longer

[2] As described in Section 1.1 above, learning to learn means learning not only a single task's internal distribution, but also the similarities across different tasks, such that a model can generalise quickly to unseen tasks in the future.

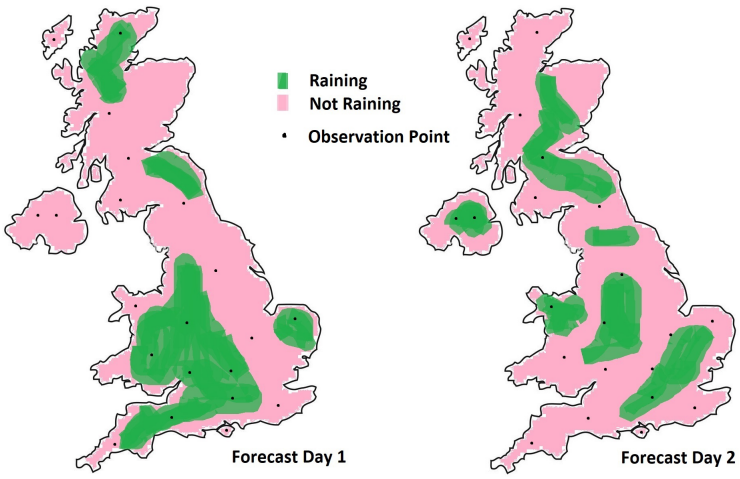


Fig. 1.1 An example demonstrating two tasks with similar specifications but different distribution of data. The task here is to predict raining or not given a training set of observation points (black dots).

true. This is shown in Figure 1.1. One solution would be to retrain the model from scratch on the new observations, but if there are not many observations then training will be difficult. Another approach therefore is to extract some information from the previous day, such as how regions of rain/no rain seem to be correlated in space, and use that information in addition to the new observations to help the model learn tomorrow's distribution faster. This is an example of meta-learning, and when repeated over many different tasks (days in this case), it allows a model to generalise rapidly to new unseen distributions with a limited number of observations for the current task.

This rainfall example is a strong use case for meta-learning, and is explored in more depth towards the end of this thesis.

1.2.2 Efficient Transfer Learning

A major motivation for neural processes and meta-learning in general is that they allow for the transfer of learned knowledge from one domain to another, as demonstrated in (Requeima et al., 2019). If done well this potentially reduces significantly the data, time, and compute required to train to the same level on the second domain. Not only is this economical, but can be crucial in many use cases.

Personalisation is a good example of this. Personalisation concerns making the outputs of a model more specific to an individual user over time, for example within the context

of recommender systems or predictive text models. In the early stages of a user using an application there is a scarcity of data that is specific to that user, and so it is difficult to effectively train a model to give them good recommendations without overfitting. What is more, the training process often requires a computationally intensive process such as backpropagation that is not feasible to run on a small device like a mobile phone.

Whilst there are a number of approaches for meta-learning (Hospedales et al., 2021), they all share a similarly extensive computation requirement during meta-training (the general training of the model over all tasks rather than specific training on a single task), with the number of tasks trained on often reaching in the thousands until a good representation can be learnt. However, once meta-training has been performed, the test-time implementation of the model on a previously unseen task is a lot cheaper and faster. Indeed in amortisation-based meta-learning approaches (Ravi and Beatson, 2019), such as those used in NPF models, there is no requirement for back-propagation at all at test time, making the computation requirements substantially smaller.[3] This means that a meta pre-trained model can be put on a smaller device such as a mobile phone and the personalisation carried out on-device in an efficient manner.

1.2.3 Handling Missing Values

The ability of neural processes to learn tasks rapidly with little training data allows them to fill in missing values for previously unseen off-the-grid data in time-series or images, for example. In the image case, a neural process can use the provided pixel values as a context set (more on these shortly) and treat the rest of the missing image grid as the target inputs. This allows for pixel value predictions to be calculated at each missing value to up-scale the image resolution (Garnelo, Rosenbaum, et al., 2018), (Garnelo, Schwarz, et al., 2018).

[3] For example the test-time performance for a ConvNP (discussed in chapter 2) is at least 5x faster than other state-of-the-art meta-learning models such as Model Agnostic Meta-Learning (MAML) that require gradient computations on unseen tasks (Requeima et al., 2019), (Finn, Abbeel, and Levine, 2017).

1.2.4 Sim-to-Real

Another use case of interest is emulation, specifically the transfer from simulated to real data.[4] This is again a method for combating data scarcity, specifically when there is a shortage of observed data but we know a basic underlying model of the problem's dynamics. Therefore a synthetic model is known, but the exact parameters are not, and real world observations contain noise that is often not well represented by a simulation. Despite this, the fact that the underlying mechanics are correct mean that the distribution of simulated data is similar to that of the real observations. The model can therefore be meta-trained on simulated data with a range of parameter settings to learn a good representation prior to having the true observations passed through to make realistic predictions at test-time.

1.2.5 Accurate Uncertainty Quantification

A final prominent motivation for neural processes is how mitigating overfitting naturally leads to strong uncertainty quantification. When a new task is presented to a meta-learner during training they are penalised for being overconfident about their predictions from the previous task as the ground-truth distribution has shifted. As a result meta-learning models, particularly generative ones like considered in this thesis, are intrinsically good at quantifying the amount of confidence behind their predictions, even after a very large number of training epochs (Garnelo, Rosenbaum, et al., 2018).[5]

The advantages of this are far-reaching, as overfitting and exaggerated confidence has been a problem that has plagued black-box models for a long time (Mena, Pujol, and Vitrià, 2021) and has led to slow uptake in domains where incorrect predictions have adverse consequences such as medicine and finance. Multi-stage models such as climate models that make use of uncertainty in their downstream modeling are also negatively effected by overconfidence, and would therefore also be advantaged by accurate uncertainty quantification. This is further discussed in Chapter 5 of this thesis.

[4] See for example the applications of Convolutional Conditional Neural Processes and Gaussian Neural Processes, which are described in Chapter 2, to predator-prey data (Gordon et al., 2020), (Markou et al., 2022)

[5] Note however that the type of uncertainty that makes up this quantification varies between models, with the original Conditional Neural Process discussed in this paper predicting almost solely aleatoric noise. This is discussed in greater detail in Chapter 2.

1.3 Thesis Outline

The organisation of the thesis takes the following order:

- **Chapter 2:** A review of the existing literature behind Neural Process Family models, discussing the task that each was built to solve and their consequent strengths and weaknesses. The existing literature has many similarities in approaches and methodology yet often use different notations to describe the same process. Therefore this chapter also provides a unification across the current models to make their comparisons more transparent.

- **Chapter 3:** Introducing and testing the new CorrConvNP model, laying out the motivations behind the adaptations and how they provide improvements to the existing methods.

- **Chapter 4:** Introducing alternative likelihoods for the CorrConvNP and exploring why it is a suitable choice for certain classification and prediction problems.

- **Chapter 5:** Investigating a real-world use case which the meta-learning and correlated predictive distribution characteristics of the new model are well suited for. This case is to predict the daily volume of rainfall over a grid given a low number of observations, and is an historically difficult problem due to the preponderance of zeros in rainfall data.

- **Chapter 6:** A summary of the investigations within this thesis and future research questions that could arise as a result of it.

Chapter 2

A Review of the Existing Literature

Since their initial conception in 2018 there have been multiple models join the neural process family, each offering a slight difference in architecture and consequent variations in their properties. This chapter investigates their chronology and builds a unifying notation under which all models can be readily compared.

2.1 Multi-Task Learning: Context and Target Sets

One feature common to all models is multi-task learning, which is the ability (and requirement) to train on many tasks with data from different underlying distributions. As mentioned at the start of Chapter 1, this is usually advantageous and rarely a restriction, as one large task can be broken up into multiple smaller ones with different distributions to give a work-around for single-task problems.

To achieve multi-task learning, all models make use of a context and target set. Whilst the terminology may seem unusual to a reader unfamiliar with meta-learning, it is similar in concept to training and test sets in traditional supervised learning. The difference, however, comes from the fact that the task at test-time has not been seen before by the model, so it does not make sense to simply pass test data through a pre-trained model in the hope that the outputs will be specific to the task at hand. Instead, after every task, be it during training or testing, the model has available to it a set of context points, D_c, and a set of target points, D_t. The difference between training and testing is that during training the labelled target points are used to learn parameters through backpropagation, whereas in

testing they are not. A diagram making clear this difference in methodology can be seen in Figure 2.1.

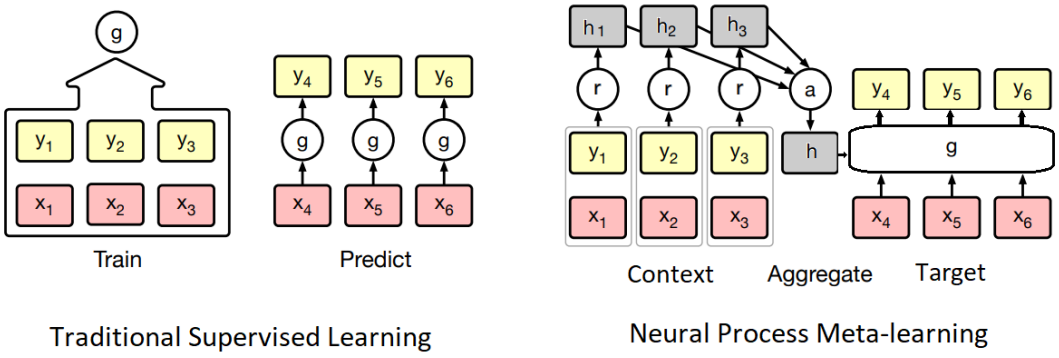


Fig. 2.1 A diagram outlining the differences between using train and test sets in traditional supervised learning (left) and using context and target sets with neural processes (right). Whilst the functional forms of r and g might change from model to model, the foundations of conditioning on a context set when predicting outputs from a target set hold true for all NPF models. Figure adapted from (Garnelo, Rosenbaum, et al., 2018).

As can be seen from the RHS of the figure, the context set is embedded by some function $r: (x_i, y_i) \to h_i$ and aggregated by $a: \{h_i\} \to h$ such that all of the target point predictions are conditioned on the context set. The context set therefore has the effect of making the meta model specific to the unseen task at hand and requires labeled (x, y) pairs, whereas the target set defines all points where an output needs to be predicted given the input.

Figure 2.1 also makes clear the similarity between neural processes and stochastic processes, as stochastic processes also make use of a set of input-output pairs to condition on when making predictions from a target set. The major advantage of the neural process approach comes from the computational complexity improvements. If there are n points in the context set and m in the target set, then a stochastic process such as a Gaussian process has complexity $\mathcal{O}((n+m)^3)$ (Rasmussen and Williams, 2006). In comparison, due to the use of neural networks, if no backpropagation is taking place (i.e. we are in a post-training regime), then the computational complexity of a neural process is $\mathcal{O}(n+m)$ (Garnelo, Rosenbaum, et al., 2018), presenting a significant speed-up comparably.

A new issue now presents itself, however. Both the context set and target set can take variable numbers of points, but neural networks generally require fixed-size inputs. What is more, the ordering of (x, y) pairs within those sets should not matter, and for

utility purposes it should be possible to continually expand both the context and target sets without having to re-run any computation on previously seen points. This is the problem of *permutation invariance* on sets (Wagstaff et al., 2019), and is solved for the target set in a variety of ways for different NPF models (discussed later on in Chapter 2). For the context set however, the solution comes for all models from deep set embeddings (Zaheer et al., 2017).

2.1.1 Deep Set Embeddings

Proposed by Zaheer et al., 2017, Deep Sets are a way of representing entire unordered sets of variable length in a single vector representation of fixed size so that the output of any function operating on them is permutation invariant.

Permutation invariance is defined as the following. Consider a function $f(\cdot)$ acting on a set of values $D = \{d_1, ..., d_N\}$ for arbitrary length N. Permutation invariance holds iff for any permutation π it is true that:

$$f(\{d_1, ..., d_N\}) \equiv f(\{d_{\pi(1)}, ..., d_{\pi(N)}\}) \tag{2.1}$$

With this in mind, the deep set theory is as follows (Zaheer et al., 2017). Any function $f(\cdot)$ is a valid set function iff it can be decomposed as:

$$f(D) = \rho\left(\sum_{d \in D} \psi(d)\right) \tag{2.2}$$

where $\psi(\cdot)$ and $\rho(\cdot)$ are suitably defined transformation functions. In the extended case that $\psi(\cdot)$ and $\rho(\cdot)$ are deep neural networks, $f(\cdot)$ becomes a deep set function.

In the context of this work, the sum is known as the *aggregation*, $\psi(\cdot)$ is referred to as a *pre-aggregation function*, and $\rho(\cdot)$ a *post-aggregation function*. The context sets are passed through this chain of functions to find an embedding, either denoted by $\boldsymbol{h}$ for FNN transformation functions, or $\mathbb{H}$ for CNN versions (this is discussed in more detail in the remainder of Chapter 2).

2.2 The Conditional Neural Process

First of the NPF models was the Conditional Neural Process (CNP) (Garnelo, Rosenbaum, et al., 2018). It is the simplest in architecture, involving only a context set embedding and a decoder. Its motivation was to mimic the functional flexibility and strong uncertainty quantification of stochastic processes such as Gaussian Processes (GPs) (Rasmussen and Williams, 2006) whilst boasting the lower computational complexity of neural networks. The structure of a CNP is modular and sequential: it can be crudely thought of as a sequence of neural networks that work in a chain-like fashion, taking as input the output of the previous network and passing the outputs forwards to another. Importantly, the final outputs are the statistics of a Gaussian distribution (namely the mean and variance), which are used to define a distribution over predicted outputs for each test input passed into the model. This is the reasoning behind saying that a CNP is a neural network approximator of a GP; it takes in a set of inputs and produces Gaussian distributions over their outputs. A diagram of a CNP can be seen in Figure 2.2.

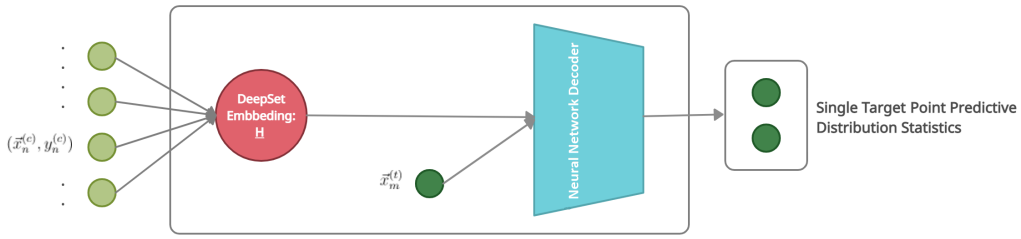


Fig. 2.2 Diagram showing the architecture of the Conditional Neural Process. The entire context set is embedded using a deep set approach as discussed in Section 2.1.1. This embedding is then passed to a neural network once concatenated with a single target input, and used to give the target predictive distribution statistics.

The functional breakdown can be described in more detail as follows:

1. There is a local embedding for each context set point, given by some pre-aggregation function (for example a Feedforward Neural Network (FNN) is used in the original paper (Garnelo, Rosenbaum, et al., 2018)):

$$\boldsymbol{h}_n = \psi(\boldsymbol{x}_n^{(c)}, y_n^{(c)}) \tag{2.3}$$

The aggregation then combines these representations in a deep set architecture as proposed in (Zaheer et al., 2017), which makes the overarching context embedding invariant to permutations and the length of the context set. A sum makes a natural choice for this:

$$\boldsymbol{h} = \sum_{n \in D_c} \boldsymbol{h}_n \tag{2.4}$$

Note that in the diagram in Figure 2.2, $\mathbb{H}$ is used as the context set embedding. This is an equivalent notation, but allows for the embedding to have more than 1 channel, for example if convolutional architectures were to be used with CNN-like pre-aggregation functions (see Section 2.5).

2. The context set embedding is then concatenated with a target set input $\boldsymbol{x}_m^{(t)}$ and passed into a decoder network, which in the original paper was once again a FNN. The output of this decoder network is therefore specific to a single target point, and gives both the mean and variance of a Gaussian distribution for the target output, $p(y_m^{(t)}|\boldsymbol{x}_m^{(t)}; D_c)$, governed by:

$$\begin{aligned} y_m^{(t)} &\sim \mathcal{N}(\mu_m, \sigma_m^2) \\ \mu_m, \sigma_m^2 &= \mathscr{F}_{\text{Decoder}}(\boldsymbol{x}_m^{(t)}, \boldsymbol{h}) \end{aligned} \tag{2.5}$$

Importantly, the fact that the model is ran separately for all points in the target set means that it it can handle a target set of any size, but that all target points must therefore be treated as independent, which can have negative consequences on tasks where correlation between points is important. This is the motivation behind the later models such as the Neural Process and Gaussian Neural Process described below.

The likelihood over the entire target set is:

$$p(\boldsymbol{y}_t|\boldsymbol{X}_t, D_c) = \prod_{m \in D_t} p(y_m^{(t)}|\boldsymbol{x}_m^{(t)}, \boldsymbol{h}(D_c)) = \prod_{m \in D_t} \mathcal{N}(y_m^{(t)}; \mu_m(\boldsymbol{x}_m^{(t)}, D_c), \sigma_m^2(\boldsymbol{x}_m^{(t)}, D_c)) \tag{2.6}$$

which demonstrates this target point independence from the fact that it is factorisable.

2.3 The Neural Process

The independence of target point distributions in the CNP ends up being problematic in many applications where correlations between input features are important. To see this we return again to the example in Chapter 1 of predicting whether it will rain or not tomorrow, and adapt it slightly. The inputs in this scenario were the spatial coordinates of a grid of locations and the outputs were a binary variable of *raining* or *not raining*. If the outputs are changed to a continuous measure of rainfall *amount*, then we can truncate at 0 our Gaussian target predictive distributions produced by the CNP and approximate the rainfall amounts by sampling from them. However, the target set independence now dictates that at any given point the amount of rainfall can be deduced from the context set only and does not depend on the rainfall at locations nearby in the target set. If the context set is small and covers very few locations then rainfall predictions far from context points will fluctuate greatly even between adjacent locations. Realistically this does not make sense, particularly on small distance scales. The amount of rain predicted a matter of meters from our location should tell us a great deal about how much rain we predict for where we are, and therefore the target predictions must be correlated.

Neural Processes were therefore introduced to remove this target prediction independence (Garnelo, Schwarz, et al., 2018) and use a latent variable (LV) architecture to prevent factorisability of the target set likelihood. Figure 2.3 below shows a diagram of a NP model, which is similar to the CNP of Figure 2.2 but has an additional encoder network inserted between the deep set embedding and the target set decoder.

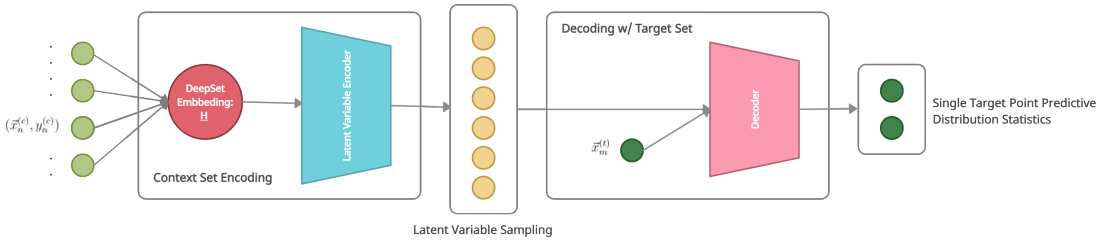


Fig. 2.3 Diagram of a Neural Process model, showing the output Gaussian statistics for a single target prediction $y_m^{(t)}$ given the entire context set. The difference from the CNP is that there is now an additional encoder after the deep set embedding which produces a distribution over latent variables. It is these latent variables that the decoder now conditions on when target inputs are passed through, and they give rise to non-factorisable target predictive distributions after marginalising them out.

The NP architecture is given as follows:

1. The encoder maps from the context set to a probability distribution over a set of LVs $\{z\}$. It takes as input the context set in embedded form and outputs the mean and variance of a multidimensional Gaussian distribution for $\{z\}$. Like with the CNP, the original paper uses FNNs for all function approximations, and so the encoder takes the form of a FNN with trainable parameters (Garnelo, Schwarz, et al., 2018).

$$\begin{aligned} \{z\} &\sim \mathcal{N}(\mu, \sigma^2) \\ \mu, \sigma^2 &= \mathscr{F}_{\text{Encoder}}(\boldsymbol{h}(D_c)) \end{aligned} \tag{2.7}$$

   where, as with the CNP, the embedding is given by:

$$\boldsymbol{h} = \sum_n \boldsymbol{h}_n = \sum_n \boldsymbol{\psi}(x_n^{(c)} \cup y_n^{(c)}) \tag{2.8}$$

2. The decoder then constructs the mapping from target inputs to distributions of the target outputs, conditioned on the LVs $\{z\}$ sampled as above. Specifically, it produces the distribution $p_\theta(\boldsymbol{y}_t | X_t, \{z\}; D_c)$ by using a neural network to approximate the Gaussian statistics via:

$$p(\boldsymbol{y}_t | \boldsymbol{X}_t, \{z\}; D_c) = \prod_{m \in D_t} \mathcal{N}(y_m^{(t)}; \mu_m(\boldsymbol{x}_m^{(t)}, \{z\}; D_c), \sigma_m^2(\boldsymbol{x}_m^{(t)}, \{z\}; D_c))) \tag{2.9}$$

   where μ_m and σ_m^2 are a 2-headed neural network output layer (the original paper again uses a FNN for this (Garnelo, Schwarz, et al., 2018)):

$$\mu_m, \sigma_m^2 = \mathscr{F}_{\text{Decoder}}(\boldsymbol{x}_m^{(t)}, \{z\}) \tag{2.10}$$

The benefit of a LV representation now becomes clear when marginalising them out to give the expression for the target set predictive:

$$p(\boldsymbol{y}_t | \boldsymbol{X}_t; D_c) = E_{p(z)} \left[p(\boldsymbol{y}_t | \boldsymbol{X}_t, \{z\}; D_c) \right] \tag{2.11}$$

which is no longer factorisable and therefore means that the target points are no longer independent given the context set.

2.4 The Gaussian Neural Process

A different approach to ensure correlation between target points is to force the target predictive distribution to be non-diagonal. This is the approach taken by the Gaussian Neural Process (GNP) proposed in Wessel Bruinsma et al., 2021. In contrast with a NP, the GNP uses a closed-form likelihood and includes only a deep set embedding and a decoder. Instead a GNP defines a covariance between each pair of points in the target set such that they are now not assumed to be independent (i.e. the covariance matrix over target points is constructed to be non-diagonal). This means that the decoder must be ran over pairs of points for each element of the covariance matrix to be constructed. The function architecture for a GNP is shown in Figure 2.4 below.

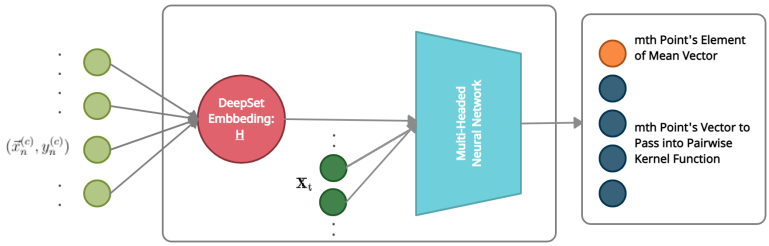


Fig. 2.4 Diagram of a Gaussian Neural Process (GNP) model. The output of the decoder is run over pairs of points to construct Gaussian statistics for the joint distribution over the whole target set, $\boldsymbol{y_t}$, given all target inputs, $\boldsymbol{X}_t$, and the entire context set.

Notice that in Figure 2.4 the decoder output now has a greater number of heads than before. The first gives the mean of the target Gaussian distribution (we have once again assumed for simplicity that the dimensions of y_c and y_t are 1), and the remainder give a fixed-size vector that is used to construct the elements of the Gaussian target predictive's covariance matrix. A more detailed breakdown of the process is as follows:

1. The likelihood distribution is now given by:

$$p(\boldsymbol{y}_t|\boldsymbol{X}_t, D_c) = \mathcal{N}(\boldsymbol{y}_t; \boldsymbol{m}(\boldsymbol{X}_t, D_c), \boldsymbol{K}(\boldsymbol{X}_t, D_c)) \tag{2.12}$$

where the mean vector and covariance matrix are defined jointly over all points in the target set. Note the diversion from CNPs here is that the covariance matrix need not be diagonal, although if $\boldsymbol{K} := \sigma^2 \hat{\boldsymbol{I}}$ for some σ^2 then we do return the CNP architecture.

2. The mean and covariances are defined using neural networks $f(\cdot)$ and $g(\cdot)$ (e.g. FNNs again), where the covariance is simplified to be calculated pair-wise over points within a positive semi-definite kernel function $k(\cdot)$:

$$\begin{aligned} \boldsymbol{m}_i &= f_\theta(\boldsymbol{x}_i^{(t)}, \boldsymbol{h}) \\ \boldsymbol{K}_{i,j} &= k(\boldsymbol{g}_\theta(\boldsymbol{x}_i^{(t)}, \boldsymbol{h}), \boldsymbol{g}_\theta(\boldsymbol{x}_j^{(t)}, \boldsymbol{h})) \end{aligned} \tag{2.13}$$

   For the architecture in Figure 2.4, $f(\cdot)$ and $g(\cdot)$ are assumed to be part of the same multi-headed neural network with parameters θ, and this decoder network is ran twice (over both $\boldsymbol{x}_i^{(t)}$ and $\boldsymbol{x}_j^{(t)}$) to construct a single $\boldsymbol{K}_{i,j}$ element.

3. For the kernel function, a common choice is the Exponentiated Quadratic (EQ) (Rasmussen and Williams, 2006) due to its theoretically infinite number of basis functions allowing it to model locally varying functions well. However, as the EQ kernel works on the difference between its vector inputs, the smallest the elements of the kernel matrix can be is 1 (i.e. on the diagonal). Therefore, the KVV kernel is used to allow near 0 covariance in regions where the target points are close to points from the context set. This is defined as follows (Markou et al., 2022):

$$\boldsymbol{K}_{i,j}^{KVV} = k_{EQ}(\boldsymbol{g}_\theta(\boldsymbol{x}_i^{(t)}, \boldsymbol{h}), \boldsymbol{g}_\theta(\boldsymbol{x}_j^{(t)}, \boldsymbol{h}))\nu(\boldsymbol{x}_i^{(t)})\nu(\boldsymbol{x}_j^{(t)}) \tag{2.14}$$

   where $\nu(\cdot)$ is a neural network layer producing a scalar that can take any real value and can therefore learn to go to 0 at the context points.

   Despite the KVV kernel giving good flexibility for learning a function shape, it has the disadvantage of not being parallelisable across an entire target set through matrix operations. Therefore, Wessel Bruinsma et al., 2021 use a low-rank kernel approximation:

$$\boldsymbol{K} = \boldsymbol{V}_\theta \boldsymbol{V}_\theta^T + \boldsymbol{\Lambda}_{\text{diag}} \tag{2.15}$$

   where $\boldsymbol{\Lambda}_{\text{diag}}$ represents non-zero but independent observation noise, and $\boldsymbol{V}_\theta$ is a matrix with any j^{th} row representing $\boldsymbol{g}_\theta(\boldsymbol{x}_j^{(t)}, \boldsymbol{h})$ from Equation (2.13).

2.5 Moving to Convolutional Architectures

2.5.1 Translation Equivariance

Translation Equivariance (TE) is an essential property for strong model generalisation (Foong et al., 2020). A model is called TE if when an input domain is shifted in space (for spatial data) or time (for temporal data) its predictions are shifted in an identical way.[1] The generalisation power of TE stems from its ability to extrapolate to unseen domains: with TE, if a model learns some function or generative distribution in an observed domain during training, it can also give good predictions on test data that follow the same distribution or functional form, even if the data lie outside of the training domain.

To achieve TE in deep learning a common approach is to use Convolutional Neural Networks (CNNs). The authors of the paper (Ravanbakhsh, Schneider, and Poczos, 2017) show that a network that exhibits parameter sharing, such as a convolutional network, has the desired properties of TE. What is more, in the paper by Kondor and Trivedi, 2018, the authors go one step further and show that convolution in every layer of a neural network is a necessary requirement for equivariance within a compact group. Equivariance here is thus not restricted just to translation (though this is the characteristic that we are most concerned with for our application) but holds for any transformation within a compact group, such as linear and affine matrix transformations (as defined in Donald House and Keyser, 2016, Appendix C).

This is the approach taken by Gordon et al., 2020, Foong et al., 2020, and Wessel Bruinsma et al., 2021, with the new additions to the NPF called the ConvCNP, ConvNP, and ConvGNP respectively. However, the use of convolutional architectures adds an additional complexity. It is no longer suitable for the context set embedding $\boldsymbol{h}$, or the LVs $\{z\}$ of encoder models, to be vectors, they have to now instead be represented by *functions*. As described in the Conv Deep Sets section of Gordon et al., 2020, the reasoning behind this is that TE is not well defined for vectors. To demonstrate this the authors consider a CNN to be a function $f(\boldsymbol{x}) : \boldsymbol{x} \to Y$, operating on inputs $\boldsymbol{x} \in \mathbb{R}^d$ with translations $\boldsymbol{\tau} \in \mathbb{R}^d$ given by a translation operator $T_{\boldsymbol{\tau}}(\cdot)$ also operating in $\mathbb{R}^d$. If Y is a vector of dimension d' then

[1] Note that this is *not* the same as translation *invariance*, which TE is often confused for. Translation invariance is instead the property that output predictions do not change when the input domain is translated.

there is no suitable definition for a translation $f(T_{\boldsymbol{\tau}}(\boldsymbol{x})) \overset{\text{TE}}{=} T_{\boldsymbol{\tau}}(f(\boldsymbol{x})) = T_{\boldsymbol{\tau}}(Y)$, as $T_{\boldsymbol{\tau}}(Y)$ is undefined when the dimension of Y can be different from the dimension of the translation operator. However, if Y is a function with inputs spanning the space $\mathbb{R}^d$, then it *is* well defined to say $f(T_{\boldsymbol{\tau}}(\boldsymbol{x})) \overset{\text{TE}}{=} T_{\boldsymbol{\tau}}(f(\boldsymbol{x})) = T_{\boldsymbol{\tau}}(Y(\boldsymbol{x}))$, and translation equivariance is satisfied.

Figure 2.5 shows a diagramatic explanation of what TE means and how it is achieved by a functional representation. The input dimensions have been made 1 here to be able to generate the figure, but in general the inputs can be any dimension.

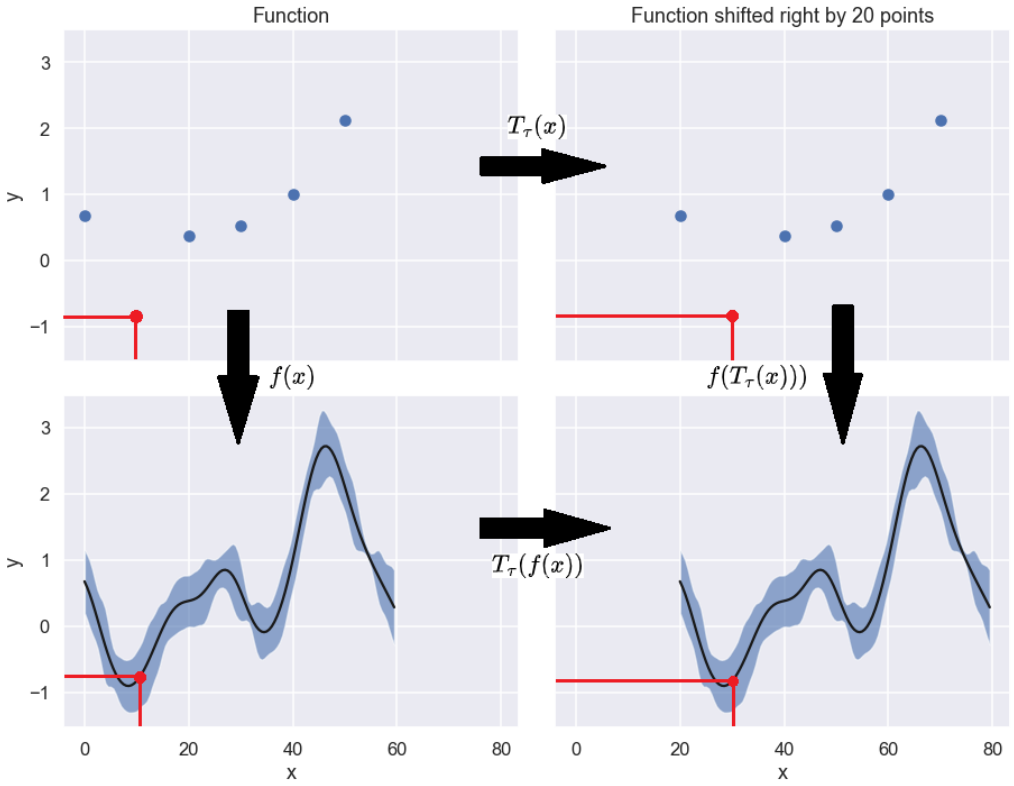


Fig. 2.5 Diagram demonstrating the effect of translation equivariance (TE). A function represented by a CNN maps the top row to the bottom row, producing a regression curve and uncertainty from context points. When the domain is shifted right 20 points in the top row, we would want the predicted CNN output function to shift accordingly. This is only achieved when the model is TE. Any non-TE model would instead fail to extrapolate to the unseen zone despite having observed the exact same pattern in a different input domain.

2.5.2 Implementation

In the non-convolutional models the context set embedding is represented by a vector $\boldsymbol{h}$, which depends on an arbitrary length context set and is permutation invariant due to the use of deep sets. For vector basis function $\boldsymbol{\psi}(\cdot)$, $\boldsymbol{h}$ is defined as (Garnelo, Rosenbaum, et al., 2018):

$$\boldsymbol{h} = \sum_{n \in D_c} \boldsymbol{\psi}(\boldsymbol{x}_n^{(c)}, y_n^{(c)}) \tag{2.16}$$

A similar approach is taken in the convolutional models, except the context set embedding now needs to be discretised to be passed as a CNN input. This discretisation is done by splitting the input domain uniformally into a user-decided number of points. Care has to be taken not to set the discretisation too high for high dimensional input spaces as the memory requirement for the discretisation increases with order $\mathcal{O}(K^d)$ for d dimensions and K discretised points per 1D domain range.

A 2-channel tensor denoted $\mathbb{H}$ is used in the ConvCNP (Gordon et al., 2020) and the ConvNP (Foong et al., 2020) as the embedding, with the first being the *data channel* and the second the *density channel*.[2]

The data channel is given by: (note the normalisation by the density channel is optional)

$$\mathbb{H}_0(\boldsymbol{t}) = \frac{\sum_{n \in D_c} y_n^{(c)} \psi(\boldsymbol{t} - \boldsymbol{x}_n^{(c)})}{\sum_{n \in D_c} \psi(\boldsymbol{t} - \boldsymbol{x}_n^{(c)})} \tag{2.17}$$

where $\boldsymbol{t}$ is a translation included to demonstrate the TE property of the convolutional embedding.

The density channel describes where there are data-points within the discretised grid, as otherwise the data channel above would give the same values for both $y_n^{(c)} = 0$ and no data-point existing at that $\boldsymbol{x}_n^{(c)}$ point at all. It is defined as:

$$\mathbb{H}_1(\boldsymbol{t}) = \sum_{n \in D_c} \psi(\boldsymbol{t} - \boldsymbol{x}_n^{(c)}) \tag{2.18}$$

[2] In the original papers they give a vector formula for $\mathbb{H}$ to represent both channels in one equation, but here we take the same notation as Wessel Bruinsma et al., 2021 and represent each channel separately for simplicity.

ConvGNPs (Wessel Bruinsma et al., 2021) also use a third channel in the embedding which they call the *source channel*. This is given by the identity matrix $\mathbb{H}_2 = \hat{\boldsymbol{I}}$, which allows the model to start out with stationary prior covariance.

Where LVs are used, they are represented as a set $\{z\}$ to indicate that they are either a set of scalars (represented by a vector) or a set of functions, depending on the use of convolutional models. Note that regardless of model, there is only one 'set' of LVs produced per task - i.e. they are global to all target points for that task.

In the convolutional case, latent functions are approximately GPs, found by sampling from a set of Gaussian distributions (one for each discretised point in the LV domain) with mean and variance parameters predicted by a CNN encoder (Foong et al., 2020). For the ConvNP these parameters are independent of each other, much like the decoder of a (Conv)CNP.

2.6 Model Training

For the models without encoders (namely the (Conv)CNP and (Conv)GNP) their generative nature means that maximum likelihood estimation (MLE) can be used as the training objective. Due to the complex relationship of the model outputs with the parameters there is no analytic solution with any of the models, so backpropagation with stochastic gradient descent is implemented using an ADAM optimiser. As discussed in Chapter 1, the combination of the models being generative and the training regime being meta-learning means that they are unlikely to overfit to a single task during training, and hence there is no need for any additional regularisation terms in the objective function.

When it comes to the LV (Conv)NP model, the objective function has to be modified in order for the encoder's parameters to be trained simultaneously to the decoder's. As the LVs are unobserved, this is achieved via an Evidence Lower-Bound (ELBO) objective, such as that defined in Section 2.2 of Kingma and Welling, 2014. The objective takes advantage of the monotonicity of the logarithm function to write a lower bound:

$$\begin{aligned}\log p_\theta(\boldsymbol{y}_t|\boldsymbol{X}_t;D_c) &= E_{\{z\}\sim q_\phi}\left[\log\left(\frac{p_\theta(\boldsymbol{y}_t|\{z\},\boldsymbol{X}_t;D_c)}{q_\phi(\{z\}|D_c)}\right)\right] + \mathrm{KL}(q_\phi(\{z\}|D_c\cup D_t)||p(\{z\}|D_c)) \\ &\geq E_{\{z\}\sim q_\phi}\left[\log\left(\frac{p_\theta(\boldsymbol{y}_t|\{z\},\boldsymbol{X}_t;D_c)}{q_\phi(\{z\}|D_c)}\right)\right] = \mathscr{L}_{ELBO}(\theta,\phi)\end{aligned} \tag{2.19}$$

where q_ϕ is the encoder neural network mapping from context set to LVs, with trainable parameters ϕ. As the term $p(\{z\}|D_c)$ is non-computable, a bootstrapping approach is suggested in Le, 2018 and Garnelo, Schwarz, et al., 2018, of approximating it with $q_\phi(\{z\}|D_c)$, giving a final *approximate* ELBO objective to maximise:

$$\mathscr{L}_{ELBO}(\theta,\phi) = E_{z\sim q_\phi(\{z\}|D_c\cup D_t)}\left[\log p_\theta(\boldsymbol{y}_t|\boldsymbol{X}_t,\{z\})\right] - \mathrm{KL}\left[q_\phi(\{z\}|D_c\cup D_t)\,||\,q_\phi(\{z\}|D_c)\right] \tag{2.20}$$

The original ConvNP paper (Foong et al., 2020) also implements a MLE objective for the LV regime due to its faster computation and better uncertainty quantification caused by no mean-field approximation (more on this below). However, using an MLE objective leads to a bias in low data scenarios.[3] Therefore for the purpose of this thesis, all encoder model training is conducted using the approximate ELBO objective. Note that in practice normalisation with the number of target points is also used, as the number of target points changes per epoch.

[3]Specifically a MLE objective pushes the latent variable towards becoming deterministic if there is low data capacity, returning in the limit a ConvCNP again, which has better uncertainty estimates, but overweights aleatoric uncertainty over epistemic (as discussed in the following section). However, as the amount of available data tends to infinity the KL term in the ELBO objective tends to 0 and the ELBO tends towards the MLE objective anyway (Foong et al., 2020).

2.7 A Summary of Existing Models and their Shortcomings

The table below summarises the existing models introduced in the chapter and outlines their characteristics and shortcomings.

Model	TE	Target Predictive Distribution	Uncertainty Quantification	Correlated Target Points
CNP	No	Diagonal Gaussian Only	Good, but overweights aleatoric uncertainty (noise)	No
NP (LV)	No	Flexible	Poor, underweights net uncertainty[4]	
GNP	No	Gaussian Only	Good	Yes
ConvCNP	Yes	Diagonal Gaussian Only	Good, but overweights aleatoric uncertainty (noise)	No
ConvNP (LV)	Yes	Flexible	Poor, underweights net uncertainty[5]	Yes
ConvGNP	Yes	Gaussian Only	Good	Yes
CorrConvNP (LV)	Yes	Flexible	Good	Yes

Table 2.1 Summary table of each of the models with their characteristics: shape of predictive distribution, ability to have correlation between samples, and uncertainty quantification. The most favourable characteristics are coloured in green. The final row is the suggested CorrConvNP introduced in the next chapter, which theoretically satisfies all desired characteristics.

The most notable results from Table 2.1 above are the strong performances of the ConvGNP and the ConvNP models, with each filling all but one of the desired properties. It is not clear exactly why the ConvNP underweights net uncertainty when trained with an ELBO objective, but it is likely caused by the mean-field approximation in VI, caused by the diagonal Gaussian (and thus independent) outputs of the ConvCNP encoder. One suggestion (Le, 2018) is that the LV encoder models the epistemic uncertainty of the underlying stochastic process, whereas the decoder models the observation noise. Therefore, by forcing the LVs to be independent there is a loss of information on the stochastic

[4]Net uncertainty is the combination of aleatoric (noise) and epistemic (functional) uncertainty.

[5]This result is for the ELBO objective, the MLE objective improves net uncertainty but overweights aleatoric uncertainty like the CNP (CovNP).

process variability, which causes epistemic uncertainty to be underweighted, leading to net uncertainty underweighting if the aleatoric noise is otherwise well-represented by the decoder.

The purpose of this thesis is therefore to propose a new model that has all of the desirable properties of Table 2.1 above (shown by the bottom row). There are multiple approaches for this, such as invertible output transformations on the ConvGNP outputs as done in Markou et al., 2022, although these require that the transformations have well-defined analytic inverses and that we know what the output distribution is, which limit their applicability. Therefore, as discussed in Chapter 3, the approach taken in this work is to start with a ConvNP basis and build in stronger uncertainty quantification by removing the mean-field approximation on the LVs.

Chapter 3

Removing the Mean-Field Approximation of (Conv)NPs

3.1 CorrConvNP Definition

The modularised design of NPF models means that their components are able to be interchanged to construct models with different properties. For example, the (Conv)NP was constructed by treating a (Conv)CNP as an encoder to generate a LV. This is then concatenated with a target input and passed to a (Conv)CNP decoder for the target prediction. In a similar fashion, the (Conv)GNP is derived from the (Conv)CNP by altering the output layer of the decoder and constructing a pair-wise dense covariance function. This is a common theme throughout both the literature and this thesis, and we take a very similar approach with the proposed new model now.

Recall that the downfall of the ConvNP is hypothesised to come from the independent LVs in the ConvCNP encoder. If we want the encoder to instead produce correlated LVs, it make sense to swap out the ConvCNP encoder for a ConvGNP instead. The encoder output is now a vector of values that parameterise both a mean vector and a non-diagonal covariance matrix using the low-rank approximation (detailed below), which allows for correlated latent functions to be sampled from a multivariate Gaussian. Due to the introduced correlation of LVs and similarity with the previous ConvNP, we call this new model the *Correlated Convolutional Neural Process* (*CorrConvNP*).

Note that this adaptation does not require major changes to the ELBO objective from Chapter 2, as only the encoder is being changed, so only the Kullback-Leibler (KL) divergence term is affected. For convenience the ELBO objective is repeated below:

$$\mathscr{L}_{ELBO}(\theta,\phi) = E_{z\sim q_\phi(\{z\}|D_c\cup D_t)}\left[\log p_\theta(\boldsymbol{y}_t|\boldsymbol{X}_t,\{z\})\right] - \mathrm{KL}\left[q_\phi(\{z\}|D_c\cup D_t)\,||\,q_\phi(\{z\}|D_c)\right] \tag{3.1}$$

With a ConvGNP encoder the distribution $q_\phi(\{z\}|\cdot)$ is now a multivariate Gaussian with a *dense* covariance matrix rather than the previous diagonal one. The calculation of the KL-divergence between two multivariate Gaussians is analytic, but unfortunately typically computationally cubic in the number of input points[1] when their covariances are non-diagonal. Therefore, Wessel Bruinsma et al., 2021 implement a low-rank approximation to the dense covariance that can lead to significant improvements in performance in this correlated LV scenario.

3.1.1 Fast Computation of the KL-Divergence

The Sherman–Morrison–Woodbury equation (Max, 1950) gives a way of approximating square dense matrices, such as a non-diagonal covariance matrix, in terms of lower rank matrices and a diagonal matrix. The consequence of this is a large computational speed-up in the calculation of dense matrix inverses. The mathematics of this operation are as follows:

If $\boldsymbol{V}_\phi$ is a $(N \text{ x } M)$ matrix from the encoder output for N input points and a chosen rank M, and $\boldsymbol{\Lambda}_{\text{diag}}$ is a $(N \text{ x } N)$ diagonal matrix representing independent Gaussian noise, then the low-rank approximation simplifies the inversion of the covariance matrix to:

$$\begin{aligned}\boldsymbol{K}^{-1} &= \left(\boldsymbol{V}_\phi\boldsymbol{V}_\phi^T + \boldsymbol{\Lambda}_{\text{diag}}\right)^{-1} \\ &= \boldsymbol{\Lambda}_{\text{diag}}^{-1} - \boldsymbol{\Lambda}_{\text{diag}}^{-1}\boldsymbol{V}_\phi\left(\hat{\boldsymbol{I}} + \boldsymbol{V}_\phi^T\boldsymbol{\Lambda}_{\text{diag}}^{-1}\boldsymbol{V}_\phi\right)^{-1}\boldsymbol{V}_\phi^T\boldsymbol{\Lambda}_{\text{diag}}^{-1}\end{aligned} \tag{3.2}$$

[1] For the encoder, the number of input points is the number of discretised points in the embedding function domain.

The effect of this a significant reduction in computational complexity whenever $M << N$, as due to the linear complexity of inverting a diagonal matrix, the entire inversion complexity for $\boldsymbol{K}$ is now $\mathcal{O}(N+M^3)$ instead of the previous $\mathcal{O}(N^3)$.

This inversion lemma helps to compute the KL-divergence in an efficient manner because the analytic form of the divergence between two multivariate Gaussians depends directly on a covariance inverse. The analytic form is given by:

$$\begin{aligned}
\mathrm{KL}\left(\mathcal{N}_1(\boldsymbol{z};\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)||\mathcal{N}_2(\boldsymbol{z};\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_2)\right) &= \int_{\boldsymbol{z}} \mathcal{N}_1(\boldsymbol{z};\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)\log\left(\frac{\mathcal{N}_1(\boldsymbol{z};\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)}{\mathcal{N}_2(\boldsymbol{z};\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_2)}\right)d\boldsymbol{z} \\
&= \frac{1}{2}E_{\mathcal{N}_1(\boldsymbol{z};\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)}\left[\log\left(\frac{||\boldsymbol{\Sigma}_2||}{||\boldsymbol{\Sigma}_1||}\right)+(\boldsymbol{z}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{z}-\boldsymbol{\mu}_2)-(\boldsymbol{z}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{z}-\boldsymbol{\mu}_1)\right] \\
&= \frac{1}{2}\left[\log||\boldsymbol{\Sigma}_2||-\log||\boldsymbol{\Sigma}_1||-d+(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2-\boldsymbol{\mu}_1)+\mathrm{Tr}(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1)\right]
\end{aligned} \tag{3.3}$$

where we have used the expectation of a quadratic form identity (Bates, 2010) in going from the second to the last line in Equation (3.3) above.

Therefore, we can see from the presence of $\boldsymbol{\Sigma}_2^{-1}$ in Equation (3.3) that if we choose a rank M to be a lot smaller than the embedding function discretisation then we will achieve a significantly reduced number of computations with each training loop iteration.

3.2 How it Fits into the Family

There were clear relations between the existing NPF models that were discussed in Chapter 2, and the new model is no departure from this. In fact, the CorrConvNP can be seen as aptly filling in the last corner of a square representation of model characteristics in model-space (or two corners of a cube if we consider also the non-convolutional models). To make this abstraction to model-space more concrete, consider the diagram in Figure 3.1. This shows a cube with a coordinate system governed by the three basis characteristics of the models: the models being conditional or latent; the models having diagonal or non-diagonal covariances (correlated samples); and the models being convolutional and TE or not.

If we ignore the convolutional axis by only considering the convolutional variants of the models (which we can do as each model has a direct convolutional alternative), then we can see that the existing literature gives three corners of a square in this space, and the new CorrConvNP simply completes the fourth.

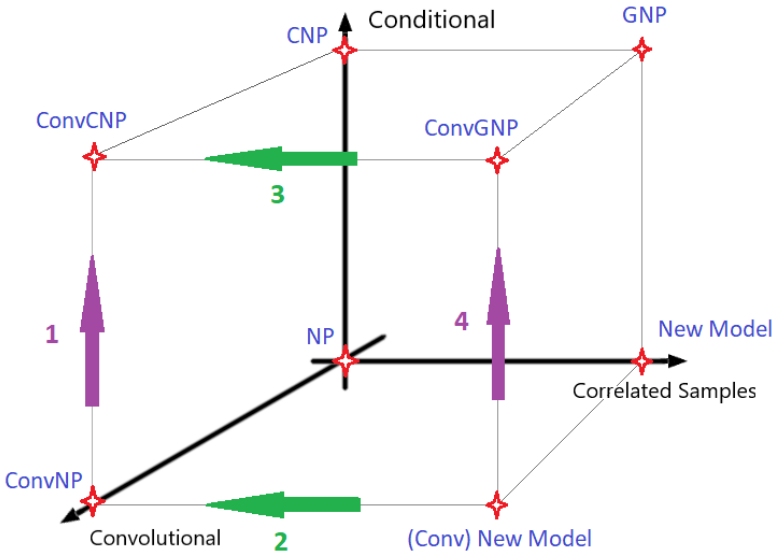


Fig. 3.1 A diagram outlining how the models differ and what changes can be made to find equivalence between them. The green arrows dictate a transformation between models that requires only a change of distribution definition and not a change of encoder/decoder architecture, and the purple arrows represent transformations which require an encoder to be made deterministic (consequently removing LVs, more on this below).

The next question is naturally then: *'If these models are so similar, then how can we move between them?'* This section aims to do just that, and provide a unifying framework under which all the models can be viewed and easily transformed between. Again, for simplicity, only the convolutional models will be considered from here-on-in. However, there is always a straightforward transformation from a convolutional to a non-convolutional model, which is to switch-out the CNNs in the embedders/encoders/decoders for MLPs, and change from latent functions to the simpler latent vectors.

In order to not repeat too much information unnecessarily, only the transformations in the directions of the arrows in Figure 3.1 will be described, but the other direction can always be achieved by doing the reverse transformation. The equivalences can therefore be found as follows:

1. The transformation from a ConvNP to a ConvCNP requires the removal of the LV aspect of the ConvNP model. In other words, the encoder of the ConvNP becomes redundant. This becomes the case if the encoder function is no longer a CNN, but instead the function:

$$\mathscr{F}_{encoder}(\mathbb{H}) = \mathbb{H} \tag{3.4}$$

   The issue now arises that the ConvNP *samples* from a Gaussian distribution based on the encoder output, whereas we desire the input of the decoder to be *equal* to the encoder output. This can be done without a change of model architecture if the encoder's Gaussian likelihood function is turned into a Dirac Delta distribution (described well in Section 1.11 of Arfken, Weber, and Harris, 2013), that is centered around $\mathbb{H}$.[2]

2. The transition from CorrConvNP to ConvNP is perhaps one of the simplest, as the distinction between them is solely the correlation that has been introduced at the encoder stage of the CorrConvNP. Equivalence is therefore found simply by restricting the LV covariance matrix of the CorrConvNP to be diagonal:

$$\boldsymbol{K}^{(z)}_{i,j} = \lambda_i \delta_{i,j} \tag{3.5}$$

   such that the LVs become independent from one-another:

$$\{z\} \sim \mathscr{N}\left(\boldsymbol{m}^{(z)}, \boldsymbol{K}^{(z)}\right) \rightarrow \prod_{i=1,\ldots,\text{len}(\{z\})} \mathscr{N}(m_i^{(z)}, \lambda_i) \tag{3.6}$$

   This transformation is implemented later on in Chapter 3, when showing statistical significance in the improvement that correlated LVs give compared to the ConvNP.

3. Moving from a ConvGNP to a ConvCNP requires a similar approach to the transformation above, except now a diagonal covariance is being introduced in the decoder rather than the encoder. The process is therefore very similar to Equation (3.5), except that target predictions replace LVs as the output:

[2] Note that a Dirac Delta distribution centered around $\boldsymbol{m}$ can be thought of as a Gaussian distribution with mean $\boldsymbol{m}$ and covariance $\boldsymbol{\Sigma} \rightarrow 0$ (Arfken, Weber, and Harris, 2013).

$$\boldsymbol{K}_{i,j} = \lambda_i \delta_{i,j} \tag{3.7}$$

$$\boldsymbol{y}_t \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K}) \to \prod_{i \in D_t} \mathcal{N}(m_i, \lambda_i) \tag{3.8}$$

From an implementation viewpoint this process is inefficient however, as a ConvGNP runs the decoder over all pairs of points in the target set and thus requires $\frac{n(n+1)}{2}$ computations to construct the dense covariance for n target points. If the covariance is subsequently restricted to be diagonal then the majority of these[3] calculations become irrelevant as only the n computations of diagonal elements are used. Therefore, if moving from a ConvGNP to a ConvCNP, it also makes sense to adapt the covariance calculation loop to iterate over the target inputs separately rather than pairwise.

4. The transformation from CorrConvNP to ConvGNP again requires an architecture change as a LV model is being restricted to a conditional one. This is similar to the ConvNP to ConvCNP transformation above, with $\mathscr{F}_{encoder}(\mathbb{H}) = \mathbb{H}$, except that now correlations have to be introduced in the decoder. This is done by expanding the dimensions of the multi-headed decoder output layer to give, for each target point i, a mean value $m_\theta(\boldsymbol{x}_i^{(t)}, \mathbb{H})$, and a vector $\boldsymbol{g}_\theta(\boldsymbol{x}_i^{(t)}, \mathbb{H})$ that is used to construct the dense covariance when ran pair-wise over the target inputs.

The final two transitions from CorrConvNP to ConvCNP and from ConvNP to ConvGNP are equivalent in process to transition 1 and transition 4 respectively. This is because once the encoders of the CorrConvNP and ConvNP models are ignored (as they are in both transitions 1 and 4), the two models become equivalent, and thus so do their transformations.

3.3 Performance on Regression Tasks

In order to test the efficacy of the new CorrConvNP model and contrast it to the existing models, we first recreate regression experiments from Foong et al., 2020 and Wessel Bruinsma et al., 2021. Specifically these are the Gaussian Process regression experiment

[3] For $n > 3$.

with an exponentiated quadratic (EQ) kernel (Rasmussen and Williams, 2006) and the bi-directional sawtooth experiment. The reasoning behind selecting these is as follows:

EQ GP:

The attraction of regressing a GP with a known underlying kernel is that we can construct the true GP, with its well-defined uncertainties, and readily use this to compare the uncertainty quantification of the different NPF models. What is more, being a GP, the true target predictive distributions are all Gaussian, which suits the conditional models (ConvCNP and ConvGNP) well, as these are fixed with a Gaussian target predictive distribution and do not have the flexibility of the encoder models. This means that all of the models are on an equal footing in terms of ability to fit the data, and the best performers will therefore be those that can best measure target point correlations and quantify the epistemic uncertainty caused by small context sets effectively.

Bi-directional Sawtooth:

As described in Foong et al., 2020, the bi-directional sawtooth task features a sawtooth function with frequencies, translations, and granularities that are randomly generated. The function is defined as:

$$y(x;s,f,K) = \frac{1}{2} - \frac{1}{\pi}\sum_{k=1}^{K}(-1)^k \frac{\sin(2\pi f k[x-s])}{k} \tag{3.9}$$

$K \in \{10,...,20\}$ dictates the granularity (how smooth the resulting function is, with higher values resulting in greater smoothness), $s \sim \mathscr{U}(-5,5)$ dictates translations along the x-axis, and $f \sim \mathscr{U}(\pm 3, \pm 5)$ dictates the frequency of peaks per domain unit. Importantly, whether a frequency is positive or negative is also chosen uniformly, with a negative frequency causing a change in direction of the sawtooth, leading to the name *bi-directional*.

Note that the task does not include any additive observation noise, so a strong performing model should be able to fit the underlying function very tightly with a sufficiently large context set. The advantage of this task specification is two-fold. Firstly, it is a difficult curve to fit, with abruptly changing shape when the frequency switches sign. The purpose of this is to highlight the advantage of the encoder models that have greater distribution flexibility than the conditional ones, particularly as uncertainty quantification

is not important for good performance on this task. Secondly, the periodic nature of this function means that its shape repeats in extrapolation domains where there are no context points. This advantages models with strong extrapolation properties that can copy over learned distribution shapes to unseen domains.

Whilst these two existing regression tasks cover many bases, it is still useful to consider the case of a periodic function with sizeable uncertainty, that is also highly non-Gaussian. This will test both the generalisation properties of the models due to the periodicity, as well as their ability to quantify noise well even when it is non-Gaussian. To this effect, the bi-modal sinusoid is introduced, and defined as follows.

Bi-Modal Sinusoid:

$$y(x;A,s,T) = A\sin\left(s+\frac{2\pi x}{T}\right)+c \tag{3.10}$$

where $A \sim \mathscr{U}(1,2)$ controls the amplitude, $s \sim \mathscr{U}(0,2\pi)$ controls the phase, and $T \sim \mathscr{U}(1,2)$ controls the time period. Here c is additive non-Gaussian noise, chosen to follow a bi-modal distribution. It is constructed from 2 Gaussians as follows:

$$c = \frac{1}{2}\mathscr{N}(x;m,\sigma^2)+\frac{1}{2}\mathscr{N}(x;-m,\sigma^2) \tag{3.11}$$

with chosen statistics $m = 0.1, \sigma^2 = 0.2$.

The plots in Figure 3.2 show the negative log-likelihood (NLL) as a function of epoch number on both the training set (from seen tasks) and validation set (from unseen tasks) for up to 100 epochs. What is immediately clear from these plots is how the smaller conditional models have a far greater distance between converged training set and validation set NLLs. The cause of this is their smaller size and lower model complexity, which allows faster adaptation of the models to individual tasks during training. This still does not lead to overfitting fortunately, as can be seen from the fact that validation scores do not increase after many epochs. This is a result of the changing distributions across tasks in meta-learning, as discussed in Chapter 1.

These comparisons can be supported further by considering the generated fits of the respective models on the same unseen tasks, as plotted in Figure 3.3.

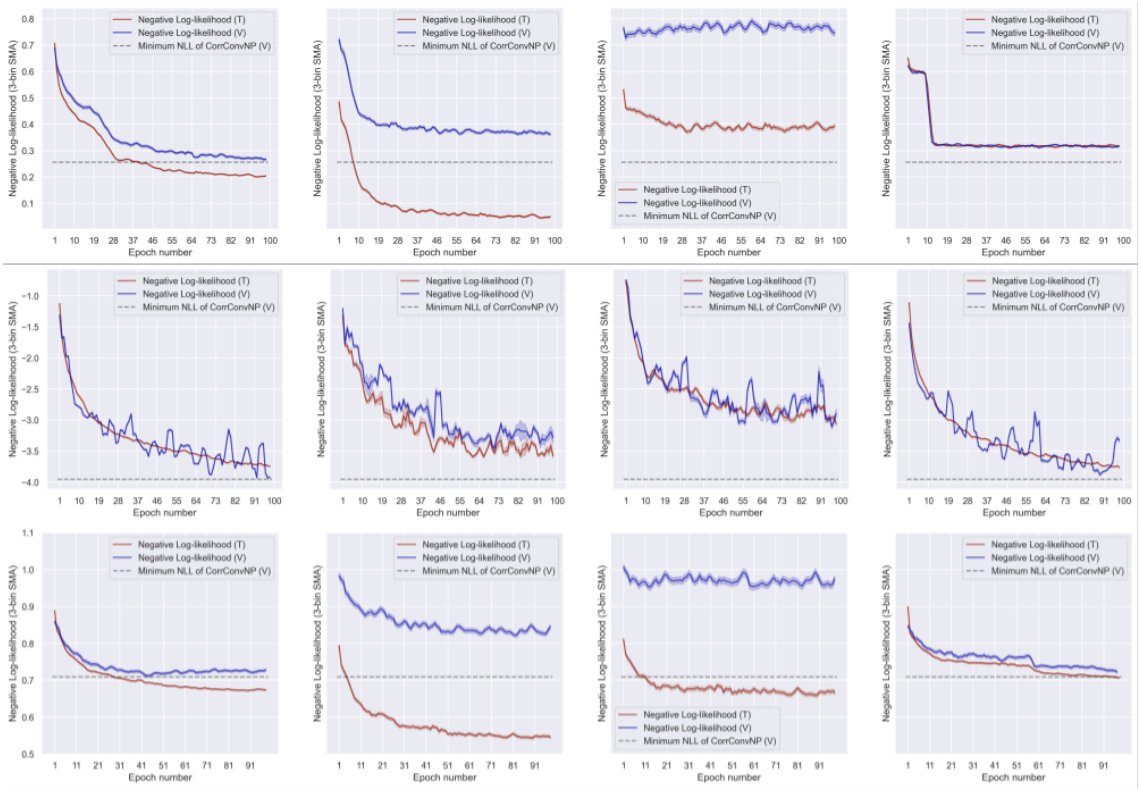


Fig. 3.2 Training NLL loss with epoch number on both the training set (red) and held-out validation set (blue) on the three regression tasks (top: EQ, middle: sawtooth, bottom: bimodal) for all convolutional models. Left to right the models are CorrConvNP, ConvGNP, ConvCNP, and ConvNP. The black dashed line represents the minimum NLL of the CorrConvNP on the validation set.

EQ GP Discussion:

In the top row of Figure 3.3 it is clear that all models have fit well to the EQ GP task despite the low number of context points given for them to condition on. The closest in fit for both mean line and uncertainty seems to be the ConvCNP, despite its poorest performance on this task in Figure 3.2. The reason for this is because the ConvCNP treats all uncertainty as aleatoric due to its independent target point samples (Markou et al., 2022). All plots in this figure have had aleatoric noise removed for visual effect,[4] and unlike the other models which include epistemic uncertainty in their samples, the ConvCNP predictions are reduced to apparently deterministic samples in the absence of it. A strong performing model will fill the ground truth uncertainty with functional uncertainty represented by samples filling the space. This is done to a large extent by the three other models, explaining their much better log-likelihoods in Figure 3.2.

[4] Aleatoric noise is still represented by the shading, which represents the 95% confidence interval for the cumulative aleatoric and epistemic uncertainty.

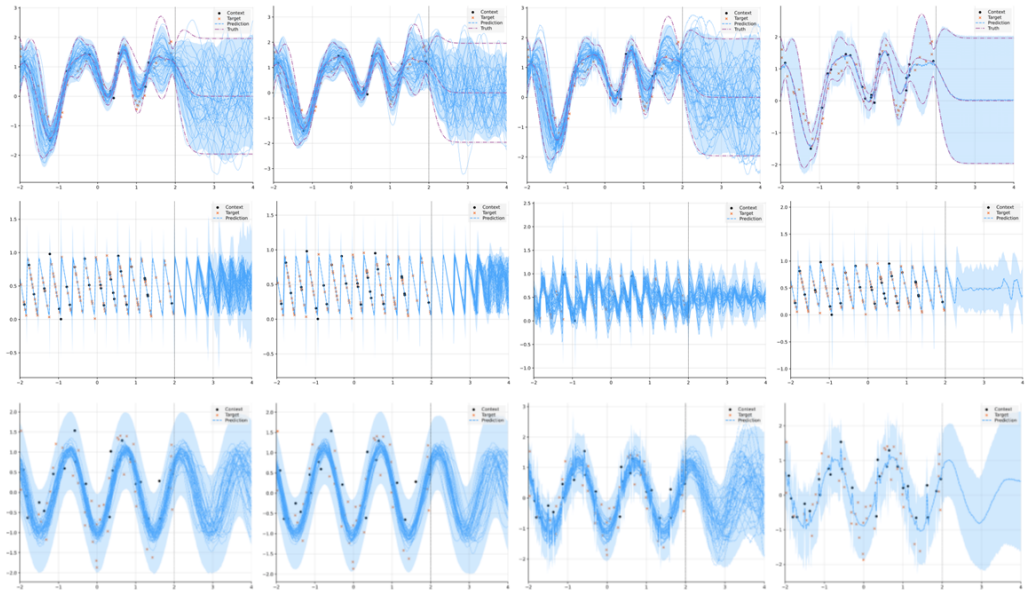


Fig. 3.3 Comparing the new model to existing models on three regression tasks with different properties. The first task (top row) is a GP with an EQ kernel and Gaussian distributed outputs. The second (middle row) is a sawtooth as generated in W. Bruinsma, 2022 demonstrating a periodic function with non-Gaussian distributed outputs. The third (bottom row) is a bi-modal sinusoid with randomly generated phase and bi-modal Gaussian noise that is very poorly approximated by a Gaussian predictive distribution. From left to right the models are: CorrConvNP, ConvNP, ConvGNP, ConvCNP.

Recall that this experiment was designed to suit well the Gaussian predictive models. It is somewhat surprising then that the ConvGNP is not the strongest performer here, as its correlated samples mean that it can represent correlations as robustly as the CorrConvNP and ConvNP, and it does not have the weakness of the ELBO-trained ConvNP that can underweight net uncertainty (as discussed in Section 2.7). Indeed, the understating of uncertainty is apparent in both encoder models, but particularly pronounced for the ConvNP, where in the extrapolation domain (to the right of the grey $x = 2$ line) it can be seen that while the balance between aleatoric and espistemic uncertainty is good (aleatoric uncertainty being negligible), the epistemic uncertainty is smaller than it should be. As mentioned in Section 2.7, this is hypothesised by Le, 2018 to be a consequence of the mean-field approximation in LV models. The LVs are thought to capture the functional uncertainty of the underlying stochastic process, so the mean-field approximation strips out information about the underlying process when it is forced to be diagonal, leading to epistemic uncertainty underweighting.

This would indicate then that the CorrConvNP should not suffer from this problem, as this new model contains correlated LVs. Whilst there is some underweighting still present in the CorrConvNP case, it is evidently reduced, and the CorrConvNP gives the best NLL performance of all the models on this data type, supporting this hypothesis. Note also that the LVs of the CorrConvNP, whilst correlated, *are still forced to be Gaussian*, which could also explain the small extent of functional uncertainty understating if the Gaussian approximation is too restrictive to fully represent the variability of the underlying stochastic process.[5]

Bi-Directional Sawtooth Discussion:

The bidirectional sawtooth experiment does not show the universally strong performance of the EQ GP one, as shown both by the variability in NLL values during training in Figure 3.2 and the poor fits of the conditional models in the fixed evaluation task in Figure 3.3. In the interpolation domain to the left of the $x = 2$ line all models seem to fit to the jagged structure of the sawtooth to some extent. However, the non-Gaussian nature of the task means that the ConvGNP struggles a lot here to find the directional shape of the function and ends up predicting far more epistemic uncertainty than the other models, causing the wider and more untidy teeth.

For the ConvCNP, the characteristic of neglecting epistemic uncertainty does not affect the fit in the interpolation domain, as the net uncertainty is almost negligible for this experiment when the context set is sufficiently large. The same reasoning explains the strong performance of the ConvNP model which previously suffered from epistemic uncertainty understating, giving the ConvNP very similar performance to the CorrConvNP. What lets the ConvCNP (and to a lesser extent the ConvGNP) down however, is the poor generalisation ability in the extrapolation domain. For the case of the ConvCNP this can be attributed in part to uncorrelated samples, meaning that the predictions decay back to the prior (which is a flat line at 0 with purely aleatoric noise) very quickly beyond the context points. The ConvGNP does have correlated samples though, so the cause of its poor extrapolation must be a consequence of its lack of an encoder and fewer parameters. A possible explanation for this follows from the theory that conditioning on the global

[5] An interesting experiment to continue this investigation would be to use a ConvNP encoder with a ConvCNP decoder, which would still give the correlated LVs, but now allow them to be non-Gaussian. The draw-back of this however would likely be high model complexity and difficulty training.

LVs provides a 'memory' about the shape of the underlying stochastic process that can be extended to unseen regions, explaining the better performance of the encoder models. As discussed shortly, there will also be the practical effect of smaller receptive fields in the convolutional conditional models that also limit their extrapolation ability.

Bimodal Sinusoid Discussion:

The advantages and disadvantages of the models from the previous two experiments are well summarised in the results from the bimodal sinusoid experiment. For the same reasons as outlined above, the ConvCNP reduces to deterministic predictions with heteroskedastic Gaussian observational noise (Foong et al., 2020) which fails to adequately model the non-Gaussianity of the true noise distribution, leading to the poor validation set performances seen in the bottom row of Figure 3.2. In a similar narrative, the ConvGNP also performs badly, as it can not predict bimodal aleatoric noise with a Gaussian distribution, so ends up exaggerating the functional uncertainty and predicting jagged spikes in the sample functions to fill the probability space, as is seen in the bottom row of Figure 3.3. The net effect of this is again a poor validation set NLL.

When it comes to the encoder models we see similar attributes between the two. Both are able to capture the shape of the underlying sinusoid and fit the noise well. These models have flexible predictive distribution shapes due to marginalisation over their LVs, which mean that they do not have the same issues modelling the non-Gaussian heteroskedastic observation noise. Whilst it is hard to see this improved fit from solely a shaded region in Figure 3.3, the lower validation set losses seen for these models in the bottom row of Figure 3.2 are testament to this. Comparing between the two they perform very similarly, as the issue with the ConvNP is to do with epistemic uncertainty understating, and there is little functional uncertainty in the bimodal tasks. That being said the CorrConvNP still achieves marginally better performance, and this is shown shown further in the following section.

3.3.1 Breakdown of Performance over Different Domains

To provide further breakdown of the log-likelihood performance we consider two different domains: the interpolation domain, where target inputs are within the range of the context points for that evaluation task; and the extrapolation domain, where target inputs are outside the range of context points for that task, but within the range of context points for *all* tasks in training.

Figure 3.4 shows the evaluation set log-likelihoods (note *not* the NLL anymore) on all regression tasks for all convolutional models when only considering target points within the former interpolation domain. In strong agreement with what was observed during training, the CorrConvNP performs the best in all cases, although only marginally better than the ConvNP in the bi-directional sawtooth and bimodal sinusoid tasks due to them having much less of the functional uncertainty that degrades ConvNP performance. The errorbars represent the 95% confidence interval across multiple evaluations on different tasks.

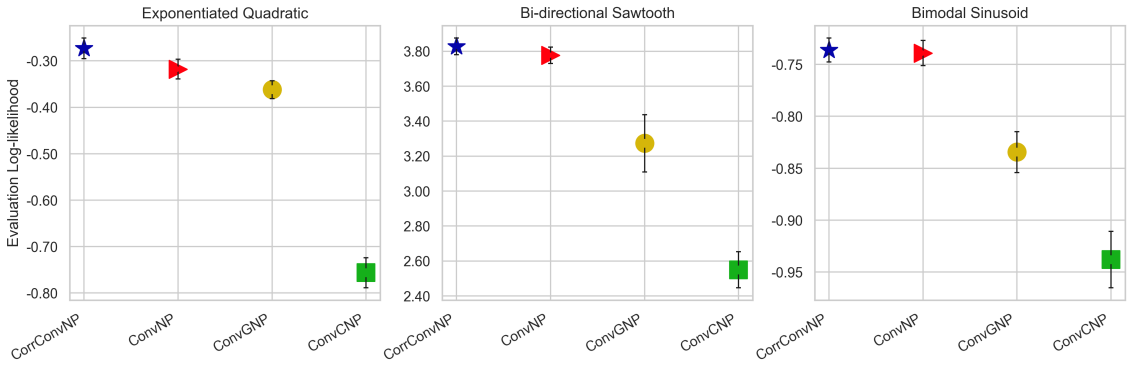


Fig. 3.4 Evaluation log-likelihoods of all convolutional models considered in this thesis on the 3 data regimes in an interpolation domain. The interpolation domain represents the domain of input values which are bounded by context points for the specific task being evaluated on.

Figure 3.5 shows the same except now for the extrapolation domain. The extrapolation domain covers regions of points that are unseen for the current task but that have been observed in previous training tasks. This can be thought of as a test of the models' meta-learning abilities. Traditional models would fit well to the current task but be largely unable to make predictions in the unseen data regime, leading to poor extrapolation log-likelihoods. We hope that by using meta-training the models will now be able to retain the same performance in the extrapolation domain as in the interpolation one, as

the distribution information learned on training tasks follows through. The fact that the extrapolation log-likelihoods mirror so closely the interpolation ones is therefore reassuring, as it demonstrates that the models do not overfit to the domain of the current task's context points.

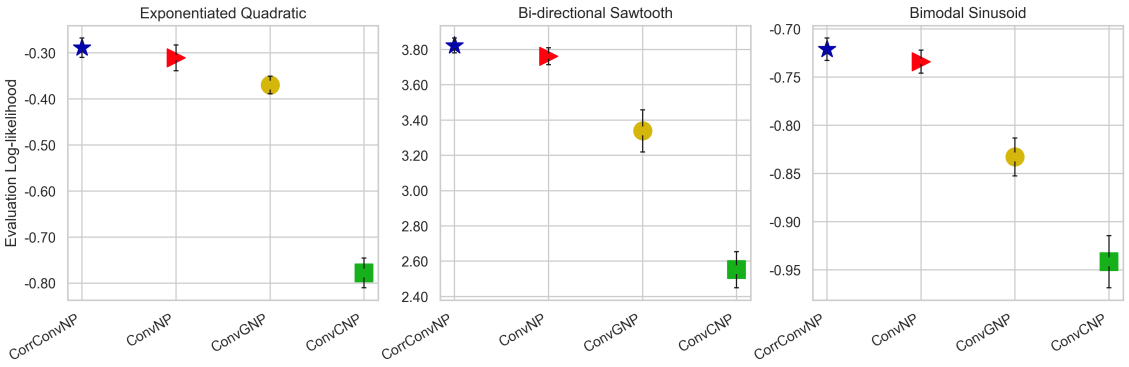


Fig. 3.5 Evaluation log-likelihoods for the 4 convolutional models on the 3 data regimes but now in an extrapolation domain. This means that the evaluation points are chosen to be *outside* the range of context points for that evaluation task. However, they can be within the range of context points for other tasks seen during training, and so this is not a complete extrapolation to an unseen domain.

The only other domain left to consider is the extrapolation domain that has not been observed by context points in *any* of the tasks during training. For periodic experiments this domain is of even more importance, as a strong model will be able to recognise the periodicity and fit to the unseen regions. However, we do not include results from such a domain, because the way that the models are implemented in practice means that the receptive fields of the convolutional encoder models are greater than those of the conditional ones, due to the additional CNN in the encoder (Araujo, Norris, and Sim, 2019). A greater receptive field allows a model to cover a greater domain for its predictions, and hence give the convolutional encoder models an unfair advantage in practice.

3.4 Significantly Better? Changing the Training Regime

So far there has been a trend that the encoder models tend to outperform the conditional models but behave similarly to each other, except for the CorrConvNP having slight improvements over the ConvNP wherever epistemic uncertainty is important. This raises the question though of whether the CorrConvNP is *significantly* better than the ConvNP or whether these improvements are coincidental and insignificant. For example, it could be

the case that the correlated LVs of the CorrConvNP become more and more independent during training (i.e. the encoder learns to predict a more and more diagonal covariance), which would mean that the CorrConvNP collapses back to the ConvNP in the high-epoch limit, mitigating the advantages of introducing correlation in the first place.

It is hard to test this however when training can change so much between models. For example, the ConvNP is known to escape local optima and drop NLL values suddenly at high-epoch numbers, so just taking a fixed number of epochs and comparing the two models could be introducing a bias and limiting the efficacy of the significance test.

Therefore, to test for significance, a training regime change experiment is proposed instead. This is a simple means of quantifying the statistical significance that the CorrConvNP achieves a better NLL than the ConvNP, where we will conclude that the CorrConvNP is better if and only if the performance is better to better than 5% significance. The process for this is as follows:

1. A CorrConvNP is trained for 20 epochs in the usual fashion without any change to the training process. This is used as the control in the significance test.

2. An identical CorrConvNP is then trained for 10 epochs, at which point the encoder's covariance matrix is forced to be diagonal, which as shown in Section 3.2 returns the ConvNP.

3. Training on this new ConvNP is then carried on for a further 10 epochs.

4. This process is repeated for another 9 training loops with different seeds for both regimes, and the mean and standard deviations of the NLL values taken for significance testing.

If, upon changing the training regime at epoch 10, we observe a spike in the validation set NLL, then this suggests that the CorrConvNP is not simply learning a diagonal LV covariance matrix, as forcing it to be diagonal is detrimental to training (at least after 10 epochs). To be more thorough however, we wish to see the long term effects of this restriction. If after a further 10 epochs the restricted model then converges to a higher NLL than the control, this is evidence for a permanent detrimental effect of having independent LVs, and is used to say that the CorrConvNP is significantly better than the ConvNP. The

multiple training runs allow for the computation of standard deviation values that can be used to gauge the statistical significance of the differences at convergence.

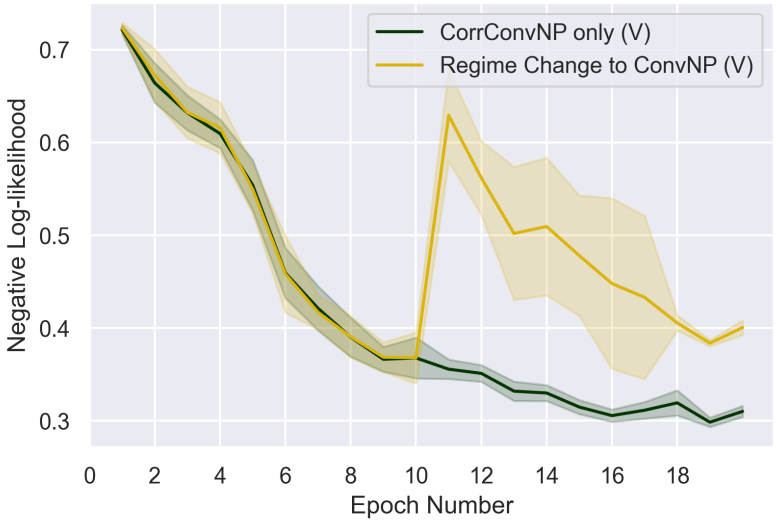


Fig. 3.6 Plot of the validation set NLL as a function of number of epochs trained on. Each epoch contains 16 distinct EQ kernel functions to approximate. The model change from CorrConvNP to ConvNP occurs at epoch 10, and an instant and lasting negative effect is observed as a result. The solid lines are mean values over 9 runs on different seeds, and the shaded regions show the 95% confidence interval.

Figure 3.6 shows the results of this experiment on the EQ GP tasks. These tasks were used because the high amount of epistemic uncertainty relative to observation noise accentuates the potential differences between the CorrConvNP and ConvNP to a greater extent than the sawtooth and bimodal sinusoid, which contain almost purely aleatoric noise. Before 10 epochs the losses follow each other closely, which is to be expected given that they are the same model at this point. At epoch 10 there is then the spike in NLL for the regime change model while the control continues to decrease in NLL. This supports the theory that the CorrConvNP encoder has learned to build non-diagonal covariances by 10 epochs that improve the training performance. Beyond 10 epochs the regime-changed model again starts training again, this time as a ConvNP, but by 20 epochs it has not reached as low a NLL as the CorrConvNP control. The shaded regions show the 95% confidence intervals over the 10 distinct training runs, so the fact that there is no overlap gives reason to say that we can be greater than 95% confident that by epoch 20 the CorrConvNP models the EQ GP tasks significantly better than the ConvNP.

However, there does not appear to be convincing convergence by epoch 20 on either regime. Therefore it could still be the case that the regime-changed ConvNP, whilst initially

negatively affected by the restriction on the LV covariance, will converge eventually to a similar loss. To show that this is not the case the process was repeated again, except now for 200 epochs rather than 20, with the change occurring after 100. This is shown by Figure 3.7.

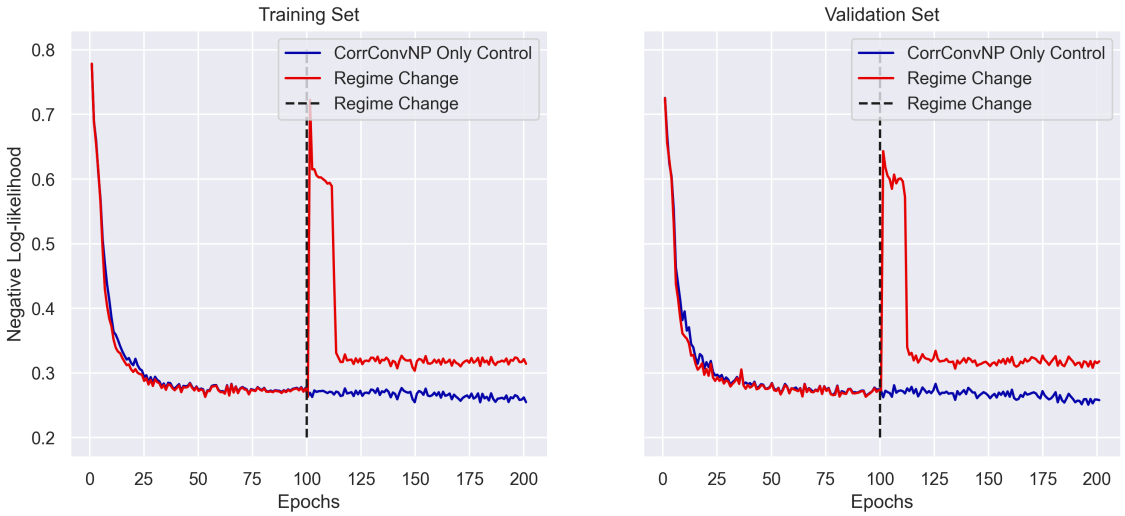


Fig. 3.7 A similar plot to in Figure 3.6 but this time for a single seed to reduce computation time. The number of epochs is scaled up 10 times to 200, with a model change at 100 epochs. The purpose of this plot was to demonstrate how the training of the two regimes converge after many epochs. It shows that the damage to the NLL from restricting the LV covariance to be diagonal is lasting, with the converged NLL being greater for the changed training regime than for the CorrConvNP control.

Whilst Figure 3.7 only includes a single training run per regime,[6] it is clear that the converged NLL values for the regime-change model are permanently damaged by moving from correlated to independent LVs. What is more, upon the initial restriction at 100 epochs, the instantaneous increase in NLL is even more severe than it was in the previous experiment after 10 epochs. This means that representation learned by the CorrConvNP encoder actually relies very heavily on its correlations to achieve a good fit, and it certainly does not learn to predict a diagonal covariance in the high-epoch limit.

[6]This is for computational reasons, as 200 epochs takes a long time to train, so repeating the process 10 times for each regime was unfeasible.

Chapter 4

Extensions to Non-Gaussian Likelihoods

Until now we have considered Gaussian likelihoods for the decoders of all models. For the conditional models this assumption was restrictive and limited performance on non-Gaussian tasks, whereas for the encoder models, marginalisation allowed for flexible distributions that depend on the distribution of $\{z\}$. That being said, there may be tasks where it is useful to use non-Gaussian decoder distributions from the start, even for the encoder models, when we have particular constraints which we wish to impose onto the distributions. These cases are considered in Chapter 4, starting with classification likelihoods in the first half, and Gamma likelihoods in the second.

4.1 Classification: Using a Bernoulli Likelihood

Unlike with the regression tasks considered so far, a continuous distribution is not suitable for fitting to the categorical targets of a classification problem. In this case a Bernoulli likelihood is better suited if there are two target classes, or a categorical distribution if there are more (for example, see Sections 4.2.3 and 4.2.4 of Murphy, 2022 respectively). For simplicity it is easier to consider binary targets and use a Bernoulli likelihood, and so this is done in this section, but a multi-class categorical likelihood is a simple extension of this.

4.1.1 Likelihood Definition

If considering only the encoder models, the decoder likelihood is a Bernoulli distribution conditioned on the LVs and is given by (Murphy, 2022):

$$p(\boldsymbol{y}_t|\boldsymbol{X}_t,\{z\};D_c) = \prod_{m\in D_t} \rho_m^{y_m^{(t)}}(1-\rho_m)^{\left(1-y_m^{(t)}\right)} \tag{4.1}$$

for $\rho_m = \mathscr{F}_{decoder}\left(\boldsymbol{x}_t^{(m)};\{z\}\right)$ the output of the decoder network. The log-likelihood of the entire target set it thus given as usual by marginalising out the LVs:

$$\log p(\boldsymbol{y}_t|\boldsymbol{X}_t;D_c) = \log E_{p(z)}\left[\prod_{m\in D_t} \rho_m^{y_m^{(t)}}(1-\rho_m)^{(1-y_m^{(t)})}\right] \tag{4.2}$$

As was the case with the Gaussian likelihoods, the correlation between target points is achieved through the non-factorisability of the (log)-likelihood, and plays a large part in the modeling of spatial data, for example. To show this, we consider two similar classification experiments of varying difficulty: a 2-class Gaussian mixture model, and a 2-class GP-cutoff model.

4.1.2 Mixture of Two Gaussians

In the same way that robust uncertainty quantification is important for downstream tasks in regression, it is also crucial for classification in many practical applications (Lakshminarayanan, Pritzel, and Blundell, 2017). This experiment is used not only to demonstrate the importance of spatial correlations in classification when there are sparse observations, but also to reinforce how the ConvCorrNP avoids overfitting even when the training tasks are simple and contain low amounts of data. This leads to better uncertainty quantification and more realistic decision boundaries than traditional classification models.

A binary mixture of Gaussians is defined in 1D with means at $\pm(1+\varepsilon)$ and variances of $0.5+0.2\varepsilon$. Each input point is sampled from one of the Gaussians with component probabilities of 0.5, and the output classed as 0 if the Gaussian centered on $-(1+\varepsilon)$ is chosen or 1 if the Gaussian centered on $+(1+\varepsilon)$ is selected instead. $\varepsilon \sim \mathscr{N}(0,1)$ represents perturbative noise that alters the covariances in such a way to ensure that each

task's distribution is unique and meta-learning is conformed to. What is more, the number of context points is reduced to between 5 and 10 per task to represent observation sparsity.

Figure 4.1 (left) shows the CorrConvNP predictions on an evaluation task. It shows the model predictions follow closely the analytic decision boundary (shown in black) and do not overfit, despite the low number of context points during training relative to the number of model parameters. In contrast, a non meta-trained classifier quickly overfits to the training points by the 30 epoch mark and the resulting decision boundary is far steeper. This is what is demonstrated by the right-hand plot, which shows the predictions of a simple MLP with only 31 parameters[1] that has been trained on the same number of points, batches and epochs as the CorrConvNP, but with data from *a single* distribution. The advantage of being able to learn by meta-learning is made clear here, as the traditional learning method overfits very easily in comparison.

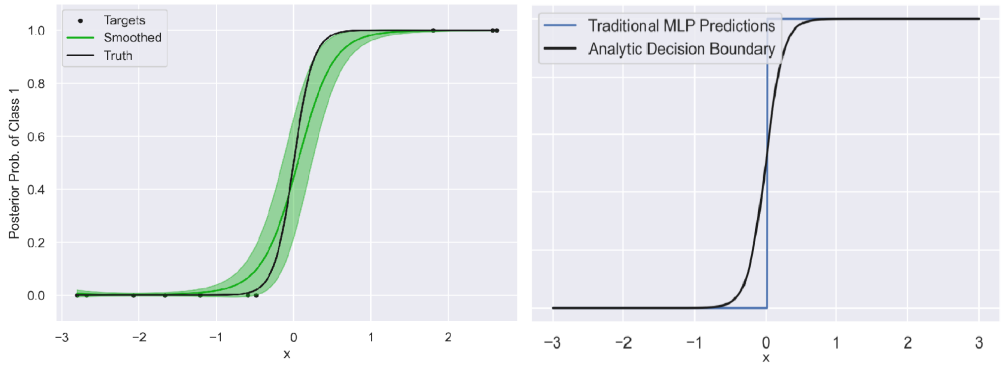


Fig. 4.1 (Left) A binary Mixture of Gaussians experiment in 1D showing the strong uncertainty quantification of the CorrConvNP. It shows the fit of the model averaged over 25 samples (green shading is the 95% confidence interval) compared to the analytic black ground truth. (Right) The fit of a 31-parameter MLP after seeing the same number of training points over the same number of epochs, but on data from a single task rather than meta-learning.

4.1.3 GP-Cutoff

The previous experiment was done to demonstrate the CorrConvNP's low capacity for overfitting in a classification setting, but has a simple linear decision boundary and does not demonstrate how the model handles more complex tasks. Therefore this next experiment is another binary classifier, but this time with a true decision boundary that is highly non-linear and varies a lot between meta tasks.

[1] In stark contrast, the CorrConvNP contains a far greater 202,371 parameters for this experiment!

It is constructed using a continuous GP sample, and a cut-off converts it into a categorical problem. For each given task, the outputs $\boldsymbol{y}_c, \boldsymbol{y}_t$ are 1 if the GP sample is greater than the median at that input point, or 0 if it is less than it. The use of the median guarantees we approximately have equal numbers of points in each class and mitigates availability bias. By using a GP we also have control over the length-scales of the samples, which by proxy lets us control the separation and width of the regions belonging to each class and hence controls the task complexity.

Figure 4.2 shows the resulting fit of the CorrConvNP on a GP-cutoff task after 100 epochs of training. Apart from the occasional region which is badly learned (see, for example, the bottom left corner), the model does an encouragingly good job of fitting to the true class memberships given that the decision boundaries change in shape and location so much between tasks.

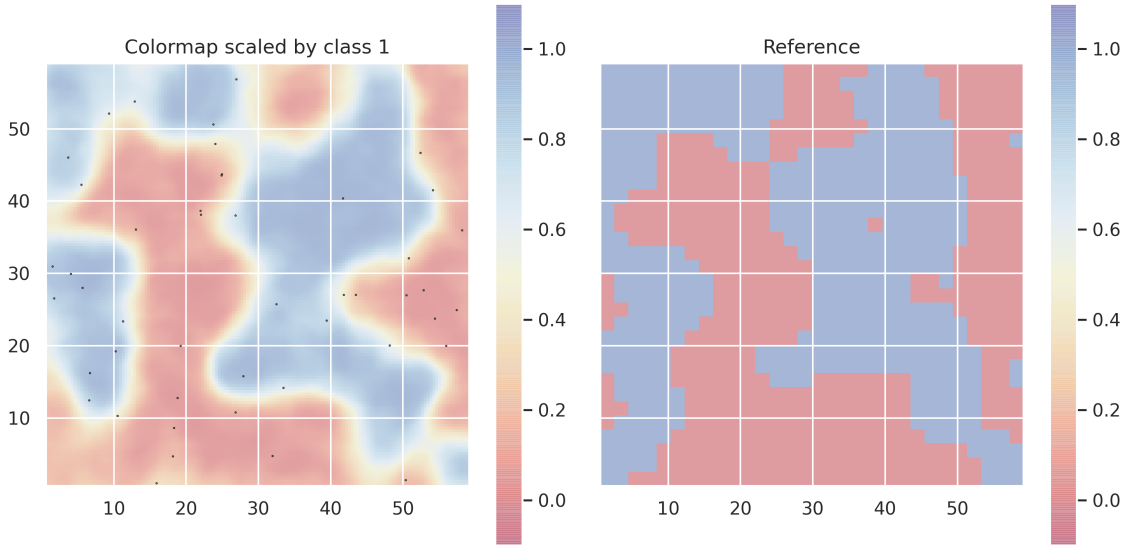


Fig. 4.2 The ConvCorrNP model predictions on an evaluation task for the 2D GP-cutoff experiment after 100 training epochs. The black dots represent the context point locations and the colour bar is scaled by the probability of belonging to class 1.

Notice how in Figure 4.2, the grid cell between (20, 20) and (30, 30) contains no context points, yet fits well the ground truth reference. This is a demonstration of strong interpolation in regions of no context points due to both TE from the convolutional structure and having target point correlation.

4.1.4 Categorical Likelihoods

If instead we did have multiple classes, then the categorical log-likelihood for an encoder model would be defined as follows (Murphy, 2022):

$$\log p(\boldsymbol{y}_t|\boldsymbol{X}_t;D_c) = \log E_{p(z)}\left[\prod_{m\in D_t}\prod_{k=1}^{K}\left(\rho_m^{(k)}\right)^{\mathbb{I}(y_m^{(t)}=k)}\right] \tag{4.3}$$

where $\mathbb{I}(\cdot)$ is the indicator function, $\mathbb{I}(y_m^{(t)}=k) := 1$ if $y_m^{(t)}=k$ and 0 otherwise. $\boldsymbol{\rho}_m = \{\rho_m^{(k)}\}_{k=1}^K = \mathscr{F}_{decoder}(\boldsymbol{x}_m^{(t)},\{z\})$ is now the K-headed output of the decoder network, with the k^{th} element corresponding to the probability that the current target point belongs to class k.

4.2 Modeling Amounts: Using a Gamma Likelihood

Even when the tasks are continuous there can be situations where a non-Gaussian likelihood gives improvements. Take, for example, the modeling of the *amount* of something, such as the amount of rainfall at a location on a given day that follows a Gamma distribution (Ye et al., 2018), or the volume of insurance claims which are Poisson or Gamma distributed (Promislow, 2014).[2] These distributions share the property that their samples are strictly positive and generally have positive skew, as theoretically samples can take the value infinity. Even when LVs are used for distributional flexibility, using a Gaussian likelihood for the decoder means that these constraints are not automatically satisfied. This can lead to non-physical results unless a stopgap approach is used (such as a truncation of the Gaussian samples at 0 or exponentiation),[3] which detracts from the generative approach of these models. This section therefore considers such cases and explores a Gamma likelihood function.

[2]Insurance claim rate is generally Poisson distributed if we are considering only a single claim, and Gamma distributed if considering the accumulation of multiple (Promislow, 2014).

[3]Exponentiation of a Gaussian variable produces a log-normal distribution, which is often used as an approximation to more complex but theoretically derived exponential distributions, but quite rarely occurs exactly in nature. The times it does are when dealing with the addition of Gaussian variables in a log-domain, for example in economics or the Weber-Fechner law from psychophysics (Fechner, 1860).

4.2.1 Likelihood Definition

Again considering only the encoder models, the decoder likelihood conditioned on $\{z\}$ is a Gamma distribution defined as follows (Pishro-Nik, 2014):

$$p(\boldsymbol{y}_t|\boldsymbol{X}_t,\{z\};D_c)=\prod_{m\in D_t}\frac{1}{\Gamma(\alpha_m)}\beta_m^{\alpha_m}\left(y_m^{(t)}\right)^{(\alpha_m-1)}\exp\left[-\beta_m y_m^{(t)}\right] \tag{4.4}$$

where the shape and rate parameters are given from the multi-head decoder output as $\alpha_m,\beta_m=\mathscr{F}_{decoder}\left(\boldsymbol{x}_m^{(t)},\{z\}\right)$ respectfully. The Gamma function is defined as $\Gamma(x)=(x-1)!$, and the scale of the distribution is the inverse rate, $1/\beta_m$. There are restrictions on these parameters to ensure that the gamma distribution is well-defined, which are that $\alpha_m,\beta_m>0\;\forall\;m$, and for the purpose of stable predictions, $\alpha_m>1\;\forall\;m$ also.

The log-likelihood of the entire model used for training and evaluation is therefore:

$$\log p(\boldsymbol{y}_t|\boldsymbol{X}_t;D_c)=\log E_{p(z)}\left[\prod_{m\in D_t}\frac{1}{\Gamma(\alpha_m)}\beta_m^{\alpha_m}\left(y_m^{(t)}\right)^{(\alpha_m-1)}\exp\left(-\beta_m y_m^{(t)}\right)\right] \tag{4.5}$$

4.2.2 Gamma Process

This experiment involves constructing a 2D Gamma Process, which is much like a Gaussian Process, except that the predictive distributions at each input point are Gamma rather than Gaussian. There is a difficulty with constructing such a process artificially however, which is that it is far more difficult to introduce correlations directly into samples from a Gamma distribution than samples from a Gaussian.[4]

Therefore, this experiment uses the slight workaround of generating samples from a non-diagonal multivariate Gaussian to introduce correlation, and then exponentiating to form a log-normal distribution. As mentioned in the second footnote of Section 4.2, the log-normal distribution gives only an approximation of the Gamma distribution, but fundamental characteristics such as bounds and asymptotic behaviour are the same. What is more, for certain parameters, the shapes of the two distributions are very similar, so the

[4]For example Bithas et al., 2007 provide an expression for the joint probability function of 2 correlated Gamma random variables, and it is clear that it is far from trivial to generalise to an arbitrary number of samples.

encoder models with their flexible marginalised likelihoods will not have a problem fitting to the lognormal tasks.

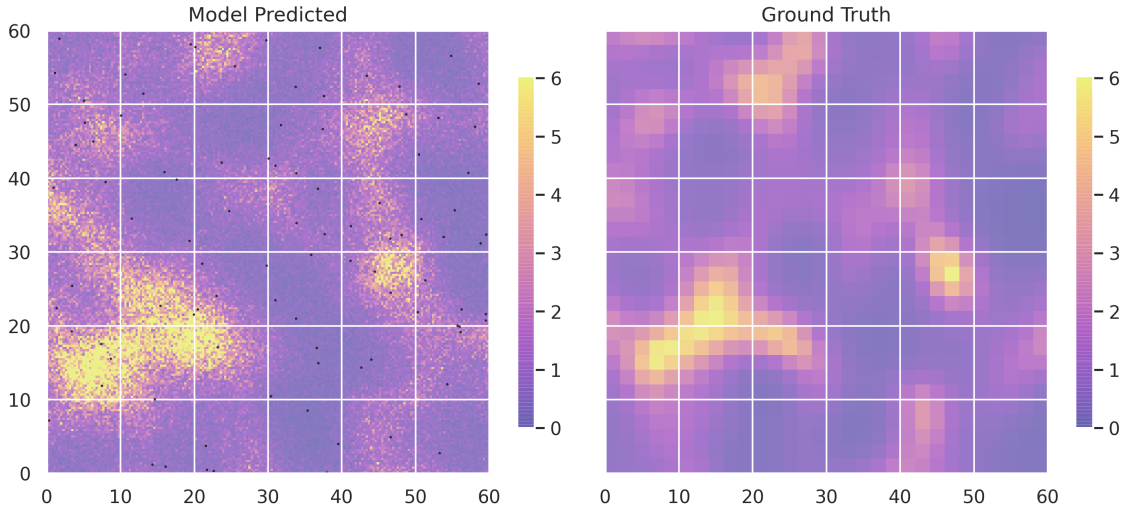


Fig. 4.3 Fit of the Gamma likelihood CorrConvNP model to a 2D synthetic Gamma process evaluation task. The black dots are again context points, and there are between 50 and 100 of them for each task.

Figure 4.3 shows the results of the Gamma Process regression with the ground truth plotted on the right and the models predictions on the left. It is visually clear that the fit is good, with clear structure and spatial correlation even in regions of few or no context points.

There are however isolated regions where the predictions are poor, highlighted for example by the dark spots in the bright yellow sections. These are either caused by too high observation noise being predicted (although this is unlikely as it is not seen across the whole domain as would be expected), or more likely it demonstrates that the Gamma distribution at those points is underdeveloped. If the latter is true, then more extensive parameter tuning would hopefully improve the issue.

4.3 Model Comparisons on Extended Likelihood Tasks

It is clear that the CorrConvNP can fit to both the classification and Gamma likelihoods well then, but it is unclear how it compares to the mean-field ConvNP and the conditional ConvCNP models. Figure 4.4 therefore shows the comparison on validation set tasks. Note that all models' decoders have the relevant adapted likelihood (Bernoulli or Gamma) for each experiment to make them comparable.

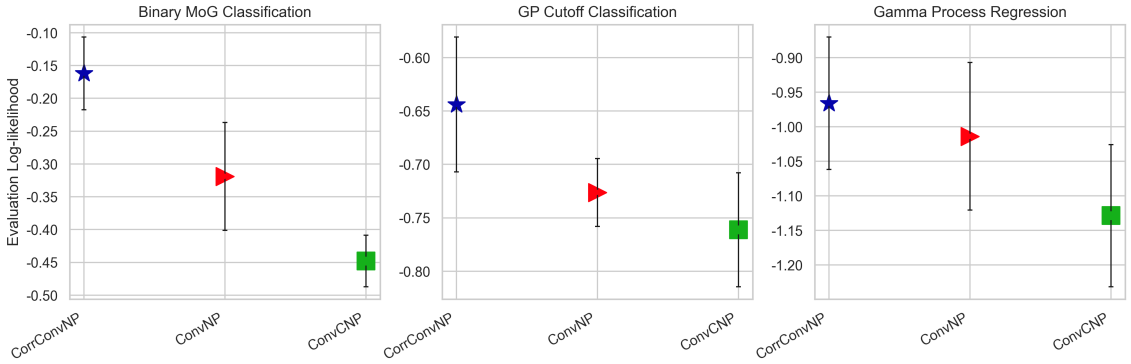


Fig. 4.4 Comparison of the evaluation log-likelihoods of the CorrConvNP, ConvNP and ConvCNP models on the extended likelihoods tasks described above, after 100 epochs of training. The binary mixture of two Gaussians is a 1D input task, whereas the GP-cutoff and Gamma process are both 2D inputs. The error bars are 95% confidence intervals.

Firstly, the binary mixture of Gaussians classification task shows the addition of LVs to give a (weakly[5]) significant advantage compared to the ConvCNP, with the CorrConvNP outperforming the ConvNP also. The GP-cutoff and Gamma process results follow the same hierarchy, except that any improvements from (i) introducing correlations in the LVs, or (ii) adding LVs to a conditional model in the first place, are not significant for these two experiments.

Like with the regression experiments of Chapter 3, the reason for this lack of significance comes down to the number of context points, specifically the fact that the 2D GP-cutoff and Gamma process experiments both use on average around 10 times more context points than the 1D binary MoG in order for the models to fit to the more complex tasks. This leads to less epistemic uncertainty (i.e. more deterministic true functions) in these tasks and a lower importance of correlated samples if more observations fill more of the domain. Therefore the ConvNP and ConvCNP achieve marginally worse but

[5]The word 'weakly' is used here as we have not carried out a thorough significance test like at the end of Chapter 3, where we changed the training regime. Instead we simply observe the 95% confidence interval on evaluation tasks after 100 epochs.

similar results to the CorrConvNP, as their weaknesses relate to underestimating functional uncertainty and having uncorrelated samples respectively.

This is further reinforced when considering the binary MoG classification experiment. Its simplicity makes it easier to learn with fewer context points, which leads to greater epistemic uncertainty due to sparser observations, and a higher importance of correlated samples. Under this regime correlated LVs become very valuable, much like with the EQ GP regression task of Chapter 3, leading to apparent significance at each stage of the hierarchy shown in the LH plot of Figure 4.4.

Chapter 5

Rainfall Modeling: A Practical Example

We have shown in Chapter 4 that the CorrConvNP can be readily extended to both classification tasks and modeling strictly positive and right-skewed quantities via the Gamma distribution. If the model can be trained on these tasks individually then it is not unreasonable to think that they may be learned in parallel to provide a solution to problems with zero-inflation. This is where an excess of 0s in a dataset, most often caused by an unobserved latent variable, leads to poor fits with traditional modeling techniques.

Modeling rainfall is a well-known example where zero-inflation causes availability bias problems when trying to fit a regression model. A historic solution has been to alter an existing distribution such that it exaggerates the probability of getting 0 and does not incorrectly shrink the variance of the distribution as a result. The Zero-Inflated Poisson (Lambert, 1992) is an example of this, and has been used with some success for the rainfall task (Dzupire, Ngare, and Odongo, 2018). We take inspiration from this approach and build zero-inflation into the Gamma model of Chapter 4 by 'gating' the Gamma model with a Bernoulli classifier first.

5.1 The Bernoulli-Gamma Likelihood Function

With this in mind, the composite Bernoulli-Gamma likelihood function is defined as follows:

$$p(\boldsymbol{Y}_t|\boldsymbol{X}_t,\{z\};D_c) = \prod_{m\in D_t}\left[\rho_m^{y_{m,0}^{(t)}}(1-\rho_m)^{\left(1-y_{m,0}^{(t)}\right)}\right]\times \left[\frac{1}{\Gamma(\alpha_m)}\beta_m^{\alpha_m}\left(y_{m,1}^{(t)}\right)^{(\alpha_m-1)}\exp\left(-\beta_m y_{m,1}^{(t)}\right)\right]^{\mathbb{I}\left(y_{m,0}^{(t)}=1\right)} \tag{5.1}$$

where $\rho_m,\alpha_m,\beta_m = \mathscr{F}_{decoder}\left(\boldsymbol{x}_m^{(t)},\{z\};D_c\right)$ is a 3-headed decoder output. Here we use a slightly different notation than previously, as we are now breaking the assumption that the context and target set outputs are 1 dimensional. Instead $\boldsymbol{Y}_\cdot = \left\{\boldsymbol{y}_i^{(\cdot)}\right\}_i$ for $\boldsymbol{y}_i^{(\cdot)} = \left(y_{i,0}^{(\cdot)},y_{i,1}^{(\cdot)}\right)^T$. $y_{i,0}^{(\cdot)}$ is a binary variable representing whether a point in the context or target set has rainfall, and $y_{i,1}^{(\cdot)}$ is then the amount of rainfall given that the binary variable is 1 (i.e. it is raining at that point).

From Equation (5.1) it can be seen that the Gamma component that is concerned with the amount of rainfall is only activated if the binary variable is 1. This is to avoid zero-inflation bias in the data, which could lead to learned Gamma distributions with underestimated variances that simply predict rainfall amounts near to 0 for all locations. The formulation of Equation (5.1) also means that during training the Bernoulli parameter is learned *independently* to the Gamma parameters, and similarly the Gamma parameters are learned independently to the Bernoulli one, once only points with rain are considered. The hope is that by having no interference between the two likelihoods the decision of whether it is raining or not can still be learned well and provide some insight, even if the Gamma distribution fits the extent of rainfall badly.

5.2 Experimental Set-Up

The real-life data used to test this model was daily rainfall data in Sweden from 2018, distributed by the Copernicus Climate Change Service, 2021. The 2D inputs were the x (West-East) and y (South-North) coordinates measured in kilometres relative to the bottom left grid value. The 2D outputs again contain a binary variable signifying if there was rain

on a given day for that (x, y) region or not, and a positive scalar variable representing the amount of rainfall in kg/m^2. Due to the size of region and consequent memory costs of running the model on very large datasets, a subset of the available region was considered, covering a $60km$ by $60km$ grid, with a granularity of observations taken every $2km$. In this way there are 900 observed points per task, of which a random subset is taken as the context set, and the remainder of the grid of points as the target inputs (at test-time).

Three example days are shown in Figure 5.1 below. The first is for a particularly rainy day on the 1^{st} of January, the second for a less rainy day on the 20^{th} of February, and the third for a day without rain on the 11^{th} of April. These plots highlight a potential challenge from working with this data: the amount of rainfall can vary not only within a task, but also greatly between them. In fact, there are many days which show no rainfall at all, which will likely cause issues when training the model.

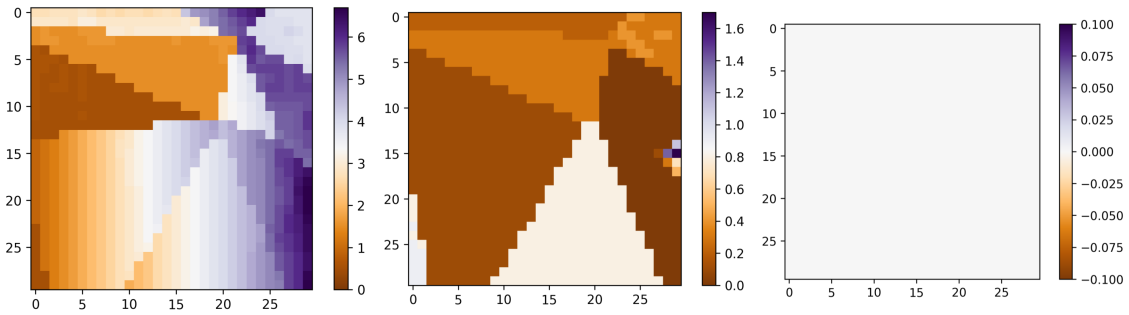


Fig. 5.1 Examples of rainfall data from 3 days that show very different properties. The first day has a lot of rain, with a lot of variation in rainfall amounts and a maximum value that is around 4 times greater than the middle plot, which represents a day with only a little rain. The middle plot also has much less variation in rainfall values, and the right hand plot shows a day with no variation as there is no rain at all. The colourbar is scaled by rainfall in kg/m^2.

With this in mind, we propose a synthetic rainfall experiment to train on, so that we can then run the pre-trained model on the real dataset at test time in a *sim-to-real* fashion (see Section 1.2.4 for details on what this entails). Training is easier to do with the synthetic data as all tasks contain regions of rainfall and no rainfall to mitigate availability bias, and the range of rainfall values between tasks are kept relatively similar. The synthetic tasks are constructed in a similar way to the GP-Cutoff experiments of Chapter 4, except that in this case the GP sample is multiplied by a ReLU function to give it regions where rainfall is strictly positive, and large regions where it is 0 to mimic zero-inflation.

5.3 Results on Synthetic Rainfall Data

At first we tried fitting the CorrConvNP with the likelihood in Equation (5.1) directly to the synthetic dataset, but encountered the problem that the model would not fit after many epochs. A hypothesis for why this is to do with the different magnitude and scaling of the two parts to the generative model loss. The Bernoulli loss is less sensitive to parameter values than the Gamma one and improves faster, meaning that model learning is dominated by learning the Gamma parameters. At the same time, the Gamma parameters are trained on only a smaller subset of the context and target points each task (because points are only included if the binary part of the output is 1), which makes the Gamma model difficult to fit, leading to noisy gradients and unstable learning.

Therefore, a workaround comes from splitting the likelihoods into *separate* models to independently predict Bernoulli and Gamma parameters, and then combining their predictions at the end. For a conditional model such as the ConvCNP, this returns exactly the closed-form joint distribution of Equation (5.1) as there is no marginalisation, and therefore has the sole effect of increasing the number of parameters. However, for the encoder models, a single-expectation joint distribution cannot be recovered due to the expectation over *different* $\{z\}$'s. The LV joint distribution from the separate models is instead given by:

$$
\begin{aligned}
p(\boldsymbol{Y}_t|\boldsymbol{X}_t;D_c) &= E_{p_1(z)}\left[\prod_{m\in D_t}\left[\rho_m^{y_{m,0}^{(t)}}(1-\rho_m)^{\left(1-y_{m,0}^{(t)}\right)}\right]\right]\times \\
&\quad E_{p_2(z)}\left[\left[\frac{1}{\Gamma(\alpha_m)}\beta_m^{\alpha_m}\left(y_{m,1}^{(t)}\right)^{(\alpha_m-1)}\exp\left(-\beta_m y_{m,1}^{(t)}\right)\right]^{\mathbb{I}\left(y_{m,0}^{(t)}=1\right)}\right] \\
&\approx E_{p(z)}\left[\prod_{m\in D_t}\left[\rho_m^{y_{m,0}^{(t)}}(1-\rho_m)^{\left(1-y_{m,0}^{(t)}\right)}\right]\times\right. \\
&\quad \left.\left[\frac{1}{\Gamma(\alpha_m)}\beta_m^{\alpha_m}\left(y_{m,1}^{(t)}\right)^{(\alpha_m-1)}\exp\left(-\beta_m y_{m,1}^{(t)}\right)\right]^{\mathbb{I}\left(y_{m,0}^{(t)}=1\right)}\right]
\end{aligned}
\tag{5.2}
$$

The single-expectation expression of Equation (5.1) could therefore again be recovered for the encoder models if the LV distributions were shared across the two encoders such that $p_1(z) \equiv p_2(z) := p(z)$, and this is something that would be interesting to explore

in the future. Nevertheless, despite being only an approximation to the true generative likelihood, the separated likelihood in Equation (5.2) still gives much improved results with the CorrConvNP compared to fitting the original combined model directly. Figure 5.2 demonstrates this by showing the output of a validation set task from the synthetic dataset after 50 epochs of training.

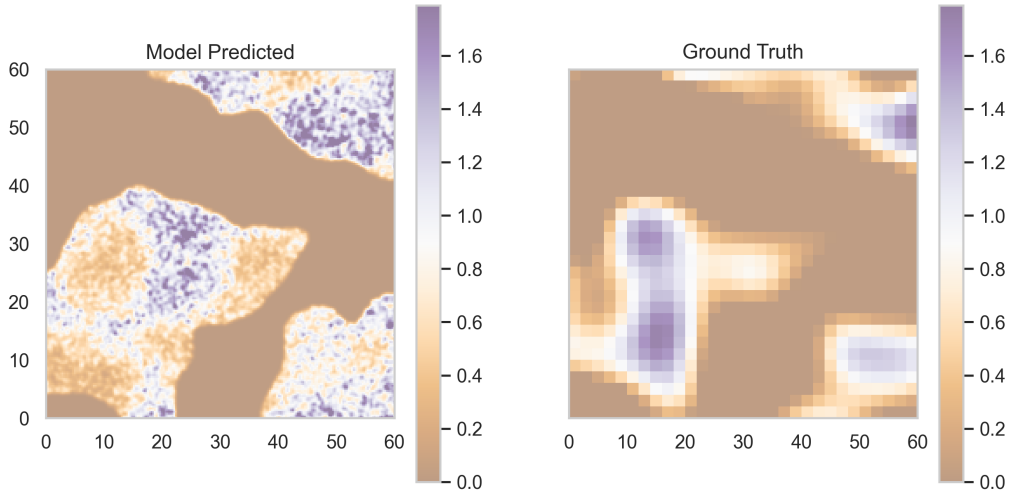


Fig. 5.2 Predicted rainfall in kg/m^2 on the LHS compared to the ground truth on the RHS, as generated by the synthetic zero-inflated Gamma process. The Bernoulli model seems to have succeeded in predicting the regions of no rainfall, but the Gamma model struggles to predict the extent of rainfall within the raining regions due to data scarcity. The total number of points available during each task varies between 0.75% and 1.15% of the number of image grid spaces available.

Whilst the Bernoulli model that predicts whether it is raining or not seems very well developed, the Gamma part seems to be lagging behind and only loosely fits the regions of high and low rainfall in the RHS reference. This is support for the hypothesis that the Gamma likelihood is far harder to fit to due to the amount of noise in backpropagation caused by a much restricted context and target set size relative to the Bernoulli model's. Again this is because the Gamma model only considers points in the context and target sets that have a binary variable of 1, and thus removes around half of the points available to the Bernoulli model each task. One solution to this would be to scale up the size of the context and target sets so that the Gamma model has more data available to it proportionally, but this is often not feasible due to the nature of low-data scenarios.

5.4 Adapting to the Real Rainfall Data

The result of running the pre-trained model on the real rain dataset at test-time can be seen in Figure 5.3. The results look positive, although this is not true for all days. As with the synthetic rainfall dataset before, what is true for all days is that the Bernoulli part of the likelihood fits well and the Gamma part lags somewhat behind. Reassuringly however, the trained model has been able to transfer the knowledge of the learned Gamma distribution from the synthetic task to achieve similar performance in the real case, reaffirming neural processes' strengths in sim-to-real adaptation.

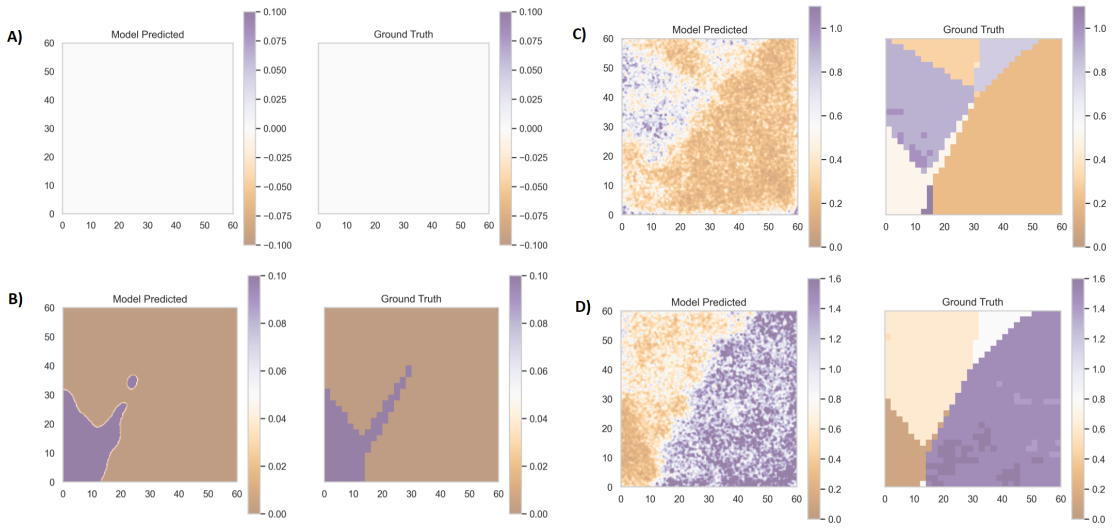


Fig. 5.3 Sim-to-real results for 4 days of the real rainfall data. The model was trained on a synthetic replica of the rainfall data and then implemented on real data without any backpropagation in a sim-to-real fashion, as described in Section 1.2.4.

What is more, by virtue of running the model on the real data in an evaluation regime only, when a zero-saturated day is encountered (like in (A) in Figure 5.3), the model training is not affected, and it is capable of recognising the zero-saturation at every occurrence. In fact, the performance on days with no rain whatsoever was better than expected, given that during training no synthetic task contained no rain at all. This is testament to the strong predictions of the Bernoulli part of the model, as the Gamma likelihood is only activated if the Bernoulli component predicts there to be rain at test-time.

In order to make the model's practical use more concrete, Figure 5.4 shows a real rainfall prediction overlaid on the geographical region that the data was taken from.

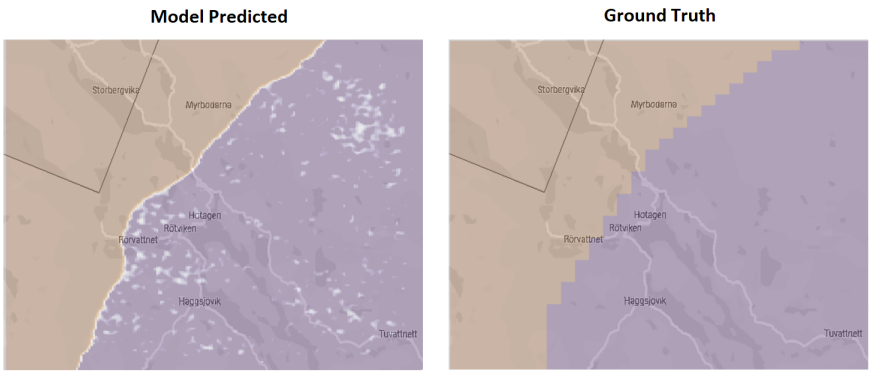


Fig. 5.4 Rainfall predictions from the real data overlaid on the geographical region where the data is taken from.

The evaluation log-likelihood on the rainfall data was -1.52, which is about inline with summing the likelihoods from the GP-cutoff classification and Gamma process regression experiments in Section 4.3, showing that using the real data gives similar performance to the synthetic sets used for training.[1]

[1]Recall that the synthetic rainfall data is fundamentally a combination of a GP-cutoff classifier and a Gamma process regressor.

Chapter 6

Conclusions and Future Work

6.1 Improvements on Baselines

The first set of experiments concerned recreating regression task results from the existing literature and comparing them to the CorrConvNP performance. This was a success story for the new model, as it performed best in all cases. Although only marginally and not statistically significantly better than the ConvNP on the sawtooth and bimodal sinusoid tasks, it *was* significantly better on the EQ GP. This is a very strong result given that of all the regression experiments, this is the one that the CorrConvNP is expected to be better on with its superior epistemic uncertainty estimation, indicating that removing the mean-field approximation improves the model.

6.2 Extension to Other Likelihoods

Further success was found when moving away from the previous Gaussian decoder likelihoods to ones that can handle categorical variables or skewed and bounded continuous ones. In both the classification and Gamma process experiments it was clear that the CorrConvNP is a versatile model, and well-suited to tasks beyond the regression specifications that had been considered in the literature and in Chapter 3. We also saw in the binary MoG classifier experiment that the CorrConvNP is notably[1] better than the ConvNP, which is

[1]The use of the word 'notably' rather than 'significantly' again refers to the lack of thorough statistical significance testing. See Section 4.3 for details.

itself notably better than the ConvCNP, as we were able to use fewer context points which increases the epistemic uncertainty. As experiment complexity increased when moving to 2D experiments, so did the number of context points required for the models to fit, which reduced epistemic uncertainty and levelled the playing field for the models. As a result, we saw that whilst the same ranking held, there was no significant advantage to adding correlation to LVs or adding LVs in the first place with these tasks. A conclusion from this could be therefore that the CorrConvNP is particularly well adapted to learning tasks with very little data.

6.3 Results of the Rainfall Case-Study

The rainfall case-study aims to demonstrate how the attributes of the CorrConvNP (and to a large extent the other NPF models) make it suitable for modeling previously hard-to-fit distributions such as zero-inflated data that occurs with rainfall. The experiments carried out demonstrate a good level of success, particularly when sim-to-real training is carried out first on a synthetic dataset to stabilise training. The zero-inflation was elegantly handled by the Bernoulli component of the likelihood, and the model was even able to fit at test-time to zero-saturated data from days when it did not rain at all. It is also clear however that the reduced number of context and target points available to the Gamma likelihood cause its parameters to not be as well learned. Apart from simply increasing the overall number of context and target points (which is infeasible in low-data scenarios), there is little that can be done to improve this underfitting with the current models. However, it is possible that another change of architecture to include deterministic pathways may yield improvements.

6.4 Improved Fitting in a Low-Data Regime: Latent and Deterministic Pathways

Attentive Neural Processes (ANPs) are another member of the NPF that use an attention mechanism in their architecture to address the original problem of underfitting in non-convolutional NPs (Kim et al., 2019). The introduction of convolutional NPF models, as discussed in detail in this thesis, also addressed this issue, and were found empirically to

give better performance than ANPs (Wessel Bruinsma et al., 2021), (Foong et al., 2020), justifying ANPs not being included thus far in the thesis.

However, part of the ANP architecture could give an interesting avenue for further research in our proposed CorrConvNP model, namely the use of deterministic and latent pathways in the encoder. Figure 6.1, adapted from Kim et al., 2019, shows a NP architecture where the encoder consists of a deterministic part and a latent part. The deterministic part includes no sampling and generates solely a deterministic vector embedding of the context set. The latent part produces a LV as seen before in Chapter 2 by sampling stochastically given the output. These latent and deterministic embeddings are them concatenated with the target input when being passed to the decoder and the decoder operates in the usual fashion.

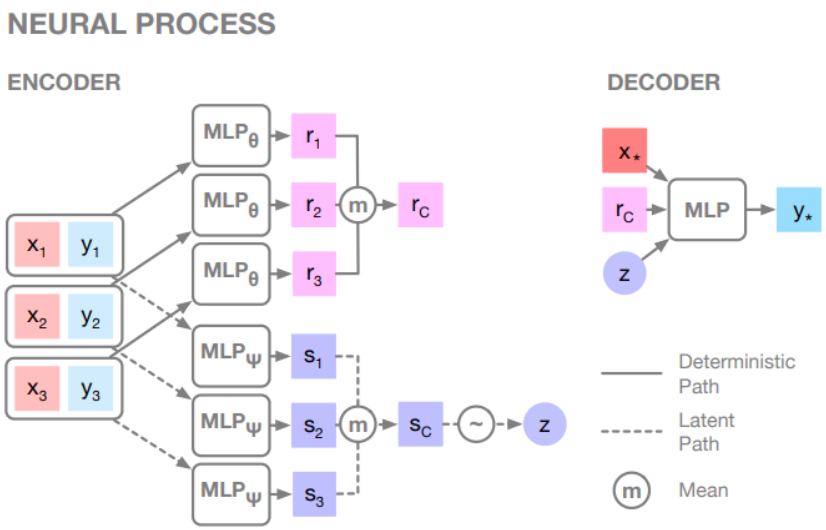


Fig. 6.1 Diagram of the inclusion of a deterministic pathway on an NP in the encoder stage. Figure adapted from Kim et al., 2019

.

Kim et al., 2019 found the inclusion of a deterministic pathway to improve NP performance. As the function of the encoder in a NP is theoretically similar to that of the CorrConvNP model, it seems reasonable to think that including a deterministic pathway on our model would also improve performance in a similar way. A consideration, however, is how to change from the MLPs of Figure 6.1 to CNNs to retain TE, as we have seen in Chapter 2 that TE requires the CNNs to operate between function spaces rather than vectors.

Bibliography

Araujo, André, Wade Norris, and Jack Sim (2019). "Computing Receptive Fields of Convolutional Neural Networks". In: *Distill*. https://distill.pub/2019/computing-receptive-fields. DOI: 10.23915/distill.00021.

Arfken, George B., Hans J. Weber, and Frank E. Harris (2013). "Chapter 1 - Mathematical Preliminaries". In: *Mathematical Methods for Physicists (Seventh Edition)*. Ed. by George B. Arfken, Hans J. Weber, and Frank E. Harris. Seventh Edition. Boston: Academic Press, pp. 1–82. ISBN: 978-0-12-384654-9. DOI: https://doi.org/10.1016/B978-0-12-384654-9.00001-3.

Bates, Douglas (Oct. 2010). *Statistics 849 notes*.

Bithas, Petros et al. (Jan. 2007). "Distributions involving correlated generalized gamma variables". In.

Brown, Tom et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Bruinsma, W. (2022). *neuralprocesses*. https://github.com/wesselb/neuralprocesses.

Bruinsma, Wessel et al. (2021). "The Gaussian Neural Process". In: *Third Symposium on Advances in Approximate Bayesian Inference*.

Copernicus Climate Change Service (2021). *Nordic gridded temperature and precipitation data from 1971 to present derived from in-situ observations*. DOI: 10.24381/CDS.E8F4A10C. URL: https://cds.climate.copernicus.eu/doi/10.24381/cds.e8f4a10c.

Donald House and John C Keyser (Dec. 2016). *Foundations of physically based modeling and animation*. Oakville, MO: Apple Academic Press.

Dzupire, Nelson Christopher, Philip Ngare, and Leo Odongo (2018). "A Poisson-Gamma Model for Zero Inflated Rainfall Data". In: *Journal of Probability and Statistics* 2018,

pp. 1–12. DOI: 10.1155/2018/1012647. URL: https://doi.org/10.1155%2F2018%2F1012647.

Fechner, G.T. (1860). *Elemente der Psychophysik*. Elemente der Psychophysik v. 1. Breitkopf und Härtel.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (Aug. 2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1126–1135.

Foong, Andrew Y. K. et al. (2020). "Meta-Learning Stationary Stochastic Process Prediction with Convolutional Neural Processes". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc. ISBN: 9781713829546.

Garnelo, Marta, Dan Rosenbaum, et al. (June 2018). "Conditional Neural Processes". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1704–1713. URL: https://proceedings.mlr.press/v80/garnelo18a.html.

Garnelo, Marta, Jonathan Schwarz, et al. (2018). "Neural Processes". In: *CoRR* abs/1807.01622. arXiv: 1807.01622.

Gordon, Jonathan et al. (2020). "Convolutional Conditional Neural Processes". In: *International Conference on Learning Representations*.

Hospedales, T. M. et al. (May 2021). "Meta-Learning in Neural Networks: A Survey". In: *IEEE Transactions on Pattern Analysis Machine Intelligence* 01, pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3079209.

Kim, Hyunjik et al. (2019). "Attentive Neural Processes". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=SkE6PjC9KX.

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun.

Kondor, Risi and Shubhendu Trivedi (Oct. 2018). "On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups". In: *Proceed-*

ings of the 35th International Conference on Machine Learning. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 2747–2755.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al., pp. 6402–6413. URL: https://proceedings.neurips.cc/paper/2017/hash/9ef2ed4b7fd2c810847ffa5fa85bce38-Abstract.html.

Lambert, Diane (1992). "Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing". In: *Technometrics* 34.1, pp. 1–14. ISSN: 00401706. URL: http://www.jstor.org/stable/1269547 (visited on 07/29/2022).

Le, Tuan Anh (2018). "Empirical Evaluation of Neural Process Objectives". In.

Markou, Stratis et al. (2022). *Practical Conditional Neural Processes Via Tractable Dependent Predictions*. DOI: 10.48550/ARXIV.2203.08775.

Max, A Woodbury (1950). "Inverting modified matrices". In: *Memorandum Rept. 42, Statistical Research Group*. Princeton Univ., p. 4.

Mena, José, Oriol Pujol, and Jordi Vitrià (Oct. 2021). "A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective". In: *ACM Comput. Surv.* 54.9. ISSN: 0360-0300. DOI: 10.1145/3477140. URL: https://doi.org/10.1145/3477140.

Murphy, Kevin P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

Pishro-Nik, Hossein (Aug. 2014). *Introduction to probability, statistics, and random processes*. Kappa Research.

Promislow, S David (Dec. 2014). *Fundamentals of actuarial mathematics*. en. 3rd ed. Nashville, TN: John Wiley & Sons.

Rasmussen, Carl Edward and Christopher K. I. Williams (2006). *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, pp. I–XVIII, 1–248. ISBN: 026218253X.

Ravanbakhsh, Siamak, Jeff Schneider, and Barnabas Poczos (2017). "Equivariance through parameter-sharing". In: *International conference on machine learning*. PMLR, pp. 2892–2901.

Ravi, Sachin and Alex Beatson (2019). "Amortized Bayesian Meta-Learning". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rkgpy3C5tX.

Requeima, James et al. (2019). "Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc.

Ronneberger, O., P.Fischer, and T. Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, pp. 234–241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

Silver, David et al. (Jan. 2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

Wagstaff, Edward et al. (2019). "On the Limitations of Representing Functions on Sets". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6487–6494. URL: http://proceedings.mlr.press/v97/wagstaff19a.html.

Ye, L. et al. (2018). "The probability distribution of daily precipitation at the point and catchment scales in the United States". In: *Hydrology and Earth System Sciences* 22.12, pp. 6519–6531. DOI: 10.5194/hess-22-6519-2018.

Zaheer, Manzil et al. (2017). "Deep Sets". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

Appendix A

Model Specifications and Training

Regression Experiments:

Model	Network Name	Channels	Num. Parameters
ConvCNP	U-Net[1]	(64, 64, 64, 128, 128, 128, 256)	1,108,164
ConvGNP	U-Net	(64, 64, 64, 128, 128, 128, 256)	1,116,484
ConvNP	U-Net	(64, 64, 64, 128, 128, 128, 256)[2]	2,222,756
CorrConvNP	U-Net	(64, 64, 64, 128, 128, 128, 256)	2,289,316

Table A.1 Details of the architectures of the convolutional networks used in the regression experiments of Chapter 3.

Bernoulli Classifier:

Model	Network Name	Channels	Num. Parameters
CorrConvNP (1D input)	U-Net	(32, 32, 32, 32, 32, 32)	202,371
ConvNP (1D input)	U-Net	(32, 32, 32, 32, 32, 32)	165,415
ConvCNP (1D input)	U-Net	(32, 32, 32, 32, 32, 32)	82,659
CorrConvNP (2D input)	U-Net	(32, 32, 32, 32, 32, 32)	869,251
ConvNP (2D input)	U-Net	(32, 32, 32, 32, 32, 32)	823,335
ConvCNP (2D input)	U-Net	(32, 32, 32, 32, 32, 32)	411,619

Table A.2 Details of the architectures of the CorrConvNP, ConvNP, and ConvCNP models used in the 1D and 2D binary Mixture of Gaussians and GP-Cutoff classification experiments of Chapter 4.

[1] Details of the U-Net architecture can be found in the original paper, (Ronneberger, P.Fischer, and Brox, 2015).

[2] These channels were the same for both the encoder and decoder for the LV models.

Gamma Process Regression:

Model	Network Name	Channels	Num. Parameters
CorrConvNP	U-Net	(32, 32, 32, 32, 32, 32)	881,408
ConvNP	U-Net	(32, 32, 32, 32, 32, 32)	835,492
ConvCNP	U-Net	(32, 32, 32, 32, 32, 32)	411,652

Table A.3 Details of the architectures of the CorrConvNP, ConvNP, and ConvCNP models used in the 2D Gamma Process regression experiment of Chapter 4.

Rainfall Prediction Experiment:

Model	Network Name	Channels	Num. Parameters
CorrConvNP	U-Net	(32, 32, 32, 32, 32, 32)	870,918

Table A.4 Details of the architectures of the CorrConvNP model used in the rainfall modeling task in Chapter 5.

Training was done for all experiments on a single Nvidia Tesla K80 GPU, with a batch size of 16 batches per epoch. For 1D regression and classification experiments the best-found learning rate was $3 \cdot 10^{-4}$, but for the 2D extended likelihood experiments, $1 \cdot 10^{-3}$ was found to perform better. All training was done with ADAM optimisation. For the regression experiments the discretisation was set to 16 points per unit, and for the classification and Gamma likelihood experiments it was set to 1 point per unit, to account for the order of magnitude increase in domain size.

Training times on this GPU varied from less than a minute per epoch for the regression experiments with the conditional models (approximately an hour and a half to train 100 epochs), to around 4 minutes per epoch for the regression experiments with the CorrConvNP (approximately 7 hours to train 100 epochs).