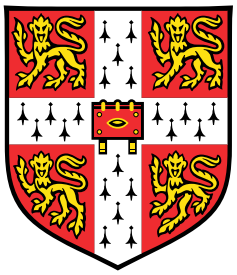
Autoregressive Conditional Neural Processes



Anthony Buonomo

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Queens' College August 16, 2022

This work is dedicated to my parents who inspire me with their abundant care and tenacity.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 15,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Anthony Buonomo
August 16, 2022

Declaration

The software used in this project builds upon and extends software from the *neuralprocesses* library [Wessel et al., 2022]—in particular by adding a few modules for the developing and testing the AR procedure detailed in this report. It also uses standard python libraries for machine learning (*pandas*, *numpy*, etc) and a few for audio preprocessing (*librosa*[McFee et al., 2015], *pydub*[Robert et al., 2018]). These software were used for Chapter 3 and Chapter 4.

Anthony Buonomo
August 16, 2022

Word Count: 14788

Acknowledgements

I would like to thank Richard Turner, Stratis Markou, Wessel Bruinsma, and James Requeima for all of their insights, guidance, and inspiration.

Abstract

Neural Processes are a rich family of meta-learning models which produce flexible priors and uncertainty-aware predictions. Typically these models, however, are limited to producing Gaussian marginal distributions. We show how the the CNP output marginal distributions can be reformulated as a Monte Carlo integration over autoregressively generated trajectory samples—yielding a GMM marginal predictive with greater ability to model multi-modal and heavy-tailed distributions—all with no changes to the CNP training procedure. We show that this method can improve held-out log-likelihoods on three tasks: two synthetic and one real-world. We explore the design space for this method, characterizing when it is most useful.

Table of contents

List of figures

List of tables

Nomenclature

C The context set

D_C The context set

D_T The target set

M The number of trajectory samples

Q_θ The trained ConvCNP model

R The trajectory length

Ξ A collection of tasks (i.e. a *meta-dataset*)

ξ A task consisting of a context set and a target set (D_C, D_T)

p The ground truth distribution

r The trajectory generator, the distribution from which trajectory inputs are drawn

CNP Conditional Neural Process

Card Cardinality

ConvCNP Convolutional Conditional Neural Process

Stochastic Process a distribution over functions

Chapter 1

Introduction

Though modern supervised deep learning models have achieved impressive performance on a variety of tasks [Lecun et al., 2015] (including vision and natural language tasks), there are some domains where their achievements have been more measured. For instance, these methods are not well-adapted to handle data regimes with multiple small data sets which are related (i.e. which are generated from the same underlying process). Meta-learning models help alleviate these issues by "learning to learn" [Lake et al., 2015]. In other words, meta-learning models are designed to learn from many datasets and make predictions on unseen datasets using contextual information.

There are many domains where this sort of "learning-to-learn" is a more natural fit to the data at hand. For example, many healthcare applications collect temporal readings from individual patients. The readings for each patient over time constitute an individual dataset. These datasets (collectively called the meta-dataset), come from the same underlying process and, therefore, they share common features. Meta-learning methods have proven effective in modeling electroencephalogram (EEG) data from multiple different patients (each corresponding to a dataset)[Markou et al., 2022]. In this setting, meta-learning was shown to outperform supervised methods which learn from one dataset at a time.

Because each individual dataset which makes up a given task (ex. forecasting EEG readings for a single patient) typically only has a small amount of data, the precise value of a prediction often cannot be pinned down with a very high confidence. Therefore, it is critical that the meta-learning strategy at hand can quantify its uncertainty in addition to achieving high accuracy. This quality of uncertainty-awareness is especially important for safety-critical applications which inform decision-makers. For example, one may like to estimate the likelihood of a combination of environmental factors (soil desiccation, rainfall, river flow volume) resulting in catastrophic flooding. Or, perhaps one would like to estimate the likelihood that a given patient may have blood clotting after a knee surgery based on

various bio-metrics. Typically, the likelihood of such events is low, but the level of risk is contingent upon precisely how likely or unlikely these events are. Many modern deep learning systems fail in this regard due to their poor uncertainty calibration (typically, their overconfidence).

Among uncertainty-aware meta-learning methods [Nguyen and Grover, 2022], Conditional Neural Processes (CNP)[Garnelo et al., 2018a] are rich models with many desirable features. These models flexibly model distributions over functions (like Gaussian Processes) while using the advantages of deep-learning to flexibility adapt to the data at hand. They do this while providing well-calibrated uncertainty estimates, and–unlike many meta-learning methods like Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017]–they make quick inferences at test time. Moreover, they are trained using a simple maximum likelihood procedure–avoiding the complications of other training procedures like Variation Inference (VI)[1].

These CNP models also elegantly handle off-the-grid and missing data, making them particularly well-adapted for many climate and healthcare related-tasks (and other spatio-temporal tasks, more generally). For example, Convolutional Conditional Neural Processes (ConvCNPs) [Gordon et al., 2019] have been shown to be effective for multi-site statistical downscaling of temperature and precipitation [Vaughan et al., 2021]–allowing predictions at arbitrary off-the-grid locations. Additionally, Conditional Neural Processes (CNPs) have been found effective at modeling Active Galactic Nuclei (AGN) light curves, a use-case which showcases their ability to model sparse, gapped, and irregular data [Cvorovic-Hajdinjak et al., 2021].

Though vanilla CNPs are prone to underfitting, the CNP architecture can be combined with other architectures to introduce specific inductive biases which take advantages in the symmetries in the data at hand [Kawano et al., 2021][Holderrieth et al., 2021]. The Attentive Neural Process [Kim et al., 2019] assumes that certain context points will be more or less relevant to predicting a given unobserved location. It therefore uses an attention mechanism which learns to attend to relevant (typically nearby) context points when making predictions. Meanwhile, the ConvCNP introduces a translation equivariance inductive bias using Convolutional Neural Networks (CNNs). Both of these architectures restrict the hypothesis space and avoid many of the underfitting issues which CNPs often encounter.

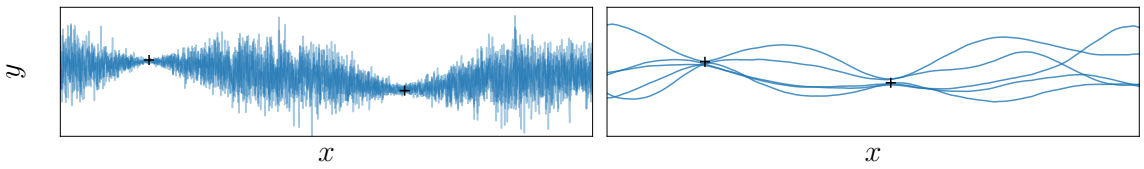


Fig. 1.1 The ConvGNP (right) makes dependent predictions and allows for drawing of function samples. The ConvCNP (left), on the other hand, makes independent predictions. Reprinted from Markou et al. [2022].

1.1 Limitations

Though CNPs gracefully handle irregularly sampled and missing data, they suffer from a few major drawbacks. First, CNP models assume independence in the output predictions–referred to as *mean-field* predictions [Markou et al., 2022]. Therefore, CNPs are unable to model dependencies between outputs and cannot produce coherent function samples. The performance of the models suffers, especially in applications which require dependent samples. For example, when estimating flood risk on may want to know whether precipitation is expected to remain above or below certain thresholds for certain periods of time. Due to their mean-field predictives, CNPs can provide answers with well-calibrated uncertainties about amounts of rainfall at particular times, but cannot provide these uncertainties when asked about spans of time–which would require dependencies in the outputs. In Figure 1.1, an example from Markou et al. [2022] can be seen which has outputs with and without dependencies.

Several adaptations to the standard CNP model have been proposed to introduce dependencies between outputs. For example, the Neural Process (NP) [Garnelo et al., 2018b] introduces a latent variable to model output dependencies. This latent variable, however, makes the likelihood analytically intractable, so NPs are trained using approximate inference (namely, Variational Inference). Training adequately while using Variational Inference often requires much expert knowledge and careful tuning–making it desirable to avoid, if possible.

The Gaussian Neural Process (referred to as FullyConvGNP)[Bruinsma et al., 2021] builds on the ConvCNP, using *translation equivariance* while modeling predictive distributions directly with Gaussian Processes (GPs) [Williams and Rasmussen, 2006]. In this way, the FullyConvGNP allows for dependencies in the outputs with an analytically tractable maximum likelihood training objective. A variant of the Gaussian Neural Process (GNP) which

[1] Latent variable variants of the CNP do require VI and, as a consequence, are more difficult to train[Garnelo et al., 2018b].

directly parametrizes the the covariance of a Gaussian predictive process, was developed to alleviate some of the computational issues with FullyConvGNPs [Markou et al., 2022].

While the GNP and FullyConvGNP perform well and provide coherent samples and output dependencies, they are limited to Gaussian predictions, making them inappropriate for many tasks. For example, distributions for daily precipitation are asymmetric, with zero density for negative values and a high density at zero millimeters. In this scenario, Gaussian predictives would have a mode shifted positive from zero millimeters, with significant probability mass over negative precipitation values.

1.2 Contributions

In this work, we explore using autoregressive (AR) sampling to model non-Gaussian marginal distributions. The method we explore uses ConvCNP architectures trained in a traditional manner (i.e. using a factorized joint for the loss) and only requires modification at test time, thereby avoiding potentially expensive and complicated training procedures. We show that our AR procedure can improve held-out log-likelihood scores on several tasks: two synthetic and one real-world. We also explore this procedure's parameter space, provide insights regarding when this procedure is particularly useful, and when it should be avoided.

In our experimental design, we do not directly address the issue of modeling output dependencies, but we propose how our work could be extended upon in order to do so.

1.3 Structure

In order to ground the discussion of this method, we start Chapter 2 by introducing meta-learning and Stochastic Processes. We then describe how CNPs fit into this paradigm and we detail their architecture and training procedure. We then discuss the some drawbacks of CNPs, and describe how ConvCNPs mitigate these drawbacks by introducing the translation equivariance inductive bias. Then we connect more closely to the problem we seek to solve–granting greater flexibility to CNP output marginal predictives.

In Chapter 3, we provide a sketch and a proof showing the potential utility of our autoregressive (AR) procedure. We then introduce the data generators for our experiments, and discuss how exactly we train ConvCNP models on these separate stochastic processes.

In Chapter 4, we provide tables showing the performance of the AR procedure when compared to the standard vanilla testing procedure, and the performance of a naive baseline model. We examine these performance differences using different parameters and different context sizes—and attempt to build intuition through figures showing these differences.

In Chapter 5, we discuss some potential future experiments which could lead to greater understanding of this AR procedure and its utility. We discuss how our work could be extended upon to model output dependencies, and how one might train a CNP model in a manner which uses our procedure in the loss function construction.

Chapter 2

Background

2.1 Metalearning

In traditional supervised learning, we are concerned with predicting some values of interest given observations. For example, suppose an astronomer directs a telescope to gather observations of the sun over the course of a month. She gathers a dataset with spatio-temporal inputs (locations around the sun over time) and outputs which represent solar activity (magnetic activity, spectra, etc.). After gathering data for a few days, she is interested in predicting the solar activity in the coming days.

We refer to these observations as a context set D_C and the data points we seek to predict as the target set D_T. Both sets consists of input, outputs pairs $D_C = \{x_j, y_j\}_{j=1}^{C}$ and $D_T = \{x_j, y_j\}_{j=1}^{T}$, where C, and T are the cardinalities of D_C and D_T, respectively. In this case x represents a particular location in space and time on sun, and y represents the solar activity at the time and location. We refer to these two datasets together as a task ξ where $\xi = (D_C, D_T)$.

In traditional supervised learning, we would approach this task ξ by fitting a model f_θ to the data we have gathered up to this point (D_C), and then using that model to make predictions of solar activity at new spatio-temporal inputs $\{x^* | \{x^*, y^*\} \in D_T\}$). The parameters of the model (θ) would be fit so as to minimize some loss function L of our predicted values of $f_\theta(x_j)$ and their true values in the context set:

$$\theta^* = \arg\min_\theta \left\{ \mathbb{E}_{\{x_j, y_j\} \in D_C} \left[L(f_\theta(x_j), y_j) \right] \right\} \tag{2.1}$$

Our astronomer then applies the model f_θ to make predictions for values in the target set D_T—where the true values of y in the target set (the future solar activity) are unknown. Our astronomer rejoices, having successfully created a model for forecasting solar activity based

on spatio-temporal information. Soon though, she laments that the model is tuned only for these particular observations and this particular time and place, and she thinks she may be able to create a superior model with more data.

Fortunately, astronomy is not a solo affair and many of the most impressive results come from collaboration and data sharing of observations from many telescopes in different places viewing the skies at different times. Let us now suppose that our astronomer has access to a collection of these datasets gathered from these disparate telescopes.

With this meta-dataset, our astronomer is now interested in a collection of tasks (a *meta-dataset* Ξ) coming from many separate observations:

$$\Xi = \{\xi_i\}_{i=1}^{N} \text{ where } \xi_i = (D_{C_i}, D_{T_i}) \tag{2.2}$$

At first, our astronomer fears that she must now train a series of new models, one for each task—a time consuming affair. She then realizes that all of these observations, tasks, come from the same underlying phenomenon, the sun. She realizes that she can utilize meta-learning in order to *learn how to learn*. In other words, she realizes she can train a meta-learning model ψ_ϑ which produces task-specific models $\hat{f}$ from context set D_C inputs—$\psi_\vartheta(D_C) = \hat{f}$. Like with supervised learning models, the meta-learning model parameters ϑ are trained by minimizing some loss function L, except this time L is a function of task-specific models f and tasks ξ—$L(f, \xi)$:

$$\vartheta^* = \arg\min_{\vartheta} \mathop{\mathbb{E}}_{\xi \sim \Xi_{train}} [L(\psi_\vartheta(D_C), \xi)] \tag{2.3}$$

In short, in traditional supervised learning, one uses the context set D_C of a single task ξ to train a model f_θ which maps from the inputs x of the target set D_T to predict outputs $\hat{y}$. In meta-learning, one uses many tasks Ξ to learn a meta-model which maps from tasks ξ to the single-task models themselves f.

In our astronomer's case, there is a key reason why it is sensible to use meta-learning in this manner: namely, the fact that all of the observations are expected to have some features in common because they arise from same underlying processes in the sun. In mathematical terms, the functions are drawn from the same distribution over functions—referred to as a *stochastic process*.

Now, let us suppose that our astronomer is interested in forecasting solar activity in order to assess the risk of geomagnetic storms impacting telecommunications. This risk assessment requires not just point predictions of solar activity, but rather probabilities of solar activity over time given observations (i.e. a new task ξ^*):

$$p(\{y_i\}_{i=1}^T|\{x_i\}_{i=1}^T, D_C) = \int p(\{y_i\}_{i=1}^T|\{x_i\}_{i=1}^T, D_C, f)p(f|D_C)df^1$$
$$\text{where } \{x_i, y_i\}_{i=1}^T = D_T \text{ and } (D_C, D_T) = \xi^* \tag{2.4}$$

So, our astronomer requires:

1. A mapping from observed context sets D_C to a stochastic processes conditioned on those context sets $p(f|D_C)$[Gordon, 2021].

2. A probability distribution of outputs $\{y_i\}_{i=1}^T$ given inputs of interest $\{x_i\}_{i=1}^T$ and a particular function f drawn from the conditional stochastic process $p(f|D_C)$.

In this discussion, we will only handle finite numbers of predictive outputs, and we will mostly concern ourselves with $p(\{y_i\}_{i=1}^T|\{x_i\}_{i=1}^T, D_C)$ directly (thus avoiding determining the full form of f).

2.2 Conditional Neural Processes

NB: We hereafter ease notation by referring to context sets as C, rather than D_C. We refer to the parameters of our CNP model as θ rather than ϑ. We overload the notation on p to refer to the hypothetical true probability distribution for whatever variable and whatever conditioning is present. We refer to the output distribution of the CNP model as Q_θ, an approximation of the true predictive. We refer to the set of target set (D_T) inputs ($\{x_i\}_{i=1}^T$), and outputs ($\{y_i\}_{i=1}^T$) as $\mathbf{x}_\tau$ and $\mathbf{y}_\tau$, respectively. When we refer to a single target point, we use $\{x^{(T)}, y^{(T)}\}$. The number of points in C is called N. We refer to the expected log-likelihood using $\mathscr{L}$, whereas we refer to its negative, the loss function, using L.

The *Conditional Neural Process* (CNP)[Garnelo et al., 2018a] is one archictecture which has been shown to work well for learning the mappings required for Equation 3.7. The architecture requires several considerations in its construction:

1. The model must produce predictive distributions given context sets and inputs of interest.

2. The model must operate on sets D_C and D_T. These sets can be of variable size and, as sets, they have no ordering. Therefore, the model should accept sets of varying sizes,

[1] $\{y_i\}_{i=1}^T$ and D_C are conditionally independent given f and $\{x_i\}_{i=1}^T$. f and $\{x_i\}_{i=1}^T$ are conditionally independent given D_C.

and the outputs should not be impacted by the ordering of these sets' representations on the computer—a feature called *permutation invariance*.

The CNP imbues these requirements in its encoder-decoder architecture. First, the encoder of the CNP assures that the model is *permutation invariant* with respect to the context set C. In particular, the encoder is of the form[Zaheer et al., 2017]:

$$r = Enc_\theta(C) = \frac{1}{N}\sum_{i=1}^{N} r_i \text{ where } r_i = h_\theta(x_i, y_i) \tag{2.5}$$

Where h_θ is a neural network (for example, a Multi-layer Perceptron). We see that permutation invariance is assured because of the commutative property of summation, wherein we average each context data point representation r_i.

We then pass this context set encoding r and a target inputs of interest x_τ into a decoder g_θ which generates parameters (μ, σ) for a Gaussian predictive:

$$(\mu, \sigma) = g_\theta(x_i, r) \text{ for all } x_i \in \mathbf{x}_\tau \tag{2.6}$$

In this manner, the CNP generates marginal predictive distributions conditioned on context sets and target points: $Q_\theta(y_i \mid C, x_i)$ for all $x_i \in \mathbf{x}_\tau$.

Now, the CNP model assures its permutation invariance to target sets D_T, by assuming a factorized distribution over its outputs:

$$Q_\theta(\mathbf{y}_\tau | C, \mathbf{x}_\tau) = \prod_{i=1}^{T} Q_\theta(y_i \mid C, x_i) \tag{2.7}$$

This factorization has the added benefit of simplifying training, but comes with the major drawback of crippling the CNP's ability to accurately assess joint predictives in many scenarios.

2.3 Training CNPs

CNPs can be trained by the procedure shown in Equation 2.3, where we have access to a meta-dataset of tasks Ξ which is partitioned into meta-train and meta-test sets, Ξ_{train} and Ξ_{test}, respectively. The meta-train set is split into many batches Ξ_{batch} for use in evaluating the loss and updating parameters. The loss function L to minimize is defined as the expected negative log-likelihood of the true data under the model with its current parameters; the loss is given by:

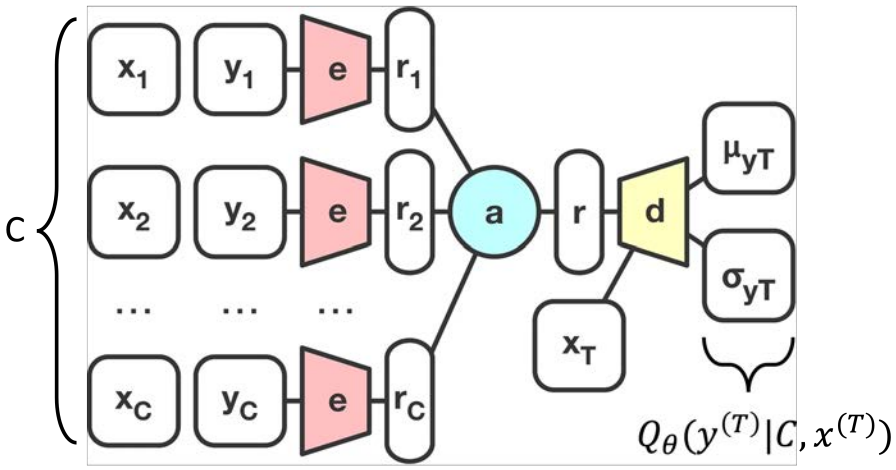


Fig. 2.1 Architecture for Conditional Neural Process model from Garnelo et al. [2018a]. Annotated with the notation we use to the context set (C), and the predictive marginal distribution ($Q_\theta(y^{(T)}|C,x^{(T)})$), where $C = D_C$ and $\{x^{(T)}, y^{(T)}\} \in D_T$.

$$L_{batch}(\theta) = -\mathop{\mathbb{E}}_{\xi\sim\Xi_{batch}}\left[\log Q_\theta\left(\mathbf{y}_\tau \mid C, \mathbf{x}_\tau\right)\right] \text{ where } \xi = (C, \mathbf{x}_\tau, \mathbf{y}_\tau) \tag{2.8}$$

Because, we assume our model Q_θ factorizes over the joint conditional predictive, we can equivalently express the loss as[2]:

$$L_{batch}(\theta) = \mathop{-\mathbb{E}}_{\xi\sim\Xi_{batch}}\left[\sum_{i=1}^{T}\log Q_\theta\left(y_i \mid C, x_i\right)\right] \tag{2.9}$$

The parameters are updated using this loss function and standard gradient descent methods[3].

2.3.1 Evaluating CNPs

We can evaluate the quality of the model by simply calculating the loss on the test set Ξ_{test}:

$$L_{test}(\theta) = \mathop{-\mathbb{E}}_{\xi\sim\Xi_{test}}\left[\sum_{i=1}^{T}\log Q_\theta\left(y_i \mid C, x_i\right)\right] \tag{2.10}$$

[4]

In our experiments, we will be looking at expected log-likelihoods $\mathscr{L}$ instead of the loss L, where the loss is simply the negative of the expected log-likelihood. Therefore, whereas we want to minimize the loss, we want to have maximal expected log-likelihoods.

[2] Noting that $\log\prod_i z_i = \sum_i \log z_i$

[3] In our experiments, we use the Adam optimizer.

2.4 Convolutional Conditional Neural Processes

While CNPs have been quite successful, they have been shown to have a few consistent failings:

1. CNPs are prone to severe underfitting. Absorbing the context into the target set for loss calculation during training is done to ameliorate this issue. Otherwise, the naive way to ameliorate this issue is to simply train for more epochs.

2. CNPs often fail to when attempting to extrapolate to areas outside of their training domain. Attempting to do so also often leads to catastrophic forgetting of data within their training regime. [Turner, 2022]

Gordon et al. [2019] introduced *Convolutional Conditional Neural Processes* (ConvCNPs) to alleviate these issues.The key decision in constructing ConvCNPs was to build the notion of *translation equivariance* into the architecture. With translation equivariant models, when input data are shifted in the domain (time, space. etc.) the outputs are correspondingly shifted. This is a reasonable inductive bias in many cases, and it often allows for the model to better learn from similar tasks (tasks where the inputs and outputs are similar when translated). This was accomplished by introducing Convolutional Neural Networks (CNNs)—which added translation equivariance to Multi-layer Perceptrons—into the CNP architecture. Accomplishing this required developing the notion of functional representations on sets which could then enrich the CNP encoder—naturally imbuing it with translation equivariance [Gordon et al., 2019].[5] In our experiments, we use a ConvCNP with UNet-like architecture, like the ConvCNPXL model used by Gordon et al.[Gordon et al., 2019].

2.5 Flexibility of Distributions

Having established an understanding of meta-learning and the CNP as a meta-learning model, we now return to key limitation of the CNP model: the marginal predictive distributions are Gaussian. Regrettably, Gaussians—though an eminently tractable distribution—lack the flexibility to adequately model a variety of datasets. We are interested in endowing the CNP models with this greater flexibility in their marginal distributions such that they can better model a variety of distributions. We are particularly interested in understanding our method's performance with respect to multi-modal and heavy-tailed distributions.

[5] Vector representations of sets, on the other hand, to not naturally lend themselves to a notion of translation equivariance.

There is a rich literature on building flexible generative distributions in this way. The method we develop and test uses ideas from *Neural Autoregressive Distribution Estimation* [Uria et al., 2016]—wherein the joint distribution is decomposed into a series of conditional distributions using the chain rule of probability. It also bears resemblance to the generative or reverse trajectory of a diffusion model [Sohl-Dickstein et al., 2015].

There is one key difference where this method stands apart from those mentioned above. The method we propose is executed solely at test time—no changes to CNP training are required in order to achieve the improvements we observe.

Chapter 3

Experiments

Having introduced meta-learning, CNPs, ConvCNPs, and their limitations—namely, those stemming from the inflexibility of Gaussian distributions—we now introduce our method. Recall, that the method introduced here requires no training procedures; the method is utilized solely at test time.

The method works as follows: Suppose we have a trained model Q_θ. This model has been trained in the typical fashion on samples generated from an underlying stochastic process P. In other words, it has been trained by sampling functions from this underlying process, sampling observations from these functions, dividing these observations into contexts and targets, and providing the model with the contexts as inputs with the goal of predicting the targets, and training through the minimum negative log-likelihood procedure described in Chapter 2.

After using this standard training scheme, we apply a non-standard process at test time (we refer to as the *AR procedure*). With the standard application of the CNP (or ConvCNP, in this case), we would produce predictive marginals for a new evaluation points $x^{(T)}$ by using the trained model to produce a Gaussian distribution, like so:

$$Q_\theta(y^{(T)}|C,x^{(T)}) = \mathcal{N}(\mu^{(T)},(\sigma^{(T)})^2) \tag{3.1}$$

In our non-standard training procedure, we use autoregressive sampling to generate Gaussian Mixture Model (GMM) marginal predictives. We start with a sketch (illustrated in Figure 3.1) of this process followed by a mathematical proof indicating its potential utility. The process is executed as follows:

Suppose, once again, we are interested in using our static trained model Q_θ along with context C to produce a predictive distribution for a input of interest $x^{(T)}$. We begin this AR procedure by first using our trained model to produce a predictive distribution at an entirely

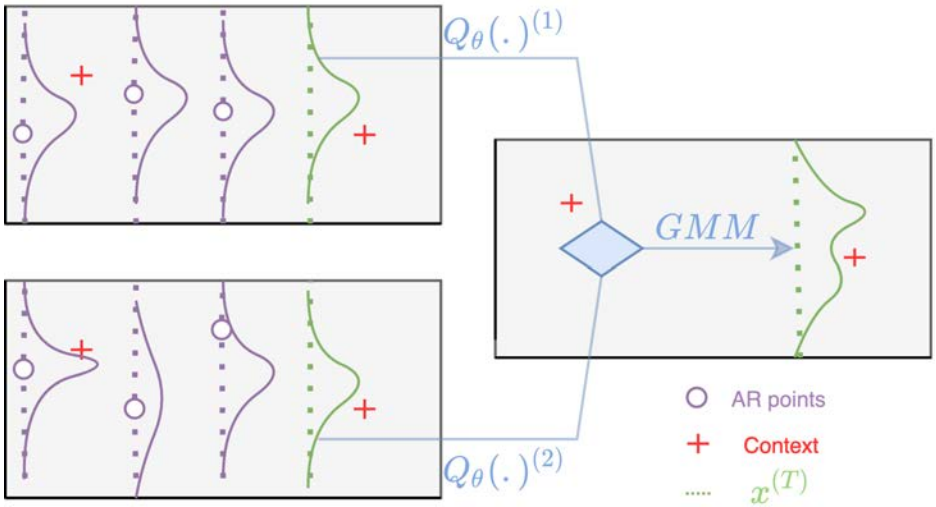


Fig. 3.1 The purple circles (sampled from the purple distributions) make up the autoregressively generated trajectories. Using the context points (red crosses) and purple circles together as pseudo context, a predictive marginal is generated at the target input location (the dotted green line). This process is executed again, with different purple distributions and purple points, leading to a different Gaussian at the green input target. These Gaussians are averaged to make a GMM on the right. This example only uses two trajectories ($M = 2$), though we typically use more.

different point x_1 (how this point is chosen will be discussed later). We then draw a sample y_1 from the predictive distribution we have produced at this input location:

$$y_1 \sim Q_\theta(y_1|C,x_1) \tag{3.2}$$

Next, we take this new point $\{x_1,y_1\}$ and include it as pseudo-context using the trained model to predict the output at yet another input location x_2. We then draw another sample, y_2 from the predictive distribution we produced in this manner[1].

$$y_2 \sim Q_\theta(y_2| \underbrace{C,\{x_1,y_1\}}_{\text{Pseudo Context Set}},x_2) \tag{3.4}$$

[1]When adopting a representation of the trained model as a function which produces parameters of a Gaussian distribution (rather than as the distribution itself), we have: $f_\theta(x^{(T)},C) = \{\mu^{(T)},\sigma^{(T)}\}$. Then, the second step of the AR process in 3.5 can be described using the pseudo-context as the input in place of C, like so:

$$f_\theta(x_2, \underbrace{\{C,\{x_1,y_1\}\}}_{\text{Pseudo Context Set}}) = \{\mu_2,\sigma_2\} \tag{3.3}$$

We continue in this fashion, choosing input locations x_j and using the previously selected points appended to the initial context set as a pseudo-context set to produce the corresponding output y_j[2]:

$$y_j \sim Q_\theta(y_j| \underbrace{C, \{x_k, y_k\}_{k=0}^{j-1}}_{\text{Pseudo Context Set}}, x_j) \tag{3.5}$$

Eventually we reach the final AR sampled point y_R (how R is chosen will be discussed later):

$$y_R \sim Q_\theta(y_R| \underbrace{\{C, \{x_k, y_k\}_{k=0}^{R-1}\}}_{\text{Pseudo Context Set}}, x_R) \tag{3.6}$$

As a result, we now have a set of AR generated points $\{x_k, y_k\}_{k=1}^{R}$ which will append to our context set C when producing a predictive distribution for our target point of interest $x^{(T)}$:

$$Q_\theta(y^{(T)}| \underbrace{\{C, \{x_k, y_k\}_{k=1}^{R}\}}_{\text{Pseudo Context Set}}, x^{(T)}) \tag{3.7}$$

Upon executing this process once, we are simply left with another Gaussian distribution, but one produced using a new context set $\{C, \{x_k, y_k\}_{k=1}^{R}\}$. We see that executing this process only once is not well justified. We are simply treating points sampled from the model as observations, thereby likely introducing bias and error into our predictive outputs.

Therefore, crucially, we must repeat this process many times to produce new series of AR generated points (hereafter referred to as *trajectories*). Doing so yields many different Gaussian distributions produced from different trajectories:

$$\begin{aligned}
&Q_\theta(y^{(T)}|\{C, \{x_k^{(1)}, y_k^{(1)}\}_{k=0}^{R}\}, x^{(T)}) = \mathcal{N}(\mu_1, \sigma_1^2) \\
&Q_\theta(y^{(T)}|\{C, \{x_k^{(2)}, y_k^{(2)}\}_{k=0}^{R}\}, x^{(T)}) = \mathcal{N}(\mu_2, \sigma_2^2) \\
&\dots \\
&Q_\theta(y^{(T)}|\{C, \{x_k^{(i)}, y_k^{(i)}\}_{k=0}^{R}\}, x^{(T)}) = \mathcal{N}(\mu_i, \sigma_i^2) \\
&\dots \\
&Q_\theta(y^{(T)}|\{C, \{x_k^{(M)}, y_k^{(M)}\}_{k=0}^{R}\}, x^{(T)}) = \mathcal{N}(\mu_M, \sigma_M^2)
\end{aligned} \tag{3.8}$$

Taking the mean of these Gaussian predictives yields a Gaussian Mixture Model (GMM):

[2]For ease of notation, we define $\{x_0, y_0\} = \emptyset$, the empty set.

$$\frac{1}{M}\sum_{i=1}^{M} Q_\theta\left(y^{(T)} \mid C, \{x_k^{(i)}, y_k^{(i)}\}_{k=0}^{R}, x^{(T)}\right) \tag{3.9}$$

What is this justification for using this process? We follow now with a proof that indicates the potential utility of this process.

First, we note that in the following derivation, we are working with the theoretical ground truth probability distributions (denoted p). We overload the notation on p such that it indicates the ground truth probability distribution of whatever variable it is applied to. With $R \in \mathbb{Z}$, and using the sum rule of probability to marginalize over $\{x_j, y_j\}_{j=0}^{R}$, we see that:

$$\begin{aligned} p(y^{(T)}|C,x^{(T)}) &= \int p(y^{(T)}, \{x_j,y_j\}_{j=0}^{R} \mid C,x^{(T)}) d\{x_j,y_j\}_{j=0}^{R} \\ &= \int p(y^{(T)}|C,\{x_j,y_j\}_{j=0}^{R},x^{(T)}) \underline{p(\{x_j,y_j\}_{j=0}^{R} \mid C,x^{(T)})} d\{x_j,y_j\}_{j=0}^{R} \end{aligned} \tag{3.10}$$

Next, we can write the underlined portion in the equation above like so:

$$\begin{aligned} & p(\{x_j,y_j\}_{j=0}^{R} \mid C,x^{(T)}) \\ &= p(\{x_j,y_j\}_{j=0}^{R} \mid C) && \text{by } \{x_j,y_j\}_{j=0}^{R} \perp x^{(T)} \mid C \\ &= \prod_{l=0}^{R} p(x_l,y_l \mid C, \{x_j,y_j\}_{j=0}^{l-1}) && \text{by chain rule} \\ &= \prod_{l=0}^{R} p(y_l) \mid C, \{x_j,y_j\}_{j=0}^{l-1}, x_l) p(x_l) && \text{by product rule} \end{aligned} \tag{3.11}$$

In the equation above, we see the autoregressive aspect of this method emerge by application of the chain rule. Substituting the last line of Equation 3.11 into 3.10, we have[3]:

[3] Upon expansion of $\{x_j,y_j\}_{j=0}^{R}$, we see that x_0 and y_0 are not included in the differential because they are not actually variables. Recall that we have defined $\{x_0,y_0\} = \emptyset$. Therefore:

$$p(y^{(T)}, \{x_j,y_j\}_{j=0}^{R} \mid C,x^{(T)}) = p(y^{(T)}, \underbrace{\{x_0,y_0\}}_{\emptyset}, \{x_j,y_j\}_{j=1}^{R} \mid C,x^{(T)}) = p(y^{(T)}, \{x_j,y_j\}_{j=1}^{R} \mid C,x^{(T)}) \tag{3.12}$$

$$
\begin{aligned}
&p(y^{(T)}|C,x^{(T)}) \\
&= \int p(y^{(T)},|\, C,\{x_j,y_j\}_{j=0}^{R},x^{(T)}) \prod_{l=1}^{R} \left[p(y_l \mid C,\{x_j,y_j\}_{j=0}^{l-1},x_l)p(x_l) \right] d\{x_j,y_j\}_{j=0}^{R} \\
&= \int\int ... \int\int p(y^{(T)},|\, C,\{x_j,y_j\}_{j=0}^{R},x^{(T)}) \prod_{l=1}^{R} \left[p(y_l \mid C,\{x_j,y_j\}_{j=0}^{l-1},x_l)p(x_l) \right] dx_1y_1...dx_Ry_R
\end{aligned} \tag{3.13}
$$

Next, to ease notation, we define $P_{y_l} = p(y_l|C,\{x_j,y_j\}_{j=0}^{l-1},x_l)$, and $P_{y^{(T)}} = p(y^{(T)}|C,\{x_j,y_j\}_{j=0}^{R},x^{(T)})$. We then rearrange the variables and their corresponding differentials in Equation 3.20, to get:

$$
= \int p(x_1) \left[\int P_{y_1} \left[... \left[\int p(x_R) \left[\int P_{y_R} \left[P_{y^{(T)}} \right] dy_R \right] dx_R \right] ... \right] dy_1 \right] dx_1 \tag{3.14}
$$

Writing this out as a series of substitutions makes it clear that we can approximate this integral through the recursive application of Monte Carlo integration:

$$
\begin{aligned}
p(y^{(T)} \mid C,x^{(T)}) &= \int g_1(x_1)p(x_1)dx_1 && \approx \frac{1}{M_1}\sum_{i=1}^{M_1} g_1(x_1), x_1 \sim p(x_1) \\
g_1(x_1) &= \int f_1(y_1)P_{y_1}dy_1 && \approx \frac{1}{N_1}\sum_{i=1}^{N_1} f_1(y_1), y_1 \sim P_{y_1} \\
f_1(y_1) &= \int g_2(x_2)p(x_2)dx_2 && \approx \frac{1}{M_2}\sum_{i=1}^{M_2} g_2(x_2), x_2 \sim p(x_2) \\
g_2(x_2) &= \int f_2(y_2)P_{y_2}dy_2 && \approx \frac{1}{N_2}\sum_{i=1}^{N_2} f_2(y_2), y_2 \sim P_{y_2} \\
&... \\
f_{l-1}(y_{l-1}) &= \int g_l(x_l)p(x_l)dx_l && \approx \frac{1}{M_l}\sum_{i=1}^{M_l} g_l(x_l), x_l \sim p(x_l) \\
g_l(x_l) &= \int f_l(y_l)P_{y_l}dy_l && \approx \frac{1}{N_l}\sum_{i=1}^{N_l} f_l(y_l), y_l \sim P_{y_l} \\
&... \\
f_{R-1}(y_{R-1}) &= \int g_R(x_R)p(x_R)dx_R && \approx \frac{1}{M_R}\sum_{i=1}^{M_R} g_R(x_R), x_R \sim p(x_R) \\
g_R(x_R) &= \int f_R(y_R)P_{y_R}dy_R && \approx \frac{1}{N_R}\sum_{i=1}^{N_R} f_R(y_R), y_R \sim P_{y_R}
\end{aligned} \tag{3.15}
$$

where $f(y_R)) = p(y^{(T)}|C, \{x_j, y_j\}_{j=0}^R, x^{(T)})$. Collapsing these Monte Carlo approximations back down, we have:

$$
\begin{aligned}
p(y^{(T)} \mid C, x^{(T)})) &\approx \left(\prod_{l=1}^{R} \frac{1}{M_l N_l} \right) \sum_{i \in S} p(y^{(T)} | C, (\{x_j, y_j\}_{j=0}^R)^{(i)}, x^{(T)}) \\
\text{where} \qquad S &= \bigcup_{l=1}^{R} \{\{1, \ldots, M_l\} \times \{1, \ldots, N_l\}\} \\
x_j^{(i)} &\sim p(x_j) \\
y_j^{(i)} &\sim p\left(y_j \mid x_j, C, (\{x_j, y_j\}_{j=0}^{l-1})^{(i)}\right)
\end{aligned}
\tag{3.16}
$$

Thus, we show that the marginal can be approximated through the iterative autoregressive sampling of trajectories $(\{x_j, y_j\}_{j=0}^R)^{(i)}$, and incorporation of these trajectories as pseudo-context for marginal prediction.

3.0.1 Approximation Using the Model

Now, it is worth revisiting the meaning of some of the terms in 3.16. First, we note that in real-world application, the true ground truth marginal predictive distributions are unknown. However, we have assumed that we have trained a CNP (Q_θ) to model these marginals. Thus, we have:

$$
p(y^{(T)}, \mid C, \{x_j, y_j\}_{j=1}^R, x^{(T)}) = Q_\theta(y^{(T)} \mid C, \{x_j, y_j\}_{j=1}^R, x^{(T)}) + \varepsilon_Q^{(T)} \tag{3.17}
$$

Where $\varepsilon_Q^{(T)}$ is the modeling error, a distribution which is a function of $y^{(T)}, \{x_j, y_j\}_{j=1}^R, C$, and $x^{(T)}$. The same substitution can be made for the distributions from which we draw our y values in our trajectory samples:

$$
p\left(y_l \mid C, (\{x_j, y_j\}_{j=0}^{l-1})^{(i)}, x_l\right) = Q_\theta(y^{(l)} \mid C, \{x_j, y_j\}_{j=1}^{l-1}, x^{(l)}) + \varepsilon_Q^{(l)} \tag{3.18}
$$

Additionally, we note that distribution $p(x)$ determines which points are context for when constructing tasks ξ. As such, we consider this distribution to be under our control, and potentially dependent upon the context set C and the specific target input in question $x^{(T)}$. We write this as $r(x \mid C, x^{(T)})$.

Substituting these values in for Equation 3.20, we have:

$$\begin{aligned} &p(y^{(T)}|C,x^{(T)}) \\ &= \int \left(Q_\theta(y^{(T)} \mid C,\{x_j,y_j\}_{j=1}^{R},x^{(T)}) + \varepsilon_Q^{(T)}\right) \\ &\prod_{l=1}^{R}\left[\left(Q_\theta(y^{(l)} \mid C,\{x_j,y_j\}_{j=1}^{l-1},x^{(l)}) + \varepsilon_Q^{(l)}\right) r(x_l \mid C,x^{(T)})\right] d\{x_j,y_j\}_{j=0}^{R} \end{aligned} \tag{3.19}$$

Expanding this product of sums results in one term with all of our model predictives, many terms with all combinations of model predictives and error predictives, and one term only with error distributions. We gather all terms with modeling error into one term denoted $\mathscr{E}_Q^{AR}$, yielding:

$$\begin{aligned} &p(y^{(T)}|C,x^{(T)}) \\ &= \int \left(Q_\theta(y^{(T)} \mid C,\{x_j,y_j\}_{j=1}^{R},x^{(T)})\right) \\ &\prod_{l=1}^{R}\left[\left(Q_\theta(y^{(l)} \mid C,\{x_j,y_j\}_{j=1}^{l-1},x^{(l)})\right) r(x_l \mid C,x^{(T)})\right] d\{x_j,y_j\}_{j=0}^{R} + \mathscr{E}_Q^{AR} \end{aligned} \tag{3.20}$$

Using the same logic applied in Equations 3.14 and 3.15, we can approximate the integral using Monte Carlo integration. In this case, we simplify the procedure by assuming all N_l and M_l values are equal $M = N_l = M_l \forall l \in 1,...,R$. [4]

$$\begin{aligned} p(y^{(T)} \mid C,x^{(T)})) &\approx \frac{1}{\mathbf{M}} \sum_{i=1}^{\mathbf{M}} Q_\theta(y^{(T)}|C,(\{x_j,y_j\}_{j=0}^{\mathbf{R}})^{(i)},x^{(T)}) \\ x_j^{(i)} &\sim \mathbf{r}(x_j|C,x^{(T)}) \\ y_j^{(i)} &\sim Q_\theta\left(y_j \mid x_j,C,(\{x_j,y_j\}_{j=0}^{l-1})^{(i)}\right) \end{aligned} \tag{3.21}$$

Because our model Q_θ only produces Gaussian outputs, the resulting model from this application of Monte Carlo integration in Equation 3.21 will be a weighted sum of Gaussians–a Gaussian Mixture Model (GMM).

Substituting on the left side with the error and using an equivalence relationship helps reveal the crux of this method's utility:

[4] There may be some utility to not making this assumption and testing different numbers of samples at different trajectory lengths, but our experiments did not include this level of complexity.

$$
\begin{aligned}
p(y^{(T)} \mid C, x^{(T)})) &= Q_\theta(y^{(T)} \mid C, x^{(T)})) + \mathscr{E}_Q^{Vanilla} \\
&= \frac{1}{M}\sum_{i=1}^{M} Q_\theta(y^{(T)} | C, (\{x_j, y_j\}_{j=0}^{R})^{(i)}, x^{(T)}) + \mathscr{E}_Q^{AR} + \mathscr{E}_{MC}
\end{aligned}
\tag{3.22}
$$

where $\mathscr{E}_{MC}$ is the Monte Carlo error. Thus, this method improves the quality of the marginal distributions when:

$$
\mathscr{E}_Q^{AR} + \mathscr{E}_{MC} < \mathscr{E}_Q^{Vanilla} \tag{3.23}
$$

We expect the number of trajectories M to impact the Monte Carlo error $\mathscr{E}_{MC}$, the trajectory length R to impact the AR model error $\mathscr{E}_Q^{AR}$, and the context selection distribution r to effect the variance reduction rate in the Monte Carlo sampling. These parameters M, R, and r are those which we are principally concerned about and will investigate in our experiments.

3.1 Trajectory Generation

Though selecting R and M is simply a matter of choosing a sensible natural number, the distribution on inputs (r)— which we refer to as the *trajectory generator*—must be more carefully constructed. In the following experiments, we use three different constructions of r, three different *trajectory generators*, which we call:

1. Random Trajectory Generator
2. Grid Trajectory Generator
3. Emanate Trajectory Generator

The Random Trajectory Generator defines a distribution r which is Uniform over the training domain and has no dependence on the context set or target point:

$$
r(x \mid C, x^{(T)}) = r(x) = Uniform(b, h) \tag{3.24}
$$

where b and h are the lower and upper bounds of the training domain, respectively.

The Grid Generator adds more structure to the trajectory generation. With the Grid Generator, we first create a grid sequence over the training domain:

$$
G = \{b, b+\Delta, ..., b+\Delta i, ..., h-\Delta, h\} \tag{3.25}
$$

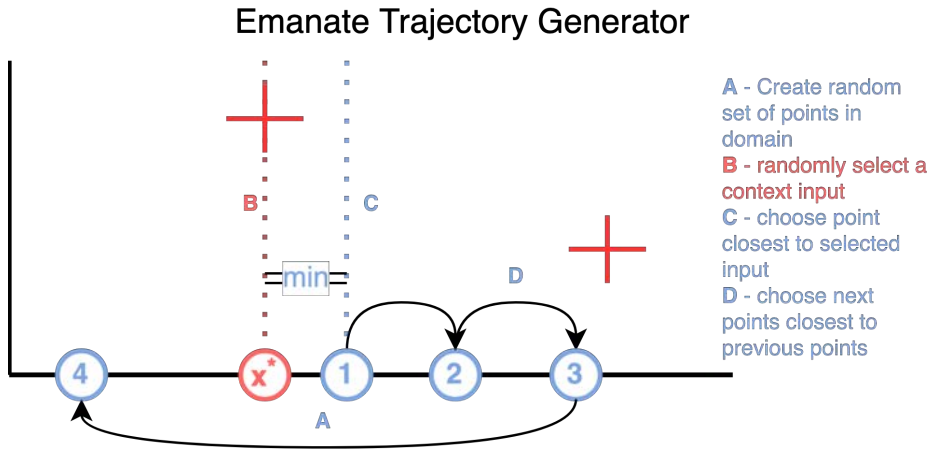


Fig. 3.2 Emanate Trajectory Generation process.

Then, for each trajectory (of which there are M in total), we apply a random permutation σ to the grid. So for trajectory i, we have:

$$\sigma^{(i)}(G) = \left\{\sigma_1^{(i)}(b), \sigma_2^{(i)}(b+\Delta), ..., \sigma_i(b+\Delta j), ..., \sigma_{R-1}^{(i)}(h-\Delta), \sigma_R^{(-1)}(h)\right\} \tag{3.26}$$

Then, for each trajectory i and each point in the trajectory l the distribution r is defined:

$$r_l^{(i)} = \delta(\{\sigma^{(i)}(G)\}_l) \tag{3.27}$$

where $\{\sigma^{(i)}(G)\}_l$ is the l^{th} element in the permutation i of grid G, and δ is the Dirac delta distribution. Here again, we note that each distribution r has no dependence on the context set C or the target point $x^{(T)}$.

We refer to the final trajectory generator as the *Emanate Generator*, and it does use information from the context set, unlike the previous two trajectory generators. For each trajectory i out of M total trajectories, the procedure is as follows:

First, we select R points uniformly between the lower and upper bounds of our training domain, inclusively:

$$x_l \sim Uniform(b,h) \forall l \in \{1,..,R\} \tag{3.28}$$

This gives us a sequence $S = \{x_1, ..., x_R\}$. Now we construct a permutation of S which uses information from the context set.

Given context set C with inputs $\mathbf{x}_\tau$, we randomly select an element $x^{(*)}$ from $\mathbf{x}_\tau$. Now, we identify the point in S which is closest to the selected context input $x^{(*)}$, and make that the first element of our permutation:

$$\sigma_1(S) = \min_{x_l \in S} \left[x_i - x^{(*)} \right] \tag{3.29}$$

Our next point in the permutation is selected to be the point in S closest to the previously selected point, excluding the previously selected:

$$\sigma_2(S) = \min_{x_l \in S \setminus \{\sigma_1(S)\}} \left[x_i - \sigma_1(S) \right] \tag{3.30}$$

The permutation is continued in this way, with each subsequent element chosen to the be the element which is closest to the previous. An illustration of the process is shown in Figure 3.2. Permutations of this sort are created for each trajectory, where the starting context input is selected randomly each time.

3.2 Experiment Setup

We assess the impact of these parameters (M, R, and r) in experiments on four different ConvCNP models: three from synthetic data generating processes, and one from real-world data. All training is done using the *neuralprocesses* library [Wessel et al., 2022].

The first experiment using a Gaussian Process was chosen because the marginals of the Gaussian Process are Gaussian, and therefore we do not expect to see improvements when adding the ability to construct GMM marginals. This experiment contrasts with all the others, as the subsequent experiments all have non-Gaussian marginals–a regime where we expect the ability to model as a GMM could be helpful.

In particular, the sawtooth experiment has bimodal marginal predictives, and the synthetic and real audio experiment marginals are heavy-tailed. We are interested in how this method performs when modeling these features which Gaussians lack.

3.2.1 Gaussian Process Experimental Setup

In this experiment, we use a data generator which is a Gaussian Process[5] with a mean function of 0 and an exponential quadratic kernel[6] with a variance of 1 and a length scale of 0.25:

[5] From *neuralprocesses* library [Wessel et al., 2022].

[6] Also referred to as the *squared exponential* kernel.

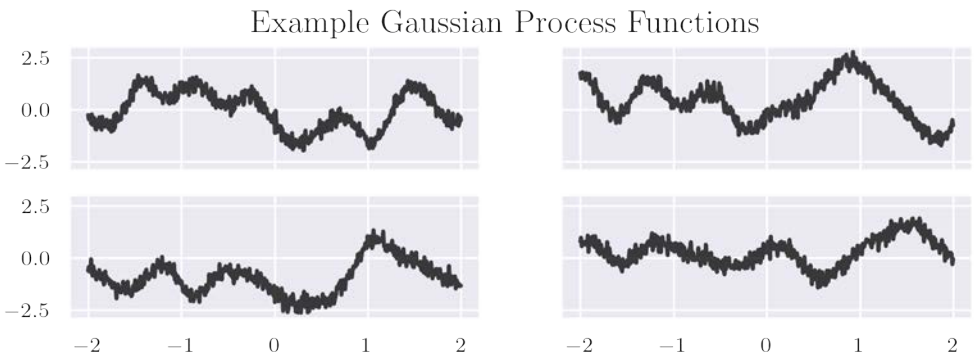


Fig. 3.3 Function drawn from GP with mean function 0 and kernel shown at 3.31

.

$$k(x,y) = \exp\left(-\frac{1}{2}\frac{||x-y||^2}{(0.25)^2}\right) \tag{3.31}$$

We add Gaussian noise to the resulting values from the function: $y = f(x) + (N)(0, 0.0025^2)$. We use a ConvCNP model with a UNet-like architecture (See *ConvCNPXL*, Gordon et al. [2019]). We train for 100 epochs using 1024 batches per epoch with a batch size of 16. We discretize the encoder by evaluating 64 points per unit. We use a margin of 0.1, and a stride length of 2 for each of the 6 layers of the UNet. Each layer has 64 channels. The receptive field size from this combination of parameters is 6.953.

For training, we sample a number of context points uniformly between 0 and 30 (inclusive) and exactly 50 target points for each function sample. Context points and target points are sampled uniformly between -2, and 2 (inclusive). We use the Adam optimizer with a learning rate of 3×10^{-4}.

3.2.2 Sawtooth Experimental Setup

In this experiment, we use a data generator for sawtooth functions[7]. The samples are generated from the following function:

$$y = [\omega(dx - \phi)] mod 1 \tag{3.32}$$

We sample frequencies ω uniformly between 3 and 5 (inclusive), direction as either -1 or 1 with equal probability, and shift ϕ uniformly conditional on the frequency in between 1 and $\frac{1}{\omega}$ (inclusive). Example functions generated from this process can be see in Figure 3.4.

[7] From *neuralprocesses* library [Wessel et al., 2022].

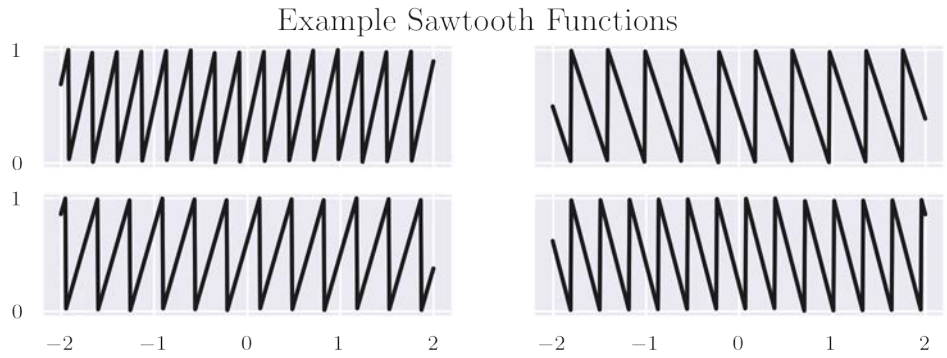


Fig. 3.4 Sawtooth functions generated by 3.32. Note that the sawtooths can have positive or negative slope, introducing bi-modality to our predictive marginals.

For training, we sample a number of context points between 0 and 75 (inclusive), and we sample exactly 100 target points. Context points and target points are sampled uniformly in between -2 and 2 (inclusive). All other hyperparameters are the same as those of the Gaussian Process model.

3.2.3 Synthetic Audio Experimental Setup

In this experiment, we create a data generator for periodic "audio-like" functions. The samples are generated by convolving a Dirac comb[8] with a truncated decaying sum of sinusoids:

$$
\begin{aligned}
s(t) &= \begin{cases} e^{\frac{-t}{(\tau)}}\left[sin(\omega_1 t) + sin(\omega_2 t)\right] & \text{for } 0 \leq t < T \\ 0 & \text{otherwise} \end{cases} \\
f(x) &= \text{Ш}_T(x) * s(x) \\
\varepsilon &\sim \mathcal{N}(\mu = 0, \sigma^2 = 0.001)
\end{aligned}
\tag{3.33}
$$

With ω_1 and ω_2 both being drawn uniformly in between 50 and 70 (inclusive). The period T being drawn uniformly in between 0.75 and 1.25 (inclusive), and the decay τ being drawn uniformly between 0.1 and 0.3 (inclusive). We truncate the waves up to the period length, because otherwise the convolution with the Dirac comb would lead to increasing amplitude, resulting in a non-stationary process. We chose to work with a simpler function instead. All other hyperparameters are the same as those of the Gaussian Process and Sawtooth setups. Example functions generated from this process can be see in Figure 3.5.

[8] Dirac comb defined as $\text{Ш}_T(t) := \sum_{k=-\infty}^{\infty} \delta(t - kT)$ for given period P.

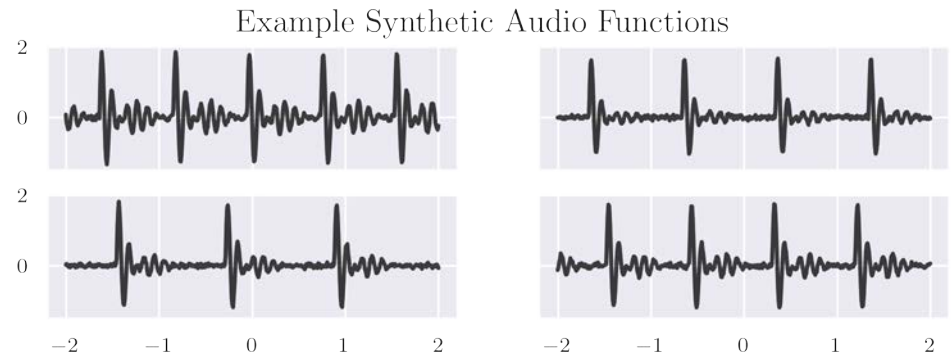


Fig. 3.5 *Synthetic Audio* functions generated by 3.33. The sparse high amplitude wave structure results in heavy-tailed marginal distributions.

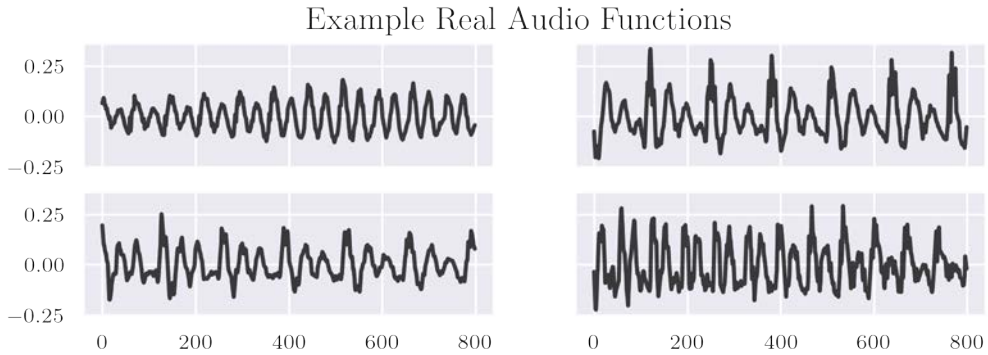


Fig. 3.6 Real audio waveforms of the phoneme /iy/ extracted from the TIMIT corpus. 800 frames is 0.05 seconds.

3.2.4 Real Audio Experimental Setup

For our real-world experiment, we look at real speech data from the TIMIT corpus [Garofolo et al., 1993], a corpus of audio recordings from 630 speakers across major dialect of American English, each reading ten sentences. The corpus is particularly useful because the speech wave form phonemes are labelled at the frame level. The waveform files are 16-bit, sampled at 16kHz.

Because of this labelling, we are able to isolate individual phonemes in the corpus. Therefore, to limit the scope of our experiment we extract the all instances of the phoneme /iy/ from corpus.

We do some preprocessing of these audio files before feeding them into the ConvCNP model. First, we cut the /iy/ audio from the audio files using the phone-level transcriptions. Next, we cut these audio in segments of 800 frames, or one twentieth of a second. We do this because we believe it should make training easier because the selection of context and

target points cannot be too far away from each-other. There is a greater resemblance to the synthetic-audio and real-audio experiments because there are fewer oscillations with these few frames.

After creating these equally sized audio segments, we then apply peak normalization on the segments using the python library *pydub* [Robert et al., 2018]. In peak normalization, we take the highest peak of the waveform and scale it -20 dBFS; all other parts of the waveform are then scaled using the same proportion. After normalizing the gain, we then normalize the signals between -1 and 1 using the python library *librosa* [McFee et al., 2015].

We then split these audio segments into train, test, and evaluation sets using the TIMIT provided separation, which are balanced for dialectic coverage. As a result, we are left with 9096 training samples, 1711 testing samples, and 1712 evaluation samples—all samples having length 800 frames. During training, when all samples have been exhausted, the same function samples are used again with different random selections for target and context points.

We train using the same ConvCNP model with UNet-inspired architecture. We use 7 channels with 1 point-per-unit encoder discretization. We use a stride size of 1 for the first layer and a stride of 2 for each subsequent layer. Each layer has 128 channels. The resulting receptive field size is 513.0.

We train for 500 epochs with 1024 batches per epoch and a batch size of 16. We use a learning rate of 2.5×10^{-5}. Like in previous experiments, the model parameters are optimized by minimizing the expected negative log-likelihood for each training batch.

For training we a number of target points uniformly between 50 and 200 (inclusive). We sample a total number of data points in 200 and 500 (inclusive). The number of contexts points is the total number of data points minus the number of target points. The data points are all sampled uniformly on integers between 1 and 800, the total number of frames.

Chapter 4

Results

Now that we have constructed our data generators and trained ConvCNP models on each one, we seek the assess the effect of using autoregressive sampling on the quality of our marginal predictions. We do so by measuring the log-likelihood of the held-out true target points (i.e. points from the tasks in the meta-test set Ξ_{test}):

$$\{(x^{(T)}, y^{(T)}) \in D_T \mid (C, D_T) \in \Xi_{test}\} \tag{4.1}$$

Here, the test tasks Ξ_{test} are constructed by sampling from our underlying data generators described in Chapter 3. For our purposes, we are interested on the impact of the context size on performance, so we also partition the meta-test set in subsets which have equivalently sized context sets, where the size is given by n:

$$\Xi_{test}^{(n)} = \{(C, D_T) \in \Xi_{test} \text{ such that } Card(C) = n\} \tag{4.2}$$

Recall from Chapter 2 that the expectation of the log-likelihood over functions drawn from our processes and the number of target context points is given by:

$$\begin{aligned} \mathscr{L}(\theta) &= \mathop{\mathbb{E}}_{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}} \left[\log Q_\theta\left(\mathbf{y}_\tau \mid C, \mathbf{x}_\tau\right)\right] \\ &= \mathop{\mathbb{E}}_{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}} \left[\sum_{k=1}^{T} \log Q_\theta\left(y_k \mid C, x_k\right)\right] \text{ where } x_k \in \mathbf{x}_\tau, y_k \in \mathbf{y}_\tau \end{aligned} \tag{4.3}$$

Again, here we assume a factorized distribution, because we are only assessing the performance on the quality of the marginals, rather than joints with dependencies.

Now, moving away from this standard method for evaluating the log-likelihoods for CNPs, and we introduce evaluation using the *AR procedure*[1]:

$$\begin{aligned}\mathscr{L}_{AR}(\theta) &= \mathop{\mathbb{E}}_{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}}\left[\log\prod_{k=1}^{T}\frac{1}{M}\sum_{i=1}^{M}Q_\theta\left(y_k \mid C,\{\{x_j,y_j\}_{j=0}^{R}\}^{(i)},x_k\right)\right] \\ &= \mathop{\mathbb{E}}_{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}}\left[\left[\sum_{k=1}^{T}\log\left(\sum_{i=1}^{M}Q_\theta\left(y_k \mid C,\{\{x_j,y_j\}_{j=0}^{R}\}^{(i)},x_k\right)\right)\right]-\sum_{k=1}^{T}\log(M)\right]\end{aligned} \tag{4.4}$$

We also often report our results simply as expected log-likelihoods of marginal predictives, which is equivalent to dividing the factorized joint by the number of target points:

$$\mathscr{L}_{AR}^{(norm)}(\theta) = \mathop{\mathbb{E}}_{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}}\left[\frac{1}{T}\left[\left[\sum_{k=1}^{T}\log\left(\sum_{i=1}^{M}Q_\theta\left(y_k \mid C,\{\{x_j,y_j\}_{j=0}^{R}\}^{(i)},x_k\right)\right)\right]-\sum_{k=1}^{T}\log(M)\right]\right] \tag{4.5}$$

Where R is the length of the trajectory and M is the number of trajectories sampled.

In the tables and figures shown in this section, we report on the performance of the Vanilla model (a ConvCNP without use of AR marginal approximation), the model with the use of the AR method, and a Naive model. We provide the log-likelihood of a Naive model to give an better sense of the scale of the differences between the Vanilla and AR performances.

The Naive model for a given experiment is defined simply by a single Gaussian distribution with mean equal to the mean of all context points for all functions in that experiment, and variance equal to the variance for all context points for all functions in that experiment:

$$\begin{aligned}&Q_{naive}(y^{(T)}) \mid C,x^{(T)}) = {}^{2}Q_{naive}(y^{(T)}) = \mathscr{N}(y^{(T)};\mu,\sigma^2) \\ &\text{where } \mu = \mathbb{E}[Y] \text{ and } \sigma^2 = Var[Y] \\ &\text{with } Y = \{y_c|(x_c,y_c)\in C_i \text{ for all } (C_i,D_{T_i})\in\Xi_{test}\}\}\end{aligned} \tag{4.6}$$

4.1 Overview

In Table 4.1, we see that using marginal approximation by autoregressive sampling improved the performance (as measured by held-out log-likelihoods) across all experiments except

[1] With a trajectory length (R) of 0, we simplify to the vanilla log-likelihood evaluation.

[2] Note that predictions made by Q_{naive} are not impacted by the particular context set and target input in question.

Experiment	Naive	Vanilla	$AR_{R=8,M=128}$
Real Audio	0.92 ± 0.10	1.74 ± 0.16	1.77 ± 0.15
Sawtooth	-0.18 ± 0.00	1.46 ± 0.30	1.64 ± 0.28
Synthetic Audio	-0.44 ± 0.02	0.12 ± 0.14	$\mathbf{0.55} \pm 0.10$
GP	-1.45 ± 0.06	-1.04 ± 0.08	-1.04 ± 0.08

Table 4.1 Using Autoregressive sampling for marginal approximation improves held-out log-likelihoods on all experiments except for the GP experiment, which is to be expected. These values are normalized by the number of target points, as shown in Equation 4.5. Values which are significantly best ($p < 0.05$) are shown in bold.

for the Gaussian Process experiment, where it achieved the same performance as the vanilla model. When we measure log-likelihoods across tasks of all context sizes, the improvement for the synthetic audio is the only one which is significant ($p < 0.05$). Later, we will divide out these results by context size and observe the differences. We hereafter refer to the standard use of the model Q_θ as the *vanilla procedure* and the AR marginal approximation method as the *AR procedure*. We avoid using the term model because we wish to highlight that both processes use the exact same trained model Q_θ.

We now further explain and contextualize these results by:

1. Defining the construction of each experiments' meta-test sets Ξ_{test}.
2. Visually comparing the marginal density estimation using the vanilla procedure and the AR procedure for each experiment.
3. Showing the impact of increasing number of trajectory samples (M) on the overall model error.
4. Investigating the impact of context set size on performance of the vanilla and AR procedures.

4.2 Examination of Particular Marginals

4.2.1 Gaussian Process

For the Gaussian Process experiment, we use the model Q_θ which has been trained on tasks from the GP data generator as described in Chapter 3. Additionally, we construct our meta-test set Ξ_{test} by sampling functions from the GP data generator. In particular, we sample a total of 96 functions–16 functions each for context set sizes 0, 1, 2, 4, 8, and 16. We use

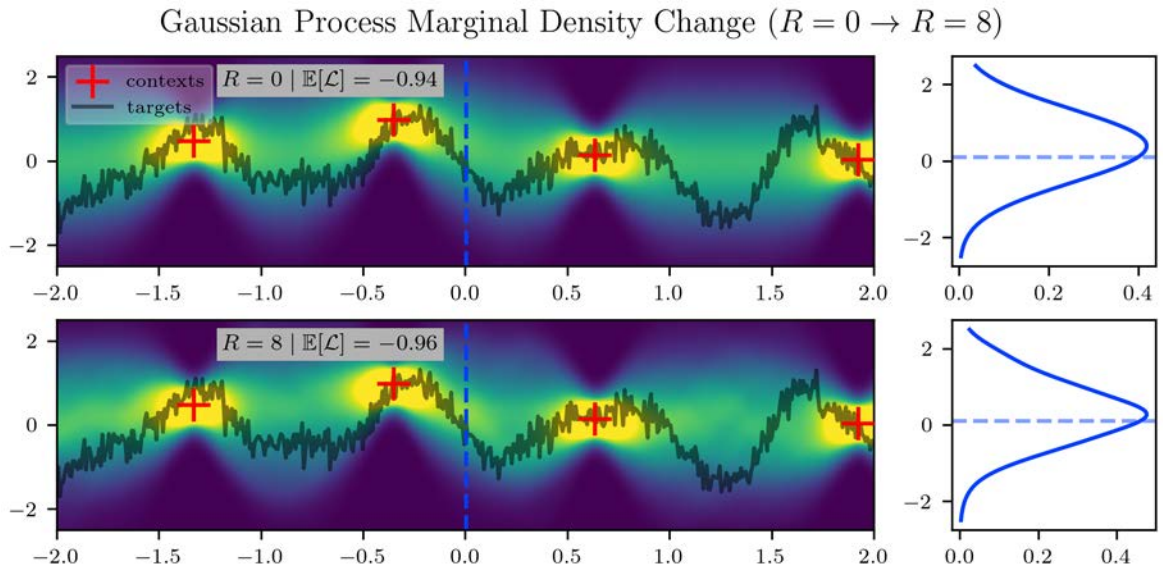


Fig. 4.1 An example task from the GP experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

trajectories lengths (R) from 0[3] to 8, inclusive. We use a number of trajectory samples (M) from 1 to 128, inclusive. For this experiment, only evaluate one method for choosing input trajectory points, the *Random Trajectory Generator* described in Section 3.1.

We find that the AR marginal approximation procedure does not improve the held-out log-likelihoods, as seen in Table 4.1. This is to be expected, because the AR procedure only introduces Monte Carlo error while not decreasing the modeling error:

$$\mathscr{E}_{MC} > 0 \text{ and } \mathscr{E}_{Q}^{AR} \geq \mathscr{E}_{Q}^{Vanilla} \implies \mathscr{E}_{MC} + \mathscr{E}_{Q}^{AR} > \mathscr{E}_{Q}^{Vanilla} \tag{4.7}$$

The modeling error is not decreased because the marginal distributions of a Gaussian Process are Gaussian [Williams and Rasmussen, 2006]. Therefore, using the AR procedure to create a GMM does not provide additional benefits.

We can see this phenomena more clearly by observing Figure 4.1. In Figure 4.1, we see one example task ξ with context set C shown with red crosses, and target set D_T shown with a grey line. The marginals are represented across the domain with the color range, and one

[3] where $R = 0$ is the vanilla model

target marginal is highlighted with a density plot on the right side of the figure. The top portion of the figure shows the vanilla procedure and the bottom shows the AR procedure. We notice a few things:

1. Visually there appears to be very little change to the marginal densities across the domain.

2. The particular target point highlighted with the right density also appears mostly unchanged, though the AR marginal appears slightly narrower near the mode.

3. The average likelihood across targets for this function $\mathbb{E}(\mathscr{L})$ is slightly less for the AR procedure than it is for the vanilla procedure.

We will take a broader look at the AR procedure performance across functions after introducing our other experimental results.

4.2.2 Sawtooth

For the Sawtooth experiment, we use a model Q_θ which has been train on tasks from the Sawtooth data generator as described in Chapter 3. Additionally, we construct our meta-test set Ξ_{test} by sampling functions from the Sawtooth data generator. In particular, we sample a total of 112 functions–16 functions each for context set sizes 0, 1, 2, 4, 8, 16, and 32. We use trajectories lengths (R) from 0[4] to 8, inclusive. We use number of trajectory samples (M) from 1 to 128, inclusive. We run this experiment using all three trajectory generators introduced in Section 3.1—Random, Grid, and Emanate.

We find that the AR procedure can improve the held-out log-likelihoods, as seen in Table 4.1. In other words, it appears that producing sufficiently long trajectories reduces the modeling error $\mathscr{E}_Q^{AR}$ and taking sufficiently many of them reduces the Monte Carlo error such that:

$$\mathscr{E}_{MC} + \mathscr{E}_Q^{AR} < \mathscr{E}_Q^{Vanilla} \tag{4.8}$$

In the next section, we will take a closer look at the Monte Carlo error, but here we try to illustrate how the modeling error is improved. Observing Figure 4.2, we notice several things. First, we see in this example that the AR procedure has produced densities which are very different from the vanilla procedure. For the particular target we have highlighted in the right plots, we see that the vanilla procedure produces a uni-modal Gaussian density, whereas the AR procedure has produced a multi-modal distribution that appears to be approaching

[4] where $R = 0$ is the vanilla model

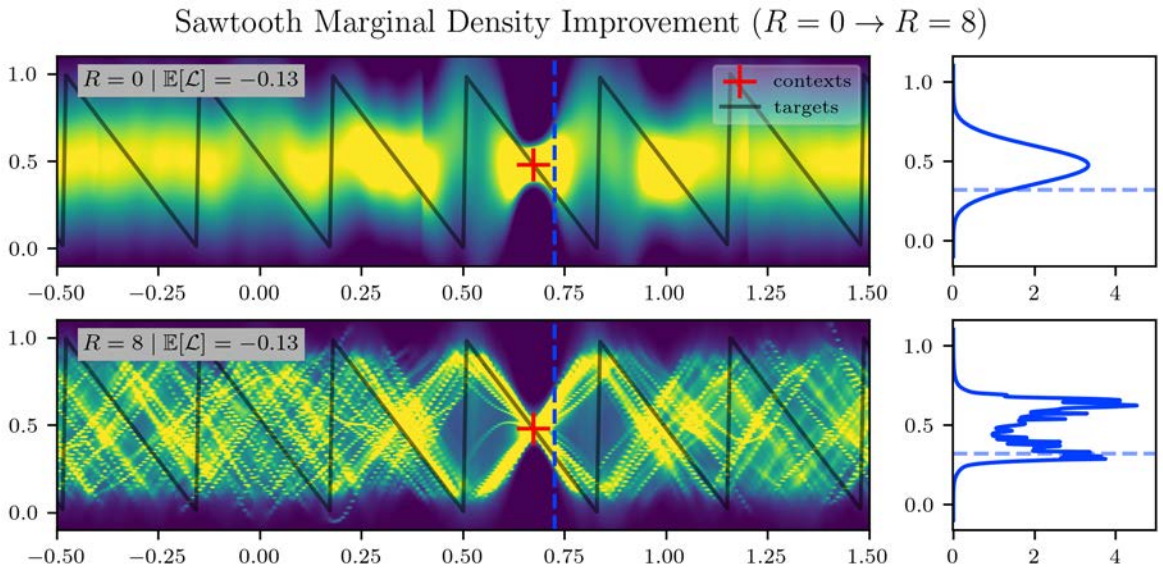


Fig. 4.2 An example task from the Sawtooth experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

a bi-modal distribution. This is precisely the sort of behaviour which we would hope to see in this model because the true marginals of the sawtooth generator are bimodal due to the randomly chosen direction of the sawtooth function. Therefore, the cross we see in the middle of the bottom left plot appears to portray the densities much more adequately than the more amorphous form in the top plot.

These are all positive signs, but one may notice that the vanilla and AR procedures achieve the same expected log-likelihood for this particular function $\mathbb{E}[(L)] = -0.13$. This is likely a result of large Monte Carlo error for this particular function, such that—despite model error improvements using the AR procedure–we end up with:

$$\mathscr{E}_{MC} + \mathscr{E}_{Q}^{AR} = \mathscr{E}_{Q}^{Vanilla} \tag{4.9}$$

Potential visual evidence of this Monte Carlo error can be seen by observing the yellow streaks (areas of high density) farther to the right and left of the context point. We see that these streaks have an orientation and form which fits the underlying true function in black, but there are many areas where they do not properly align with the underlying true function. In a subsequent section, we try to shed more light on a question raised by this observation:

How do our parameter (R and M) selections impact the two error sources $\mathscr{E}_{MC}$ and $\mathscr{E}_{Q}^{AR}$?

4.2.3 Synthetic Audio

For the synthetic audio experiment, we use a model Q_θ which has been trained on tasks from the synthetic audio data generator as described in Chapter 3. Additionally, we construct our meta-test set Ξ_{test} by sampling functions from the synthetic audio data generator. In particular, we sample a total of 96 functions–16 functions each for context set sizes 0, 1, 2, 4, 8, and 16. We use trajectories lengths (R) from 0[5] to 8, inclusive. We use number of trajectory samples (M) from 1 to 128, inclusive. We run this experiment using all three trajectory generators introduced in Section 3.1—Random, Grid, and Emanate.

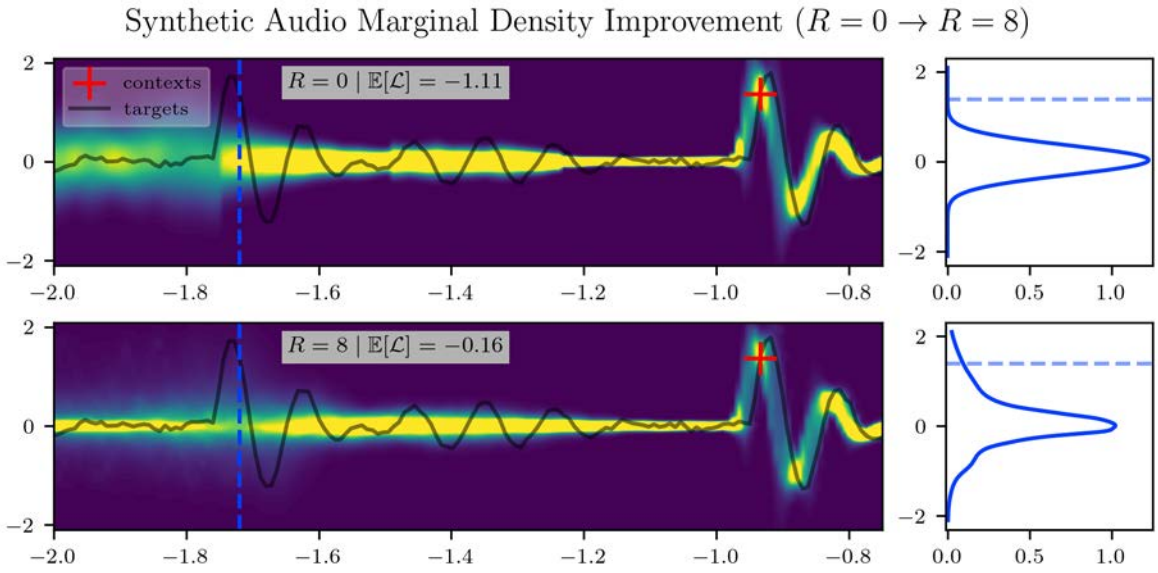


Fig. 4.3 An example task from the Sawtooth experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

Like the previous two examples, the top plot of Figure 4.3 shows the marginal density produced using the vanilla procedure and the bottom shows the marginal density produced by the AR procedure ($R = 8$, $M = 128$). The blue plots on the right show a particular target density of interest.

[5] where $R = 0$ is the vanilla model

The difference between these plots is not as visually striking at that of the sawtooth experiment. However, we find that the AR procedure is beneficial here—increasing the log-likelihood from -1.11 to -0.16.

By taking a closer look at the specific marginal density of interest shown in blue, we can gain intuition for why we see this improvement. Namely, we see that the AR marginal places a much higher density on the target point (shown with the horizontal blue line). It does this by producing a heavy-tailed distribution. Though both procedures place a similar density at the mode of zero, this heavy-tailed distribution places a much higher density at the true target point.

4.2.4 Real Audio

In this Real Audio experiment, we use a model Q_θ which has been trained on tasks from the real audio $\backslash iy \backslash$ phoneme data generator as described in Chapter 3. Additionally, we construct our meta-test set Ξ_{test} by sampling functions from the this data generator. In particular, we sample a total of 96 functions–16 functions each for context set sizes 0, 10, 20, 40, 80, and 160. We chose to use more context for this experiment because this task is more difficult than our synthetic experiments. Later, we discuss how this decision may have been misguided for the purposes of exploring the efficacy of the AR procedure. Like the previous experiments, we use trajectories lengths (R) from 0 to 8, inclusive. We use number of trajectory samples (M) from 1 to 128, inclusive. We run this experiment using all three trajectory generators.

Here, as seen in Table 4.1 we also an improvement by using the AR procedure over the vanilla procedure, though these results show a smaller improvement over use of the vanilla model as compared to the sawtooth and real-audio experiments. Taking a closer look at the marginals for the particular function in Figure 4.4 may help reveal why.

First, we notice that the top plot (vanilla procedure) and the bottom plot (AR procedure), appear to place the areas of highest density in very similar locations. We also notice that there appears to be little difference between vanilla in AR in the particular marginal density highlighted in the blue plots on the right of the figure. Nevertheless, there does still appear to be an improvement in expected log-likelihood for this particular function (from 1.08 to 1.14). We suspect that the modeling errors have been reduced by the AR procedure's ability to model heavy-tailed distributions, as shown in the synthetic audio experiment. Though we also suspect that the marginals in this case are closer to Gaussian, perhaps because we are using more context points. In an upcoming section we will seek to better understand this question:

How does the context size impact the efficacy of the AR procedure?

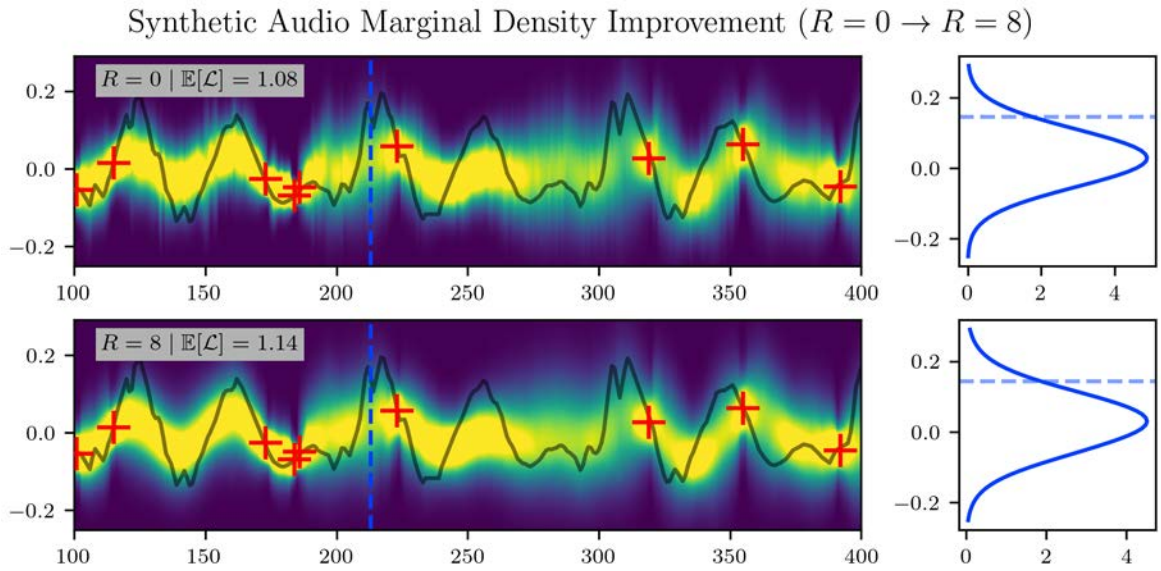


Fig. 4.4 An example task from the real audio experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

4.3 Monte Carlo Stopping Criteria and Convergence

Having built some understanding of the effects of using the AR procedure on a few example functions, we seek to better understand two parameters—the trajectory length (R) and the number of trajectories (M). First, we take a closer look at the two error sources ($\mathscr{E}_{MC}$ and $\mathscr{E}_{Q}^{AR}$, and $\mathscr{E}_{Q}^{Vanilla}$).

In order to better understand these error sources, we need to discuss notion of Monte Carlo error reduction and convergence. Note that we have overloaded the error terms to mean both the distributions which would make our marginals equivalent to the ground truth model as well as the difference in expected log-likelihood from the ground truth model over some meta-test set.

First, we present Figure 4.5 to illustrate the relationship between these sources of error. In this illustration, we assume we have utilized the AR procedure with a fixed trajectory length and varying number of trajectories, yielding the purple curve. We also have measured the vanilla procedure's expected log-likelihood over the same tasks. At the selected point in black, the Monte Carlo error is very large, making the AR procedure inferior to the vanilla procedure ($\mathscr{E}_{MC} + \mathscr{E}_{Q}^{AR} > \mathscr{E}_{Q}^{Vanilla}$). However, if we move further to the right along the purple

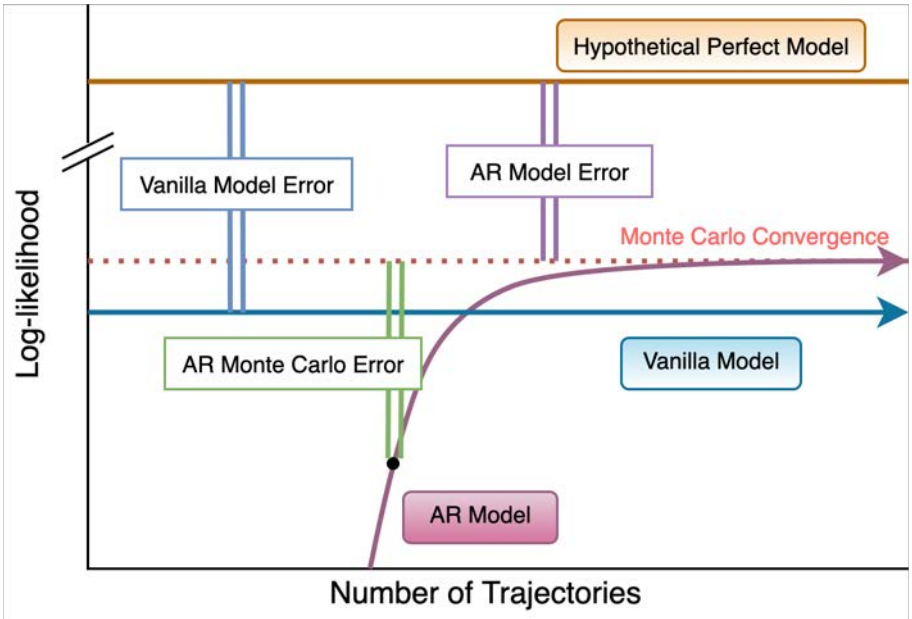


Fig. 4.5 Illustration of the interplay of the three error terms of interest: the AR Monte Carlo Error $\mathscr{E}_{MC}$, the AR modeling error $\mathscr{E}_Q^{AR}$, and the vanilla modeling error $\mathscr{E}_Q^{Vanilla}$. Here we see that the vanilla model (in blue) does not have any Monte Carlo error, the line is perfectly horizontal. For the AR model, however, we see that increasing the number of trajectories decreases the Monte Carlo error as it approaches Monte Carlo convergence. Notice that this convergence falls short of the log-likelihood of the hypothetical perfect ground truth model. This is because our AR procedure does not have access to the ground truth marginals, only models of these marginals Q_θ (See Section 3.0.1). The AR procedure is helpful when the sum of the Monte Carlo and AR modeling error is less than the vanilla modeling error. In this case, for the black point shown in this figure, the AR procedure is inferior to the vanilla procedure due to high Monte Carlo error. However, if we were to take a point further along the purple AR procedure curve, we would eventually surpass the vanilla procedure performance as our Monte Carlo error shrank.

AR curve, our Monte Carlo error reduces such that the AR procedure becomes superior to the vanilla procedure ($\mathscr{E}_{MC} + \mathscr{E}_Q^{AR} < \mathscr{E}_Q^{Vanilla}$). This occurs because we can reduce the Monte Carlo error arbitrarily by taking more and more trajectory samples, but the AR modeling error is less than the vanilla model. We expect this to be the case when the vanilla model's restriction to producing Gaussians makes it mal-adapted for modeling the marginals of a particular experiments' tasks. On the other hand, we expect the vanilla model to be at the line of AR Monte Carlo convergence when Gaussians can model the marginals well.

Because we can take arbitrarily many trajectory samples in order to decrease the Monte Carlo error, we are interested in identifying a number of trajectory samples after which taking

more samples provides little to no practical benefit. To this end we construct a heuristic for convergence.

First, we set a trajectory length and measure the expected log-likelihood using this trajectory length for differing number of trajectories (producing a curve like the purple one in Figure 4.5), giving us a sequence:

$$S = \left\{ \mathscr{L}^{AR}(\theta, n) \forall n \in 1, ..., M \right\} \tag{4.10}$$

where $\mathscr{L}^{AR}(\theta, n)$ is defined as the expected log-likelihood for some fixed trajectory length and a number of trajectories n. Next, we find differences between subsequent terms in this sequence, yielding:

$$D = \left\{ \mathscr{L}^{AR}_{diff}(\theta, n) = \mathscr{L}^{AR}(\theta, n) - \mathscr{L}^{AR}(\theta, n) \forall n \in \{2, ..., M\} \right\} \tag{4.11}$$

We expect this sequence to approach zero as the differences in subsequent terms decrease with more samples. Next, take the rolling mean with a window size of 5 over this sequence D, yielding:

$$D_{roll} = \left\{ \mathscr{L}^{AR}_{roll}(\theta, n) = \frac{1}{5} \sum_{k=0}^{4} \mathscr{L}^{AR}_{diff}(\theta, n-k) \forall n \in \{6, ..., M\} \right\} \tag{4.12}$$

We then select the convergence number of trajectories n^* as the number of trajectories after which there are no values of $\mathscr{L}^{AR}_{roll}(n)$ greater than 0.001. Under this convergence threshold, we expect to require more than 10 additional trajectory samples to achieve a log-likelihood reduction of 0.01, or about a 1% reduction in the predictive interval—which we consider to be the smallest reduction of practical significance. We consider this to be a reasonable bounds on the number of trajectories which mitigates an explosion in computational requirements. Having established this idea of convergence, we return to take a closer look at the AR procedure results for different selections of M and R.

4.3.1 Gaussian Process

Returning to the Gaussian Process experiment, in Figure 4.6, we see measures comparing the AR procedures with differing number of trajectories (M) and trajectory lengths (R). We increase our expected log-likelihood [6] as we increase the number of trajectory samples (M). This occurs because we are reducing our Monte Carlo error $\mathscr{E}_{MC}$ by taking more samples. Additionally, we notice that none of the AR procedures achieve performance equal

[6] Normalized by the number of target points

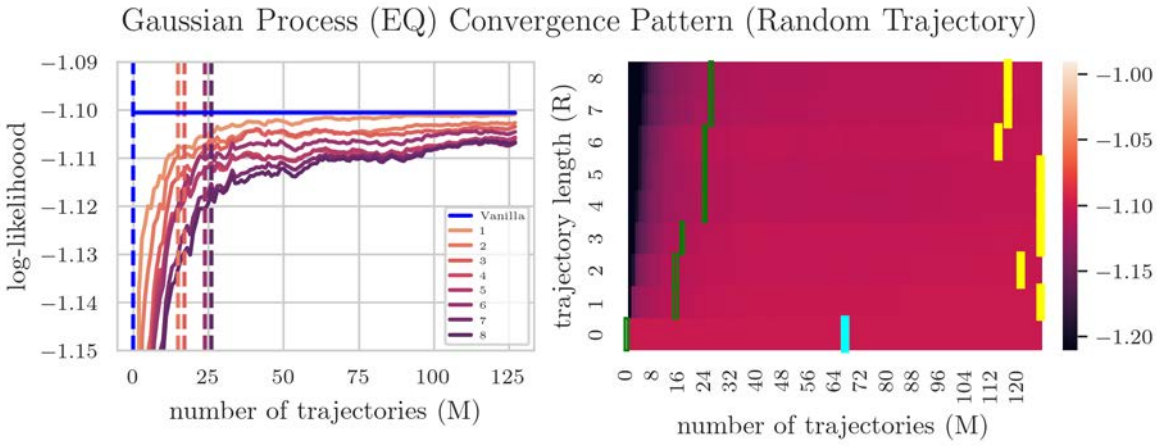


Fig. 4.6 The two plots seek to reveal patterns when changing the number of trajectories M and trajectory length R. On the left, lines representing different trajectory lengths. As we increase the number of trajectories , we increase the expected log-likelihood over all functions in this experiment. This happens because we are reducing the Monte Carlo error $\mathscr{E}_{MC}$ by taking more samples. The dotted vertical lines indicate the number of trajectories needed for the given procedure's log-likelihood to converge (as defined in Section 4.3). Here the convergence threshold is 0.001. The right plot shows the same information using a heatmap. The color indicates the log-likelihood, each cell corresponding to a particular number of trajectories and trajectory length. The green blocks indicate the convergence number of trajectories for the given trajectory length (rows). The yellow cells show the best number of trajectories for each given trajectory length. The teal cell shows the best performing combination of trajectory length and number of trajectories.

to the vanilla procedure. This further illustrates the point made earlier regarding Gaussian Processes—the vanilla procedure already models the marginals sufficiently well, so the AR procedure introduces Monte Carlo $\mathscr{E}_{MC}$ error without reducing the modeling error $\mathscr{E}_{Q}^{AR}$.

The right plot shows the same information in a different way. The expected log-likelihood is shown by the color and each cell represents the AR procedure log-likelihood for a particular trajectory length and number of trajectories. The green cells indicate the number of trajectories at which the given trajectory length achieves convergence (as defined in Section 4.3). The yellow cells show the number of trajectories that lead to the highest log-likelihood for a given trajectory length. The teal cell indicates the parameter selection which leads to the best performance overall. We see in this case that the best log-likelihood is achieved using the vanilla procedure[7].

[7]Only one cell is highlighted in teal for a given trajectory length, though all number of trajectories are equivalent under the vanilla procedure.

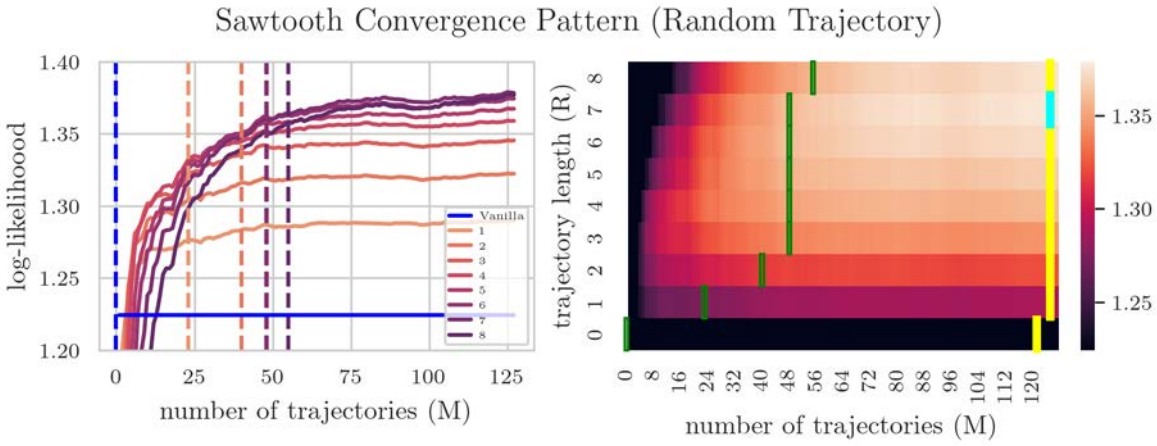


Fig. 4.7 The two plots seek to reveal patterns when changing the number of trajectories M and trajectory length R. On the left, lines representing different trajectory lengths. As we increase the number of trajectories , we increase the expected log-likelihood over all functions in this experiment. This happens because we are reducing the Monte Carlo error $\mathscr{E}_{MC}$ by taking more samples. The dotted vertical lines indicate the number of trajectories needed for the given procedure's log-likelihood to converge (as defined in Section 4.3). Here the convergence threshold is 0.001. The right plot shows the same information using a heatmap. The color indicates the log-likelihood, each cell corresponding to a particular number of trajectories and trajectory length. The green blocks indicate the convergence number of trajectories for the given trajectory length (rows). The yellow cells show the best number of trajectories for each given trajectory length. The teal cell shows the best performing combination of trajectory length and number of trajectories.

4.3.2 Sawtooth

For the Sawtooth experiment, we observe Figure 4.7. Like in the Gaussian Process experiment, we see that the expected log-likelihood increases for all trajectory lengths as we take a larger number of trajectories. Again, this is due to the reduction in Monte Carlo error. Noting the location of the vanilla procedure performance (blue), we see how this experiment differs from the Gaussian Process experiment. Longer trajectory lengths appear to generally decrease the AR procedure model error E_Q^{AR}[8] as the log-likelihood ceiling gets higher for large number of trajectories. Moreover, we see that procedures for all trajectory lengths surpass the vanilla procedure log-likelihood after about a dozen trajectory samples.

In the plot on the right, we can see the convergence pattern more clearly. Recalling that the green values indicate convergence number of trajectories, we see that the longer

[8] $R = 8$ is below $R = 7$ with $M = 128$, perhaps because $R = 8$ still hasn't sufficiently reduced Monte Carlo error to surpass $R = 7$ performance.

trajectories—though they seem to generally decrease model error by more—require more trajectory samples to converge.

Here we find that the best performance (shown in teal) is achieved using a trajectory length of 7 with and 127 trajectory samples.

4.3.3 Synthetic Audio

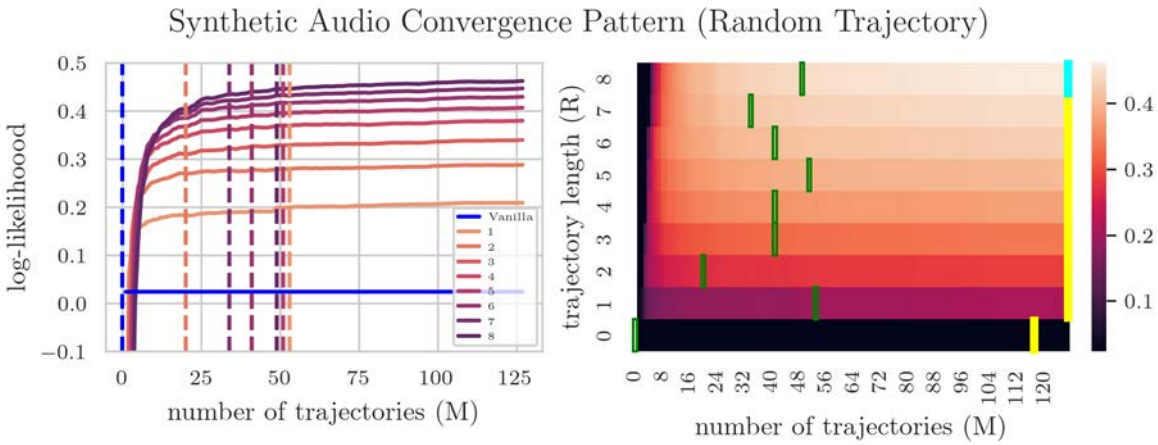


Fig. 4.8 The two plots seek to reveal patterns when changing the number of trajectories M and trajectory length R. On the left, lines representing different trajectory lengths. As we increase the number of trajectories , we increase the expected log-likelihood over all functions in this experiment. This happens because we are reducing the Monte Carlo error $\mathscr{E}_{MC}$ by taking more samples. The dotted vertical lines indicate the number of trajectories needed for the given procedure's log-likelihood to converge (as defined in Section 4.3). Here the convergence threshold is 0.001. The right plot shows the same information using a heatmap. The color indicates the log-likelihood, each cell corresponding to a particular number of trajectories and trajectory length. The green blocks indicate the convergence number of trajectories for the given trajectory length (rows). The yellow cells show the best number of trajectories for each given trajectory length. The teal cell shows the best performing combination of trajectory length and number of trajectories.

For the synthetic audio experiment, we see in Figure 4.8 a similar pattern to that seen in the sawtooth experiment in Figure 4.7. The Monte Carlo error decreases as we take more trajectory samples, and the log-likelihood ceiling gets higher with longer trajectory lengths—due to a decrease in modeling error. In this case, as in the sawtooth case, all trajectory lengths surpass the log-likelihood of the vanilla model (in blue).

Observing the right plot, the convergence pattern is less clear. Trajectory length 1, for example, does not converge until more than 50 trajectory samples, more than any of the

others trajectory lengths. Though at the scale presented, the trajectory length curves appear quite smooth, closer inspection reveals kinks in the curve—numbers of trajectories which lead to a significant jump in log-likelihood.

Though this requires more careful inspection, we believe this phenomena arises from the sparsity of the distributions in question. Recall that at each step of our AR sampling process (see 3.21), we create a target sample $y_j^{(i)}$ by drawing from the model $Q_\theta\left(y_j \mid C, (\{x_j, y_j\}_{j=0}^{l-1})^{(i)}, x_j\right)$, which is a Gaussian. If the true marginals are heavy tailed, this may lead to some portions of the density being under-explored because these samples are too focused around the mode. This phenomena may motivate the exploration of using importance sampling with the AR procedure when handling distributions suspected to be heavy-tailed.

4.3.4 Real Audio

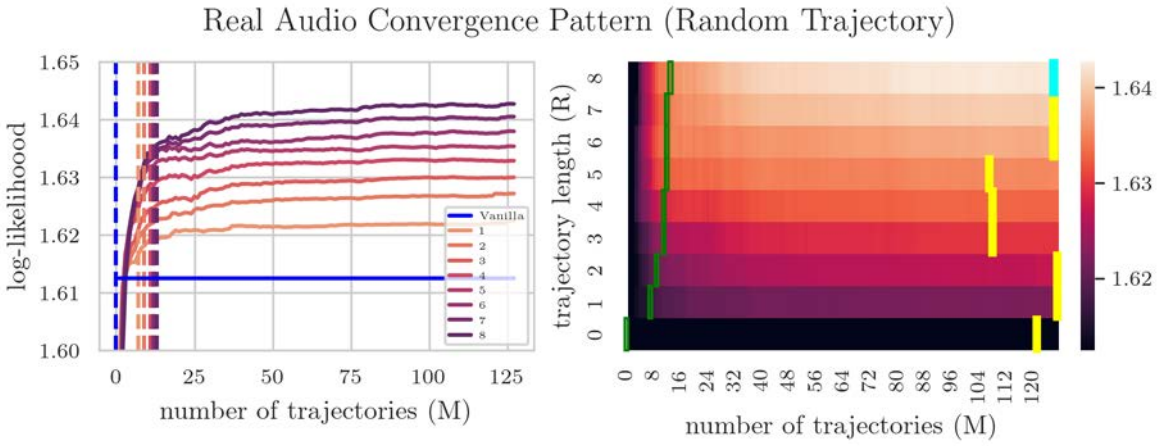


Fig. 4.9 The two plots seek to reveal patterns when changing the number of trajectories M and trajectory length R. On the left, lines representing different trajectory lengths. As we increase the number of trajectories , we increase the expected log-likelihood over all functions in this experiment. This happens because we are reducing the Monte Carlo error $\mathscr{E}_{MC}$ by taking more samples. The dotted vertical lines indicate the number of trajectories needed for the given procedure's log-likelihood to converge (as defined in Section 4.3). Here the convergence threshold is 0.001. The right plot shows the same information using a heatmap. The color indicates the log-likelihood, each cell corresponding to a particular number of trajectories and trajectory length. The green blocks indicate the convergence number of trajectories for the given trajectory length (rows). The yellow cells show the best number of trajectories for each given trajectory length. The teal cell shows the best performing combination of trajectory length and number of trajectories.

For the real audio experiment, we again see in Figure 4.9 a familiar pattern: decreasing Monte Carlo error with larger number of trajectories; decreasing AR model error with

increasing trajectory length. And again, all AR procedures eventually achieve better log-likelihoods than the vanilla model—all after just a few trajectory samples.

Observing the plot on the right, we see one notable difference from the previous experiments: the Monte Carlo convergence occurs with fewer trajectory samples (smaller M). We believe this is due to a major experimental difference between this experiment and the others: we use much larger context sizes (0, 10, 20, 40, 80, 160). This larger context reduces the variance on the trajectory samples and leads to marginals which are more Gaussian. This leads to both a faster reduction in Monte Carlo error and a smaller improvement over the vanilla model. The relationship between context size and the AR and vanilla procedure performances will be explored in the next section.

4.4 Understanding Impact of Context Size

As we've taken closer looks at some specific marginals on specific functions and some log-likelihood convergence patterns in expectation, questions have arisen regarding the impact of the context size on the AR procedure's performance. Here we seek gain better purchase on these questions by examining a few plots.

We are interested in the performance differences between the AR procedure and the vanilla procedure for different context sizes. Additionally, we want to see the relative difference between the AR procedure and the vanilla procedure when compared to the Naive model.

4.4.1 Gaussian Process

Observing the bottom plot of Figure 4.10, we can get a visual intuition for the relative difference between vanilla procedure (in blue) and the AR procedure (in ruddy hues) and the Naive baseline model (in black). For the Gaussian Process experiment, the bottom plot of absolute log-likelihoods[9] shows a few patterns:

1. The vanilla procedure performs better than the baseline. This indicates to us that our model is performing as expected.

2. The vanilla procedure achieves increasingly higher log-likelihood with greater context size. This is to be expected as more context should narrow our predictive estimates. The Naive model, on the other hand, is unchanged with more context[10].

[9] in expectation

[10] The Naive model receives no new information upon seeing this context.

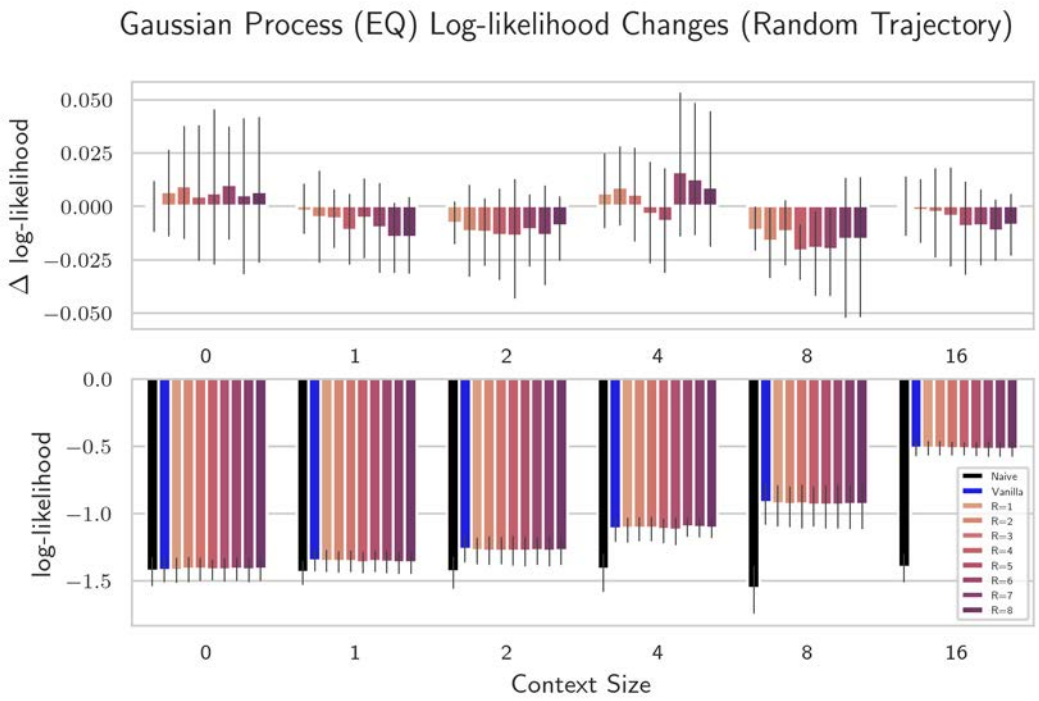


Fig. 4.10 In the bottom plot we see the absolute expected log-likelihood using the AR procedure for different trajectory lengths and context sizes. For each trajectory length, the number of trajectories used is that at which that trajectory length achieves Monte Carlo convergence (as defined in 4.3). In the top plot, we see the difference between AR procedures of differing trajectories and the vanilla procedure. Error bars represent 95% confidence interval obtained through bootstrapping. No difference between AR procedures and vanilla procedure are significant. Log-likelihoods can be seen at Table A.8.

3. The vanilla and AR procedures appear to be quite close in absolute log-likelihoods, when compared to the Naive baseline.

In the top plot of Figure 4.10, we take a look at the difference of log-likelihood between the AR procedures and the vanilla procedure. We find that only a few AR procedures lead to an improvement over the vanilla procedure and that those improvements are small (0.01) with large confidence intervals.

Overall, the AR and vanilla procedures appear to have approximately the same performance across context sizes. This is unsurprising, as we do not expect the AR procedure to help with GP marginal modeling.

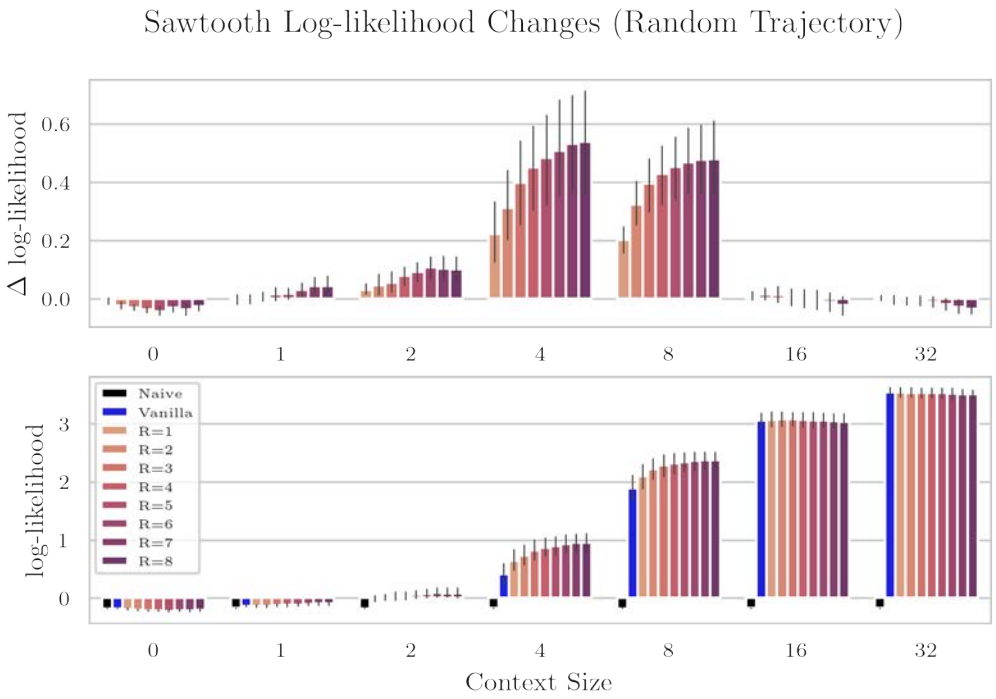


Fig. 4.11 In the bottom plot we see the absolute expected log-likelihood using the AR procedure for different trajectory lengths and context sizes. For each trajectory length, the number of trajectories used is that at which that trajectory lengths achieves Monte Carlo convergence (as defined in 4.3). In the top plot, we see the difference between AR procedures of differing trajectories and the vanilla procedure. Error bars represent 95 % confidence interval obtained through bootstrapping. Log-likelihoods can be seen at Table A.2.

4.4.2 Sawtooth

Now, we observe Figure 4.11 for the Sawtooth experiment. Observing the bottom plot, we see a few interesting trends:

1. As in the previous experiment, the vanilla procedure (blue) outperforms the Naive model (black)—increasingly so with larger context size.

2. The AR procedure outperforms the vanilla procedure for context sizes 1, 2, 4, and 8.

3. For these context sizes, a larger trajectory length (darkening ruddy hues) leads to greater log-likelihood.

4. The relative improvement for increasing the trajectory falls off as we use larger trajectory lengths.

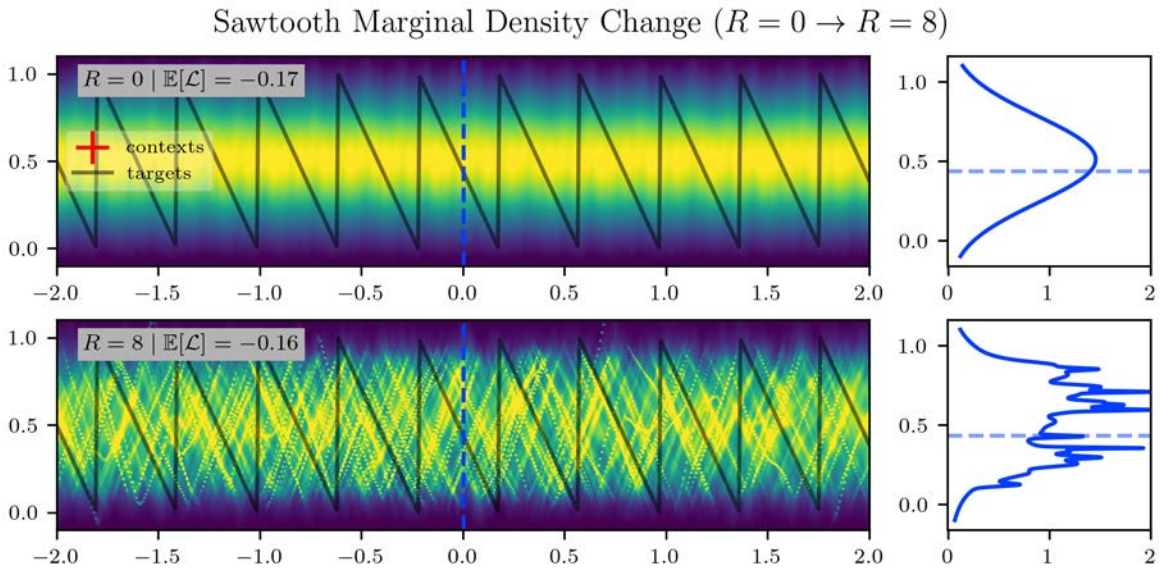


Fig. 4.12 An example sawtooth task with zero context. We see that the marginals produced by the AR procedure are very spike-y. With zero context, the AR procedure appears to have slower rates of Monte Carlo error reduction.

5. For context sizes 0, 16, and 32, the AR procedure appears to typically achieve lower log-likelihoods than the vanilla procedure.

Regarding 2, we believe that for context set size of zero, the AR procedure struggles to improve over the vanilla procedure's Gaussian marginal one major reason: with a context set size of the zero, the randomness of the sawtooth shift ϕ (see sawtooth data generator in Section 3.2.2) means that the sawtooth function could be placed anywhere within the domain. This means we will not end up with bimodal marginals. In fact, the marginals will be uniform between 0 and 1. In this scenario, the AR procedure may recover its Monte Carlo error very slowly because the resulting GMM requires many mixing components to adequately model the infinite modes of the uniform distribution. Consequently, without taking very many trajectory samples, simply using the single Gaussian from the vanilla procedure leads to better log-likelihoods. The phenomena can be observed in Figure 4.12.

For context set sizes 1, 2, 4, and 8, on the other hand, the AR procedure increases the log-likelihood substantially. This occurs because of the bi-modality of the true marginal leading to a high error using the vanilla procedure, which attempts to place a Gaussian to sufficiently cover both modes. The AR procedure, on the other hand, can model both marginals using the generated GMM (Figure 4.2).

Observing a particular task with context set size 4 in Figure 4.13, we notice that one mode has come to mostly dominate the density at the particular target shown in blue on the

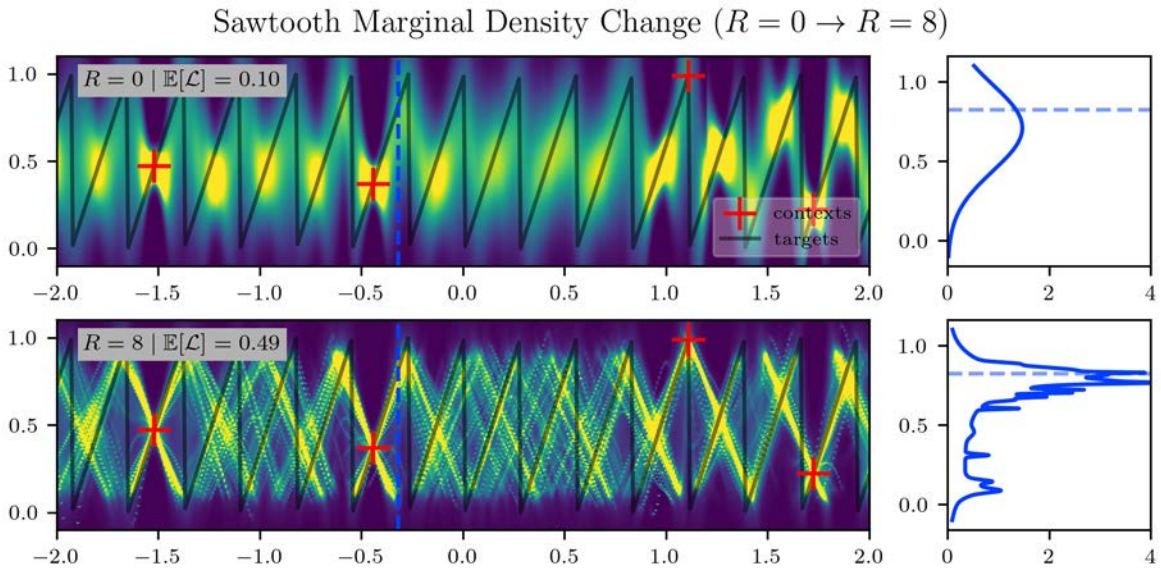


Fig. 4.13 An example task from the Sawtooth experiment with 4 context points, and the target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

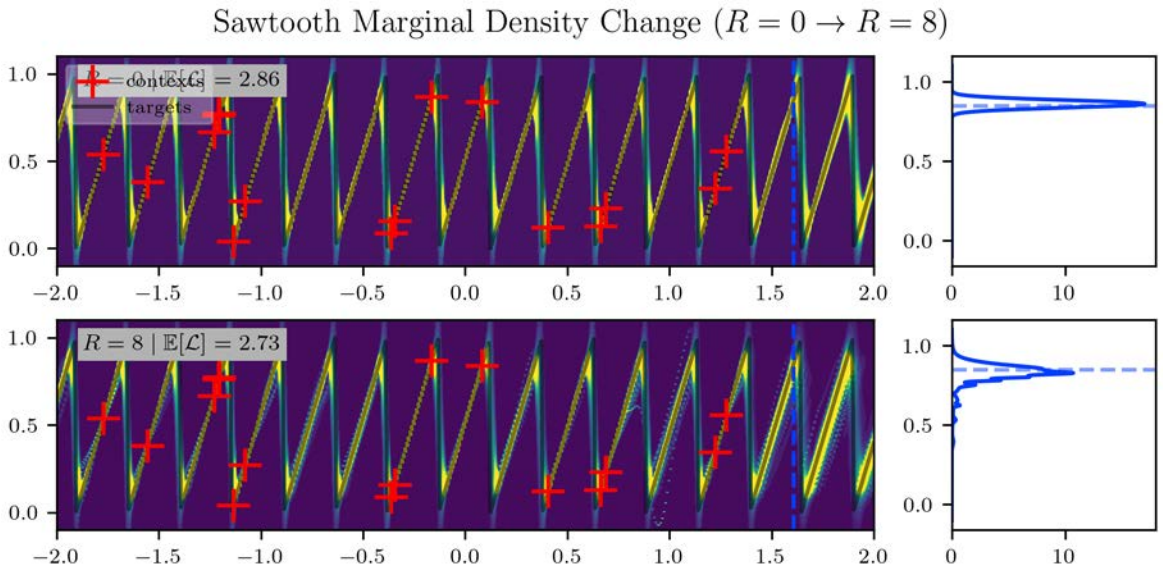


Fig. 4.14 An example task from the Sawtooth experiment with 16 context points, and the target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output.

right plots. The AR procedure places a much higher density at this location, which is close to the true target, whereas the vanilla procedure's mode is shifted slightly downwards and is of much lower density.

What happens when the true sawtooth marginal density is almost entirely dominated by a single mode? In this scenario, the AR procedure produces little to no benefit, which we observe with context sizes 16 and 32. Figure 4.14 shows this scenario. There is sufficient information from these context points to allow the vanilla Gaussian to perform well by placing its single-mode, light-tailed density such that: $\mathscr{E}_{MC} + \mathscr{E}_{Q}^{AR} > \mathscr{E}_{Q}^{Vanilla}$.

We can also observe the convergence patterns for differing contexts sizes in Figure 4.15. We that with a context set size of 2 and 4, we never actually achieve convergence for some trajectory lengths. Therefore, there may still be significant log-likelihood gains to be had by adding more trajectory samples for these context sizes. For a context set size of 16, on the other hand, convergence is reached more quickly—likely due to lower variance of trajectory samples.

Now, we wish to gain a better understanding of the computational costs of these log-likelihood changes. Given the procedure outlined in Equation 3.21, we see that for a

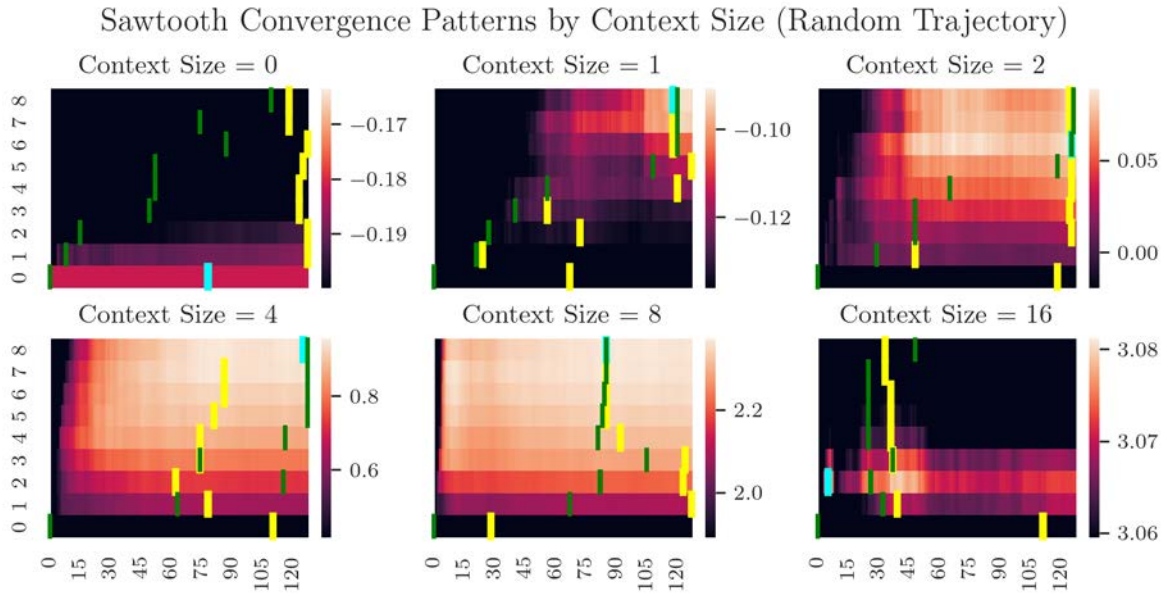


Fig. 4.15 Here we see several heatmaps revealing convergence patterns for various context sizes on the sawtooth experiment. Each cell represents the expected log-likelihood (indicate by the hue) of a given number of trajectories (columns) and trajectory length (rows). The cells highlighted in green show the number of trajectories for which the AR procedure using a given trajectory length achieves convergence (as defined in Section 4.3). The yellow cells indicate the number of trajectories for a given trajectory length where the AR procedure has the best performance. The teal cell shows the combination of number of trajectories and trajectory lengths that yields the best performance overall for that context size.

trajectory length of R, M trajectory samples, we must execute the model $(M) \times (R+1)$ times. Recall that in Figure 4.11, we use the convergence number of trajectories for each given trajectory length. Therefore, we can compute the total number of model executions for each trajectory length and normalize the log-likelihood changes by this number. The results of this calculation can be seen in Figure 4.16.

In the bottom plot we see the total number of model executions for each trajectory length. In the top plot, we see the log-likelihood change normalized by model executions. From context set sizes 2, 4, and 8, we see a trajectory length of 1 appears to lead to the greatest log-likelihood change per model-execution, with the gains trending down thereafter.

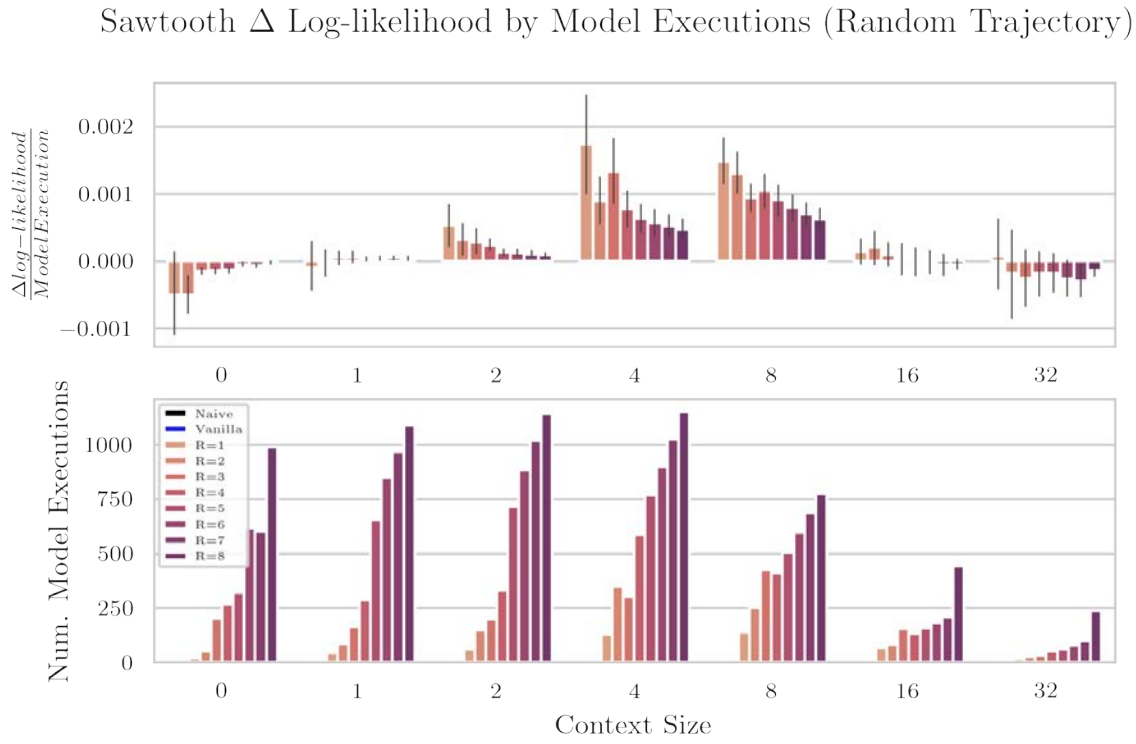


Fig. 4.16 In the bottom plot, we see the number of executions (forward passes) of the model Q_θ for each trajectory length. For a given trajectory length R, convergence number of trajectory samples M is used (see Section4.3). The number of model executions is then $M \times (R+1)$. The top plot then takes the difference between the AR procedure and vanilla procedure log-likelihood for each trajectory length and divides is by the number of model executions, yielding the difference per model execution.

4.4.3 Synthetic Audio

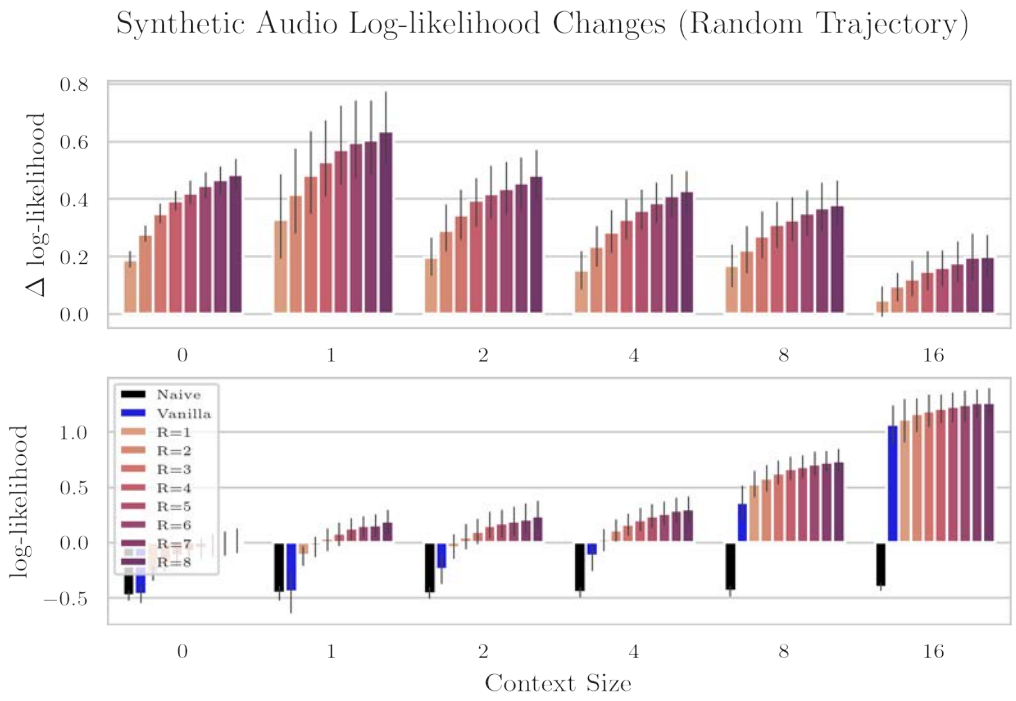


Fig. 4.17 In the bottom plot we see the absolute expected log-likelihood using the AR procedure for different trajectory lengths and context sizes. For each trajectory length, the number of trajectories used is that at which that trajectory lengths achieves Monte Carlo convergence (as defined in 4.3). In the top plot, we see the difference between AR procedures of differing trajectories and the vanilla procedure. Error bars represent 95 % confidence interval obtained through bootstrapping. Log-likelihoods can be seen at Table A.4.

Now, we comment on the impact of context size on the AR vs vanilla procedure performance on the synthetic audio experiment. In Figure 4.17, we see some similarities with the sawtooth results, and a few differences. First, we note that the the vanilla model improves over the naive model—increasingly so with more context. Like before, this indicates our model Q_θ is well trained. Second, we also see that larger trajectory lengths lead to greater log-likelihood gains at a decreasing rate.

Now, for some notable differences. One, the AR procedure significantly outperforms the vanilla procedure with zero context. This occurs because the true marginal with zero context should be heavily concentrated around zero with heavy tails which can place some density on the wave peaks. Therefore, the true marginal with zero context should bear more resemblance to a Laplace distribution than to a Gaussian. We see that the AR procedure better captures this distribution in Figure 4.18.

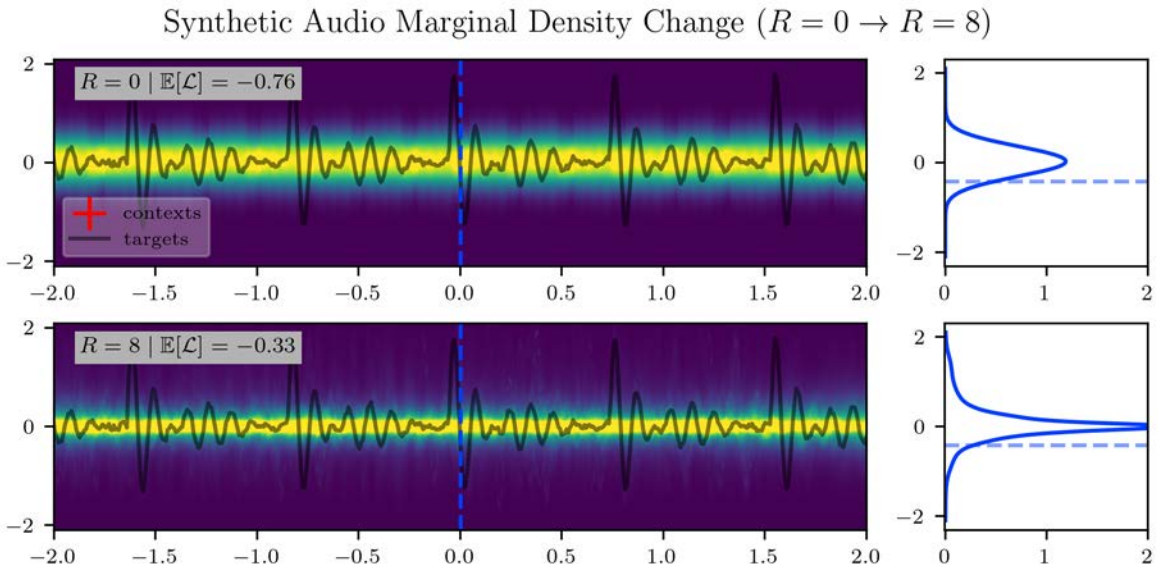


Fig. 4.18 An example task from the synthetic audio experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output. This example uses zero context points.

We also note that—unlike in the sawtooth experiment—the AR procedure outperforms the vanilla procedure even with a context set size of 16. We believe this occurs because the synthetic audio has a random period and that many context point observations close to zero are common and relatively uninformative. Therefore, the vanilla model does not have sufficient context to place a very high density on the target such that the model error is commensurate with the density placed by the peakier, heavier-tailed AR procedure density. This question could be more adequately answered by using larger context sizes with the synthetic audio model.

Now, we can observe the convergence patterns by context size in Figure 4.20. We see a general trend of larger trajectory lengths requiring more trajectory samples to reach convergence, though this trend is not present for a context set size of 16.

As before, we can also normalize the log-likelihoods by the number of model executions, giving us Figure 4.21. Like in the sawtooth experiment, we see a trend of shorter context lengths giving more log-likelihood gains per model execution—with a few exceptions. For example, for context set size 4, trajectory length of 2 yields more gains per model execution

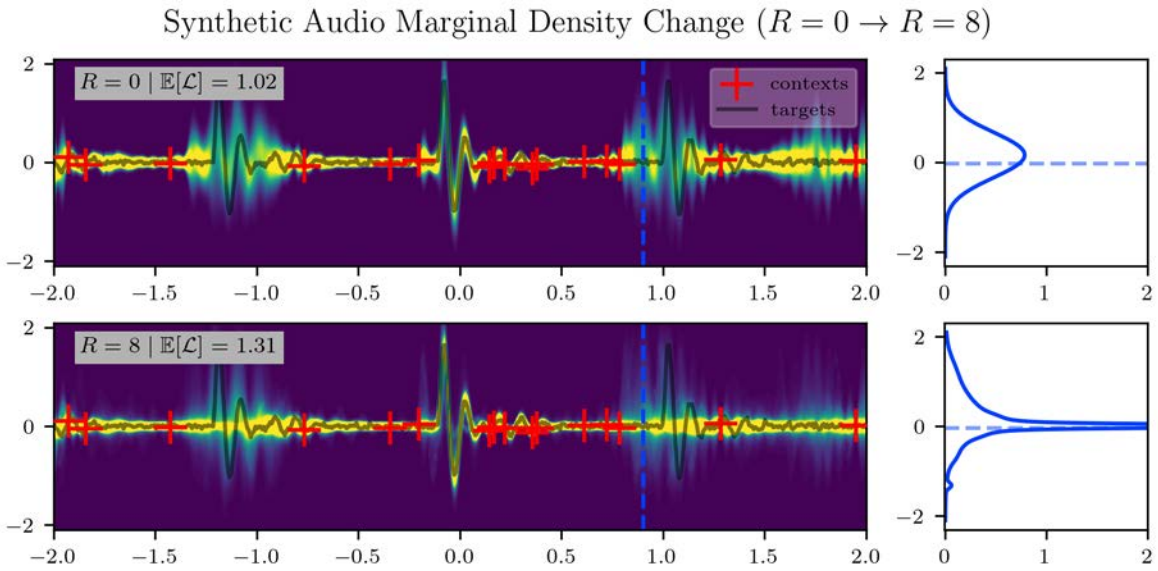


Fig. 4.19 An example task from the real audio experiment with target marginal densities created with two methods. In the top plot, we use use the Vanilla model ($R = 0$) to create the marginal densities, indicated by the color (yellow as areas of high density, purple as areas of low density). The bottom plot uses an AR model with $R = 8$, and $M = 128$. The two plots on the right show particular the density on one particular target input indicated with the vertical dotted blue line on the left plots. The horizontal dotted blue line on the right plots shows the location of the true target output. This example uses 16 context points.

than trajectory length of 1. We can see that this is due to fewer overall execution being run for trajectory length 2 due to faster convergence.

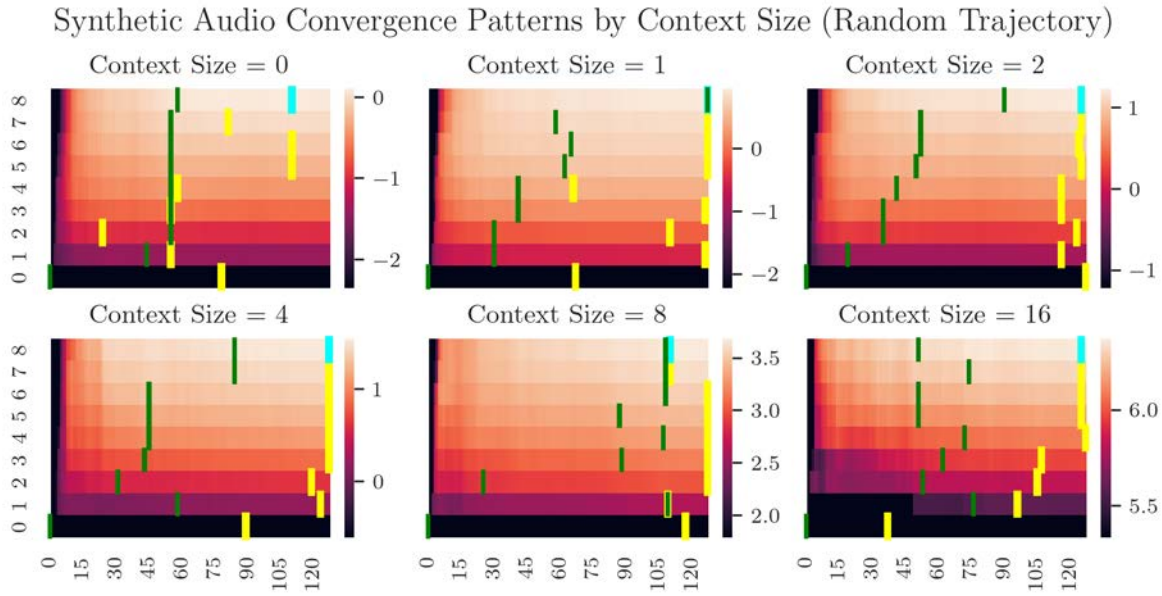


Fig. 4.20 Here we see several heatmaps revealing convergence patterns for various context sizes on the synthetic audio experiment. Each cell represents the expected log-likelihood (indicate by the hue) of a given number of trajectories (columns) and trajectory length (rows). The cells highlighted in green show the number of trajectories for which the AR procedure using a given trajectory length achieves convergence (as defined in Section 4.3). The yellow cells indicate the number of trajectories for a given trajectory length where the AR procedure has the best performance. The teal cell shows the combination of number of trajectories and trajectory lengths that yields the best performance overall for that context size.

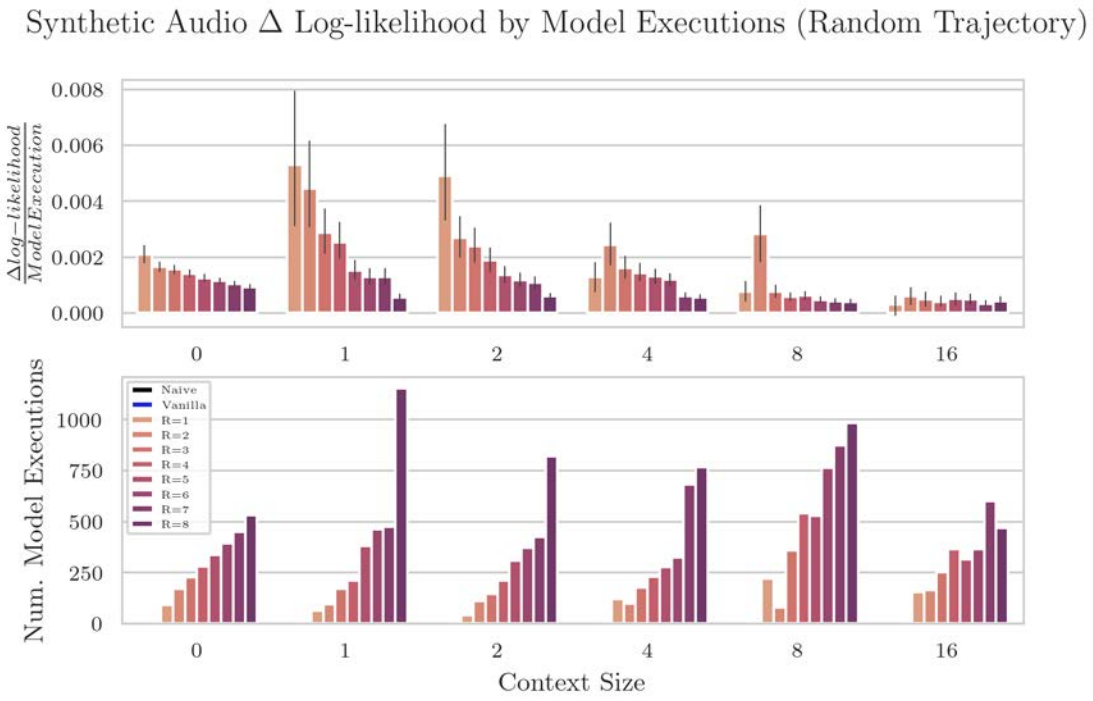


Fig. 4.21 In the bottom plot, we see the number of executions (forward passes) of the model Q_θ for each trajectory length. For a given trajectory length R, convergence number of trajectory samples M is used (see Section4.3). The number of model executions is then $M \times (R+1)$. The top plot then takes the difference between the AR procedure and vanilla procedure log-likelihood for each trajectory length and divides is by the number of model executions, yielding the difference per model execution.

4.4.4 Real Audio

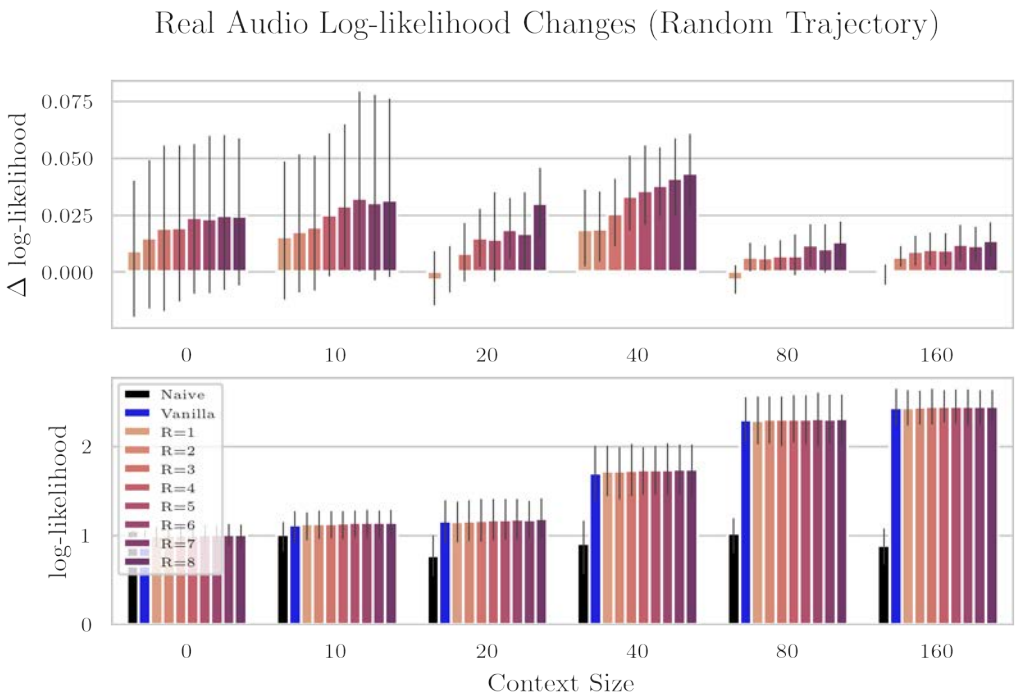


Fig. 4.22 In the bottom plot we see the absolute expected log-likelihood using the AR procedure for different trajectory lengths and context sizes. For each trajectory length, the number of trajectories used is that at which that trajectory lengths achieves Monte Carlo convergence (as defined in 4.3). In the top plot, we see the difference between AR procedures of differing trajectories and the vanilla procedure. Error bars represent 95 % confidence interval obtained through bootstrapping. Log-likelihoods can be seen at Table A.6.

Finally, we investigate the effect of context size on the AR vs vanilla procedure performance for the real audio experiment. We notice in Figure 4.22 that the vanilla model outperforms the naive model more and more so as more context is added. The AR procedure does not appear to give substantial gains over the vanilla procedure relative to the naive model. Taking a closer look at the AR-vanilla procedure performance differences in the top plot, we see that the most substantial differences are seen with a context set size of 40. We also see the familiar pattern where longer trajectory lengths lead to increasing log-likelihoods at a decreasing rate. We note, however, than none of these differences are significant ($p < 0.05$, see Table A.6).

In Figure 4.23, we see the convergence patterns by context size. We note that log-likelihood generally achieves convergence sooner in this experiment than in the synthetic

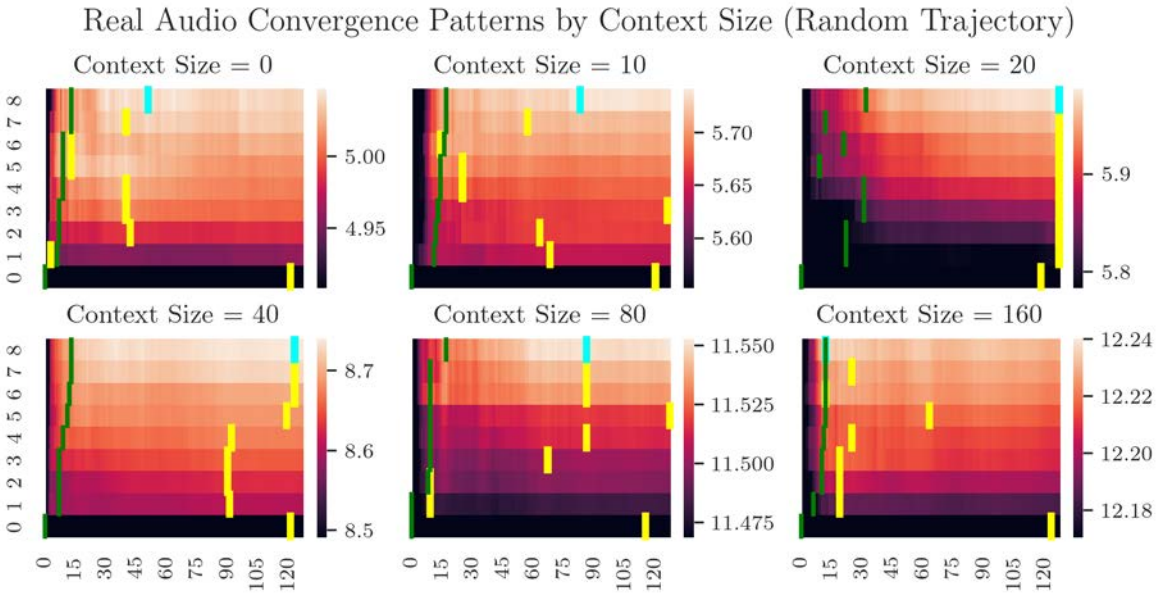


Fig. 4.23 Here we see several heatmaps revealing convergence patterns for various context sizes on the real audio experiment. Each cell represents the expected log-likelihood (indicate by the hue) of a given number of trajectories (columns) and trajectory length (rows). The cells highlighted in green show the number of trajectories for which the AR procedure using a given trajectory length achieves convergence (as defined in Section 4.3). The yellow cells indicate the number of trajectories for a given trajectory length where the AR procedure has the best performance. The teal cell shows the combination of number of trajectories and trajectory lengths that yields the best performance overall for that context size.

audio and sawtooth experiments. This likely occurs because we are using larger context sizes, leading to a lower variance in trajectory samples.

We can also look at the log-likelihood changes normalized by model executions in Figure 4.24. We find that context set size of 40 generally obtains the greatest log-likelihood increase per execution. We also see, again, that the shorter trajectories generally have greater log-likelihood delta by execution.

Overall, we find that the AR gains over the vanilla procedure are more measured in this experiment, especially when compared to the synthetic audio experiment. This may be due to a Gaussian being more appropriate in this context because the data is much noisier than in the synthetic audio experiment. In the synthetic audio experiment, a very peaky heavy-tailed distribution could be placed due to low noise, but there is not enough certainty to do so here because of the noise.

Further work should explore the effect of noise. For example, one may try many variations of the synthetic audio experiment with increasing noise levels.

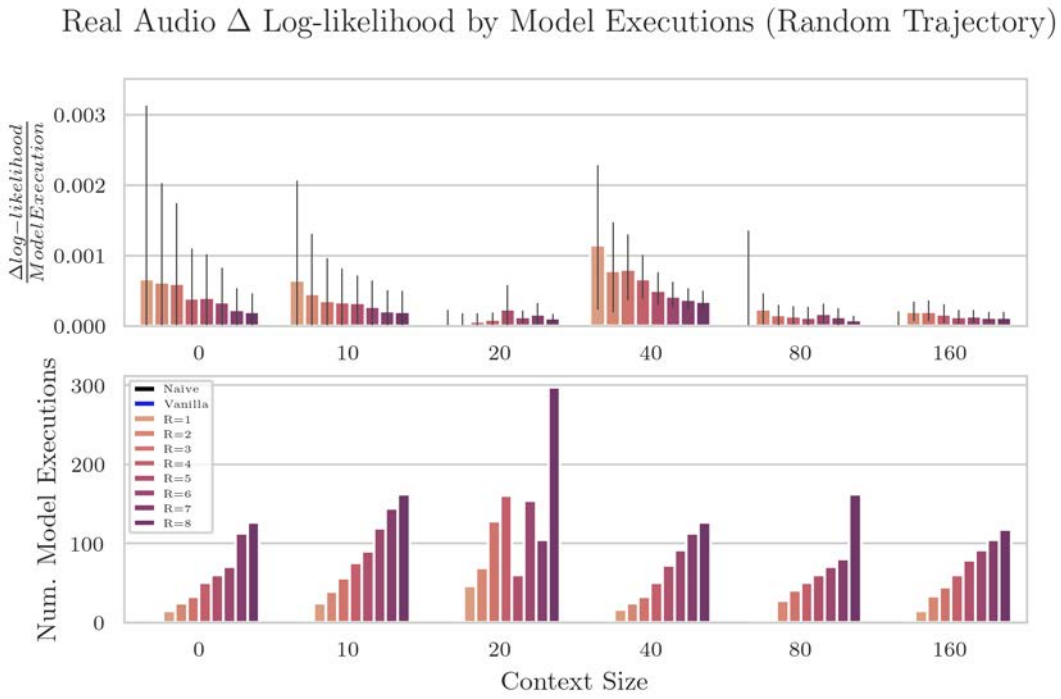


Fig. 4.24 In the bottom plot, we see the number of executions (forward passes) of the model Q_θ for each trajectory length. For a given trajectory length R, convergence number of trajectory samples M is used (see Section4.3). The number of model executions is then $M \times (R+1)$. The top plot then takes the difference between the AR procedure and vanilla procedure log-likelihood for each trajectory length and divides is by the number of model executions, yielding the difference per model execution. This plot has a minimum of 0, which cuts off some of the lower parts of the confidence intervals.

Table 4.2 Log-likelihood Changes with Different Trajectory Generators

Experiment	Emanate	Grid	Random
Real Audio	1.77 ± 0.15	1.76 ± 0.16	1.77 ± 0.16
Sawtooth	1.65 ± 0.29	1.57 ± 0.28	1.64 ± 0.29
Synthetic Audio	0.56 ± 0.10	0.54 ± 0.11	0.55 ± 0.10

Table 4.3 Expected log-likelihoods using different trajectory generation methods described in Section 3.1. Context set size of 0 has been omitted in the calculation of these log-likelihoods because using zero context is ill-defined for the Emanate trajectory generator.

4.5 Effect of Trajectory Generation Method

In the above analysis, we have been focused on focused on the impact of trajectory length R and number of trajectory samples M as they relate to the AR procedure performance for varying context sizes. The above analysis has been limited to random trajectory generation. Now we take a look at the impact of the trajectory generation on the AR procedure's performance.

In Table 4.3, we see an overview of the expected log-likelihoods across all tasks with non-zero context [11]. We did not find any significant differences between using these different trajectory generation strategies in the resulting log-likelihoods.

We note that the Random and Grid trajectory generators are agnostic to context or the target point which we are predicting for. The Emanate trajectory generator does use information from the context set when creating trajectories. None of them, however, use information about the specific target point of interest. Follow on work might investigate trajectory generation strategies which do use this information.

[11] Emanate trajectory generator requires non-zero context

Chapter 5

Conclusion

We have presented a novel method for using a trained CNP model to generate flexible marginal distributions by altering the test-time procedure using Monte Carlo integration and Autoregressive Sampling. We have shown this model to be effective at creating multi-modal and heavy tailed distributions which achieve higher expected log-likelihoods across three tasks: two synthetic and one real-world.

In our discussion, we found a key equation that–when true–tells us that our method improves over the vanilla model:

$$\mathscr{E}_{MC} + \mathscr{E}_{Q}^{AR} < \mathscr{E}_{Q}^{Vanilla} \tag{5.1}$$

We observed in situations where the true marginals are heavy tailed or multi-modal, we could use the AR procedure such that $\mathscr{E}_{Q}^{AR} < \mathscr{E}_{Q}^{Vanilla}$, and then we could take sufficiently many trajectory samples to reduce our Monte Carlo error $\mathscr{E}_{MC}$.

We found that for the sawtooth and real audio experiments, having a large context set led to the marginals being modeled sufficiently well by Gaussians such that using the AR procedure degraded marginal quality through the introduction of Monte Carlo error. We also found that the AR procedure struggled to recover the uniform marginal from having zero context on the sawtooth experiment.

Generally, we found that this method has an impressive capacity for improving marginal quality and does so reliably if sufficient trajectory samples are taken. The model performs particularly well in scenarios when the marginals have a few modes or are heavy-tailed, but is not justified when there are many context points.

Further experiments should examine the impact of aleatoric uncertainty in the underlying stochastic process on the effectiveness of this method. The insignificant improvements using

the AR procedure on the real world audio may indicate that the procedure works best when the data we model have little noise.

The potential usage of importance sampling in trajectory generation could be explored in cases where it is expected that sampling trajectory outputs from a Gaussian leads to samples which are too focused around the Gaussian mode. There could be improvements in the Monte Carlo convergence rates, especially for processes with heavy-tailed marginals.

The effectiveness of this process could also be explored further by again applying the chain rule of probability over the target points to evaluate the joint (as opposed to using a factorized joint). For example, by:

$$\mathscr{L}(\theta) = \log \sum_{i=0}^{M} \prod_{k=1}^{T} Q_\theta(y_k^{(T)} \mid \overbrace{C, \{\{x_j, y_j\}_{j=0}^{R}\}^{(i)}, \{x_l^{(T)}, y_l^{(T)}\}_{l=1}^{k-1}}^{Pseudo-Context}, x_k^{(T)}) \tag{5.2}$$

Finally, one might train a CNP model using the AR procedure—by updating parameters using the negative of Equation 5.3 as the loss function:

$$\mathscr{L}_{AR}(\theta) = \underset{(C,(\mathbf{x}_\tau,\mathbf{y}_\tau))\sim\Xi_{test}}{-\mathbb{E}} \left[\left[\sum_{k=1}^{T} \log \left(\sum_{i=1}^{M} Q_\theta \left(y_k \mid C, \{\{x_j, y_j\}_{j=0}^{R}\}^{(i)}, x_k \right) \right) \right] - \sum_{k=1}^{T} \log(M) \right] \tag{5.3}$$

In this way, the training procedure would be explicitly focused on maximizing the AR procedure's construction of the marginal predictives. Training in this manner might take some cues from diffusion models [Sohl-Dickstein et al., 2015], which have a similar construction.

References

Wessel P. Bruinsma, James Requeima, Andrew Y. K. Foong, Jonathan Gordon, and Richard E. Turner. The Gaussian Neural Process. pages 1–34, 2021. URL http://arxiv.org/abs/2101.03606.

Iva Cvorovic-Hajdinjak, Andjelka B. Kovacevic, Dragana Ilic, Luka C. Popovic, Xinyu Dai, Isidora Jankov, Viktor Radovic, Paula Sanchez-Saez, and Robert Nikutta. Conditional Neural Process for non-parametric modeling of AGN light curve. *Astronomische Nachrichten*, 343(1-2), nov 2021. doi: 10.1002/asna.20210103. URL http://arxiv.org/abs/2111.09751http://dx.doi.org/10.1002/asna.20210103.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. mar 2017. URL http://arxiv.org/abs/1703.03400.

Marta Garnelo, Dan Rosenbaum, Chris J. Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J. Rezende, and S. M.Ali Eslami. Conditional neural processes. *35th International Conference on Machine Learning, ICML 2018*, 4:2738–2747, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural Processes. 2018b. URL http://arxiv.org/abs/1807.01622.

John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93:27403, 1993.

Jonathan Gordon. Advances in Probabilistic Meta-Learning and the Neural Process Family. (November), 2021.

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional Conditional Neural Processes. (i):1–32, 2019. URL http://arxiv.org/abs/1910.13556.

Peter Holderrieth, Michael J Hutchinson, and Yee Whye Teh. Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes. In *International Conference on Machine Learning*, pages 4297–4307. PMLR, 2021.

Makoto Kawano, Wataru Kumagai, Akiyoshi Sannai, Yusuke Iwasawa, and Yutaka Matsuo. Group equivariant conditional neural processes. *arXiv preprint arXiv:2102.08759*, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive Neural Processes. jan 2019. URL http://arxiv.org/abs/1901.05761.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, dec 2015. ISSN 10959203. doi: 10.1126/science.aab3050. URL https://www.sciencemag.org/lookup/doi/10.1126/science.aab3050.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. 521(7553):436–444, May 2015. ISSN 14764687. doi: 10.1038/nature14539.

Stratis Markou, James Requeima, Wessel P. Bruinsma, Anna Vaughan, and Richard E. Turner. Practical Conditional Neural Processes Via Tractable Dependent Predictions. mar 2022. URL http://arxiv.org/abs/2203.08775.

Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

Tung Nguyen and Aditya Grover. Transformer Neural Processes: Uncertainty-Aware Meta Learning Via Sequence Modeling. jul 2022. URL http://arxiv.org/abs/2207.04179.

James Robert, Marc Webbie, et al. Pydub, 2018. URL http://pydub.com/.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

Richard E. Turner. Neural Processes for Data Efficient Machine Learning. 2022.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural Autoregressive Distribution Estimation. may 2016. URL http://arxiv.org/abs/1605.02226.

Anna Vaughan, Will Tebbutt, J. Scott Hosking, and Richard E. Turner. Convolutional conditional neural processes for local climate downscaling. jan 2021. URL http://arxiv.org/abs/2101.07950.

Bruinsma Wessel, Tom Andersson, Stratis Markou, and James Requeima. neuralprocesses, 5 2022. URL https://github.com/wesselb/neuralprocesses.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Appendix A

Tables

Table A.1 Sawtooth Experiment Log-likelihoods.

	1	2	4	8	16	32
Naive	-0.17 $\pm$ 0.01	-0.18 $\pm$ 0.00	-0.18 $\pm$ 0.01	-0.18 $\pm$ 0.00	-0.18 $\pm$ 0.00	-0.17 $\pm$ 0.00
Vanilla	-0.14 $\pm$ 0.02	-0.02 $\pm$ 0.05	0.42 $\pm$ 0.19	1.89 $\pm$ 0.23	3.06 $\pm$ 0.13	3.54 $\pm$ 0.08
1	-0.14 $\pm$ 0.02	0.01 $\pm$ 0.05	0.64 $\pm$ 0.17	2.09 $\pm$ 0.22	3.07 $\pm$ 0.13	3.54 $\pm$ 0.08
2	-0.14 $\pm$ 0.02	0.03 $\pm$ 0.07	**0.73** $\pm$ 0.17	**2.21** $\pm$ 0.19	3.08 $\pm$ 0.13	3.54 $\pm$ 0.09
3	-0.13 $\pm$ 0.02	0.04 $\pm$ 0.07	**0.81** $\pm$ 0.17	**2.29** $\pm$ 0.18	3.07 $\pm$ 0.12	3.53 $\pm$ 0.08
4	-0.12 $\pm$ 0.03	0.06 $\pm$ 0.07	**0.87** $\pm$ 0.16	**2.32** $\pm$ 0.17	3.06 $\pm$ 0.13	3.53 $\pm$ 0.08
5	-0.12 $\pm$ 0.03	0.07 $\pm$ 0.07	**0.90** $\pm$ 0.15	**2.34** $\pm$ 0.16	3.06 $\pm$ 0.13	3.53 $\pm$ 0.09
6	-0.11 $\pm$ 0.03	**0.09** $\pm$ 0.07	**0.92** $\pm$ 0.16	**2.36** $\pm$ 0.15	3.06 $\pm$ 0.13	3.52 $\pm$ 0.09
7	**-0.09** $\pm$ 0.04	**0.08** $\pm$ 0.08	**0.95** $\pm$ 0.16	**2.37** $\pm$ 0.15	3.05 $\pm$ 0.13	3.51 $\pm$ 0.08
8	-0.09 $\pm$ 0.04	0.08 $\pm$ 0.08	**0.95** $\pm$ 0.16	**2.37** $\pm$ 0.15	3.04 $\pm$ 0.13	3.51 $\pm$ 0.08

Table A.2 Log likelihood values for varying context sizes using AR procedures and vanilla and naive models. AR procedure log-likelihoods are shown in bold when they are significantly different than the vanilla log-likelihood ($p < 0.05$). Column headers are context sizes. Row names are methods. Errors represent 95% confidence interval obtained through bootstrapping.

Table A.3 Synthetic Audio Experiment Log-likelihoods.

	1	2	4	8	16
Naive	-0.46 ± 0.06	-0.46 ± 0.05	-0.45 ± 0.04	-0.44 ± 0.04	-0.40 ± 0.03
Vanilla	-0.45 ± 0.18	-0.24 ± 0.12	-0.12 ± 0.12	0.36 ± 0.15	1.07 ± 0.18
1	**-0.12** ± 0.08	**-0.05** ± 0.10	0.03 ± 0.09	0.52 ± 0.12	1.11 ± 0.20
2	**-0.03** ± 0.09	**0.05** ± 0.11	**0.11** ± 0.10	**0.58** ± 0.11	1.16 ± 0.15
3	**0.04** ± 0.09	**0.10** ± 0.12	**0.16** ± 0.10	**0.63** ± 0.11	1.19 ± 0.14
4	**0.08** ± 0.10	**0.15** ± 0.11	**0.20** ± 0.10	**0.67** ± 0.11	1.21 ± 0.13
5	**0.13** ± 0.10	**0.17** ± 0.12	**0.24** ± 0.10	**0.68** ± 0.10	1.23 ± 0.14
6	**0.15** ± 0.10	**0.19** ± 0.13	**0.26** ± 0.10	**0.71** ± 0.10	1.24 ± 0.13
7	**0.16** ± 0.10	**0.21** ± 0.13	**0.29** ± 0.11	**0.72** ± 0.10	1.26 ± 0.12
8	**0.19** ± 0.11	**0.24** ± 0.12	**0.30** ± 0.12	**0.74** ± 0.10	1.27 ± 0.13

Table A.4 Log likelihood values for varying context sizes using AR procedures and vanilla and naive models. AR procedure log-likelihoods are shown in bold when they are significantly different than the vanilla log-likelihood ($p < 0.05$). Column headers are context sizes. Row names are methods. Errors represent 95% confidence interval obtained through bootstrapping.

Table A.5 Real Audio Experiment Log-likelihoods.

	1	2	4	8	16
Naive	1.01 ± 0.16	0.77 ± 0.22	0.90 ± 0.30	1.02 ± 0.20	0.88 ± 0.20
Vanilla	1.11 ± 0.18	1.16 ± 0.22	1.70 ± 0.29	2.29 ± 0.26	2.43 ± 0.19
1	1.13 ± 0.15	1.15 ± 0.22	1.72 ± 0.28	2.29 ± 0.27	2.43 ± 0.19
2	1.13 ± 0.15	1.16 ± 0.22	1.72 ± 0.28	2.30 ± 0.27	2.44 ± 0.19
3	1.13 ± 0.15	1.16 ± 0.22	1.72 ± 0.28	2.30 ± 0.26	2.44 ± 0.19
4	1.14 ± 0.15	1.17 ± 0.22	1.73 ± 0.28	2.30 ± 0.27	2.44 ± 0.19
5	1.14 ± 0.15	1.17 ± 0.22	1.73 ± 0.29	2.30 ± 0.27	2.44 ± 0.19
6	1.14 ± 0.15	1.18 ± 0.22	1.74 ± 0.28	2.31 ± 0.27	2.45 ± 0.19
7	1.14 ± 0.14	1.17 ± 0.22	1.74 ± 0.28	2.30 ± 0.27	2.45 ± 0.19
8	1.14 ± 0.15	1.19 ± 0.22	1.74 ± 0.28	2.31 ± 0.27	2.45 ± 0.19

Table A.6 Log likelihood values for varying context sizes using AR procedures and vanilla and naive models. AR procedure log-likelihoods are shown in bold when they are significantly different than the vanilla log-likelihood ($p < 0.05$). Column headers are context sizes. Row names are methods. Errors represent 95% confidence interval obtained through bootstrapping.

Table A.7 Gaussian Process Experiment Log-likelihoods.

	1	2	4	8	16
Naive	-1.44 $\pm$ 0.08	-1.44 $\pm$ 0.12	-1.42 $\pm$ 0.13	-1.56 $\pm$ 0.17	-1.40 $\pm$ 0.10
Vanilla	-1.36 $\pm$ 0.08	-1.27 $\pm$ 0.09	-1.12 $\pm$ 0.08	-0.92 $\pm$ 0.14	-0.52 $\pm$ 0.06
1	-1.35 $\pm$ 0.08	-1.28 $\pm$ 0.10	-1.11 $\pm$ 0.08	-0.92 $\pm$ 0.14	-0.51 $\pm$ 0.05
2	-1.35 $\pm$ 0.08	-1.28 $\pm$ 0.10	-1.11 $\pm$ 0.08	-0.92 $\pm$ 0.15	-0.51 $\pm$ 0.05
3	-1.35 $\pm$ 0.08	-1.28 $\pm$ 0.11	-1.12 $\pm$ 0.08	-0.93 $\pm$ 0.15	-0.52 $\pm$ 0.05
4	-1.36 $\pm$ 0.08	-1.28 $\pm$ 0.10	-1.12 $\pm$ 0.09	-0.93 $\pm$ 0.15	-0.52 $\pm$ 0.05
5	-1.35 $\pm$ 0.08	-1.29 $\pm$ 0.11	-1.12 $\pm$ 0.09	-0.94 $\pm$ 0.16	-0.52 $\pm$ 0.05
6	-1.35 $\pm$ 0.08	-1.28 $\pm$ 0.10	-1.13 $\pm$ 0.09	-0.93 $\pm$ 0.14	-0.52 $\pm$ 0.05
7	-1.35 $\pm$ 0.08	-1.29 $\pm$ 0.10	-1.13 $\pm$ 0.10	-0.92 $\pm$ 0.14	-0.52 $\pm$ 0.05
8	-1.35 $\pm$ 0.08	-1.29 $\pm$ 0.10	-1.13 $\pm$ 0.09	-0.93 $\pm$ 0.15	-0.52 $\pm$ 0.05

Table A.8 Log likelihood values for varying context sizes using AR procedures and vanilla and naive models. AR procedure log-likelihoods are shown in bold when they are significantly different than the vanilla log-likelihood ($p < 0.05$). Column headers are context sizes. Row names are methods. Errors represent 95% confidence interval obtained through bootstrapping.