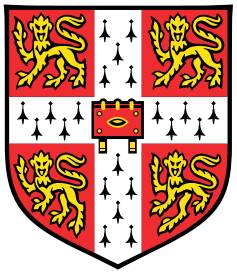
Autoregressive Diffusion Neural Processes



Lorenzo Bonito

Department of Engineering
University of Cambridge

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

St John's College August 2023

Alla mia famiglia, e ad amici vecchi e nuovi.

Declaration

I, Lorenzo Bonito of St John's 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.

The software utilised in this project builds upon the `neuralprocesses` library [Bruinsma et al., 2023a], to which a significant amount of code has been added to implement the models and experiments described in Chapters 3 and 4 below (the reader should refer to the author's fork[1] of the above GitHub repository for a detailed account of said additions).

This dissertation contains 14600 words including tables, footnotes, captions, equations and abstract, but excluding declarations, bibliography, photographs and diagrams.

Lorenzo Bonito
August 2023

[1] https://github.com/lorenzobonito/neuralprocesses

Acknowledgements

I would firstly like to thank my supervisor, Professor Richard Turner, for proposing the central idea upon which this project is based, for his invaluable guidance and support throughout my research endeavours, and for his endless enthusiasm, which has constituted a constant source of inspiration over the past four months.

I would also like to thank my co-supervisors, James Requeima and Aliaksandra Shysheya, for sharing their insight and knowledge, ultimately enabling me to take the project in the right direction.

Lastly, a big, heartfelt thank you goes to Nicola, Gergő, Avery and Ilaria, without whom life in Cambridge would simply not have been quite the same.

Abstract

Over the last two decades, machine learning research has made enormous strides, enabling the scientific community (and, more recently, the general population) to tackle tasks which, just at the beginning of the millennium, were deemed to lie outside of the realm of possibility. This success is due, in large part, to advances in the area of deep learning (DL), which, through ever larger models trained on ever larger datasets, has attained astounding performance in the most disparate of fields.

These achievements notwithstanding, many circumstances still exist in which DL models cannot successfully be deployed. These include, amongst others, applications where data is scarce, or which require accurate uncertainty estimates to be provided alongside the generated predictions. This thesis focuses precisely on these scenarios, and, more specifically, on models that are especially well-suited to operating within them: Neural Processes (NPs).

In particular, the project work presented in this document proposes and develops a novel framework for NPs which, by drawing inspiration from the diffusion literature, aims to address the main shortcomings affecting present state-of-the-art approaches. The resulting models (termed Autoregressive Diffusion Neural Processes) are trained and tested on a variety of tasks, and their performance is shown to be quite impressive, in many cases significantly outclassing that attained by even the best currently available alternatives.

Table of contents

List of figures and tables

Chapter 1

Introduction

1.1 Overview and Motivation

When examining technological progress over the past decade, few advances measure up, in both scope and significance, to those made in the area of machine learning. Unarguably, the main driving force behind this AI Renaissance has been deep learning, which has, time and again, proved to be an invaluable tool in the most disparate of fields. Indeed, modern deep neural networks have been able to all but exhaustively solve a variety of extremely complex tasks within an impressively diverse multitude of subject areas, ranging from image recognition [Krizhevsky et al., 2012] and natural language processing [Vaswani et al., 2017] to game playing [Silver et al., 2016] and bioinformatics [Jumper et al., 2021].

Despite this premise, one would be remiss to think that these now near-ubiquitous approaches are capable of successfully handling any challenge presented to them. In fact, data for deep learning applications is usually required to be highly structured, and, more importantly, abundant [Goodfellow et al., 2016], and one need not stray far from day-to-day life to identity numerous circumstances in which neither of these prerequisites is (or can be) met. What is more, even when the conditions for employing these models are satisfied, the resulting predictions usually lack any form of uncertainty estimate [Gal and Ghahramani, 2016], which greatly hinders their applicability in many real-life scenarios. To illustrate the above points, let us imagine that an algorithm were tasked with inferring information about a patient's health based on a small set of biophysical measurements: said collection of data would certainly not be big enough to train a deep neural network, and, even if it were, any doctor

would likely want a measure of the predictor's confidence before administering life-changing treatment based on its outputs.

As the reader might have already inferred from the title of this dissertation, Neural Processes constitute a highly effective approach in addressing both of the above issues. Indeed, NPs are specially designed to operate in the small data regime, and also deliver reliable uncertainty estimates alongside their predictions. Thanks to these desirable properties, these models have, since their original conception [Garnelo et al., 2018a,b], garnered an increasing amount of popularity, with multiple improvements on the original formula having been developed throughout the past 5 years.

As discussed and demonstrated at length throughout Chapter 2, each new NP variation generally aims to address its predecessors' deficiencies (be these related to modelling performance, computational costs, or a combination of the two), but usually also comes with its own, new set of limitations. For example, Latent Neural Processes [Garnelo et al., 2018b] were designed to utilise latent variables to model statistical dependencies in their outputs (in contrast to Conditional Neural Processes [Garnelo et al., 2018a], which make predictions at different input locations independently). This alteration, however, renders the marginal likelihood intractable, thereby making these models more challenging to train [Le et al., 2018]. Analogously, Gaussian Neural Processes [Bruinsma et al., 2021; Markou et al., 2021] do away with said latent variables, simplifying training whilst still offering correlated predictions, but their intrinsic assumption of a Gaussian predictive is often unsuitable for practical applications (where multi-modal, heavy-tailed or otherwise non-Gaussian predictives are often optimal). Lastly, Autoregressive Conditional Neural Processes [Bruinsma et al., 2023b] (the latest addition to and current state of the art in the NP Family) also follow a similar pattern: whilst predictives far down the autoregressive chain do achieve higher complexity than those produced by GNPs, variables generated early on in the autoregressive process are, regrettably, still limited to elementary Gaussian distributions.

In line with the above trend, the work described in this thesis proposes, implements and evaluates a new NP variant which aims to overcome this intrinsic weakness of AR CNPs: namely, Autoregressive Diffusion Neural Processes. Drawing inspiration from the diffusion literature [Ho et al., 2020; Sohl-Dickstein et al., 2015], AR DNPs are able, through conditioning on various noisy data sources, to model complex, highly dependent and highly general distributions at *every* input location, rather than yielding simpler predictives for some subset of these (as is the case for AR CNPs). In addition, depending on the specific architecture and learning task at hand, the proposed approach is designed to be less computationally

intensive than AR CNPs during deployment (requiring fewer forward passes). However, one should note that, as was the case for the examples outlined above, the above improvements are also coupled with some disadvantages, which should be carefully weighed when electing to employ these models under certain circumstances.

1.2 Thesis Contributions

The main contributions of this work are as follows:

- **The development of a novel framework for NP models**, referred to in this document as Autoregressive Diffusion Neural Process (AR DNP for short). As hinted at above, this new addition to the NP family aims to directly tackle some of the shortcomings plaguing Autoregressive Conditional Neural Processes, and is grounded in the findings resulting from the aforementioned literature review.
- **A meticulous evaluation of the performance of AR DNPs** across a variety of tasks, model design choices and hyperparameter settings. Since the proposed model architecture naturally inserts itself within the broader NP and meta-learning literature, comparisons between its efficacy and that of relevant, pre-existing baselines are also presented.

1.3 Thesis Outline

The remainder of the main body of this document is organised as follows:

- **Chapter 2** presents a thorough, comparative review of the main existing members of the Neural Process (NP) Family, with a specific focus on each model's advantages and drawbacks in relation to its peers. This analysis sheds light on common patterns in the literature, and highlights potential areas of improvement with respect to the current state of the art, thereby expanding upon the motivations behind this project (already hinted at in Section 1.1 above).
- **Chapter 3** explains the central idea at the basis of the proposed Autoregressive Diffusion Neural Processes, and systematically describes the techniques employed to implement and test these models.
- **Chapter 4** showcases the most relevant results obtained throughout the project, and calls attention to notable patterns in the available data to make qualitative and quanti-

tative remarks on the performance of AR DNPs compared to current state-of-the-art baselines. The influence of key hyperparameters on the model's predictive power is also explored, in an effort to render the findings easily reproducible and extensible.

- **Chapter 5** looks back upon the work as a whole, discussing its main impacts and the key conclusions that should be drawn from it. A brief summary of potential avenues of future work is also included.

Chapter 2

Background

This chapter lays the theoretical foundations upon which the remainder of this thesis shall be constructed. Firstly, Section 2.1 examines the general concept of meta-learning, inserts the models belonging to the Neural Process Family within this framework, and specifies a significant portion of the notation employed throughout the rest of this document. Secondly, Sections 2.2 to 2.5 centre around the members of the NPF which are most relevant for the work undertaken throughout the project (namely, CNPs, ConvCNPs, GNPs and AR CNPs), with a specific focus on the motivations, architectures, advantages and shortcomings of each. Lastly, Section 2.6 summarises the main findings from the Chapter, drawing systematic comparisons between the discussed models and reinforcing the motivations for the project work already outlined in Section 1.1.

2.1 Meta-Learning and the Neural Process Family

In the realm of ML, the term "meta-learning" is usually understood to refer to classes of models which focus on *learning about learning*. Given the evident looseness of this definition, it should come as no surprise to the reader that the actual meaning behind this term is usually heavily dependant upon the specific context in which one encounters it. For the purposes of this dissertation, this expression shall be used, in direct opposition to "supervised learning", to describe the behaviour of systems which, instead of being trained on and asked to make predictions about a single dataset, are exposed to a *set* of datasets, and tasked with uncovering and modelling the underlying *process* that generated said collection, in an effort to construct a flexible model which is able to easily adapt to a variety of different (albeit still similar) tasks.

To give a more technical description of the above distinction, some notation is needed. Let a supervised learning task ψ be defined as a tuple comprising of a context set $\mathcal{D}_c$ (containing the observations to condition upon when making future predictions) and a target set $\mathcal{D}_t$ (containing said future "observations"). One should note that both sets consist of a fixed number of input-output pairs (i.e. $\mathcal{D}_c = \{(x_n^{(c)}, y_n^{(c)})\}_{n=1}^{N_c}$ and $\mathcal{D}_t = \{(x_n^{(t)}, y_n^{(t)})\}_{n=1}^{N_t}$), where each member of said pair can take on an arbitrary dimensionality (for ease of notation, boldface will never be employed, unless the discussion specifically only applies to vector quantities). Additionally, the values of $y_n^{(t)}$ are, naturally, unknown, and the ultimate goal of training is to produce accurate predictions for them.

Having made this premise, in a standard supervised learning setting, one would tackle a given task $\psi = (\mathcal{D}_c, \mathcal{D}_t)$ by first fitting a parametric model f_θ to the available context data $\mathcal{D}_c$, and then employing said model to make predictions for the desired target inputs $x_n^{(t)} \in \mathcal{D}_t$. Specifically, the set of optimal model parameters θ^* would be selected in such a way as to minimise the expected value of a given loss function $\mathcal{L}$ which quantifies some "distance" (in the most general sense of the term) between the predicted context outputs $f_\theta(x_n^{(c)})$ and the true context outputs $y_n^{(c)}$:

$$\theta^* = \underset{\theta \in \Theta}{\arg\min} \left\{ \mathbb{E}_{(x_n^{(c)}, y_n^{(c)}) \sim \mathcal{D}_c} \left[\mathcal{L}\left(f_\theta(x_n^{(c)}), y_n^{(c)} \right) \right] \right\} \tag{2.1.1}$$

Armed with values for θ^*, predictions for the true, unknown target outputs $y_n^{(t)}$ are then easily obtained as $y_n^* = f_{\theta^*}(x_n^{(t)})$.

In a meta-learning setting, an analogous procedure applies, with the crucial difference that, as mentioned above, the training instead utilises a collection $\Psi = \{\psi_i\}_{i=1}^N$ of N tasks ψ generated by the same underlying process. As a result, the goal is to fit a meta-model ξ_θ which itself generates task-specific models $\tilde{f}$ given context sets $\mathcal{D}_c$ as input. Similarly to the supervised case, the optimal model parameters θ^* are still obtained by minimising the expectation of a loss function $\mathcal{L}$, which, however, now quantifies the performance of a given output model $\tilde{f} = \xi_\theta(\mathcal{D}_c)$ on a specific task ψ:

$$\theta^* = \underset{\theta \in \Theta}{\arg\min} \left\{ \mathbb{E}_{\psi_i \sim \Psi} \left[\mathcal{L}\left(\xi_\theta(\mathcal{D}_c), \psi_i \right) \right] \right\} \tag{2.1.2}$$

Crucially, after obtaining values for θ^*, predictions for the true, unknown target outputs $y_n^{(t)}$ are produced by first running the meta-model on the context set $\mathcal{D}_c$ to obtain the predictor $\tilde{f}$ specific to a given task, and then using said function to infer $y_n^* = \tilde{f}(x_n^{(t)})$ (following a slightly different procedure than for supervised learning).

In summary, in a regular supervised scenario, one considers a single task $\psi = (\mathcal{D}_c, \mathcal{D}_t)$ and employs the context set $\mathcal{D}_c$ to train a model f_θ which maps from target inputs $x_n^{(t)} \in \mathcal{D}_t$ to predictions for the unknown target outputs $y_n^{(t)} \in \mathcal{D}_t$. On the contrary, in a meta-learning approach, one utilises multiple tasks $\Psi = \{\psi_i\}_{i=1}^N$ to train a meta-model ξ_θ which maps from tasks ψ (or, specifically, context sets $\mathcal{D}_c$) to task-specialised models $\tilde{f}$.

Given meta-learning's ability to condense information originating from a variety of tasks, it is trivial to see why this technique would be particularly well-suited to handle problems in which each individual ψ only consists of few data points. Referring back to Section 1.1, this is precisely one of the two desirable properties that traditional deep learning approaches lack. To achieve the second one (the ability to produce uncertainty estimates), a little more mathematical machinery is needed.

As discussed above, meta-learning can be conceptualised as directly learning a map from context sets $\mathcal{D}_c$ to predictor functions $\tilde{f}$. Intuitively, limiting the search to a single predictor $\tilde{f}$ is needlessly restrictive: indeed, a significantly more desirable feature to aim for would be a *distribution* over predictors, from which different versions of $\tilde{f}$ could be sampled to account for the intrinsic uncertainty in this function's exact form. The resulting distribution over functions is usually referred to as a stochastic process.

Having established how to satisfy the need for uncertainty estimates, the Neural Process Family can finally be properly introduced. In line with the above, models in the NPF can simply be understood as utilising neural networks to *meta-learn a map from datasets to predictive stochastic processes*. This framing is especially useful, as it enables one to draw rigorous mathematical conclusions about NPs, and also allows one to compare the NPF to Gaussian processes (GPs), another popular ML method employed for stochastic process predictions [Dubois et al., 2020].

Before delving into a thorough analysis of the members of the NPF required to comprehend the work described in this document, let us outline some key characteristics that, unless otherwise stated, are common to all models in this category:

- Neural Processes treat the context set as a *set* (in the mathematical connotation of the word): this means that, unlike models that operate with traditional, vector-valued inputs, NPs must be able to deal with *varying-sized* inputs, and the value of their outputs must also not depend on the ordering of said inputs (a property called *permutation invariance*). These two properties must necessarily be satisfied for the corresponding NP to behave like a traditional stochastic process.

- To attain the above behaviour, Neural Processes make use of neural networks in an encoder-decoder architecture. Specifically:

  – The encoder Enc_θ maps the entire context set $\mathcal{D}_c$ to some representation R through a relation of the form:

  $$R = \text{Enc}_\theta(\mathcal{D}_c) = \varphi\left(\sum_{n=1}^{N_c} \rho\left(x_n^{(c)}, y_n^{(c)}\right)\right) \tag{2.1.3}$$

  for appropriate φ and ρ functions, whose specifications vary based on the kind of NP being considered. The summation in Equation 2.1.3 is responsible for handling varying input sizes and for guaranteeing permutation invariance.

  – The decoder Dec_θ takes in the representation R produced by Enc_θ, and generates predictive distributions p_θ at the desired target inputs $x_n^{(t)}$: $p_\theta\big(\{y_n^{(t)}\}_{n=1}^{N_t} | \{x_n^{(t)}\}_{n=1}^{N_t}, \mathcal{D}_c\big)$. Importantly, for some key models described in the following, p_θ factorises conditioned on R:

  $$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t} | \{x_n^{(t)}\}_{n=1}^{N_t}, \mathcal{D}_c\right) = \prod_{n=1}^{N_t} p_\theta\left(y_n^{(t)} | x_n^{(t)}, R\right) \tag{2.1.4}$$

  meaning the model makes independent predictions at each target location.

  – A general schematic illustrating the above points is shown in Figure 2.1.1 below.

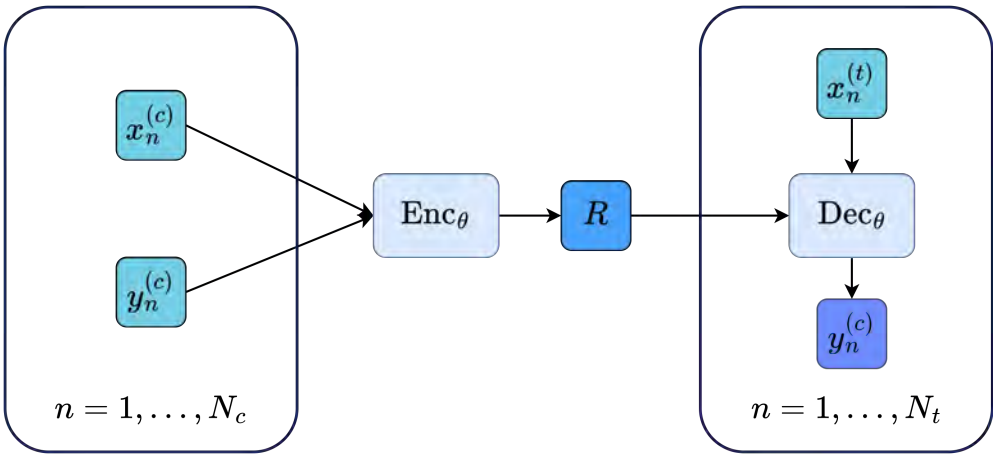


Figure 2.1.1: General schematic for models belonging to the Neural Process Family. The outer blue squircles indicate that a specific operation is repeated for each of the indices printed at the bottom.

- Since comparisons shall often be drawn between Gaussian processes and Neural Process, it shall prove useful to know that, unlike GPs, which scale with $\mathcal{O}\left((N_c+N_t)^3\right)$ [Rasmussen and Williams, 2006], NPs boast a linear running time complexity, and scale with $\mathcal{O}(N_c+N_t)$. This makes them particularly enticing for applications requiring small computational overheads and fast predictions.

Armed with the above general background on NPF models, the reader should now be ready to delve into the topics discussed in Sections 2.2 to 2.5 without much difficulty. As mentioned previously, what follows constitutes a rather complete account of the NP literature, with only a few seminal papers [Garnelo et al., 2018b; Kim et al., 2019] having been omitted as not directly related to the thesis' subject matter.

2.2 Conditional Neural Processes (CNPs)

Conditional Neural Processes were the first kind of NP model to be developed [Garnelo et al., 2018a]. As such, they hold a very special place in the literature, and are still quite relevant to date, despite having fallen substantially far behind, in terms of performance, compared to the current state of the art.

Utilising the framework discussed in Section 2.1 as a point of reference, CNPs make use of a Deep Sets [Zaheer et al., 2017] function approximator to produce a latent representation R of the context set $\mathcal{D}_c$ that satisfies the ordering- and size-invariance requirements. More specifically, the overall operation of the model's encoder can be summarised through the following expressions:

$$R_n = h_\theta\left(x_n^{(c)}, y_n^{(c)}\right), \qquad \forall (x_n^{(c)}, y_n^{(c)}) \in \mathcal{D}_c \tag{2.2.1}$$

$$R = R_1 \oplus R_2 \oplus \ldots \oplus R_{N_c} \tag{2.2.2}$$

where the function h_θ is parameterised by a neural network (in this case, a simple multilayer perceptron), and $\oplus$ indicates a commutative operator which maps a variable number of elements of $\mathbb{R}^d$ to a single element of $\mathbb{R}^d$ (where d is the chosen dimensionality for the context set representation R). In this case, $\oplus$ is set to be a simple averaging operation (though, in principle, other choices would also be viable), which yields the following functional form for the encoder Enc_θ:

$$\text{Enc}_\theta(\mathcal{D}_c) = \frac{1}{N_c}\sum_{n=1}^{N_c} h_\theta\left(x_n^{(c)}, y_n^{(c)}\right) \tag{2.2.3}$$

One should note that the above expression fits the general NPF encoder definition given in Equation 2.1.3.

Having obtained the context representation R, a decoder Dec_θ is then employed to obtain predictions at the given target locations $\{x_n^{(t)}\}_{n=1}^{N_t}$:

$$\text{Dec}_\theta(x_n^{(t)}, R) = g_\theta\left(x_n^{(t)}, R\right), \qquad \forall x_n^{(t)} \in \mathcal{D}_t \tag{2.2.4}$$

where the function g_θ is again parameterised by a neural network (a multilayer perceptron). This decoder outputs a mean μ_n and a variance σ_n^2 for each $x_n^{(t)}$, and the predictives at each target input are modelled as *independent* Gaussian distributions with said parameters:

$$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t} | \{x_n^{(t)}\}_{n=1}^{N_t}, \mathcal{D}_c\right) = \prod_{n=1}^{N_t} \mathcal{N}\left(y_n^{(t)}; \mu_n, \sigma_n^2\right) \tag{2.2.5}$$

where, collapsing all steps into one expression:

$$(\mu_i, \sigma_i^2) = g_\theta\left(x_i^{(t)}, \frac{1}{N_c}\sum_{j=1}^{N_c} h_\theta\left(x_j^{(c)}, y_j^{(c)}\right)\right) \tag{2.2.6}$$

A schematic illustrating the functioning of a CNP is presented in Figure 2.2.1 below.

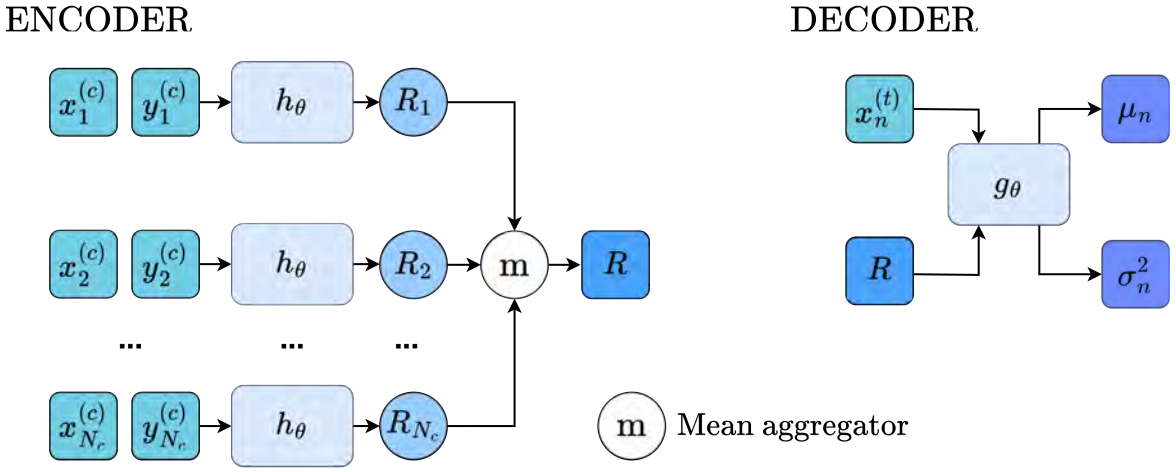


Figure 2.2.1: Schematic showcasing the functioning of a CNP model. The encoder $\text{Enc}_\theta(\mathcal{D}_c)$ is depicted on the left, and the decoder $\text{Dec}_\theta(x_n^{(t)}, R)$ is shown on the right. The reader should compare this diagram with Figure 2.1.1 to better comprehend how this kind of NP fits within the broader Neural Process Family.

With the above architecture in mind, let us now turn our attention to the training procedure required to fit a CNP to a given meta-dataset $\Psi = \{\psi_i\}_{i=1}^N$. To find optimal values θ^* for the encoder-decoder neural network parameters θ, a simple maximum likelihood objective is utilised:

$$\theta^* = \arg\max_{\theta \in \Theta} \left\{ \mathbb{E}_{\psi_i \sim \Psi} \left[\mathbb{E}_{n \in \{1,\ldots,N_t\}} \left[\log p_\theta \left(y_n^{(t)} | x_n^{(t)}, R \right) \right] \right] \right\} \tag{2.2.7}$$

This learning approach makes CNPs extremely easy to train, and, by mimicking the way the system is deployed at test time, leads to stronger overall predictive performance [Gordon et al., 2019].

Before moving on to the next kind of NP, one would be remiss not to mention that, since CNPs produce *independent* marginal predictions at each target location, they are unable to produce coherent function samples (Figure 2.2.2). This is due to the model's architecture, and, more specifically, to the fact that the decoder $\text{Dec}_\theta(x_n^{(t)}, R)$ outputs a single value of σ_n^2 for each target input $x_n^{(t)}$ (which means that the joint Gaussian predictive is parameterised by a diagonal covariance matrix). This deficiency often makes these models not viable for downstream estimation tasks (e.g. flood prediction), and has been addressed in a variety of ways, with Latent Neural Processes [Garnelo et al., 2018b] and Gaussian Neural Processes [Bruinsma et al., 2021; Markou et al., 2021] constituting the two most notable examples.

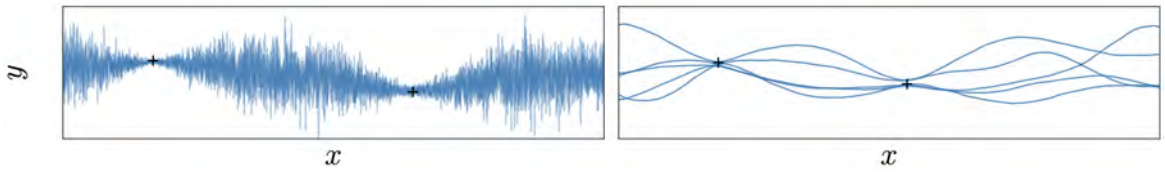


Figure 2.2.2: Samples drawn from a (Convolutional) Conditional Neural Process (left) and a (Convolutional) Gaussian Neural Process (right). Because of their architectures, CNPs yield independent predictions at each target locations, which prevents them from producing *coherent* function samples. Image sourced from Markou et al. [2022].

Lastly, one should also mention that, whilst CNPs' performance is generally satisfactory across a variety of tasks, these models regrettably tend to consistently underfit the context set $\mathcal{D}_c$. As hypothesised in Kim et al. [2019], this behaviour is likely due to the mean aggregation step, which, by giving the same weight to each context point, makes it significantly harder for the decoder to learn which specific elements of $\mathcal{D}_c$ provide relevant information for predictions at a given target location $x_n^{(t)}$. As suggested in the same paper [Kim et al., 2019], one way of addressing this shortcoming is to introduce an attention mechanism in the NP's architecture, which, however, increases complexity from $\mathcal{O}(N_c + N_t)$ to $\mathcal{O}(N_c(N_c + N_t))$.

2.3 Convolutional Conditional NPs (ConvCNPs)

Having thoroughly described the architecture, advantages and shortcomings of a traditional Conditional Neural Process, we shall now turn our attention to models which build upon said design to attain either altogether improved performance or otherwise desirable predictive properties. As suggested by the above title, this Section focuses on Convolutional Conditional Neural Process [Gordon et al., 2020], which achieve both of these goals by endowing CNPs with *translation equivariance*.

Intuitively, a model is said to be translation equivariant if a given translation in its inputs results in a similar translation in the corresponding prediction. For clear reasons, this is a crucial inductive bias for many learning tasks (e.g. time series modelling, spatial data, and images).

In the case of NPs, this characteristic essentially means that, if a given context set $\mathcal{D}_c = \{(x_n^{(c)}, y_n^{(c)})\}_{n=1}^{N_c}$ is mapped to function samples $\tilde{f}$, then the translated context set $\mathcal{D}_c = \{(x_n^{(c)} + k, y_n^{(c)})\}_{n=1}^{N_c}$ should be mapped to the same function samples $\tilde{f}$ translated by a factor of k.

To attain this property, ConvCNPs broaden the work carried out by Zaheer et al. [2017] and introduce Convolutional Deep Sets, directly extending the theory of neural representations of sets to include *functional representations*. This is necessary because, as discussed above, CNP encoders map context sets to an embedding R in a vector space $\mathbb{R}^d$, in which equivariance with respect to translations in the inputs is not well-defined. If, however, the encoder is modified to map to a *function*, an unambiguous definition of translation equivariance is recovered. Hence, ConvCNPs make predictions in the same way a regular CNP does:

$$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t} | \{x_n^{(t)}\}_{n=1}^{N_t}, \mathcal{D}_c\right) = \prod_{n=1}^{N_t} \mathcal{N}\left(y_n^{(t)}; \mu_n, \sigma_n^2\right) \tag{2.3.1}$$

with the important difference that, in this new scenario,

$$\left(\mu_i, \sigma_i^2\right) = \Phi_\theta(\mathcal{D}_c)\left(x_i^{(t)}\right) \tag{2.3.2}$$

where Φ_θ is a ConvDeepSet. More specifically, the ConvDeepSet function $\Phi_\theta(\mathcal{D}_c)$ can be defined as the following function composition:

$$\Phi_\theta(\mathcal{D}_c) = \rho(E(\mathcal{D}_c)) \tag{2.3.3}$$

where $E(\mathcal{D}_c)$ is an encoder that maps the context set $\mathcal{D}_c$ into a functional space and ρ is a decoder which acts as a translation equivariant map between two function spaces. In particular:

- The encoder $E(\mathcal{D}_c)$ takes the following form:

$$E(\mathcal{D}_c) = \sum_{n=1}^{N_c} \phi\left(y_n^{(c)}\right) \lambda\left(\cdot - x_n^{(c)}\right) \tag{2.3.4}$$

  In the above expression, λ denotes a positive-definite kernel associated with a Reproducing Kernel Hilbert Space. This kernel is responsible for E's desirable properties, and is customarily set to an exponentiated-quadratic (EQ) form with a learnable length scale parameter. $\phi(y_n)$, on the other hand, takes the form $\left(y_n^0, y_n^1, \ldots, y_n^K\right)$, where K is a crucial parameter which indicates how many outputs are associated with each input location (multiplicity). The first output ϕ_1 provides the model with information concerning the locations at which data has been observed, and its functional representation h is often referred to as "density channel".

- The decoder ρ is parameterised by a Convolutional Neural Network. Since CNNs are only able to handle discrete (on-the-grid) input and output spaces, the specific architecture of ρ depends on whether the data one is interested in modelling is on-the-grid or off-the-grid. For the purposes of this dissertation, only the latter scenario is relevant, and the former shall not be discussed in detail (although one should note that similar considerations to those made below still generally apply). For off-the-grid (continuous) data, one must first discretise the input to ρ, then pass it through the CNN, and, lastly, map the CNN's output back to a continuous function. An illustrated summary of these steps is presented in Figure 2.3.1 below.

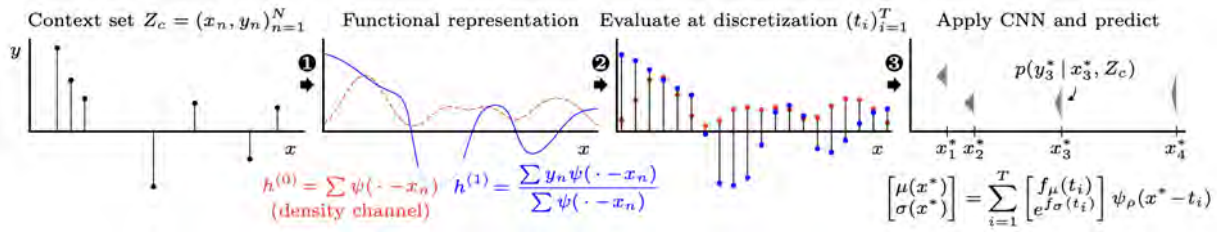


Figure 2.3.1: Illustration of the key steps in the ConvCNP forward pass for off-the-grid data. Image sourced from Gordon et al. [2020].

Much like the regular CNP, the ConvCNP is also trained via a maximum likelihood approach similar to that already formalised in Equation 2.2.7 (although some slight modifications are

introduced to account for the modified architecture). Additionally, for the same reasons discussed in Section 2.2 for the CNP, the ConvCNP also produces *independent* predictions at each target point (as evidenced by the factorised distribution on the right-hand side of Equation 2.3.1), meaning coherent function samples cannot be drawn.

A schematic illustrating the functioning of a ConvCNP is presented in Figure 2.3.2 below.

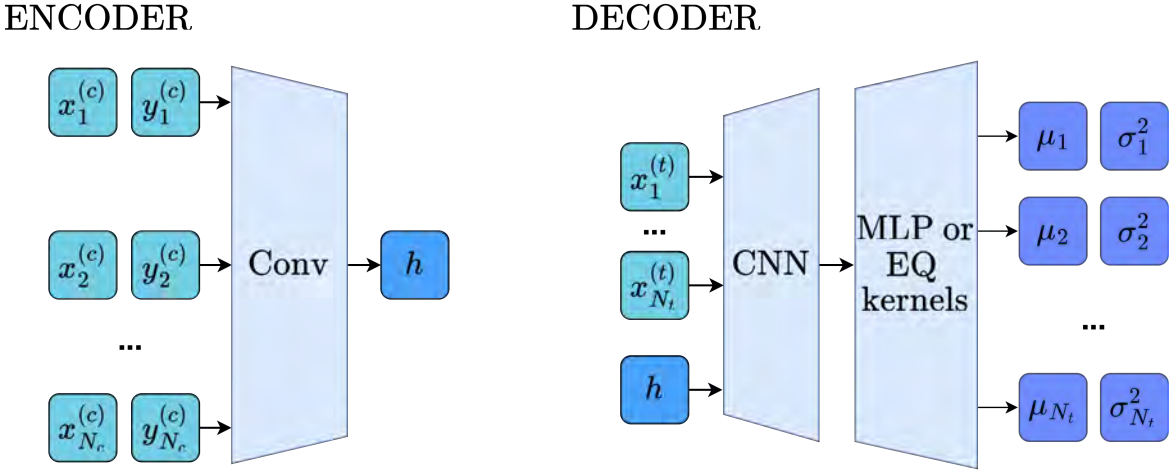


Figure 2.3.2: Schematic showcasing the functioning of a ConvCNP model. The encoder $E(\mathcal{D}_c)$ is depicted on the left, and the decoder ρ is shown on the right. One should note that, in the case of off-the-grid data, EQ kernels are employed in the decoder, whilst MLPs are utilised in the opposite scenario. The reader should, once again, compare this diagram with Figure 2.1.1 to better comprehend how this type of NP fits within the NPF.

2.4 Gaussian Neural Processes (GNPs)

As already discussed at length in both Section 2.2 and Section 2.3, traditional Conditional Neural Processes (as well as their Convolutional variant) are unable to produce coherent function samples, owing to the factorised form taken by their joint predictives (Equations 2.2.5 and 2.3.1). As also previously mentioned, two main ways of circumventing this issue have been explored in the literature published to date:

- Utilising the context set representation R to define a global latent variable $\mathbf{z} \sim p_\theta(\mathbf{z}|R)$ conditioned on which the predictive distribution is then factorised (as opposed to the latter being factorised conditioned on R directly). This approach (first proposed in

Garnelo et al. [2018b]) is then characterised by the following predictive:

$$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t}|\{x_n^{(t)}\}_{n=1}^{N_t},\mathcal{D}_c\right)=\int\prod_{n=1}^{N_t}p_\theta\left(y_n^{(t)}|x_n^{(t)},\mathbf{z}\right)p_\theta(\mathbf{z}|R)\,\mathrm{d}\mathbf{z} \tag{2.4.1}$$

which, regrettably, results in an intractable likelihood, requiring the model to be trained via approximate methods such as amortised Variational Inference and significantly increasing complexity compared to utilising a simple maximum likelihood objective.

- Modifying the NP architecture so that the decoder also outputs the covariance between any two points in the target set (rather than just a single variance at each location, as in the case of CNPs and ConvCNPs). This is the technique employed by Gaussian Neural Processes [Bruinsma et al., 2021; Markou et al., 2021], discussed in more detail throughout the remainder of this Section.

In line with the above, a GNP's predictive distribution may be expressed as follows:

$$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t}|\{x_n^{(t)}\}_{n=1}^{N_t},\mathcal{D}_c\right)=\mathcal{N}\left(\{y_n^{(t)}\}_{n=1}^{N_t};m\left(\{x_n^{(t)}\}_{n=1}^{N_t},\mathcal{D}_c\right),k\left(\{x_n^{(t)}\}_{n=1}^{N_t},\mathcal{D}_c\right)\right) \tag{2.4.2}$$

where m and k are functions of an appropriate form that yield, respectively, a mean vector $\mathbf{m}$ and a *full* covariance matrix $\mathbf{K}$ (as opposed to the CNP case, in which the equivalent of $\mathbf{K}$ would be restricted to being diagonal).

To implement the mean map m, a ConvDeepSet (already described in Section 2.3) is used. For the kernel map k, however, this approach is not directly viable, and the authors of Bruinsma et al. [2021] thus modify this architecture as follows. Since covariance functions are maps $\mathcal{X}\times\mathcal{X}\to\mathbb{R}$, for discrete $\mathcal{X}=\{1,\ldots,N\}$ they can be conceptualised as "images" of sorts. As such, whilst the ConvCNP produces a prediction for the mean by embedding data into a 1D array and passing it through 1D convolutions, k generates a prediction for a covariance matrix by embedding data into a *2D image* and passing it through 2D convolutions. More formally, this happens in three separate steps:

1. An encoder maps the context set $\mathcal{D}_c$ to an encoding $\mathbf{H}\in\mathbb{R}^{M\times M\times 3}$ at a pre-specified grid $\mathbf{Z}\in\mathbb{R}^{M\times M}$ (for a given $M\in\mathbb{N}$):

$$\mathbf{H}=\mathrm{Enc}(\mathcal{D}_c,\mathbf{Z}) \tag{2.4.3}$$

Through $\mathbf{Z}$, this step discretises the input so that convolutions can later be applied. $\mathbf{H}$ comprises three channels (respectively, data, density and source), only the first two of which have a ConvDeepSet counterpart.

2. The encoding $\mathbf{H}$ is passed through a CNN (producing an $M \times M$ matrix), and the output is projected, through the operator $\Pi^{\text{p.s.d.}}$, onto the nearest positive semi-definite matrix $\mathbf{K}$ (with respect to the Frobenius norm):

$$\mathbf{K} = \Pi^{\text{p.s.d.}} \text{CNN}(\mathbf{H}) \tag{2.4.4}$$

This step is crucial because, naturally, a valid covariance matrix must be PSD. One should mention that, in Bruinsma et al. [2021], the operation in Equation 2.4.4 above is actually not carried out, and positive semi-definiteness is, instead, enforced through the simpler transformation $\mathbf{K} = \text{CNN}(\mathbf{H})\,\text{CNN}(\mathbf{H})^\top$.

3. Lastly, the obtained PSD matrix $\mathbf{K}$ is interpolated, through a decoder, to the desired covariances $\mathbf{K}^{(t)}$ corresponding to the given target inputs $\{x_n^{(t)}\}_{n=1}^{N_t}$:

$$\mathbf{K}^{(t)} = \text{Dec}\left(\mathbf{K}, \{x_n^{(t)}\}_{n=1}^{N_t}\right) \tag{2.4.5}$$

Regrettably, whilst theoretically sound, the above approach is quite costly and, additionally, not particularly viable when handling data whose input dimensionality is larger than 1. This is because, for a given D-dimensional dataset, this technique requires one to carry out $2D$-dimensional convolutions, which, for $D > 1$, are often computationally expensive and poorly supported. For this reason, Markou et al. [2021] proposed a different GNP implementation, in which the elements of the mean vector $\mathbf{m}$ and covariance matrix $\mathbf{K}$ are parameterised as follows:

$$\mathbf{m}_i = f\left(x_i^{(t)}, R\right) \tag{2.4.6}$$

$$\mathbf{K}_{ij} = k\left(g\left(x_i^{(t)}, R\right), g\left(x_j^{(t)}, R\right)\right) \tag{2.4.7}$$

where f and g are neural networks, R is some latent encoding of the context set $\mathcal{D}_c$ mediated by a neural network r, and k is a carefully selected positive-definite function (the authors suggest either a linear covariance $\mathbf{K}_{ij} = g(x_i^{(t)}, R)^\top g(x_j^{(t)}, R)$ or a so-called "kvv" covariance $\mathbf{K}_{ij} = k(g(x_i^{(t)}, R), g(x_j^{(t)}, R))v(x_i^{(t)}, R)v(x_j^{(t)}, R)$, in which k is an EQ kernel and v is another neural network). A GNP constructed with this approach can easily be trained via maximum likelihood (Equation 2.2.7) to find appropriate parameters for the networks r, f and g.

Before proceeding, one should note that, unless specifically stated otherwise, any future mention of GNPs (or their convolutional variants ConvGNPs) in the remainder of this document shall be in direct reference to this second, more efficient architecture. Additionally, one should also highlight that, whilst undoubtedly highly performant under most circumstances, GNPs are still limited in terms of the functional classes that they are able to successfully model, being significantly restricted by the assumption of a *Gaussian* predictive (Equation 2.4.2).

2.5 Autoregressive Conditional NPs (AR CNPs)

Having thoroughly examined a variety of older pieces of NP literature, let us now turn our attention towards the latest (and, by many relevant metrics, best-performing) addition to the Neural Process Family: Autoregressive Conditional Neural Processes [Bruinsma et al., 2023b].

As already briefly touched upon in Section 1.1, AR CNPs were developed in an effort to retain the advantages of GNPs (i.e. dependent predictions without the need for latent variables, intractable likelihoods and consequent approximate training) whilst doing away with the aforementioned limiting assumption of a Gaussian predictive (Figure 2.5.1).

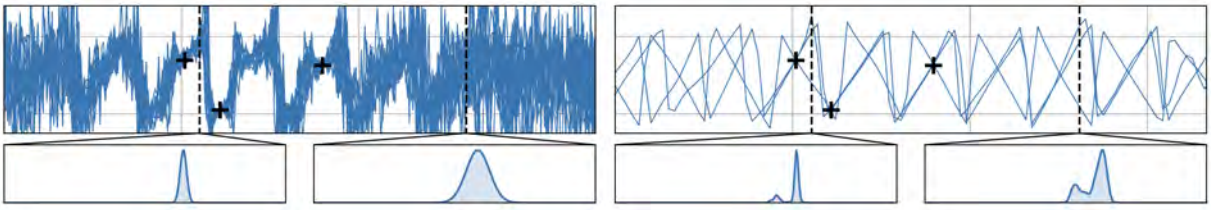


Figure 2.5.1: Outputs of a ConvCNP trained on sawtooth functions and deployed in standard mode (left) and in autoregressive mode (right). The black crosses indicate context points, the blue curves show model samples, and the bottom plots depict the marginal predictives at the target locations highlighted by the dashed lines. As evidenced by these results, the standard CNP models each output with an independent Gaussian, while the AR CNP is able to produce coherent samples, alongside *multi-modal*, *non-Gaussian* predictives. Image sourced from Bruinsma et al. [2023b].

Interestingly, AR CNPs achieve these impressive feats not by introducing new elements in the established NP architectures or training procedures, but rather by simply altering the way in which these models are deployed at test time. More specifically, the joint predictive is defined as a product of conditionals, where each factor is modelled with a CNP whose

effective context set consists of a union of the original context set $\mathcal{D}_c$ *and* the predictions already generated at previous target locations. For example, for the case of three target points:

$$\begin{aligned} p_\theta\left(\{y_n^{(t)}\}_{n=1}^3 | \{x_n^{(t)}\}_{n=1}^3, \mathcal{D}_c\right) = & p_\theta\left(y_1^{(t)} | x_1^{(t)}, \mathcal{D}_c\right) \cdot \\ & p_\theta\left(y_2^{(t)} | x_2^{(t)}, \{(x_1^{(t)}, y_1^{(t)})\}, \mathcal{D}_c\right) \cdot \\ & p_\theta\left(y_3^{(t)} | x_3^{(t)}, \{(x_1^{(t)}, y_1^{(t)})\}, \{(x_2^{(t)}, y_2^{(t)})\}, \mathcal{D}_c\right) \end{aligned} \tag{2.5.1}$$

or, for the more general case of N_t target points:

$$p_\theta\left(\{y_n^{(t)}\}_{n=1}^{N_t} | \{x_n^{(t)}\}_{n=1}^{N_t}, \mathcal{D}_c\right) = \prod_{n=1}^{N_t} p_\theta\left(y_n^{(t)} | x_n^{(t)}, \mathcal{D}_c \cup \{(x_j^{(t)}, y_j^{(t)})\}_{j=1}^{n-1}\right) \tag{2.5.2}$$

where, once again, the model's previous predictions are fed back into it in an autoregressive fashion by adding to the context set to generate each new conditional factor. Crucially, this AR procedure yields correlated, coherent samples even when p_θ is not trained to model dependencies between the target outputs (as is the case with a CNP).

Albeit extremely performant (constituting the indisputable state of the art under most circumstances), AR CNPs still present three main issues which somewhat limit their widespread applicability:

- As evidenced by Figure 2.5.2 below, the predictions generated early on in the autoregressive process are still restricted to being *Gaussian* (or near-Gaussian).

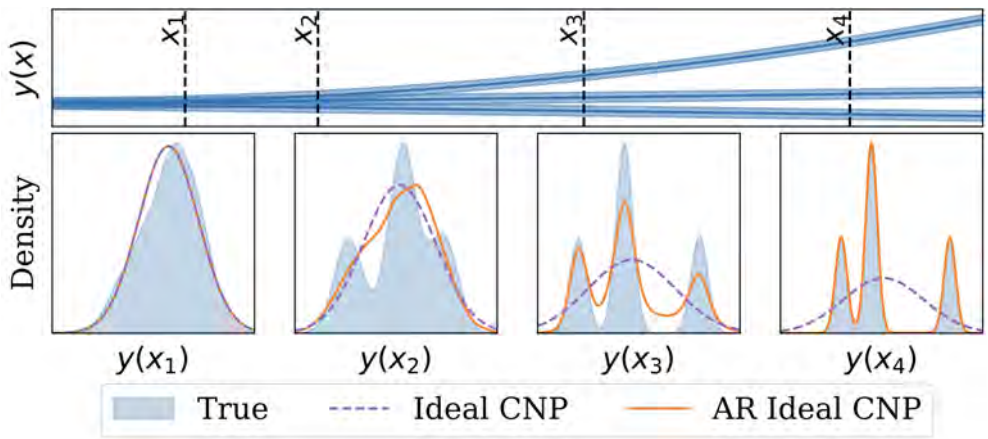


Figure 2.5.2: Mixture model of three deterministic functions with additive Gaussian noise (top) and true and predicted distributions at the four target locations denoted by dashed lines (bottom). Early AR CNP predictions remain close to normal distributions (or, in the case of $y(x_1)$, are exactly normally distributed). Image sourced from Bruinsma et al. [2023b].

- Since each prediction in the AR process is itself conditioned upon earlier predictions, the order with which one elects to sweep through the target locations necessarily influences the form of the final output. Since the final joint is not permutation- or marginalisation-invariant, AR CNPs renounce the fundamental property of *consistency*. This violates the central conditions of the Kolmogorov extension theorem [Øksendal, 2014], and prevents the AR predictions from defining consistent stochastic processes. In turn, this greatly expands the design space one must consider when employing AR CNPs, as factors that usually do not affect the outputs of other NPs can now negatively impact performance if overlooked.

- AR CNP deployment is significantly more computationally expensive than its regular counterpart, requiring N_t forward passes to complete: when sampling targets on a very fine grid, this can become prohibitively costly (or even unstable, if the number of target points is much greater than the size of the context sets encountered during training).

2.6 Summary and Models Comparison

A comparative summary of the models examined throughout Chapter 2 is presented in Table 2.6.1 below. The NP approach proposed in this document is also included in the last column. As highlighted by the two rightmost columns, the work described in this document ultimately aims to address the three AR CNP shortcomings listed in Section 2.5. That is, in an ideal scenario, AR DNPs would satisfy the following three conditions (which AR CNPs fail to meet):

- Be able to model complex, highly dependent and highly non-Gaussian predictions at *every* target location $\{x_n^{(t)}\}_{n=1}^{N_t}$, rather than yielding simple normal (or normal-adjacent) distributions for low n.

- Be relatively *cheap to deploy* at test time, even if at the price of increased computational costs during training.

- Produce *consistent* predictions which do not depend on the specific ordering of the target set $\mathcal{D}_t$, and that are invariant to marginalisation (in a bid to satisfy the conditions of the Kolmogorov extension theorem [Øksendal, 2014], which is necessary to ensure that AR DNPs define valid stochastic processes). One should note that, as shall be discussed in greater detail in Section 3.2, this condition being met hinges on a specific design choice relating to the input locations selected when generating the augmented

training dataset (and, consequently, on the design of the corresponding AR deployment procedure).

	CNPs	LNPs	GNPs	AR CNPs	AR DNPs (this work)
Correlated predictions	✗	✓	✓	✓	✓
Exact training	✓	✗	✓	✓	✓
Non-Gaussian predictions	✓	✓	✗	✓*	✓
Consistent predictions	✓	✓	✓	✗	✓*
Cheap deployment	✓	✓	✓	✗	✓

Table 2.6.1: Comparison of the main current NP models and the AR DNP (proposed in this project) across a variety of key benchmarks. Convolutional models are not included because, since translation equivariance is not being considered here, their entries would be identical to those of the corresponding base models. The asterisk in the fourth column serves to indicate that AR CNPs only really produce non-Gaussian predictions for target locations encountered *later* in the AR chain, and the same symbol in the fifth column calls attention to the fact that AR DNPs only produce fully consistent predictions if a specific design choice (discussed in Section 3.2) is made.

Armed with this plan of action and with an extensive theoretical background, we are now finally ready to examine in detail the techniques employed to achieve the three goals set out above. This will constitute the bulk of the material presented in Chapter 3.

Chapter 3

Methods and Implementation

This Chapter examines the key techniques at the basis of the proposed Autoregressive Diffusion Neural Process models and discusses in detail how each step of this novel approach to NPs is implemented. Firstly, Section 3.1 eases into the topic by providing an intuitive overview of the central idea behind AR DNPs and by defining the relevant notation. Section 3.2 describes how data pre-processing is carried out for this specific task, and Section 3.3 then builds upon this foundation to present the two main architectures arising from this setup. Lastly, to conclude the Chapter, the procedures employed to train and deploy the models are illustrated in detail in Section 3.4.

Before proceeding, one should note that, in an effort to ensure that the relevant concepts come across as clearly and as effortlessly as possible, no code is directly included in this document. The reader is, however, reminded that they may refer to the relevant GitHub repository, should they be interested in the more technical and pragmatic details of the implementation, or in reproducing the experiments described in Chapter 4.

3.1 The AR DNP Concept

As implied by their name, Autoregressive Diffusion Neural Processes take inspiration from the diffusion literature [Ho et al., 2020; Sohl-Dickstein et al., 2015] to attempt and rectify the AR CNP issues brought to the reader's attention in Chapter 2. More specifically, the approach revolves around two key steps:

- Firstly, an augmented dataset consisting of different fidelity levels perturbed by varying amounts of additive Gaussian noise is produced. While a more in-depth discussion of this noising process is postponed to Section 3.2, a few details are presented here, to ensure this introductory Section is as self-contained as possible. Given a task $\psi = (\mathcal{D}_c, \mathcal{D}_t)$ consisting of a context set $\mathcal{D}_c = \{(x_n^{(c)}, y_n^{(c)})\}_{n=1}^{N_c}$ and a target set $\mathcal{D}_t = \{(x_n^{(t)}, y_n^{(t)})\}_{n=1}^{N_t}$, a series of F augmented target fidelity levels $\{y_{n,f}^{(t)}\}_{n=1,f=0}^{N_t,F}$ are computed through an autoregressive process in which the zeroth fidelity level simply consists of the original target outputs $\{y_n^{(t)}\}_{n=1}^{N_t}$, and all subsequent fidelities $f \in \{1, \dots, F\}$ are obtained by adding Gaussian noise onto the previous level.

- Secondly, the entire augmented dataset is modelled by one (or multiple) NP model(s) (whose specifications shall be discussed at a later point), in an *autoregressive denoising process* wherein the coarsest fidelity level $\{y_{n,F}^{(t)}\}_{n=1}^{N_t}$ is first generated from the original context set $\mathcal{D}_c$, and each subsequent fidelity $f \in \{F-1, \dots, 0\}$ is then obtained by conditioning on both $\mathcal{D}_c$ *and* the previously-obtained fidelities $\{y_{n,f'}^{(t)}\}_{n=1,f'=f+1}^{N_t,F}$, with this operation being repeated until the original, noise-free target points $\{y_{n,0}^{(t)}\}_{n=1}^{N_t}$ are recovered.

By applying the above autoregressive denoising procedure, the augmented dataset is used to induce a non-Gaussian and complex distribution on the observed data. Crucially, this technique cleverly solves the first problem highlighted above for standard AR CNPs, as the first variables generated in the AR chain ($\{y_{n,F}^{(t)}\}_{n=1}^{N_t}$) are, while definitely normally distributed, not actually of interest themselves. Additionally, since each fidelity level is modelled jointly, the ordering of the target inputs has no influence on the final predictions, meaning *consistency* is also retained (provided that, as discussed in more detail in Subsection 3.2.1 below, the marginalisation invariance property is also satisfied).

Before proceeding, one should call attention to the fact that, as hopefully clarified by the above paragraphs, the term "autoregressive" has a different connotation for AR DNPs than it does for AR CNPs: namely, in the former case, autoregression occurs *between* fidelity levels, whilst, in the latter, the technique is applied *within* a single layer (effectively, within the *only* existing layer, since AR CNPs are not characterised by a layered structure). Hence, the reader should not be misled by the similarity in naming conventions, and keep in mind that the two approaches are, indeed, quite different (and, specifically, that the former is not directly derived from the latter, but rather constitutes a parallel branch within the broader Neural Process Family).

3.2 Data Augmentation Through Noising

Having set the scene around AR DNPs, let us now delve into more specific implementation details, starting, in this Section, with an in-depth look at the process utilised to generate the altered training datasets mentioned in Section 3.1.

3.2.1 General Setup

As already briefly discussed above, the augmented dataset shall consist of a number of different fidelity levels, each generated by adding increasing amounts of Gaussian noise on top of the original target output values $\{y_n^{(t)}\}_{n=1}^{N_t}$. To account for this addition, let us modify the original definition of a task $\psi = (\mathcal{D}_c, \mathcal{D}_t)$ to also include what we shall henceforth refer to as the *auxiliary context dataset* $\mathcal{D}_a$:

$$\widehat{\psi} = (\mathcal{D}_c, \mathcal{D}_t, \mathcal{D}_a) \tag{3.2.1}$$

Crucially, unlike $\mathcal{D}_c$ and $\mathcal{D}_t$, $\mathcal{D}_a$ is not a set of input-output tuples, but rather an *ordered collection* of sets of input-output tuples (the aforementioned fidelity levels). For example, for F fidelities:

$$\mathcal{D}_t = \{(x_{n,0}^{(t)}, y_{n,0}^{(t)})\}_{n=1}^{N_{t,0}} \tag{3.2.2}$$

and:

$$\mathcal{D}_a = \left(\{(x_{n,1}^{(t)}, y_{n,1}^{(t)})\}_{n=1}^{N_{t,1}}, \ldots, \{(x_{n,F}^{(t)}, y_{n,F}^{(t)})\}_{n=1}^{N_{t,F}}\right) \tag{3.2.3}$$

where usually, for simplicity, the target input locations $\{x_{n,f}^{(t)}\}_{n=1,f=0}^{N_{t,f},F}$ are taken to be the same across all different values of f, so that the corresponding output values $\{y_{n,f}^{(t)}\}_{n=1,f=0}^{N_{t,f},F}$ can be more easily generated, without needing to separately noise slightly different copies of the same underlying signal (though this is also a viable option, and, indeed, occasionally leads to more stable behaviour at training and test time). Importantly, one should mention that, from a theoretical perspective, this seemingly minor choice relating to $\mathcal{D}_a$'s input locations has profound impacts on the model's properties:

- If the auxiliary context locations $\{x_{n,f}^{(t)}\}_{n=1,f=1}^{N_{t,f},F}$ are selected to match the original target locations $\{x_{n,0}^{(t)}\}_{n=1}^{N_{t,0}}$, the resulting predictions are *not* marginalisation-invariant, meaning the model does not define valid stochastic processes (despite the fact that ordering invariance is satisfied, unlike in the case of AR CNPs). In this scenario, we denote $N_{t,0} = N_{t,1} = \cdots = N_{t,F} = N_t$.

- If the auxiliary context locations $\{x_{n,f}^{(t)}\}_{n=1,f=1}^{N_{t,f},F}$ are selected *independently* from the original target locations $\{x_{n,0}^{(t)}\}_{n=1}^{N_{t,0}}$, the final outputs are *both* ordering- and marginalisation-invariant, satisfying the conditions of the Kolmogorov extension theorem [Øksendal, 2014] and yielding consistent predictions.

For ease of implementation, this project shall mostly focus on the first scenario, postponing a more thorough analysis of the implications of this choice to future investigations.

Having stated how both $\{x_{n,f}^{(t)}\}_{n=1,f=0}^{N_t,F}$ and $\{y_{n,0}^{(t)}\}_{n=1}^{N_t}$ are selected, to fully characterise $\mathcal{D}_a$, one need only describe the autoregressive procedure employed to generate each element of $\{y_{n,f+1}^{(t)}\}_{n=1}^{N_t}$ given the corresponding element of $\{y_{n,f}^{(t)}\}_{n=1}^{N_t}$:

$$y_{n,f+1}^{(t)} = \sqrt{1-\beta_f}\, y_{n,f}^{(t)} + \varepsilon_n \beta_f; \qquad \varepsilon_n \sim \mathcal{N}(0,1) \tag{3.2.4}$$

That is, each fidelity level is generated by first scaling down its predecessor by a factor of $\sqrt{1-\beta_f}$, and then adding standard Gaussian noise scaled by β_f on top. Whilst, as suggested by the subscript f, the parameter β_f could, potentially, be set to a different value for each fidelity, this option shall not be explored here, and a single value β shall be utilised instead.

3.2.2 The β Parameter

Given the above setup, one can easily observe that, as the depth (number of layers) of the AR DNP model is incremented (more on this in Section 3.3 below), the total amount of noise in the last fidelity level also proportionally increases (as one extra fidelity level is included in the augmented dataset for each new layer added to the model). Hence, the value of β should be set in such a way that the F-th fidelity is characterised by a specific, pre-specified noise variance: this will greatly simplify comparisons between architectures with different depths, and ensure that the differences observed are, in fact, due to this factor, rather than to a difference in total noising.

To obtain an expression for the noise present at the F-th fidelity level, one may proceed as follows (the subscript n and the superscript (t) are dropped for convenience, and the remaining subscript indicates the fidelity level $f \in \{0,1,\ldots,F\}$). For $f=1$:

$$y_1 = \sqrt{1-\beta}\, y_0 + \varepsilon\beta \tag{3.2.5}$$

or, since $\varepsilon \sim \mathcal{N}(0,1)$:

$$y_1 = \sqrt{1-\beta}\, y_0 + \delta_1; \qquad \delta_1 \sim \mathcal{N}\left(0, \beta^2\right) \tag{3.2.6}$$

For $f = 2$:

$$\begin{aligned} y_2 &= \sqrt{1-\beta}\, y_1 + \varepsilon\beta \\ &= \sqrt{1-\beta}\, \left(\sqrt{1-\beta}\, y_0 + \varepsilon\beta\right) + \varepsilon\beta \\ &= (1-\beta)\, y_0 + \beta\sqrt{1-\beta}\, \varepsilon + \varepsilon\beta \end{aligned} \tag{3.2.7}$$

or:

$$y_2 = (1-\beta)\, y_0 + \delta_2; \qquad \delta_2 \sim \mathcal{N}\left(0, \beta^2 + \beta^2\,(1-\beta)\right) \tag{3.2.8}$$

For $f = 3$:

$$\begin{aligned} y_3 &= \sqrt{1-\beta}\, y_2 + \varepsilon\beta \\ &= \sqrt{1-\beta}\, \left((1-\beta)\, y_0 + \beta\sqrt{1-\beta}\, \varepsilon + \varepsilon\beta\right) + \varepsilon\beta \\ &= (1-\beta)^{\frac{3}{2}}\, y_0 + \beta\,(1-\beta)\, \varepsilon + \beta\sqrt{1-\beta}\, \varepsilon + \varepsilon\beta \end{aligned} \tag{3.2.9}$$

or:

$$y_3 = (1-\beta)^{\frac{3}{2}}\, y_0 + \delta_3; \qquad \delta_3 \sim \mathcal{N}\left(0, \beta^2 + \beta^2\,(1-\beta) + \beta^2\,(1-\beta)^2\right) \tag{3.2.10}$$

By comparing Equations 3.2.6, 3.2.8 and 3.2.10, it is trivial to conclude that, for the general case of level f, the following expression holds true:

$$y_f = (1-\beta)^{\frac{f}{2}}\, y_0 + \delta_f; \qquad \delta_f \sim \mathcal{N}\left(0, \sum_{j=1}^{f} \beta^2\,(1-\beta)^{j-1}\right) \tag{3.2.11}$$

Since δ_f's variance is the sum of the first f elements of a geometric progression of the form $a, ar, ar^2, \ldots$ (with $a = \beta^2$ and $r = (1-\beta)$), the signal at the last level F may be neatly expressed as:

$$y_F = (1-\beta)^{\frac{F}{2}}\, y_0 + \delta_F; \qquad \delta_F \sim \mathcal{N}\left(0, \beta - \beta\,(1-\beta)^F\right) \tag{3.2.12}$$

Armed with the above expression and desired values for F and for the noise variance σ^2 in fidelity level F (henceforth referred to as *maximum noise variance*), the appropriate value for β can easily be obtained by numerically solving the following equation:

$$\sigma^2 = \beta - \beta\,(1-\beta)^F \tag{3.2.13}$$

Additionally, as shown in Figure 3.2.1, the above expression can also be employed to examine how the relative magnitudes of the original signal and the noise evolve, for a given value of β, as the depth of the AR DNP model varies:

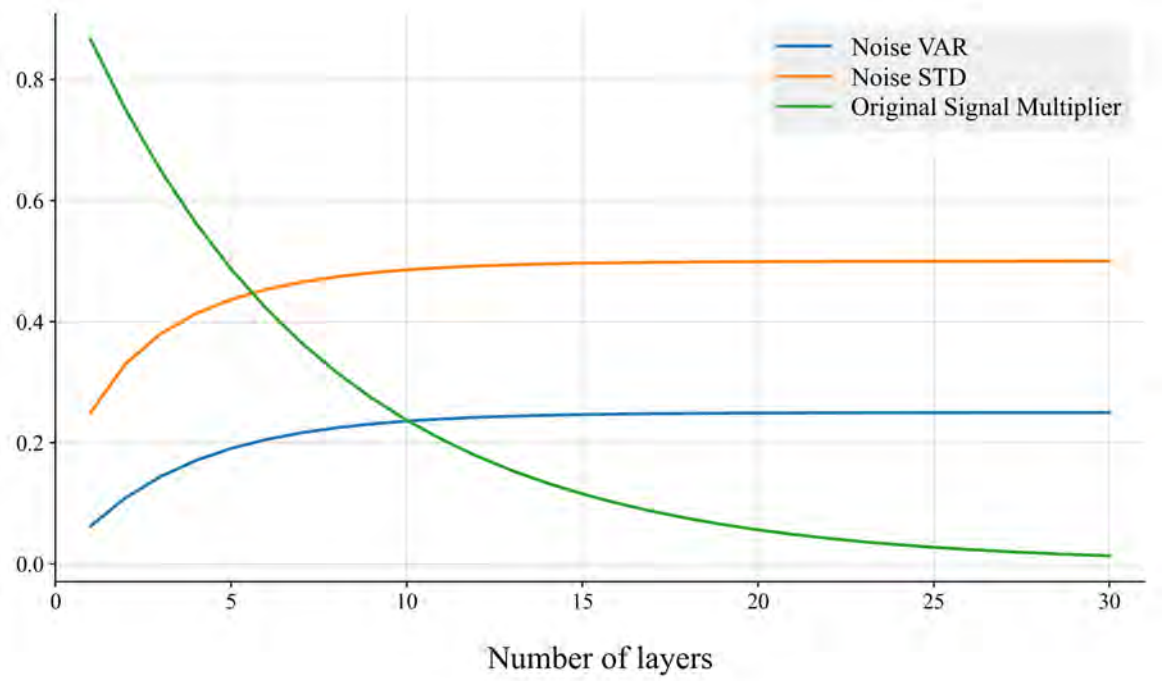


Figure 3.2.1: Evolution of the last layer's noise variance σ^2, noise standard deviation σ and original signal multiplier $(1-\beta)^{\frac{F}{2}}$ as the total number F of layers in the AR DNP model is increased (for a fixed $\beta = 0.25$). If the amplitude/range of the original signal is assumed to be equal to one, the crossing of the green curve and the orange curve indicates the point at which the magnitude of the injected noise become comparable to that of the original signal.

Lastly, Equation 3.2.12 also implies that, as $F \to \infty$:

$$y_\infty = \delta_\infty; \qquad \delta_\infty \sim \mathcal{N}(0, \beta) \tag{3.2.14}$$

meaning that, at least in principle, a very large number of layers can be added to the AR DNP model, without fearing that the magnitude of the resulting training data will increase without bound (though naturally, in the limit, no trace of the original signal will remain, possibly limiting the practical benefits of employing high values of F).

3.2.3 Synthetic 1D Datasets

To conclude this look at the data augmentation and pre-processing, let us briefly discuss the kinds of datasets upon which all quantitative and qualitative analyses presented in the following shall be based.

Given the somewhat stringent time constraints imposed on the project, the entirety of the work focused on synthetic 1D datasets, which are both easy to source, and also constitute the most common basic benchmark for NP performance, being present in virtually all of the relevant literature. Naturally, potential future expansions upon the research presented here shall involve *natural* 1D dataset, as well as 2D datasets (both of which are significantly more interesting to investigate when aiming to optimise models for practical, real-world applications).

With this premise, three 1D functional forms shall be considered: sawtooth waves, square waves and samples from Gaussian processes (Figures 3.2.2, 3.2.3 and 3.2.4 respectively).

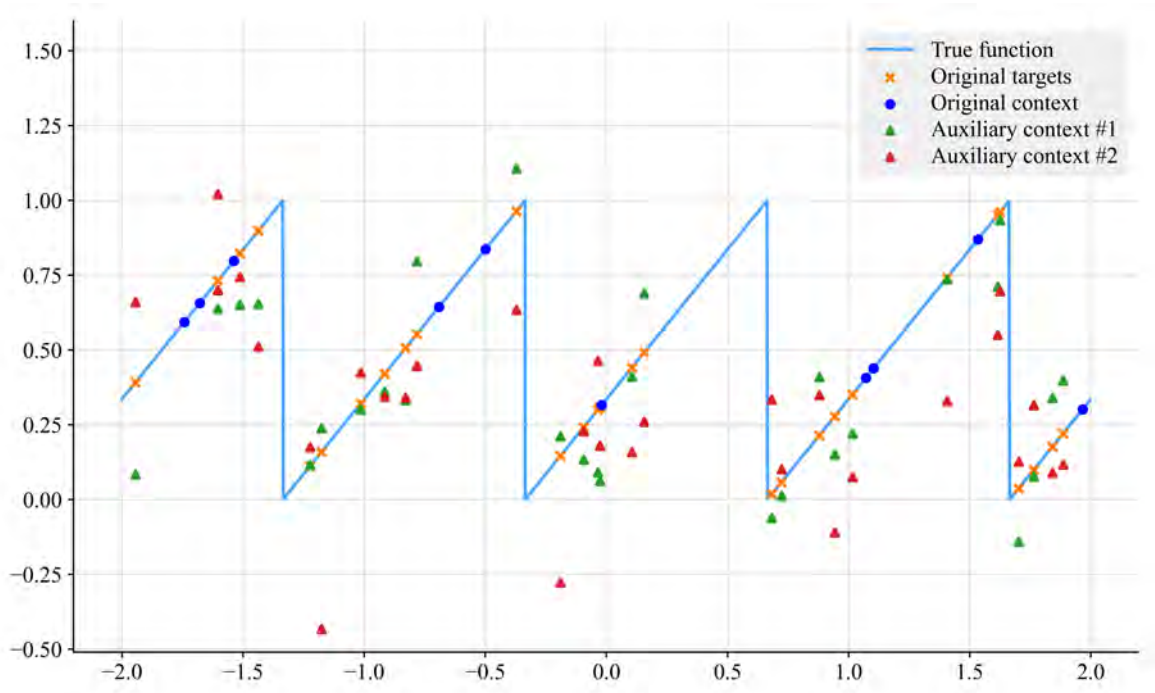


Figure 3.2.2: Example of a typical training task based on a sawtooth wave. The true function the model is meant to predict is shown, alongside the original context and target sets, as well as two auxiliary context sets (i.e. the first and second elements of $\mathcal{D}_a$, according to the definition given in Equation 3.2.3). Since two fidelity levels are present, this task would be tackled by a three-layer AR DNP (more on this in Section 3.3 below).

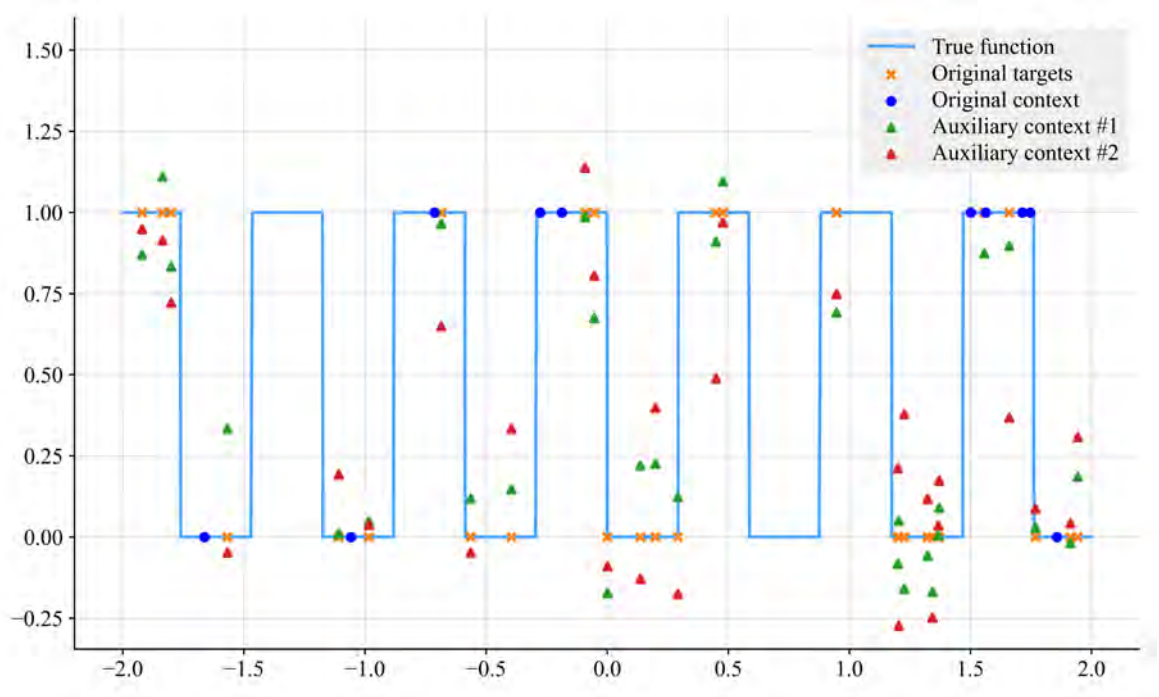


Figure 3.2.3: Example of a typical training task based on a square wave. Identical considerations to those made in Figure 3.2.2's caption also apply here.

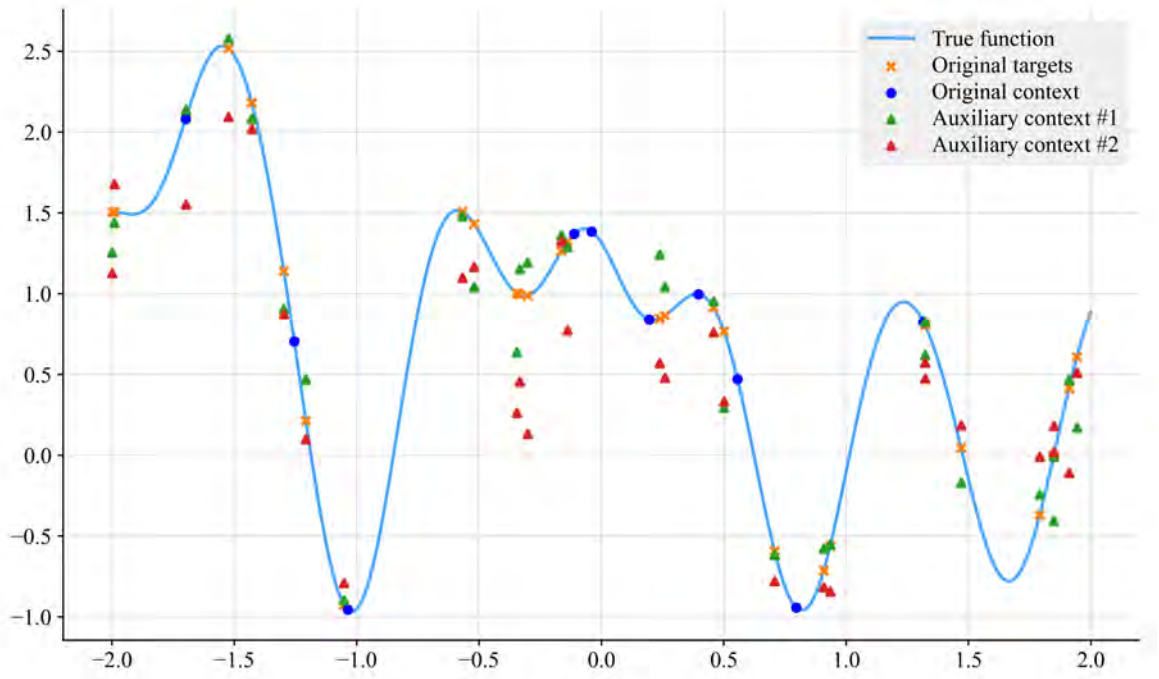


Figure 3.2.4: Example of a typical training task based on a function sample from a GP. The same points made in Figure 3.2.2's caption also hold here.

To create the three meta-datasets, different functions of each form are generated as follows:

- For the **sawtooth wave** dataset, each task's true function is generated by first sampling a wave frequency $\omega \sim \mathcal{U}(2,4)$, a direction d (-1 or 1 with equal probability) and a phase shift $\phi \sim \mathcal{U}\left(\frac{1}{\omega},1\right)$, and then combining these variables through the following expression:

$$f(x) = [\omega(dx-\phi)] \bmod 1 \tag{3.2.15}$$

- For the **square wave** dataset, each task's true function is produced by first sampling a wave frequency $\omega \sim \mathcal{U}(1,3)$ and a phase shift $\phi \sim \mathcal{U}\left(\frac{1}{\omega},1\right)$, and then combining these variables through the following expression:

$$f(x) = [\![\lfloor \omega x - \phi \rfloor] \bmod 2 = 0]\!] \tag{3.2.16}$$

  where $\lfloor\cdot\rfloor$ indicates the floor function, and $[\![P]\!]$ is the Iverson bracket of P, which takes on a value of 1 if the statement P is true, and 0 otherwise.

- For the **GP** dataset, each task's true function is simply sampled from a Gaussian process with an Exponentiated Quadratic (EQ) kernel with lengthscale $\ell = 0.25$.

3.3 AR DNP Model Architecture

Armed with appropriately pre-processed and augmented training datasets, let us now take a closer look at the modelling approach taken by AR DNPs. In an effort to simplify the notation employed throughout this Section, we shall henceforth denote $\{x_{n,f}^{(t)}\}_{n=1}^{N_t}$ as $\mathbf{x}_f^{(t)}$ and, similarly, $\{y_{n,f}^{(t)}\}_{n=1}^{N_t}$ as $\mathbf{y}_f^{(t)}$ (where, once again, for $f=0$, these expressions are understood to refer to the elements contained in the original target set $\mathcal{D}_t$).

3.3.1 General Architecture

The joint predictive is then defined as the following, appropriately normalised mixture model:

$$p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) \approx \sum_{s=1}^{S} \frac{1}{S} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right); \qquad \mathcal{D}_{a,s} \sim p_\theta(\mathcal{D}_{a,s} | \mathcal{D}_c) \tag{3.3.1}$$

where each mixture component is generated by the same NP p_θ, conditioned on a *different* instance $\mathcal{D}_{a,s}$ of the auxiliary context dataset $\mathcal{D}_a = \left(\mathcal{D}_a^1, \ldots, \mathcal{D}_a^F\right)$, each of which is obtained

by repeating the following autoregressive process (starting from different random states):

$$\begin{aligned}
\mathcal{D}_a^F &= \left(\mathbf{x}_F^{(t)}, \mathbf{y}_F^{(t)}\right); \qquad \mathbf{y}_F^{(t)} \sim p_\theta\left(\mathbf{y}_F^{(t)} | \mathbf{x}_F^{(t)}, \mathcal{D}_c\right) \\
\mathcal{D}_a^{F-1} &= \left(\mathbf{x}_{F-1}^{(t)}, \mathbf{y}_{F-1}^{(t)}\right); \qquad \mathbf{y}_{F-1}^{(t)} \sim p_\theta\left(\mathbf{y}_{F-1}^{(t)} | \mathbf{x}_{F-1}^{(t)}, \mathcal{D}_c, \mathcal{D}_a^F\right) \\
\mathcal{D}_a^{F-2} &= \left(\mathbf{x}_{F-2}^{(t)}, \mathbf{y}_{F-2}^{(t)}\right); \qquad \mathbf{y}_{F-2}^{(t)} \sim p_\theta\left(\mathbf{y}_{F-2}^{(t)} | \mathbf{x}_{F-2}^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{F-1:F}\right) \\
&\vdots \\
\mathcal{D}_a^f &= \left(\mathbf{x}_f^{(t)}, \mathbf{y}_f^{(t)}\right); \qquad \mathbf{y}_f^{(t)} \sim p_\theta\left(\mathbf{y}_f^{(t)} | \mathbf{x}_f^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{f+1:F}\right) \\
&\vdots \\
\mathcal{D}_a^1 &= \left(\mathbf{x}_1^{(t)}, \mathbf{y}_1^{(t)}\right); \qquad \mathbf{y}_1^{(t)} \sim p_\theta\left(\mathbf{y}_1^{(t)} | \mathbf{x}_1^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{2:F}\right)
\end{aligned} \tag{3.3.2}$$

For clarification, three main notes should be made about the two expressions presented above:

- In Equation 3.3.1, the $\approx$ sign indicates that the RHS originates from a Monte-Carlo approximation of the more general expression:

$$p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) = \int p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right) p_\theta(\mathcal{D}_{a,s} | \mathcal{D}_c) \, \mathrm{d}\mathcal{D}_{\mathrm{a,s}} \tag{3.3.3}$$

  Having formally stated this, in the following, an equals sign shall be used for simplicity instead.

- In Equation 3.3.2, the numerical superscripts on $\mathcal{D}_a$ indicate which element(s) of the ordered collection are being referred to, again based on the expression given in Equation 3.2.3. Specifically, the notation $\mathcal{D}_a^{f:F}$ is employed as a shorthand for the collection $\left(\left(\mathbf{x}_f^{(t)}, \mathbf{y}_f^{(t)}\right), \ldots, \left(\mathbf{x}_F^{(t)}, \mathbf{y}_F^{(t)}\right)\right)$.

- In Equation 3.3.2 (and in the previous bullet point), in a slight abuse of notation, the tuples $\left(\mathbf{x}_f^{(t)}, \mathbf{y}_f^{(t)}\right)$ are improperly utilised in lieu of the more cumbersome $\{(x_{n,f}^{(t)}, y_{n,f}^{(t)})\}_{n=1}^{N_t}$. This convention is also employed at other points throughout the remainder of this Section.

To summarise the discussed AR DNP approach, the procedures set out in Equations 3.3.1 and 3.3.2 are pictorially exemplified through the diagram presented in Figure 3.3.1 below (for the case of $F = 2$).

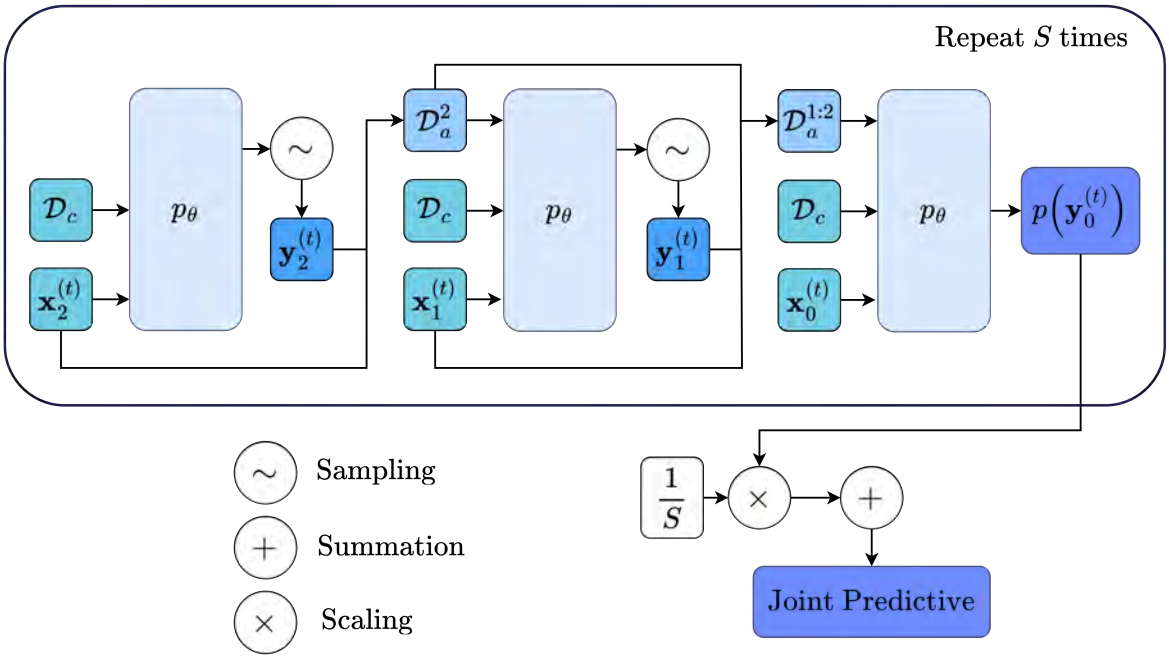


Figure 3.3.1: Schematic illustrating the functioning of a three-layer AR DNP model.

Since the architecture of the NP p_θ used to generate both the mixture components and the noised target outputs $\{\mathbf{y}_f^{(t)}\}_{f=1}^F$ can, at least in principle, be selected to match any of the ones already described at length in Chapter 2, the mathematical framework for AR DNPs is, quite elegantly, entirely described by Equations 3.3.1 and 3.3.2. Clearly, however, the actual precise design of p_θ should still be discussed in some detail. This is the goal of the following Subsection.

3.3.2 Split vs. Joint Model

In an effort to ensure that AR DNPs can seamlessly be inserted within the Neural Process Family framework, these models were designed to be a direct superset of existing NPs. Indeed, as mentioned above, the architecture of the base NP p_θ can be chosen from any of the variants discussed in Chapter 2 (though, in most cases, a ConvGNP will prove to be the best option), and, if F is set to 0 (to yield a one-layer model with an empty auxiliary context dataset $\mathcal{D}_a$), said basic architecture is recovered. Additionally, if one elects to also allow *intra-layer* AR predictions (rather than only employing the *inter-layer* AR procedure formalised in Equation 3.3.2), regular AR CNP behaviour can also, in principle, be obtained (this approach has been briefly examined, but a more thorough investigation is postponed to future work).

Having made this premise, two main AR DNP architectures shall be investigated:

- A **split** model, consisting of a *series* of NPs $\left(p_{\theta_0}, \dots, p_{\theta_F}\right)$, each tasked with modelling *one* of the fidelity levels. In this case, Equations 3.3.1 and 3.3.2 take the following forms:

$$p_{\theta_0}\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) = \sum_{s=1}^{S} \frac{1}{S} p_{\theta_0}\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right) \tag{3.3.4}$$

and:

$$\begin{aligned}
\mathcal{D}_a^F &= \left(\mathbf{x}_F^{(t)}, \mathbf{y}_F^{(t)}\right); \qquad \mathbf{y}_F^{(t)} \sim p_{\theta_F}\left(\mathbf{y}_F^{(t)} | \mathbf{x}_F^{(t)}, \mathcal{D}_c\right) \\
&\vdots \\
\mathcal{D}_a^f &= \left(\mathbf{x}_f^{(t)}, \mathbf{y}_f^{(t)}\right); \qquad \mathbf{y}_f^{(t)} \sim p_{\theta_f}\left(\mathbf{y}_f^{(t)} | \mathbf{x}_f^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{f+1:F}\right) \\
&\vdots \\
\mathcal{D}_a^1 &= \left(\mathbf{x}_1^{(t)}, \mathbf{y}_1^{(t)}\right); \qquad \mathbf{y}_1^{(t)} \sim p_{\theta_1}\left(\mathbf{y}_1^{(t)} | \mathbf{x}_1^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{2:F}\right)
\end{aligned} \tag{3.3.5}$$

  In this approach, each model in $\left(p_{\theta_0}, \dots, p_{\theta_F}\right)$ is constructed to handle a context set with $F+1$ channels (1 for $\mathcal{D}_c$ and F for D_a), and to produce single-channel target predictions given single-channel target locations. One should note that, in principle, the number of context channels could be varied based on which level the model is constructed to handle, but this option was not explored, so that all instances of p_θ would share the same architecture, and just differentiate themselves based on their parameters θ_f.

- A **joint** model, consisting of a *single* NP p_θ tasked with jointly modelling *all* fidelity levels. In this approach, p_θ is constructed to handle a context set with $F+1$ channels (1 for $\mathcal{D}_c$ and F for D_a), and to produce $F+1$-channel target predictions given $F+1$-channel target locations. This means that Equations 3.3.1 and 3.3.2 take the following forms:

$$p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) = \sum_{s=1}^{S} \frac{1}{S} p_\theta\left(\mathbf{y}_0^{(t)}, \cancel{\mathbf{y}_1^{(t)}}, \dots, \cancel{\mathbf{y}_F^{(t)}} | \mathbf{x}_0^{(t)}, \cancel{\mathbf{x}_1^{(t)}}, \dots, \cancel{\mathbf{x}_F^{(t)}}, \mathcal{D}_c, \mathcal{D}_{a,s}\right) \tag{3.3.6}$$

and:

$$\begin{aligned}
\mathcal{D}_a^F &= \left(\mathbf{x}_F^{(t)}, \mathbf{y}_F^{(t)}\right); \qquad \mathbf{y}_F^{(t)} \sim p_\theta\left(\cancel{\mathbf{y}_0^{(t)}}, \cancel{\mathbf{y}_1^{(t)}}, \dots, \mathbf{y}_F^{(t)} | \right. \\
&\qquad\qquad \left. \cancel{\mathbf{x}_0^{(t)}}, \cancel{\mathbf{x}_1^{(t)}}, \dots, \mathbf{x}_F^{(t)}, \mathcal{D}_c\right) \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}_a^f = \left(\mathbf{x}_f^{(t)}, \mathbf{y}_f^{(t)}\right); \quad & \mathbf{y}_f^{(t)} \sim p_\theta\Big(\cancel{\mathbf{y}_0^{(t)}}, \ldots, \mathbf{y}_f^{(t)}, \ldots, \cancel{\mathbf{y}_F^{(t)}} \mid \\
& \qquad \cancel{\mathbf{x}_0^{(t)}}, \ldots, \mathbf{x}_f^{(t)}, \ldots, \cancel{\mathbf{x}_F^{(t)}}, \mathcal{D}_c, \mathcal{D}_a^{f+1:F}\Big) \\
\vdots & \\
\mathcal{D}_a^1 = \left(\mathbf{x}_1^{(t)}, \mathbf{y}_1^{(t)}\right); \quad & \mathbf{y}_1^{(t)} \sim p_\theta\Big(\cancel{\mathbf{y}_0^{(t)}}, \mathbf{y}_1^{(t)}, \ldots, \cancel{\mathbf{y}_F^{(t)}} \mid \\
& \qquad \cancel{\mathbf{x}_0^{(t)}}, \mathbf{x}_1^{(t)}, \ldots, \cancel{\mathbf{x}_F^{(t)}}, \mathcal{D}_c, \mathcal{D}_a^{2:F}\Big)
\end{aligned} \tag{3.3.7}$$

where the cancellation signs are meant to indicate that, when dealing with a specific fidelity level f', all target inputs $\{\mathbf{x}_f^{(t)}\}_{f \neq f'}$ are replaced by an empty tensor, and, consequently, the resulting target predictions $\{\mathbf{y}_f^{(t)}\}_{f \neq f'}$ are also empty (and are thus ignored, yielding the same behaviour as in the split scenario, despite p_θ's joint modelling). Incidentally, as discussed in more detail in the next Section, this also ensures that log-likelihood computations carried out during training only take into account the relevant fidelity level, despite the predictions being technically made jointly across all levels.

3.4 Training and Deployment Procedures

Having described AR DNPs' key architectural details, to close out this Chapter, let us now briefly discuss how these models are trained (Subsection 3.4.1), and, subsequently, deployed at test time (Subsection 3.4.2).

3.4.1 Training

As already mentioned in the previous Section, AR DNPs were designed as an architectural superset of existing NPs. Because of this, it should hopefully come as no surprise that the procedure employed to train these models is very similar to that utilised for the regular NP variants that they directly extend (that is, a maximum likelihood objective). However, one main structural difference between AR DNPs and their non-diffusion counterparts still subsists: namely, the presence, in the former, of multiple layers, each dealing with a differently-noised version of the same data. As a result, to fully characterise how AR DNPs are fit to a specific meta-dataset, one new technique needs to be introduced to address this discrepancy: that is, *task masking*.

To explain this notion, let us refer back to the concepts discussed in Section 3.2, and once again consider an augmented task $\widehat{\psi} = (\mathcal{D}_c, \mathcal{D}_t, \mathcal{D}_a)$, each element of which is respectively defined as follows (using the less formal, but more intuitive and less cumbersome notation established in Section 3.3):

$$\mathcal{D}_c = \left(\mathbf{x}^{(c)}, \mathbf{y}^{(c)}\right) \tag{3.4.1}$$

$$\mathcal{D}_t = \left(\mathbf{x}_0^{(t)}, \mathbf{y}_0^{(t)}\right) \tag{3.4.2}$$

$$\mathcal{D}_a = \left(\left(\mathbf{x}_1^{(t)}, \mathbf{y}_1^{(t)}\right), \ldots, \left(\mathbf{x}_F^{(t)}, \mathbf{y}_F^{(t)}\right)\right) \tag{3.4.3}$$

Given this setup, the term "task masking" refers to the process of altering an augmented task $\widehat{\psi}$ in such a way that it may be utilised to train a *specific* layer f of a given AR DNP model. As can easily be inferred by examining the mathematical framework set out in Section 3.3, a task masked to be handled by the f-th layer of an AR DNP (henceforth denoted as $\widehat{\psi}_f$) would necessarily take the following form:

$$\widehat{\psi}_f = \left(\mathcal{D}_c, \mathcal{D}_t^f, \mathcal{D}_a^{f+1:F}\right) \tag{3.4.4}$$

where $\mathcal{D}_t^f = \mathcal{D}_a^f$ if $f \neq 0$, and simply equals $\mathcal{D}_t$ otherwise (i.e. in the case of the last/output layer). In other words, when training a given layer f, the model should learn to condition on $\mathcal{D}_c$ and on all the noisier fidelity levels $\mathcal{D}_a^{f+1:F}$ to predict (as targets) the input-output pairs contained within fidelity level f (so that the AR procedures detailed in Equation 3.3.2 may be carried out as required during deployment).

Having described how a task is masked to train a single layer f, the specific procedure employed to train *all* layers $0, \ldots, F$ depends on whether a split or a joint model is being considered:

- In the case of a **split** model, a different NP (with a distinct set of parameters) is instantiated to model each layer. Therefore, training is quite straightforward, and $F+1$ separate models are trained, each through a maximum likelihood objective defined over a series $\{\widehat{\psi}_{f,i}\}_{i=1}^N$ of N appropriately-masked tasks $\widehat{\psi}_f$:

$$\theta_f^* = \underset{\theta_f \in \Theta}{\arg\max} \left\{ \mathbb{E}_{\widehat{\psi}_{f,i} \sim \Psi} \left[\log p_{\theta_f} \left(\mathbf{y}_f^{(t)} | \mathbf{x}_f^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{f+1:F} \right) \right] \right\} \tag{3.4.5}$$

- In the case of a **joint** model, a single NP (with a single set of parameters θ) is used to model all layers. Hence, for each training batch, a layer index f is first randomly

sampled from $\mathcal{U}(0,F)$, and each augmented task within the batch is then masked accordingly. Subsequently, a batched forward pass is carried out, the relevant log-likelihood $p_\theta\left(\mathbf{y}_f^{(t)} | \mathbf{x}_f^{(t)}, \mathcal{D}_c, \mathcal{D}_a^{f+1:F}\right)$ is computed, and the results are employed for backpropagation as usual. Importantly, given the masking procedure, only predictions for $\mathbf{y}_f^{(t)}$ are yielded (despite the model's joint-across-levels approach), meaning gradient computations are not contaminated by data that should not be available to the model at a given training instance.

3.4.2 Deployment

Armed with a trained model, the deployment technique employed at test time is quite straightforward, and directly based upon the noised AR procedure already presented in Equation 3.3.2, as well as on the mixture model predictive given in Equation 3.3.1. To more concretely illustrate the workings of this process, let us imagine that a four-layer AR DNP has been properly fit to a suitable augmented meta-dataset. Then, one need only select a positive integer S denoting how many AR samples should be drawn (i.e. of how many mixture components the final joint predictive should consist), and repeat the following procedure S times to populate different versions $\mathcal{D}_{a,s}$ of the auxiliary context dataset $\mathcal{D}_a$ (as a reminder, at test time, no augmented data outputs are available, and one must exclusively rely on the original context set and target locations to obtain target predictions):

$$\begin{aligned}
\mathcal{D}_{a,s}^3 &= \left(\mathbf{x}_3^{(t)}, \mathbf{y}_{3,s}^{(t)}\right); \qquad \mathbf{y}_{3,s}^{(t)} \sim p_\theta\left(\mathbf{y}_3^{(t)} | \mathbf{x}_3^{(t)}, \mathcal{D}_c\right) \\
\mathcal{D}_{a,s}^2 &= \left(\mathbf{x}_2^{(t)}, \mathbf{y}_{2,s}^{(t)}\right); \qquad \mathbf{y}_{2,s}^{(t)} \sim p_\theta\left(\mathbf{y}_2^{(t)} | \mathbf{x}_2^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}^3\right) \\
\mathcal{D}_{a,s}^1 &= \left(\mathbf{x}_1^{(t)}, \mathbf{y}_{1,s}^{(t)}\right); \qquad \mathbf{y}_{1,s}^{(t)} \sim p_\theta\left(\mathbf{y}_1^{(t)} | \mathbf{x}_1^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}^{2:3}\right) \\
\mathcal{D}_{a,s} &= \left(\mathcal{D}_{a,s}^1, \mathcal{D}_{a,s}^2, \mathcal{D}_{a,s}^3\right)
\end{aligned} \tag{3.4.6}$$

Imagining $S = 5$, the final, full joint predictive may then be fully written out as follows (by directly applying Equation 3.3.1):

$$\begin{aligned}
p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) = &\frac{1}{5} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,1}\right) + \frac{1}{5} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,2}\right) + \\
&\frac{1}{5} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,3}\right) + \frac{1}{5} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,4}\right) + \\
&\frac{1}{5} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,5}\right)
\end{aligned} \tag{3.4.7}$$

As a reminder, to compute overall test log-likelihood, one can simply apply the LogSumExp function to the list of individual log-likelihoods achieved by each mixture component (remembering to divide by the number S of noised AR samples):

$$\begin{aligned} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right) &= \sum_{s=1}^{S} \frac{1}{S} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right) \\ \log\left[p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c\right)\right] &= \log\left[\frac{1}{S}\sum_{s=1}^{S} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right)\right] \\ &= \log\left[\sum_{s=1}^{S} p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right)\right] - \log(S) \\ &= \text{LogSumExp}\left\{\log p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right)\right\}_{s=1}^{S} - \log(S) \end{aligned} \tag{3.4.8}$$

One should also mention that, to ensure fair comparisons between tasks, each log-likelihood in $\left\{\log p_\theta\left(\mathbf{y}_0^{(t)} | \mathbf{x}_0^{(t)}, \mathcal{D}_c, \mathcal{D}_{a,s}\right)\right\}_{s=1}^{S}$ is also normalised by dividing by the number of targets.

Having meticulously discussed the inner workings of AR DNP models, the next natural step is to thoroughly assess the performance of this diffusion-inspired approach to NPs on a variety of tasks. This shall constitute the main goal of Chapter 4.

Chapter 4

Experimental Setups and Results

This Chapter presents the main results obtained by deploying a number of AR DNP models on a variety of tasks. Firstly, Section 4.1 benchmarks the performance of AR DNPs on the three synthetic 1D regression meta-datasets introduced in Chapter 3 (sawtooth, square wave and GP, in Subsections 4.1.1, 4.1.2 and 4.1.3 respectively). The performance of other NP models (both state of the art and not) is also measured on the same tasks, and provided as a baseline for comparison. Key observations that are valid across all three datasets are also included in Subsection 4.1.4. Secondly, Section 4.2 investigates the most relevant hyperparameters that were found to affect AR DNP predictive power, and advances hypotheses concerning the reasons behind the observed behaviours. Lastly, to close out the Chapter, Section 4.3 briefly and informally discusses the computational costs tied to training and deploying AR DNPs, and compares these with those arising from AR CNPs.

4.1 1D Regression on Synthetic Datasets

Before delving into the 1D regression results, a few notes should be made about the plots presented below.

Firstly, unless specifically mentioned otherwise, all the models examined in this Chapter share the same setup (both in terms of training[1] and underlying architecture[2]). Indeed, the

[1] All models were trained for 500 epochs (each consisting of 2^{14} tasks), using a batch size of 16 and the Adam optimiser [Kingma and Ba, 2015] with a learning rate of 0.0003.

[2] All models utilise a U-Net [Ronneberger et al., 2015] architecture with 6 channels, each with size 64 and stride 2. All (single) models have a parameter count comprised between 330k and 350k.

only hyperparameters that are varied in any way between experiments are those specifically discussed in Section 4.2.

Secondly, each of the three test meta-datasets utilised for benchmarking is composed of 310 tasks, evenly split across context sizes $|\mathcal{D}_c|$ comprised between 0 and 30 (i.e. 10 tasks for each). In all cases, the cardinality $|\mathcal{D}_t|$ of the original target set is fixed at 50. This setup allows for a highly granular analysis of AR DNP performance across varying context sizes (which, as evidenced by the following Subsections, shall prove crucial in drawing specific conclusions about the behaviour of different models).

Thirdly, the numerous likelihood plots (Figures 4.1.1, 4.1.3, 4.1.5 et cetera) analysed in the remainder of this Chapter do not include error bars. This is a conscious decision, made in an effort to present as clear of a picture as possible. In fact, whilst support for this feature has been implemented, the relatively small sample size (each data point is obtained by averaging over only 10 values) coupled with the high inter-task variability (due to some context distributions being intrinsically harder to predict from) leads to error bars whose large size partially obfuscates the underlying trends. One should also note that this issue affects all models equally (baselines included), and is not a quirk specific to AR DNPs.

Lastly, the observant reader might notice that the likelihood plots also do not showcase an AR ConvGNP baseline. This omission is not voluntary, but regrettably due to an unidentified bug in the pre-existing code, which seemingly prevents ConvGNPs from being successfully deployed AR (some results can be obtained, but these are most likely wrong, and were thus not deemed worthy of inclusion). Thankfully, this setback only really detracts from the GP results (Subsection 4.1.3), wherein the Gaussian nature of the underlying data would likely lead to a somewhat significant improvement in performance. Nevertheless, rectifying this issue is at the top of the list of potential future improvements upon the project work.

Having made this premise, Subsections 4.1.1-4.1.3 present the best results obtained by running a variety of different AR DNP architectures on the three meta-datasets.

4.1.1 Noised Sawtooth

Figure 4.1.1 below compares the performance of an AR DNP (Joint ConvGNP architecture, 4 layers, 0.02 maximum noise variance, 5000 AR samples) with that of the relevant available baselines (ConvCNP, ConvGNP and AR ConvCNP) across sawtooth benchmark tasks with different context sizes.

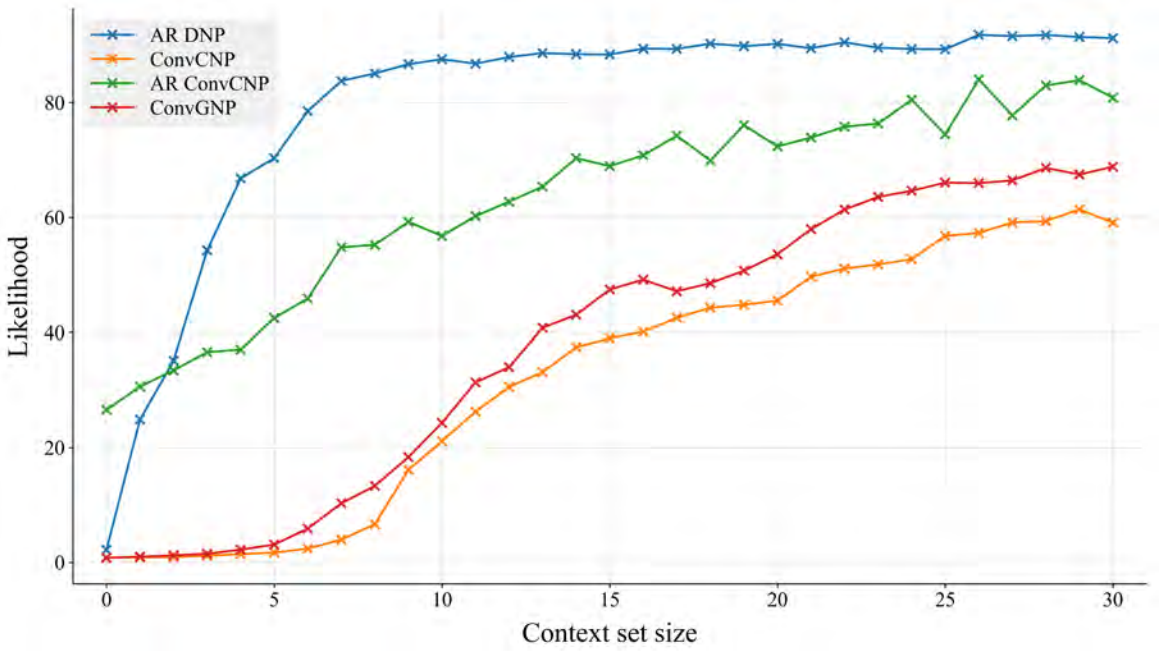


Figure 4.1.1: Comparison of AR DNP performance against relevant baselines on the sawtooth benchmark meta-dataset. Each data point is obtained by averaging the likelihoods obtained across 10 tasks with a specific context size $|\mathcal{D}_c|$.

As evidenced by these curves, the selected AR DNP model vastly outperforms both basic NP variants (ConvCNP and ConvGNP) at all context sizes, and, more interestingly, also significantly overtakes AR ConvCNPs (the current state of the art) on all tasks but those characterised by extremely small context sizes $|\mathcal{D}_c| \leq 1$ (more on this specific phenomenon in Subsection 4.1.4). This performance is altogether quite promising, and seemingly suggests that AR DNPs are, indeed, a very viable alternative to AR CNPs (at least when modelling this specific sawtooth functional family).

Naturally, before drawing any clear-cut conclusions, other datasets should also be investigated. Before doing so, however, let us take a brief look at an example of the kind of predictions produced by the above AR DNP on one of the benchmarks tasks. Figure 4.1.2 below showcases how *one* of the S mixture components in a specific task's final joint predictive is generated, starting from just the original context in layer 3 (bottom right) and autoregressively building upon this to yield the output predictive in layer 0 (top left).

As demonstrated by these four plots (and especially the one relating to the zeroth layer), the AR DNP is able, in this specific instance, to recover the full original signal from just two context points. This is quite impressive and, despite the fact that this process is based

on *random* samples (meaning other mixture components s might not behave as ideally), suggests that AR DNPs are able to recover a significant portion of the underlying structure of the data-generating distributions, even when provided with relatively minimal context information.

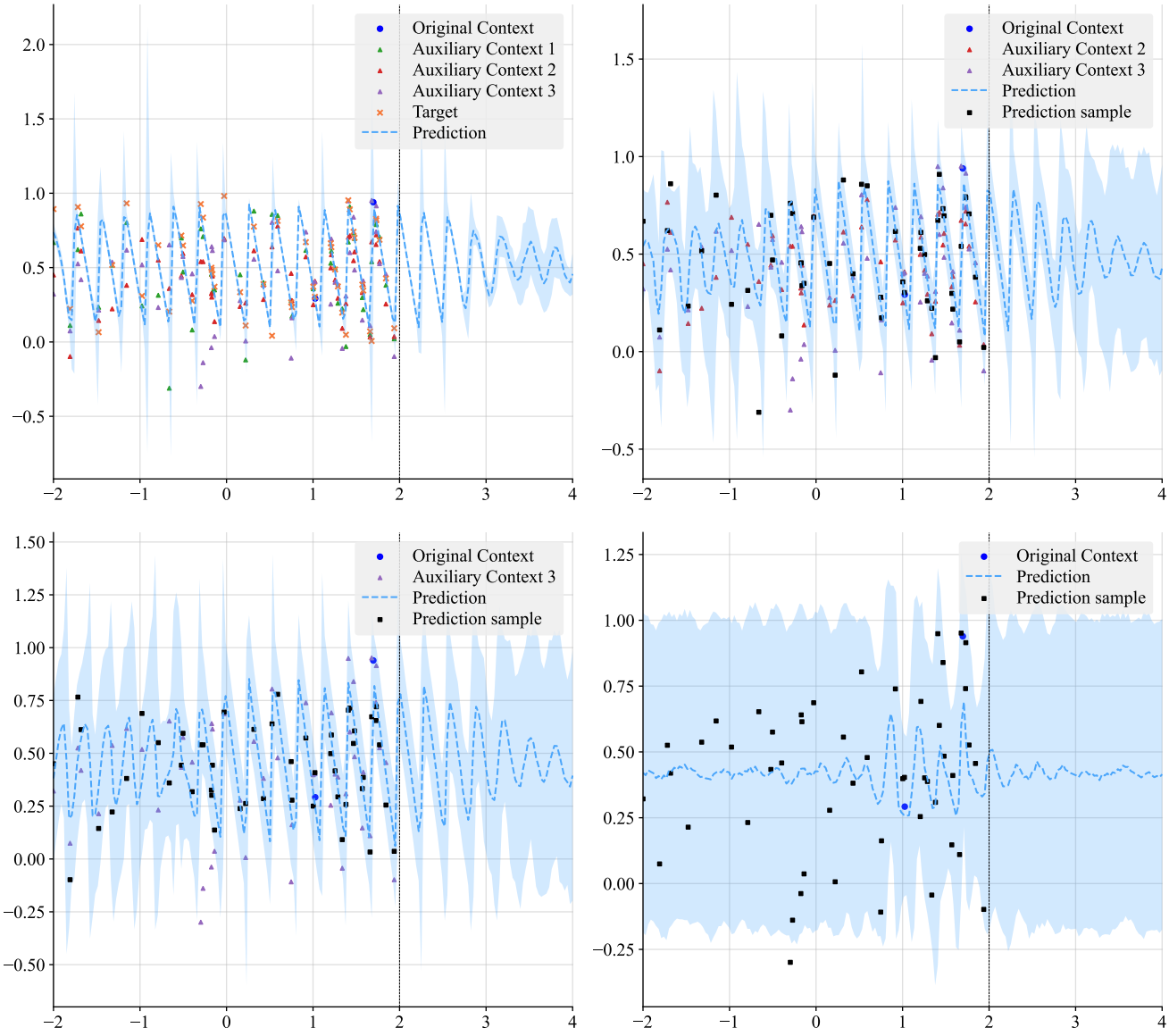


Figure 4.1.2: AR DNP joint predictives in layers 0 (top left), 1 (top right), 2 (bottom left) and 3 (bottom right) for a benchmark task with $|\mathcal{D}_c| = 2$. The AR deployment process starts in layer 3, where a prediction is made by conditioning on the context set. Samples from this prediction are used to populate the third channel of the auxiliary context ($\mathcal{D}_a^3$), which is then conditioned upon in layer 2. This procedure is repeated until the full auxiliary context is generated, and subsequently utilised (in layer 0) to predict at the original target locations.

4.1.2 Noised Square Wave

Figure 4.1.3 below compares the performance of an AR DNP (Joint ConvGNP architecture, 4 layers, 0.06 maximum noise variance, 5000 AR samples) with that of the relevant available baselines (ConvCNP, ConvGNP and AR ConvCNP) across square wave benchmark tasks with different context sizes.

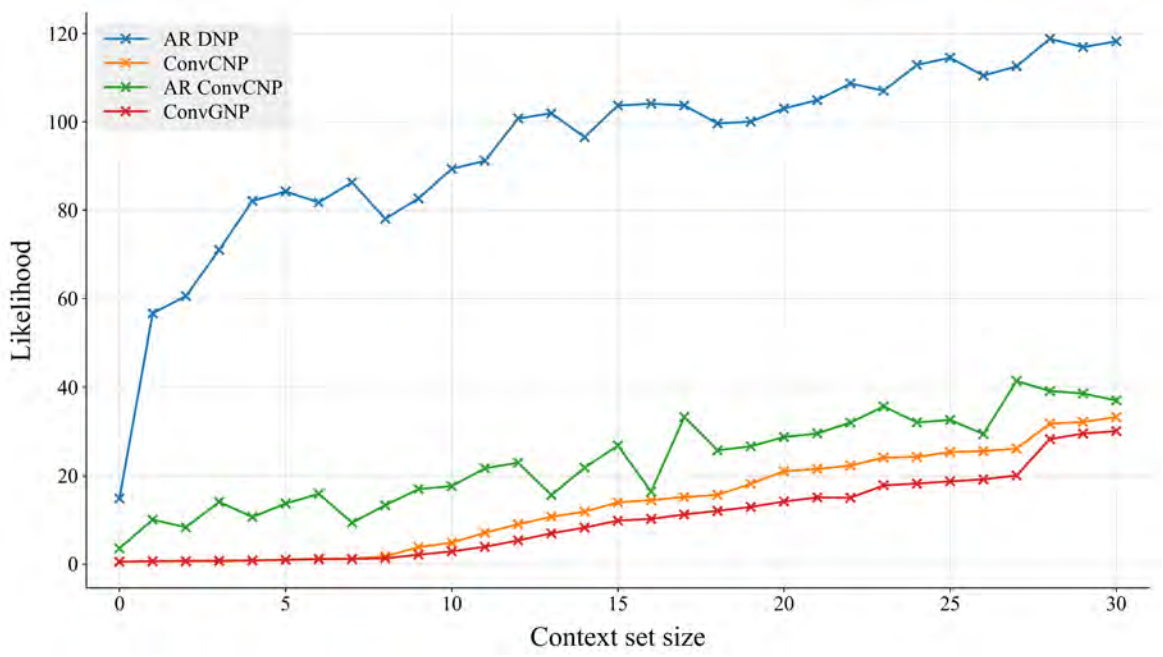


Figure 4.1.3: Comparison of AR DNP performance against relevant baselines on the square wave benchmark meta-dataset. Each data point is obtained by averaging the likelihoods obtained across 10 tasks with a specific context size $|\mathcal{D}_c|$.

Once again, the selected AR DNP model vastly outperforms both basic NP variants (ConvCNP and ConvGNP) at all context sizes. In this instance, however, the diffusion-based approach *also* substantially outclasses the state-of-the-art AR ConvCNPs on *all* tasks, regardless of context size (unlike in the case of the sawtooth meta-dataset, where the latter technique was shown to be better suited than the former in handling low-context scenarios). As such, the performance measured above is extremely promising (even more so than that presented in the previous Subsection), and supports the conclusion that AR DNPs are a definitely viable alternative to AR CNPs when attempting to model square waves.

Hypotheses as to the reason behind this observed difference in behaviour across datasets are advanced in Subsection 4.1.4.

Before proceeding, let us once again look at an example of how a typical mixture component of a random task's joint predictive is generated (Figure 4.1.4).

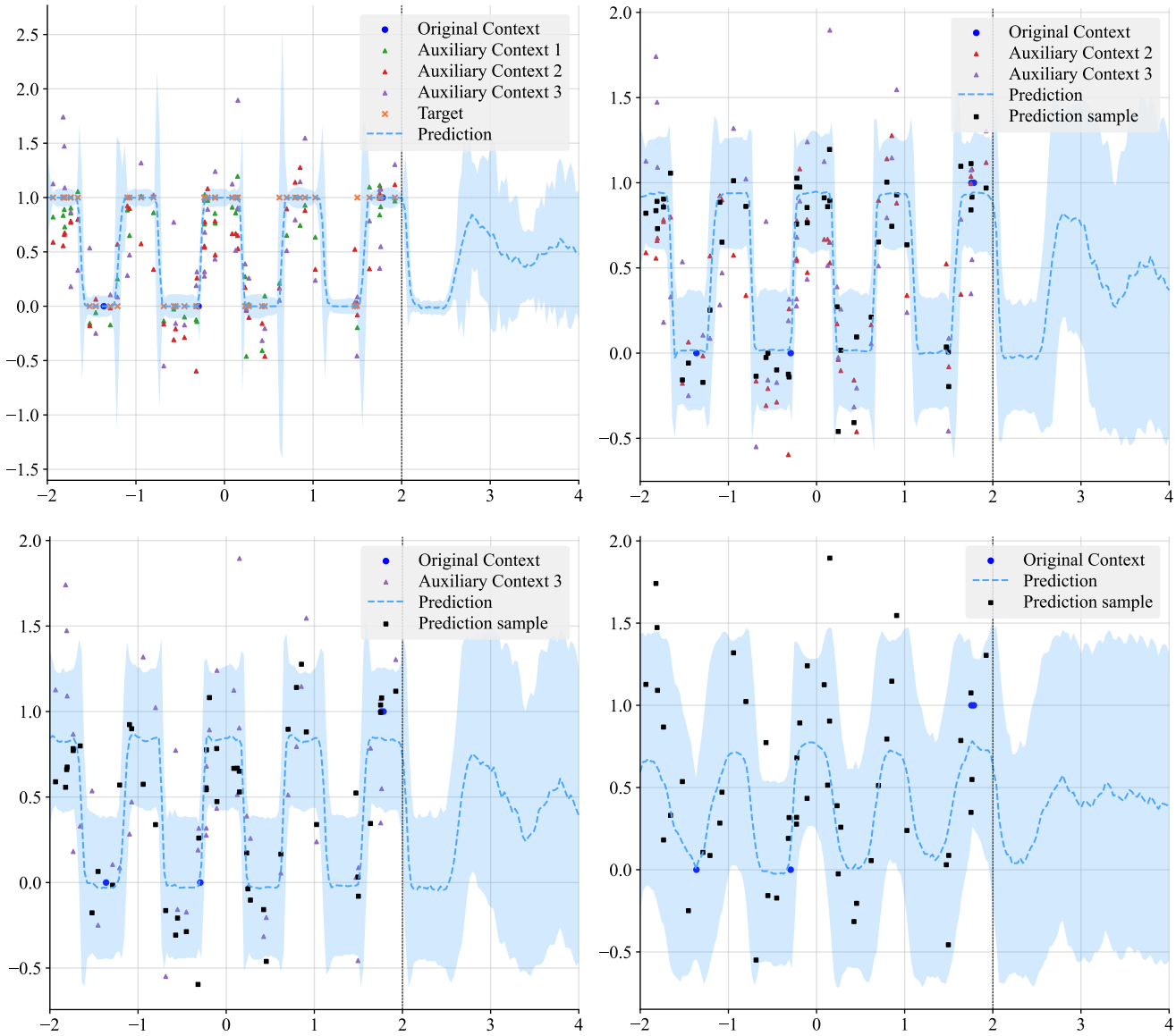


Figure 4.1.4: AR DNP joint predictives in layers 0 (top left), 1 (top right), 2 (bottom left) and 3 (bottom right) for a benchmark task with $|\mathcal{D}_c| = 4$. The AR deployment process starts in layer 3, where a prediction is made by conditioning on the context set. Samples from this prediction are used to populate the third channel of the auxiliary context ($\mathcal{D}_a^3$), which is then conditioned upon in layer 2. This procedure is repeated until the full auxiliary context is generated, and subsequently utilised (in layer 0) to predict at the original target locations.

4.1.3 Noised GP

Figure 4.1.5 below compares the performance of an AR DNP (Joint ConvGNP architecture, 3 layers, 0.08 maximum noise variance, 5000 AR samples) with that of the relevant available baselines (ConvCNP, ConvGNP, AR ConvCNP and Diagonal GP) across EQ GP benchmark tasks with different context sizes.

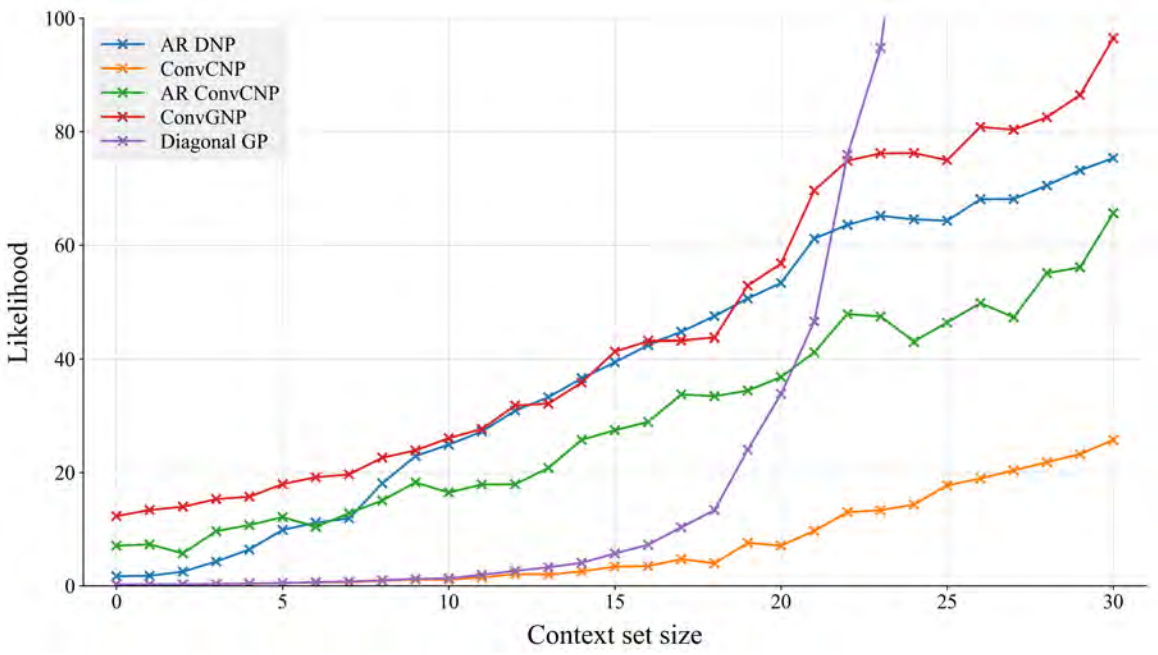


Figure 4.1.5: Comparison of AR DNP performance against relevant baselines on the Gaussian process benchmark meta-dataset. Each data point is obtained by averaging the likelihoods obtained across 10 tasks with a specific context size $|\mathcal{D}_c|$. The "diagonal GP" curve denotes the likelihood attained by the exact GP predictive (with correlations removed).

As one can easily note, Figure 4.1.5 paints quite a different picture compared to Figures 4.1.1 and 4.1.3. In fact, whilst the AR DNP model still vastly outperforms the basic ConvCNP, and even the AR ConvCNP for context sizes $|\mathcal{D}_c| \geq 5$, is is outclassed by the vanilla ConvGNP for both low and high context sizes (interestingly, the two models are mostly matched in predictive power for $10 \leq |\mathcal{D}_c| \leq 20$). Unsurprisingly, all models besides the vanilla ConvCNP initially perform significantly better than the diagonal GP baseline (due to the fact that they model correlations through a full covariance matrix), but the diagonal GP predictive still pushes ahead for high context scenarios (which is expected, considering that a GP conditioned on many points would be able to very accurately narrow down on the underlying function, with little uncertainty, and, thus, high predictive log-likelihoods).

Before delving into the implications of the data shown above, let us first, for the last time in this Chapter, look at an illustration of the process responsible for generating one of the mixture components of a random task's joint predictive (Figure 4.1.6).

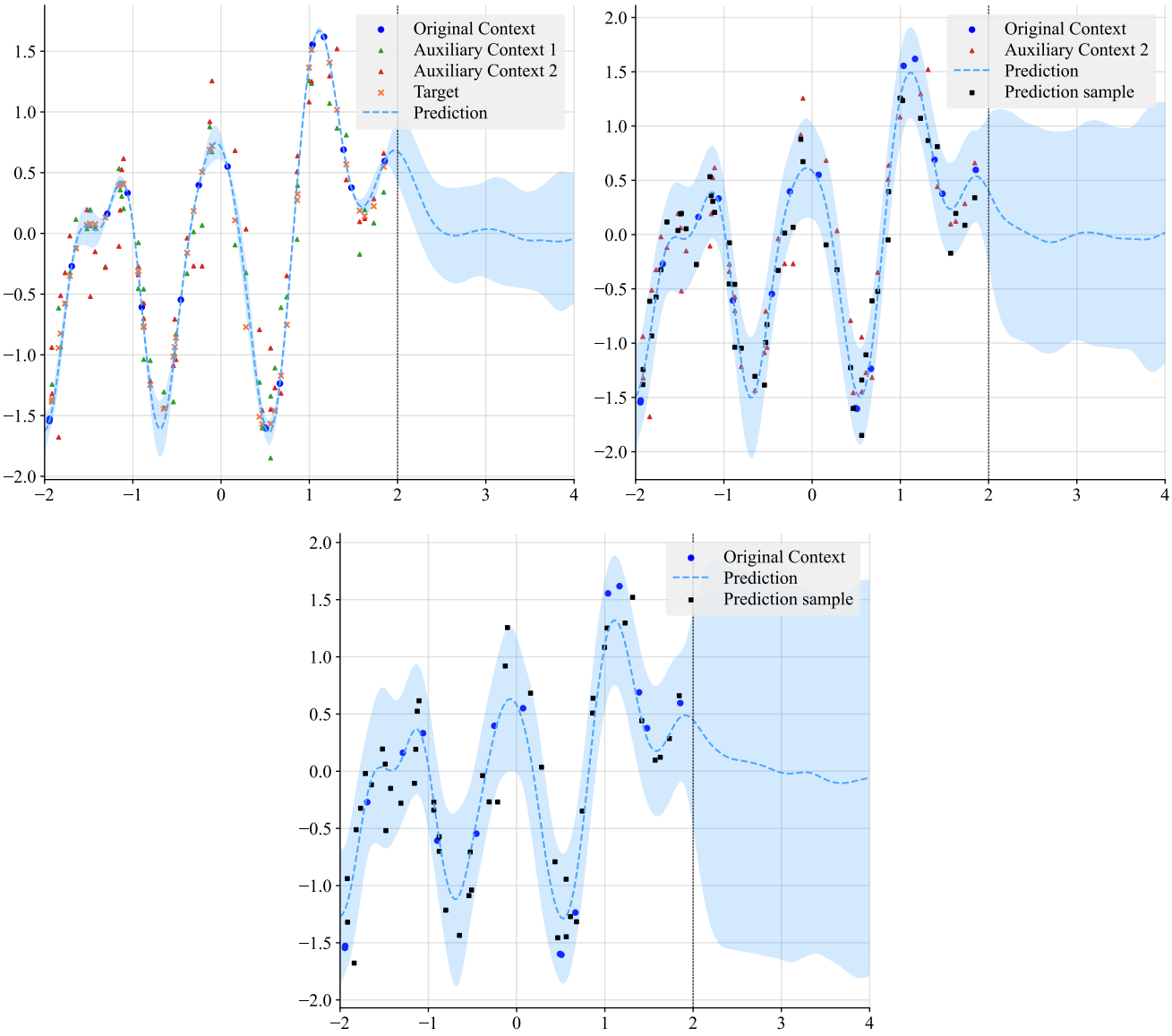


Figure 4.1.6: AR DNP joint predictives in layers 0 (top left), 1 (top right) and 2 (bottom) for a benchmark task with $|\mathcal{D}_c| = 17$. The AR deployment process starts in layer 2, where a prediction is made by conditioning on the context set. Samples from this prediction are used to populate the second channel of the auxiliary context ($\mathcal{D}_a^2$), which is then conditioned upon in layer 1. Similarly, the prediction made in layer 1 is sampled from to populate the first channel of the auxiliary context ($\mathcal{D}_a^1$). Lastly, both $\mathcal{D}_a^1$ and $\mathcal{D}_a^2$ are used (alongside $\mathcal{D}_c$) to generate predictions at the original target locations in layer 0.

Given the results presented in Figure 4.1.5, AR DNPs regrettably appear to not be particularly well-suited to modelling data generated by a GP (although one should note that, despite their not outperforming the state of the art, the results they produce are still quite decent). This is somewhat surprising, as GP modelling is usually handled relatively easily by most other members of the NPF. At the time of writing, the reason behind this peculiar phenomenon is not entirely understood. Indeed, this behaviour may be caused by something as banal as inappropriate hyperparameter settings, or may constitute an intrinsic limitation of AR DNPs.

Naturally enough, identifying the source of this disparity in performance is one of the top priorities for future research endeavours. For the time being, however, some preliminary hypotheses based on empirical observations are advanced in Subsection 4.1.4 below.

4.1.4 General Observations

Generally speaking, the likelihood comparisons depicted in Figures 4.1.1, 4.1.3 and 4.1.5 support the conclusion that, under the right circumstances, Autoregressive Diffusion Neural Processes can offer a significant boost in predictive power, and, in many cases, even **outperform the current state of the art in the NP literature**. This is, of course, quite an impressive feat, especially when considering the relative simplicity and elegance of the process upon which these models are based. What is even more interesting, AR DNPs can also, at least in principle, be **more computationally efficient to deploy than AR CNPs** (although, as discussed in Section 4.3, this point is somewhat contentious, and its validity is highly dependant upon the specific modelling scenario in which one finds themselves).

Nevertheless, despite the generally positive results showcased above, two main concerns should be called to the reader's attention:

- For both the GP and the sawtooth dataset, AR DNPs seem to under-perform, compared to the state of the art, in the low context regime. This issue is significantly more pronounced in the former case, but still noticeable, for very low context sizes, in the latter. It is presently not clear whether this limitation is intrinsically tied to the AR DNP approach itself, or whether some clever design tweaks (e.g. including proportionally more low-context tasks in the training meta-dataset) could be employed to solve this problem. One should also note that, while, as demonstrated in Subsection 4.2.5, increasing the number S of AR samples does help, this comes at the cost of substantial increases in deployment runtimes (and might thus not always constitute a viable solution).

- The performance of AR DNPs on the GP dataset is, compared to the NP state of the art, relatively poor. As already mentioned in Subsection 4.1.3, the reasons behind this are, at present, not fully understood. However, by contrasting this phenomenon with the considerable improvements observed for both the sawtooth and the square wave dataset, one could hypothesise that AR DNPs are simply better suited to modelling *strongly non-Gaussian* data, and that their potential is thus less evident when benchmarked on a GP dataset. This theory is empirically supported not only by the mediocre EQ performance, but also by the fact that, as evidenced by Figures 4.1.1 and 4.1.3, the margin between AR DNP and AR ConvCNP is much larger when testing on the square wave dataset than when deploying on the sawtooth one (and, in some sense, the former data source is markedly less "Gaussian-like" than the latter[3]). Naturally, this is only speculation, and more extensive research on this topic shall be conducted in the future.

4.2 Key Hyperparameters Study

Having thoroughly analysed the performance of AR DNPs across all three selected meta-datasets, let us now turn our attention to examining how varying specific hyperparameters impacts these models' predictive power (again taking care to focus on likelihoods across varying context sizes). For the sake of brevity, only one plot (deemed to be most representative for the topic described) shall be included in each of the following Subsections, and most discussions (besides those in Subsections 4.2.1 and 4.2.2) shall thus revolve around a single dataset. The sawtooth benchmark has been selected to fill this role, as this was the first data source to be implemented in the codebase, and, hence, the one over which the most experiments have been run over the course of the project.

4.2.1 ConvGNP vs. ConvCNP

To start things off, let us examine how the choice of base NP model used to construct the AR DNP affects the overall system's performance. Given the desire to retain the aforementioned translation equivariance property, only two of the NP variants reviewed in Chapter 2 constitute viable options: namely, the ConvCNP and the ConvGNP. As a reminder, from a conceptual, implementation-agnostic standpoint, the main difference between the two lies in the way

[3]For the square wave, at any given input location, the marginal distribution will be *strongly* bimodal, with all output values taking on either a value of 0 (when in a trough) or 1 (when at a peak). On the contrary, for the sawtooth, the marginal distribution at any given input location will be more spread out, because the wave covers a wider range of values as it slopes up or down. Hence, whilst the marginals are, in both cases, decidedly non-Gaussian, one could say that the latter case is more "Gaussian-like" than the former.

the joint predictive is specified: whilst the former approach elects to make independent predictions at each target location (thereby yielding a diagonal covariance matrix), the latter attempts to also model correlations between target locations (producing a full covariance matrix in the process).

To empirically establish which of the two alternatives is better suited for the task at hand, various AR DNPs with identical hyperparameters (barring the choice of base model) were benchmarked against one another. The relevant results are summarised in Figure 4.2.1 below.

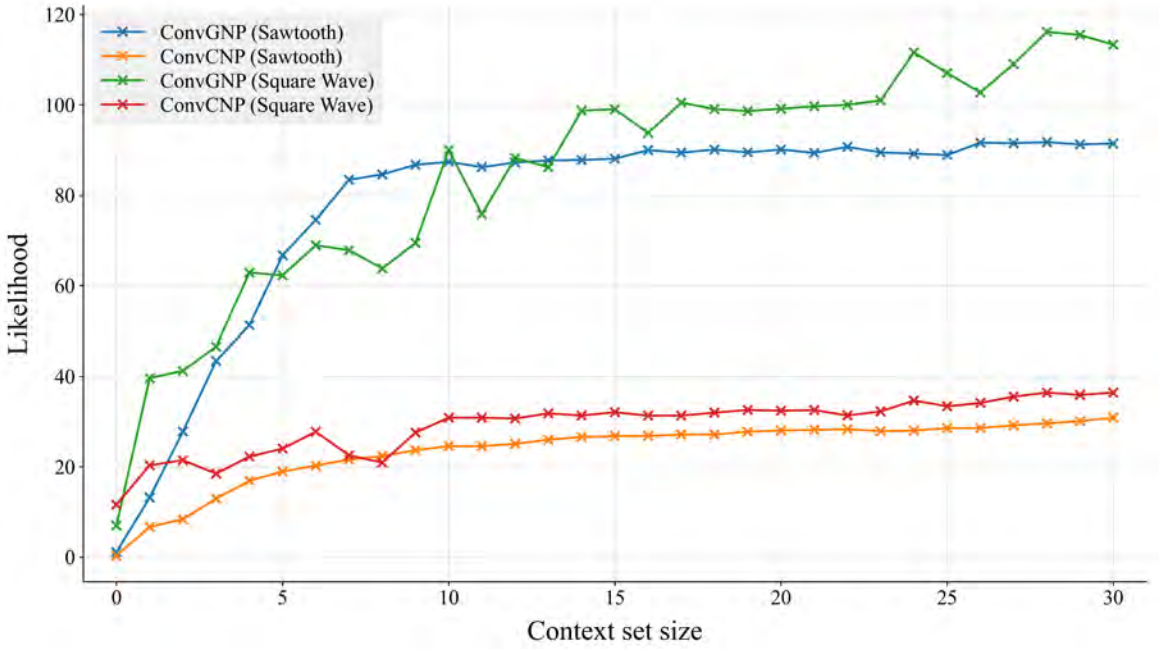


Figure 4.2.1: Comparison of ConvGNP-based AR DNP performance against ConvCNP-based AR DNP performance on both the sawtooth and the square wave meta-dataset (the GP meta-dataset is not included, as a better performance on the ConvGNP's part would likely be simply due to the highly Gaussian nature of the data source). The models' hyperparameters are set to the same values highlighted in Section 4.1 (i.e. joint 4 layers with 0.02 maximum noise variance for the sawtooth, and joint 4 layers with 0.06 maximum noise variance for the square wave), with the only difference being the use of 1000 AR samples rather than 5000 (to reduce deployment time).

As evidenced by the above curves, the ConvGNP-based AR DNP substantially outperforms its ConvCNP-based counterpart on both meta-datasets across virtually all context sizes. The reason behind this phenomenon is hypothesised to lie in the nature of the process employed to generate the augmented datasets utilised in training. Indeed, as described at length in Chapter 3, each element of $\mathcal{D}_a$ is generated by adding noise on top of a single, underlying functional sample. Hence, despite the fact that the additive Gaussian noise at each target location is

independent, modelling correlations in the data is still likely to yield better predictions, which, in turn, renders each channel in the augmented dataset more useful to the overall AR DNP system. This is a similar advantage to the one that a regular ConvGNP has over a regular ConvCNP, with the difference that, since AR DNPs generate multiple predictions (one per layer) for each AR sample, the positive effect of a full covariance matrix is compounded at each step, and, thus, greatly amplified overall.

4.2.2 Split vs. Joint

Having established that ConvGNPs are a better building block for AR DNPs, let us now turn our attention to the second key design choice one must make when constructing these models: that is, whether to implement a split or a joint architecture. To inform this decision, two split/joint AR DNP pairs with identical hyperparameters were benchmarked against one another on both the sawtooth and the GP meta-dataset. The relevant results are summarised in Figure 4.2.2 below.

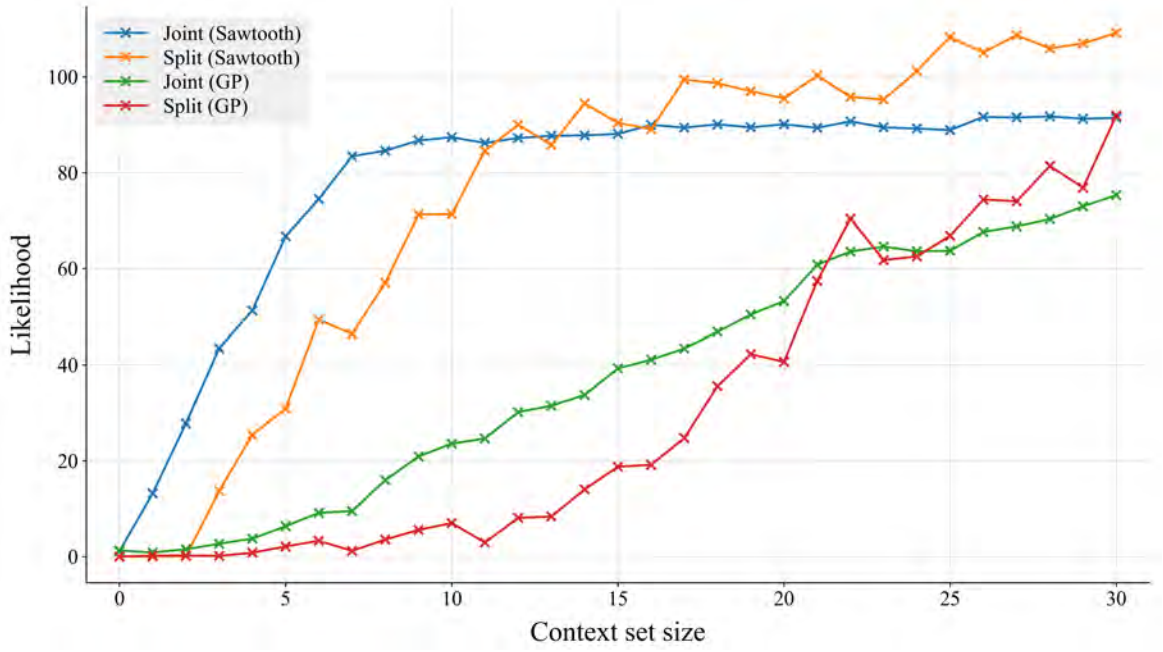


Figure 4.2.2: Comparison of split AR DNP performance against joint AR DNP performance on both the sawtooth and the GP meta-dataset. The models' hyperparameters are set to the same values highlighted in Section 4.1 (i.e. ConvGNP with 4 layers and 0.02 maximum noise variance for the sawtooth, and ConvGNP with 3 layers and 0.08 maximum noise variance for the GP). The joint models make use of 1000 AR samples, whilst their split counterparts utilise only 200 (due to the fact that, as discussed below, these are slower to deploy).

As demonstrated by the above curves, on both meta-datasets, the joint AR DNP outperforms its split counterpart for smaller context sizes, while the opposite is true for larger values of $|\mathcal{D}_c|$. With that being said, the improvement for large $|\mathcal{D}_c|$ is not particularly substantial, and, in most cases, one would be more interested in better predictions in the low context regime, rather than in marginally less uncertain ones in high context scenarios. Additionally, split AR DNPs come with quite a few undesirable qualities:

- For an AR DNP with $F+1$ layers, $F+1$ separate models must be trained. Thanks to task masking, each of these various NPs is fit independently from its peers, meaning that training can be efficiently parallelised. However, this still incurs much greater computational costs that the joint case, and this overhead only gets worse as model depth increases.

- The trained model is significantly more unwieldy, being composed of $F+1$ times as many parameters as its joint counterpart. For large AR DNPs, this can cause issues with GPU memory limitations, and occasionally even noticeably slow down the deployment procedure, as a different model must be loaded and run to deal with each fidelity level.

For the above reasons, joint AR DNPs are, overall, deemed to be the superior option, and, as a result, our investigations in the remainder of this Chapter shall focus exclusively on this architecture.

4.2.3 Model Depth

Armed with the above knowledge, let us continue our hyperparameter study by focusing on model depth (i.e. number of layers) next. To examine the influence of this setting on predictive power, four AR DNPs with 3, 4, 5 and 6 layers (and identical designs besides this) were benchmarked against one another on the sawtooth meta-datasets. The ensuing results are presented in Figure 4.2.3.

As evidenced by the curves shown below (with the exception of the one reporting the likelihoods obtained by the six-layers model), as the number of layers is increased, the low-context performance appears to slightly increase, at the price of a somewhat sizeable decrease in high-context performance (which, once again, based on the specific application, might make for an advantageous trade-off). Generally speaking, however, as long as the depth is kept at a reasonably low value (e.g. 3 or 4), its effect on the overall system appears to be relatively negligible (or, at the very least, certainly less impactful than either the ConvGNP/ConvCNP or the split/joint choice).

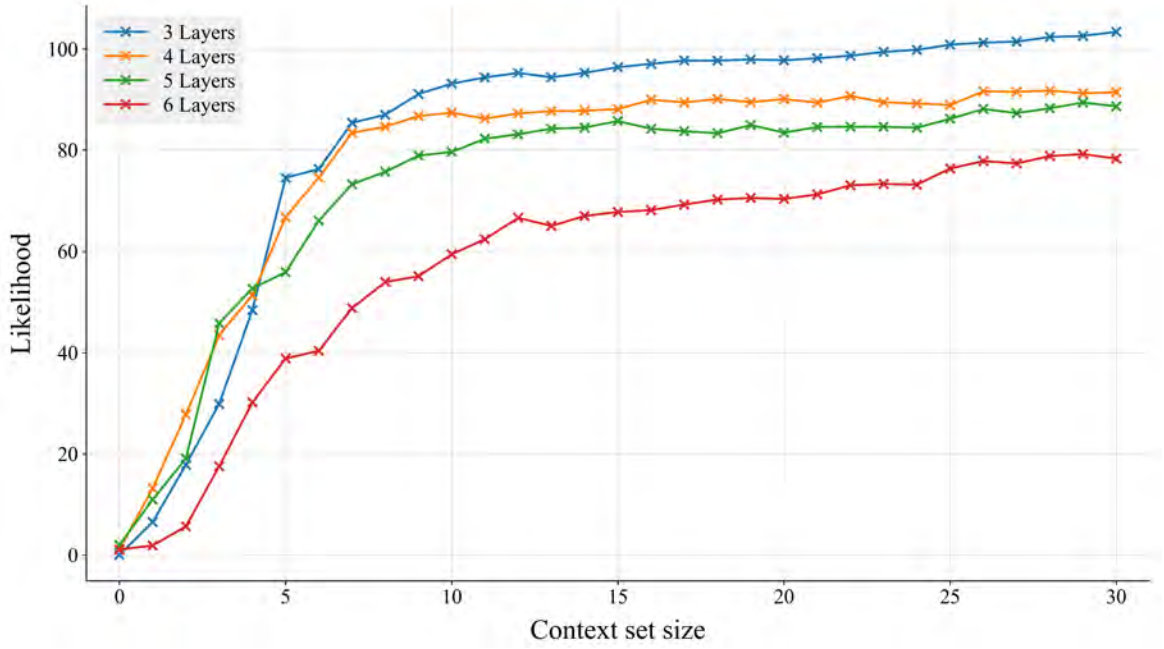


Figure 4.2.3: Performance comparison of AR DNPs with 3, 4, 5 and 6 layers on the sawtooth meta-dataset. The models' hyperparameters are set to the same values highlighted in Section 4.1 (i.e. joint ConvGNP-based with 0.02 maximum noise variance), with the only difference being the use of 1000 AR samples rather than 5000 (once again to reduce deployment time).

Interestingly, the performance of the six-layers model is decidedly worse than that of its peers across all context sizes $|\mathcal{D}_c|$. This is, all things considered, quite surprising, as one would intuitively expect the overall predictive power to increase (or, at least, stay constant) as the depth is augmented (in an analogous way as to how, generally speaking, the performance of diffusion models improves when making use of more timesteps). This discrepancy might be caused by the fact that, as more channels are added to the joint AR DNP, each of them gets exposed to comparatively fewer tasks (since the total number of training tasks is held constant), and, thus, develops less expressive power overall.

Before proceeding, one should also note that the chosen setting of the maximum noise variance hyperparameter (discussed in the next Subsection) has been found to affect which depth leads to best performance, with increased noise occasionally leading to deeper models outclassing their shallower counterparts. This means that higher numbers of layers should not at all be disregarded, and that, to achieve the best possible results, one should jointly optimise depth and noise variance whenever conducting hyperparameter search.

4.2.4 Maximum Noise Variance

Having examined the impact of model depth, let us now proceed by exploring that of the maximum noise variance hyperparameter. As a reminder, this setting indicates how much total noise variance should be present in the noisiest fidelity level (which is used to compute the appropriate value of β for each layer, as already explained in Subsection 3.2.2). Repeating a procedure similar to that described in the previous paragraphs, five AR DNPs with 0.02, 0.04, 0.06, 0.08 and 0.1 maximum noise variance (and identical designs besides this) were benchmarked against one another on the sawtooth meta-datasets. The results are presented in Figure 4.2.4 below.

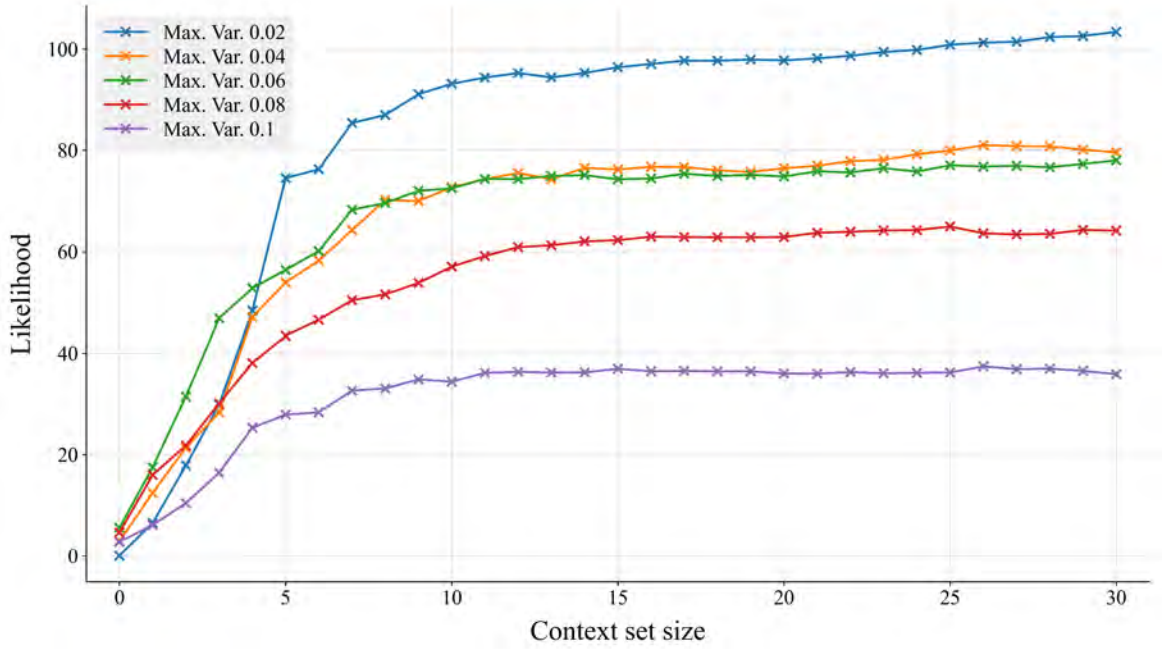


Figure 4.2.4: Performance comparison of AR DNPs with 0.02, 0.04, 0.06, 0.08 and 0.1 maximum noise variance on the sawtooth meta-dataset. All models are three-layers joint ConvGNP-based deployed for 1000 AR samples.

Similarly to what was observed for model depth in Subsection 4.2.3, generally speaking, as the maximum noise variance increases, the likelihoods obtained in the low-context regime improve, but the performance in high-context scenarios regrettably degrades at the same time. This outcome makes intuitive sense, as the added noise causes the AR DNP to yield more uncertain predictions, which, in turns, lowers the likelihood for high $|\mathcal{D}_t|$ (where more confident predictions would be warranted), but also boosts this quantity for low $|\mathcal{D}_t|$ (where more conservative estimates are more ideal). Additionally, the results obtained for a

maximum variance of 0.1 suggest that increasing the noise beyond a certain threshold has a negative effect regardless of context size (which is, again, a perfectly sensible finding).

4.2.5 Number of AR Samples

To close out this Section, let us assess the performance differentials caused by increasing the number S of AR samples (and, thus, of mixture components in the final joint predictive) generated during the deployment procedure. To do this, the same model (specifically, the joint ConvGNP-based AR DNP presented in Subsection 4.1.1) was deployed over the sawtooth benchmark meta-dataset for varying values of S. The corresponding results are showcased in Figure 4.2.5 below.

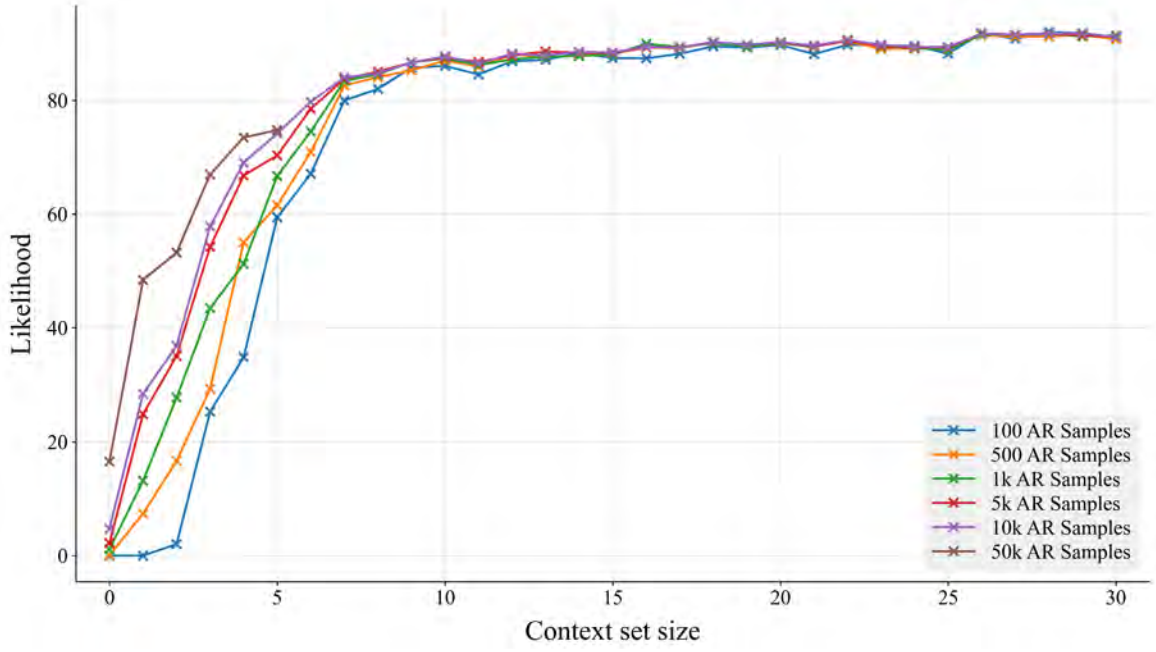


Figure 4.2.5: Comparison of performances obtained by deploying the same AR DNP (four-layers joint ConvGNP-based with 0.02 maximum noise variance) on the sawtooth meta-dataset for increasing values of S. The curve for 50000 AR samples only includes datapoints for context sizes 0-5, as the corresponding run was cut short to save on compute (since, as exemplified by the above, no significant improvements are obtained for large S in the high-context regime anyway).

As evidenced by the above curves, increasing the number of AR samples leads to quite a significant performance boost in the low-context regime. Since, as discussed in Subsection 4.1.4, AR DNPs can occasionally struggle to outperform the state of the art (AR CNPs) in low-context scenarios, these findings are quite encouraging, demonstrating that, as long as

higher computational costs do not constitute a hindrance, higher likelihoods can, at least in principle, be attained (as one would expect, considering that, as the value of S increases, the accuracy of the Monte-Carlo estimate of the integral given in Equation 3.3.3 improves).

4.3 Considerations on Computational Costs

Before concluding this Chapter, one would be remiss to not spend a few words discussing how the computational costs incurred by AR DNPs compare to those arising when dealing with AR CNPs.

When considering training costs, the two NP frameworks are, in general, quite evenly matched (as long as one limits oneself to employing joint AR DNPs, for the reasons already laid out in Subsection 4.2.2). Naturally, Autoregressive Diffusion Neural Processes do require a bit more effort to train (since the relevant ConvGNP architectures employ more channels and, consequently, have slightly higher parameter counts), but this difference has been found to be relatively negligible overall.

More interestingly, on the other hand, AR CNPs and AR DNPs scale *very* differently at test time with respect to the cardinality $N_t = |\mathcal{D}_t|$ of the target set. In fact, using a base model's forward pass (FP) as a reference unit:

- An AR CNP performs N_t forward passes (as each target prediction calls for the model to be re-conditioned upon $\mathcal{D}_c$ and the previously-obtained predictions).
- An AR DNP with $F+1$ layers taking S AR samples performs $S(F+1)$ forward passes (since the overall system is run S times, each of which requires $F+1$ forward passes to get through all of the fidelity levels).

Hence, as the target set size N_t increases, the number of FPs *scales linearly* in the former approach, but *remains constant* in the latter. Thus, assuming a single FP requires the same amount of operations in both scenarios[4], deploying an AR DNP will be cheaper as long as the following relation is satisfied:

$$S(F+1) < N_t \tag{4.3.1}$$

[4] In practice, this is not exactly the case (since the two systems condition on different amounts of points), but the difference is likely negligible for large target sets, and still does not detract from the fact that the computational cost of deploying an AR DNP only scales with N_t insofar as the forward pass' cost does (whereas, in AR CNPs, the N_t factor also affects the *number* of forward passes, alongside their complexity).

As a result, AR DNPs are more computationally efficient than AR CNPs when predictions must be made at a large number of target locations, with this effect becoming more and more apparent as N_t increases. Naturally enough, for the toy synthetic datasets examined in this project (in which $N_t = 50$), this advantage is hard to demonstrate (and, actually, AR CNPs do end up being less expensive for high values of S and F). In real-life applications, however, N_t often takes on high enough values[5] for Equation 4.3.1 to be satisfied for appropriate settings of S and F, making AR DNPs particularly enticing not only for their sheer modelling power, but also for their comparatively small deployment runtimes.

[5]Two main examples come to mind. When working with image data (e.g. in the case of image completion, a popular NP benchmark), the number of target points grows very rapidly as resolution is increased. Similarly, when modelling environmental features (e.g. cloud coverage, temperature or precipitation) on the Earth's surface, the number of target points required to generate predictions at small enough scales is very large (and, indeed, as specifically stated in Bruinsma et al. [2023b], AR CNPs are simply not viable when sampling functions on a very fine grid).

Chapter 5

Conclusion

5.1 Discussion

In summary, the work presented in this thesis set out to theorise, implement and evaluate a novel Neural Process framework in an attempt to address some notable shortcomings identified in the current state of the art (AR CNPs). The resulting Autoregressive Diffusion Neural Process models have been shown to possess both enticing theoretical properties and impressive practical advantages:

- As described in Chapter 3, AR DNPs yield **correlated**, highly **non-Gaussian** and, under the right circumstances, **consistent** predictions.
- As demonstrated in Chapter 4, these models attain **state-of-the-art performance** on two of the three meta-datasets examined throughout the project.
- As also discussed in Chapter 4, when operating in scenarios in which large target sets must be handled (that is, in most practical, real-world applications), AR DNPs also outclass the AR CNP state of the art in terms of **computational efficiency**.

Given the above accomplishments, the author feels confident in stating that, generally speaking, the project has been quite successful in achieving its original goal of extending the Neural Process Family with a new, high-performance class of models which retain most of the advantages of their predecessors, whilst also making considerable strides in mitigating their limitations.

The above notwithstanding, this work could, naturally enough, benefit from further research endeavours. We thus conclude this dissertation by listing (and briefly expanding upon) potential avenues of future work in the following Section.

5.2 Future Work

With more time at the author's disposal, the following research directions would be explored in an effort to complement the work presented in this document:

- As mentioned in Subsection 3.2.3, more AR DNP experiments could be conducted on other kinds of regression datasets (besides synthetic 1D). This could include real 1D and 2D data drawn from a variety of subject areas (as with other NPs, applications in climate sciences appear especially enticing, and should be extensively investigated). In particular, any scenario in which high-resolution sampling is required (i.e. any task with very large target set sizes) should be carefully inspected, as AR DNPs show uniquely great promise in these specific data regimes.

- As discussed in Subsection 3.2.1, the choice of input locations for the auxiliary dataset $\mathcal{D}_a$ dictates whether or not the overall AR DNP approach generates consistent predictions. For ease of implementation, the results presented in this document were obtained by employing versions of $\mathcal{D}_a$ that cause the corresponding models to renounce this property (by violating the marginalisation invariance condition). In the future, this design aspect should be studied further, in an effort to establish whether consistency can be achieved without significant losses in terms of performance, and whether this property is even, in actuality, of any practical interest in the first place.

- As stated in Subsection 3.3.2, AR DNPs were designed to be a direct superset of existing NPs. Specifically, if one introduces *intra-layer* AR predictions alongside the already present *inter-layer* AR procedure, the behaviour of AR CNPs can be recovered. Whilst this approach has briefly been examined during the project (with mixed results), a more thorough investigation should be conducted to establish whether this idea could enable one to attain even more impressive performances at the price of increased computational costs.

- As reported in Section 4.1, in an effort to decrease the design space over which to conduct model tuning, certain hyperparameters (e.g. the chosen U-Net architecture) were fixed to pre-determined values. More experiments could be carried out to assess

whether any partially overlooked setting happens to have an unexpectedly profound impact on AR DNP behaviour.

References

Wessel Bruinsma, James Requeima, Andrew Y. K. Foong, Jonathan Gordon, and Richard E. Turner. The Gaussian Neural Process. In *3rd Symposium on Advances in Approximate Bayesian Inference*, 2021.

Wessel Bruinsma, Stratis Markou, James Requeima, and Tom Andersson. neuralprocesses. https://github.com/wesselb/neuralprocesses, 2023a.

Wessel Bruinsma, Stratis Markou, James Requeima, Andrew Y. K. Foong, Tom Andersson, Anna Vaughan, Anthony Buonomo, Scott Hosking, and Richard E. Turner. Autoregressive Conditional Neural Processes. In *11th International Conference on Learning Representations*, 2023b.

Yann Dubois, Jonathan Gordon, and Andrew Y. K. Foong. Neural Process Family. https://yanndubs.github.io/Neural-Process-Family, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *33rd International Conference on Machine Learning*, 2016.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Teh, Danilo Rezende, and Ali Eslami. Conditional Neural Processes. In *35th International Conference on Machine Learning*, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural Processes. *ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018b.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E. Turner. Meta-Learning Probabilistic Inference for Prediction. In *7th International Conference on Learning Representations*, 2019.

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional Conditional Neural Processes. In *8th International Conference on Learning Representations*, 2020.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583 – 589, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive Neural Processes. In *7th International Conference on Learning Representations*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Tuan Anh Le, Hyunjik Kim, Marta Garnelo, Dan Rosenbaum, Jonathan Schwarz, and Yee Whye Teh. Empirical Evaluation of Neural Process Objectives. *NeurIPS 2018 Workshop on Bayesian Deep Learning*, 2018.

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

Stratis Markou, James Requeima, Wessel P. Bruinsma, Anna Vaughan, and Richard E. Turner. Practical Conditional Neural Processes Via Tractable Dependent Predictions. In *10th International Conference on Learning Representations*, 2022.

Bernt Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Springer, 6th edition, 2014.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484 – 503, 2016.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *32nd International Conference on Machine Learning*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30, 2017.