Enhancing VAE-learning on spatial priors using graph convolutional networks

Candidate Number: 1059912

Submitted in partial fulfillment for the degree of

Master of Science in Computer Science

Trinity Term 2022

Word Count: **13,303**

Obtained using `texcount main.tex -inc -sum -total`, TeXcount version 3.1.1.

Abstract

Bayesian inference on discrete spatial data is most popularly done using Gaussian processes (GPs) in a hierarchical Bayesian formulation. This is because GPs have the ability to capture both spatial autocorrelations and fit to a large family of functions. However, GPs present a huge computational challenge which makes performing Bayesian inference using them impractical in real-world problems.

PriorVAE [Semenova et al., 2022] presents a two-stage process to overcome this challenge when the spatial structure is fixed: train a variational autoencoder (VAE) on a large class of GP priors, and then the learnt decoder replaces the GP within a Bayesian hierarchical model, leading to highly efficient spatial inference. In previous work, the VAE that was used strongly disregarded the spatial structure in the input data; rather than retaining the neighbourhood structure of spatial data, the VAE used a flattening operator to make the input into a vector, thus significantly limiting learning capability.

In this dissertation, we extend PriorVAE by introducing a method to carefully augment a graph convolutional network in the VAE so that information aggregation can now be performed locally whilst at the same time spatial inference performance is maintained at high efficiency levels. We further introduce a *local-to-global* scheme that helps localized information to be aggregated across the whole space, since we realized that GPs are often defined by covariance kernels that inherently capture global information.

Our experiments show that our proposed method is able to successfully improve PriorVAE estimates in a series of Bayesian spatial inference tasks, achieving significantly lower mean squared error in all of them. Furthermore, our results demonstrate that the local-to-global scheme has a significant positive impact in learning spatial priors that are highly complex, evident from an increase in effective sample size and lower inference time as compared to the approach where the local-to-global scheme is not considered.

Contents

1 Introduction

1.1 Overview

Small area estimation [Rao, 2017] of discrete spatial data (also known as areal units) is important for policy decision-making in fields that include epidemiology and the environmental sciences. It is often the case that we require reliable areal unit level estimates despite having only unreasonably small sample sizes; possibly none. In epidemiology, for example, one might be interested in verifying whether non-pharmaceutical interventions can mitigate the spread of a disease within local areas in an ongoing pandemic. Wrongful verification of such interventions could lead to catastrophic consequences including damaging the global economy and unfavourably reducing the happiness index of a country. The main issue is that some areal units might not have samples readily available when we want to compute the relevant statistics, for example, due to a delay in reporting by hospitals or local authorities. As such, "borrowing strength" across neighbouring areal units is integral for making reliable estimates to ensure a well-informed policy decision-making which, for our particular example, is deciding whether to apply an intervention in a specific area.

A popular approach for modelling spatial data is the use of *Gaussian process* (GP) priors in a hierarchical Bayesian formulation. This should not come as a surprise since GP are non-parametric and hence by definition, are able to capture a broad family of functions. Furthermore, the natural setup when using GPs also allows for capturing model uncertainty which is key in statistical modelling. From the practitioner's point of view, GPs are especially favoured because they are relatively easy to implement. In fact, there are already several efficient implementations of GPs that can be used by researchers on the fly without them having to write everything from scratch (for example, see GPy [GPy, since 2012] and GPflow [Matthews et al., 2017]). On top of that, they are also preferred for their mathematical simplicity. They are not particularly discouraging for the average practitioner since, for example, if one is already familiar with the Gaussian distribution, then learning about GPs should just require an extra reading.

Despite their advantages, GPs do come with several computational challenges. Firstly, performing Bayesian inference using a GP prior would involve computing the inverse and determinant of a (covariance) matrix. For data collected over N areal units, these operations has a cost of $\mathcal{O}(N^3)$ which can be problematic as the number of areal units increases. For example, one might want to consider finer administrative units within a predefined domain or even a completely new domain with granular administrative regions. As such, the use of GP priors is impractical for large N and other approaches should be explored. Furthermore, Stephenson et al. [2022] has demonstrated that a poor GP kernel choice could lead to unsatisfactory learning of the desired underlying stochastic process. Unfortunately for most, it is typical to inherently choose a "wrong" kernel function due to limited domain knowledge or a poor modelling assumption. To combat this problem, the authors suggested that choosing the right inference algorithm becomes crucial when learning using a GP, evident from their empirical evaluations.

Naturally, *Markov chain Monte Carlo* (MCMC) is thus the preferred choice when choosing an algorithm for performing Bayesian inference using GPs. The ergodic theorem guarantees that samples we get from MCMC are asymptotically exact. That is, if the Markov chain is simulated for a sufficiently long period of time, then the samples we obtained should approximate the desired posterior distribution really well. This key feature of MCMC makes it reliable for performing spatial inference using GPs as it helps mitigate the issue highlighted in [Stephenson et al., 2022]. Unfortunately, MCMC comes with its own set of limitations. For example, MCMC does not scale well with increasing dimensionality of the target parameter. Furthermore, the simulated Markov chain are typically highly autocorrelated, which is especially true when we perform inference in a spatial model. As such, expensive techniques such as burn-in and thinning have to be performed which could end up making inference intractable.

An alternative method to MCMC is *variational inference* [Hoffman et al., 2013] which is faster in general and is scalable to not only more complex models but also to larger size datasets. Unlike the sampling-based MCMC, variational inference is optimization-based and aims to approximate the target posterior using a variational family of densities that is chosen *a priori* by the practitioner. *Expectation propagation* [Minka, 2013] is another alternative to MCMC that does an approximation scheme similar to variational inference. Another popular alternative method that are specifically optimized for inference using latent Gaussian models (and hence, GPs) is the *integrated nested Laplace approximation* (INLA; [Rue et al., 2009]).

This approximation scheme aims at approximating the marginal distribution (of the model parameters) using a variety of Gaussian approximation, which makes inference computationally cheap. Unfortunately, despite being efficient, there are no guarantees that the aforementioned approximation schemes simulates the target posterior exactly like MCMC does. In fact, the posterior estimates obtained from the approximation may not even be accurate evident from a variational inference review by [Yao et al., 2018].

Recall that these approximation schemes are considered because performing MCMC for sampling suffers from intractability when the target posterior is high-dimensional and due to Markov chain samples being highly autocorrelated. But what if we can significantly reduce the dimension of the target parameter such that the remaining dimensions are uncorrelated and still meaningfully capture the underlying spatial process? Then MCMC becomes practical for spatial inference! Remarkably, Semenova et al. [2022] suggested that we **can** do this by using the *variational autoencoder* (VAE) architecture from deep learning and combining it into a Bayesian spatial inference framework via a simple two-step procedure, which they dubbed the PriorVAE. Their idea is to train a variational autoencoder on a wide class of GP priors and then use the learnt decoder at inference and/or prediction time. Since the dimensions are greatly reduced and are uncorrelated, inference time is significantly reduced. Furthermore, because GPs are not used directly when performing inference, we also avoid the need for $\mathcal{O}(N^3)$ operations due to matrix inversions and determinant computations.

Despite being able to successfully replace GPs in spatial inference, we believe that PriorVAE can still benefit from further improvements especially at the architecture level. Upon inspection of the architecture, we realized that input data is propagated through the VAE without any regard for the local neighbourhood structure of each areal units. In particular, the input data is flattened before being propagated through an *multi-layer perceptron* (MLP). Without regularization techniques such as dropout, this means that the VAE views each areal unit as being adjacent to one another which can limit its potential in learning highly complex spatial priors. Thus, our goal is to explore an approach that overcomes this limitation whilst simultaneously maintain the highly efficient inference performance of PriorVAE.

1.2 Contributions

In this dissertation, we present a novel generative modelling approach for learning spatial priors to enable small area estimation that solve some of the issues highlighted in PriorVAE whilst maintaining the power and advantages discussed previously. In particular, our proposed approach actually utilizes the local neighbourhood spatial structure without flattening the input data. We are able to do this by augmenting a *graph convolutional network* (GCN) layer into the VAE, forming a variational graph autoencoder architecture, hence we call it **PriorVGAE**. Utilizing GCN layers also allows us to favourably reduce the number of learnable parameters, enabling computationally cheap training even for highly complex problems. Thus, despite also having a fixed spatial structure, out-of-sample predictions now becomes tractable as we can restart VAE training when required.

The key contributions of our work are summarized as follows:

- We extend the first of the two-stage PriorVAE process by introducing a GCN layer in the VAE which allows learning (of spatial priors) that actually exploits the local neighbourhood spatial structure.
- We introduce a so-called *local-to-global* scheme that helps *variational graph autoencoder* (VGAE) learn spatial priors more effectively as compared to naively stacking graph convolutional layers.
- We show that VGAE priors are useful for small area estimation by performing Bayesian inference on simulated data where the ground truth is known.
- We demonstrate that priors learnt using this novel graph convolutional network scheme leads to estimates that are far superior as compared to using PriorVAE priors. In particular, we show that inference using the proposed VGAE results in favourably reduced mean squared error as compared to the PriorVAE, especially as the complexity of the spatial structure increases.

1.3 Outline of Dissertation

This dissertation is organized as follows.

- In Chapter 2, we present a brief overview of Bayesian inference and the deep learning concepts relevant to this work. We also give a review of the problem of interest followed by a thorough discussion of the existing PriorVAE approach.

- Chapter 3 presents PriorVGAE, our proposed novel approach for doing inference in a spatial setting. It first argues that a certain local-to-global information passing scheme is necessary if we want to learn the desired spatial priors effectively. Then it details a key step that helps speed up inference when using a VGAE. This is then followed by a theoretical formulation that underlies our VGAE architecture. Finally, it gives a detailed description of how the model is implemented in practice: from input data to its latent representation and further to its reconstruction.

- Chapter 4 then details our experimental findings. Here, we give the exact implementation details used in each experiment (e.g., the number of hidden layers used) and we present key visualizations and metrics that demonstrate the performance of our model.

- Finally, we summarize and discuss our findings in Chapter 5, and suggest potential future work avenues.

2 Preliminary Material

2.1 Bayesian Inference

Suppose we are interested in learning the probability distribution of a latent variable $\mathbf{z}$ given some observed data $\mathbf{x}$. Formally, we first have to assume a model $p(\mathbf{x}, \mathbf{z})$ on the joint distribution of $\mathbf{z}$ and $\mathbf{x}$. By the chain rule of densities, this model can be decomposed into

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}),$$

where $p(\mathbf{x}|\mathbf{z})$ is called the *data likelihood* or just *likelihood* of the data $\mathbf{x}$ given $\mathbf{z}$, and $p(\mathbf{z})$ is called the *prior distribution* over $\mathbf{z}$ which captures our prior knowledge regarding $\mathbf{z}$ before observing any data. Because of this decomposition, we know that assuming a prior distribution $p(\mathbf{z})$ over $\mathbf{z}$ and a likelihood function $p(\mathbf{x}|\mathbf{z})$ is enough to define the joint distribution $p(\mathbf{x}, \mathbf{z})$ in consideration. Once we have a complete model for $p(\mathbf{x}, \mathbf{z})$, we can then obtain the *posterior distribution* of $\mathbf{z}$ given $\mathbf{x}$ using Bayes' theorem

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}, \tag{2.1}$$

where $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$ is called the *normalising constant* which, as its name suggests, normalizes the posterior $p(\mathbf{z}|\mathbf{x})$. While the numerator in (2.1) is clearly tractable (because we specified the model), the normalising constant in the denominator typically does not have an analytical solution and is often intractable especially when the dimensionality of $\mathbf{z}$ increases. This becomes an issue as lacking the normalising constant would lead to inaccurate evaluations of the posterior $p(\mathbf{z}|\mathbf{x})$. However, since the normalising constant is independent of $\mathbf{z}$, it is often sufficient to only know the *unnormalised posterior density*, i.e., only knowing the posterior density up to a constant

$$p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z})p(\mathbf{x}|\mathbf{z}).$$

This is because the posterior density can be numerically estimated using sampling methods such as Markov chain Monte Carlo, which we shall discuss next.

2.1.1 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a general sampling technique to sample from a possibly unnormalised probability density $\pi(\mathbf{z})$, often called the *target density* [Gelman et al., 2013]. In our current context, this target density is the posterior $p(\mathbf{z}|\mathbf{x})$. The main idea behind MCMC is to simulate an ergodic Markov chain $\left\{\mathbf{z}^{(i)}\right\}_{i\in\mathbb{N}}$ that admits $\pi(\mathbf{z})$ as its unique limiting distribution. Of course in practice we cannot simulate the chain indefinitely, but the idea is to run the simulation long enough to get a simulated chain $\left\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N)}\right\}$, where $N \in \mathbb{N}$ is a sufficiently large number, that comes from some distribution that approximates $\pi(\mathbf{z})$.

MCMC is usually favoured over simpler inference algorithms such as *simple Monte Carlo* and importance sampling, especially when the support of the target density $\pi(\mathbf{z})$ is high-dimensional. This is because in higher dimensions, most of the probability mass does not accumulate around the mode of the density, but rather within a vanishingly thin region known as the *typical set* [Betancourt, 2017]. Consequently, inference algorithms such as importance sampling or *rejection sampling* becomes very inefficient as the former would yield a lot of samples with very small weights while the latter would have bizarrely high rejection rates.

The Metropolis-Hastings algorithm [Hastings, 1970; Metropolis et al., 1953] forms the foundation for most of MCMC algorithms used today. In this algorithm, we first choose a proposal density (also known as a transition kernel) that can be chosen *a priori* arbitrarily. The purpose of this density, as its name suggests, is to propose a new value in the latent space given some current value via a finite number of transitions. The proposed value will then go through an accept/reject procedure with probability given by the so-called *Hastings' ratio*, and this step acts as a correction mechanism that ensures the simulated Markov chain indeed approaches the desired limiting distribution $p(\mathbf{z}|\mathbf{x})$.

While MCMC has proven to be very successful as compared to simpler inference algorithms that we have discussed, it does has its limitations. For example, the simulated samples drawn using MCMC are often highly correlated and not truly i.i.d as we desired. Therefore, computationally expensive techniques such as burn-in (discarding often large number of initial samples) and thinning (keep only every Tth sample after burn-in) have to be employed to mitigate these problems. Moreover, sometimes there are just large correlations between dimensions which consequently makes chain mixing slow. This is especially true in spatially referenced data which we are interested in.

2.2 Variational Autoencoders

Suppose we have a dataset $\mathcal{D} = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}$ of $N > 1$ datapoints and assume that $\mathbf{x}^{(i)}$ is sampled from an unknown underlying distribution $p^*(\mathbf{x})$ over the support $\mathbb{R}^D$. A common criterion for learning this underlying distribution is to choose a parametric model $p_{\boldsymbol{\theta}}(\mathbf{x})$, representing the likelihood of $\mathbf{x}$ given $\boldsymbol{\theta}$, and maximizing this likelihood (or more commonly the *log-likelihood* $\log p_{\boldsymbol{\theta}}(\mathbf{x})$) with respect to the parameter $\boldsymbol{\theta}$. A known technique to add more expressivity is to further introduce latent variables $\mathbf{z} \in \mathbb{R}^M$ and consider the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ over $\mathbb{R}^D \times \mathbb{R}^M$ instead. In this setup, one can recover the likelihood by marginalizing over $\mathbf{z}$:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int_{\mathbb{R}^M} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z} = \int_{\mathbb{R}^M} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z})\, d\mathbf{z}. \tag{2.2}$$

This distribution is often called the *marginal likelihood* (due to marginalizing).

Modern day generative modelling has shown that using neural networks to approximate distributions in $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ allows us to successfuly model complex data that are often high-dimensional in nature. This approach, known as *deep latent variable models*, does not however come for free as one could expect. Using neural networks together with high-dimensional continuous latent variables causes the integral in (2.2) to be intractable since we have to integrate out $\mathbf{z}$. Consequently, we cannot simply learn via maximizing the log-likelihood without doing any modifications. The trick, however, is to use Bayes' theorem and realize the connection between the likelihood and the posterior of the latent variable $\mathbf{z}$ given $\mathbf{x}$:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})}; \text{ or equivalently } p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}. \tag{2.3}$$

Now $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ is tractable, but the likelihood and the posterior are not in the setting of deep latent variable models. However, observe that because of Bayes' theorem, if we can somehow make one of them tractable, then we can compute the other one; and this is where variational autoencoders come in.

The *variational autoencoder* (VAE) [Kingma and Welling, 2014; Rezende et al., 2014] provides a scalable framework to efficiently learn latent variable models $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ via variational inference. In the VAE setup, we again use a neural network for the *generative model* $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ as in deep latent variable models; but now, we introduce an *inference model* $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ parameterized over $\boldsymbol{\phi}$ to approximate the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. Typically, this parametrization is done by using a neural network. For example, in a

vanilla VAE we often approximate the posterior using a factorized Gaussian in the following way:

$$(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \log \boldsymbol{\sigma}_{\boldsymbol{\phi}}) = \text{InferenceNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x}), \tag{2.4}$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}_{\boldsymbol{\phi}}), \tag{2.5}$$

where InferenceNeuralNet is the inference model, $\mathcal{N}(\cdot)$ denotes the normal (or Gaussian) distribution, and the neural network output $\boldsymbol{\mu}_{\boldsymbol{\phi}}$ and $\boldsymbol{\sigma}_{\boldsymbol{\phi}}$ are the mean and standard deviation of the factorized Gaussian. The generative and inference models are sometimes called the *decoder* and *encoder* respectively because put together, the whole VAE has a similar encoder-decoder architecture like an autoencoder [Goodfellow et al., 2016]. In fact, we will always refer to them as the decoder and the encoder from now on to avoid confusion with *spatial inference*, the task that we are ultimately interested in. Thanks to the relation (2.3), we shall see that optimizing $\boldsymbol{\phi}$ in fact leads to maximizing the marginal log-likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$ which has been our main goal from the very beginning.

2.2.1 Evidence Lower Bound

Because $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is just an approximation to the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, we do not maximize the (exact) marginal log-likelihood over $\boldsymbol{\theta}$ but instead maximize a lower bound on this log-likelihood, known as the *evidence lower bound* (ELBO), over both $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. This lower bound arises naturally due to Jensen's inequality as follows

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log \int_{\mathbb{R}^M} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z} \tag{2.6}$$

$$= \log \int_{\mathbb{R}^M} \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \tag{2.7}$$

$$= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right] \tag{2.8}$$

$$\geqslant \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right) \right]}_{\equiv \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x})} \tag{2.9}$$

where $\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x})$ is our notation for the ELBO. There is also an alternative derivation of the ELBO that is less natural but provides a better insight regarding the gap

between the approximate posterior and the true posterior:

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{2.10}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.11}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.12}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right) + \log\left(\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.13}$$

$$= \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right)\right]}_{\equiv \mathsf{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))} \tag{2.14}$$

where $\mathsf{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))$ is the *Kullback-Leibler* (KL) *divergence* (or simply KL-divergence), a measure of dissimilarity between the distributions $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ borrowed from information theory. A key property of the KL-divergence is that it is non-negative. This property establishes a lower bound on $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ as follows

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathsf{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})) \leqslant \log p_{\boldsymbol{\theta}}(\mathbf{x}). \tag{2.15}$$

Furthermore, the KL-divergence is identically zero only when the posterior matches, i.e., $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ in which case $\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x})$. Consequently, from equation (2.15) we can deduce that maximizing the ELBO not only improves the approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ of the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ but also maximizes the marginal log-likelihood $p_{\boldsymbol{\theta}}(\mathbf{x})$; where both of which are our core objectives when learning the latent variable model $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$ using a VAE. In practice, we do not maximize the exact formula as in (2.15) but instead the following alternative formulation:

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.16}$$

$$= \int_{\mathbb{R}^M} \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \tag{2.17}$$

$$= \int_{\mathbb{R}^M} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\, d\mathbf{z} + \int_{\mathbb{R}^M} \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \tag{2.18}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] + \mathsf{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z})), \tag{2.19}$$

where the first term $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]$ in equation (2.19) is called the *reconstruction error* and the second term is the KL-divergence between the approximate posterior and a prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ over the latent space. This KL-divergence term can be viewed as a regularizer, ensuring that the approximate posterior is as close to $p_{\boldsymbol{\theta}}(\mathbf{z})$. Since the factorized Gaussian is often used as the approximate posterior, the latent space prior is typically chosen to be a unit Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$ although more complex priors have been considered (e.g., see [Tomczak and Welling, 2018]). The reason for considering the formulation as given in (2.19) is because it involves distributions that we know and can sample from using Monte Carlo methods.

2.2.2 Reparametrization Trick

A key advantage of using the ELBO is that it allows us to use stochastic gradient descent with backpropagation to simultaneously optimize the encoder parameters $\boldsymbol{\phi}$ and model parameters $\boldsymbol{\theta}$. Optimizing the model parameter is quite straightforward:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right) \right] \tag{2.20}$$

$$= \nabla_{\boldsymbol{\theta}} \int_{\mathbb{R}^M} \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right) q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \, d\mathbf{z} \tag{2.21}$$

$$= \int_{\mathbb{R}^M} q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right) d\mathbf{z} \tag{2.22}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} \log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \right) \right] \tag{2.23}$$

$$= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \right] \tag{2.24}$$

where we have used the Leibniz integral rule to go from equation (2.21) to (2.22) and the fact that $\nabla_{\boldsymbol{\theta}} \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = 0$ to obtain the last equality in (2.24). We can then use a simple Monte Carlo estimator

$$\frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_i), \;\; \mathbf{z}_i \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$$

to get an unbiased estimate of the gradient. Efficient optimization of the encoder parameters, on the other hand, is only possible via the *reparametrization trick*: we express latent samples $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ as a deterministic bijective transformation $f_{\boldsymbol{\phi}}$ as

follows

$$\mathbf{z} = \mathbf{z}(\boldsymbol{\epsilon}; \boldsymbol{\phi}) = f_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}, \mathbf{x}),$$

where $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ is a random variable independent of $\mathbf{x}$ and $\boldsymbol{\phi}$. As an example, the reparametrization trick for the vanilla VAE is applied as follows:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}, \tag{2.25}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of the Gaussian obtained as output from the encoder as in equation (2.4), $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\odot$ denotes element-wise product. Using this reparametrization, we can then write the ELBO as an expectation with respect to $p(\boldsymbol{\epsilon})$ instead of $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$,

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right].$$

This form allows us to use the Leibniz integral rule to push the gradient operator inside the expectation:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}}\, \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) &= \nabla_{\boldsymbol{\phi}}\, \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right] && (2.26)\\
&= \nabla_{\boldsymbol{\phi}} \int \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right) p(\boldsymbol{\epsilon})\, d\boldsymbol{\epsilon} && (2.27)\\
&= \int \nabla_{\boldsymbol{\phi}} \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right) p(\boldsymbol{\epsilon})\, d\boldsymbol{\epsilon} && (2.28)\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\nabla_{\boldsymbol{\phi}} \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right)\right] && (2.29)
\end{aligned}$$

from which we can then use a simple Monte Carlo estimator to obtain an unbiased estimate of the gradient $\nabla_{\boldsymbol{\phi}}\, \mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x})$:

$$\frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\phi}} \log\left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\right), \text{ with } \mathbf{z} = \mathbf{z}(\boldsymbol{\epsilon}_i; \boldsymbol{\phi}), \boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon}).$$

2.3 Graph Neural Networks

Notation. We write $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for any graph $\mathcal{G}$ with nodes $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and put $N = |\mathcal{V}|$. We denote $\mathbf{A} \in \mathbb{R}^{N \times N}$ for the adjacency matrix associated to $\mathcal{G}$, and write $\mathbf{X} \in \mathbb{R}^{N \times D}$ for the node feature matrix.

2.3.1 Graph Convolutional Networks

A *graph convolutional network* (GCN) [Kipf and Welling, 2017] is a neural network model for learning on graph-structured data. It can be seen as a generalization of the convolutional neural network [Lecun et al., 1998] to arbitrary graphs which can have very irregular structures. For an undirected graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, a single GCN layer is defined by the propagation rule

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{2.30}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops, $\mathbf{I}$ is the identity matrix with same dimensionality as $\mathbf{A}$, and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the diagonal node degree matrix of $\tilde{\mathbf{A}}$. ℓ is the index of the current neural network layer and $\mathbf{W}^{(\ell)}$ is a learnable weight matrix in the ℓ-th layer. $\mathbf{H}^{(\ell)}$ is the node hidden matrix in the ℓ-th layer where we put $\mathbf{H}^{(0)} = \mathbf{X}$, the input node feature matrix. $\sigma(\cdot)$ is a non-linear function such as the *rectified linear unit* (ReLU) [Nair and Hinton, 2010] defined as $\mathsf{ReLU}(x) = \max(x, 0)$ or the *exponential linear unit* (ELU) [Clevert et al., 2016] given by

$$\mathsf{ELU}(x) = \begin{cases} x, & \text{if } x > 0, \\ \exp(x) - 1, & \text{if } x \leqslant 0, \end{cases}$$

both of which are applied element-wise. We shall write $\mathsf{GCN}(\mathbf{H}, \mathbf{A})$ for a single application of the GCN layer to a graph with adjacency matrix $\mathbf{A}$ and node hidden matrix $\mathbf{H}$, indexing it as $\mathsf{GCN}^{(\ell)}$ for the ℓ-th layer if the emphasis is required.

Collectively, the product

$$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} \tag{2.31}$$

is a symmetric normalized version (with added self-loops) of the adjacency matrix $\mathbf{A}$. The self-loops are added (via addition of the identity matrix to $\mathbf{A}$) to allow the feature vector of the current node to be considered as well when aggregating over neighbouring nodes. This is a feature that we typically want although possibly not for every layer. The symmetric normalization, sometimes called the *renormalization trick*, is performed instead of the naive layer $\mathbf{H}^{(\ell+1)} = \sigma\left(\mathbf{A}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right)$ to ensure that repeated application of the GCN does not scale the feature matrix too much, leading to vanishing gradients and numerical instabilities. Note that for directed graphs, we use the non-symmetric normalization $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ instead [Schlichtkrull et al., 2018] which makes $\hat{\mathbf{A}}\mathbf{H}^{(\ell)}$ simply a node feature neighbourhood average computation.

Observe that for each layer ℓ, the weight matrix $\mathbf{W}^{(\ell)}$ are shared by all the nodes

in the graph. This is an important feature of the GCN as it greatly reduces the number of parameters to be learnt, yielding a scalable and efficient way of learning on large graphs. It is also the reason why we call such networks "convolutional".

2.3.2 Variational Graph Autoencoders

The *variational graph autoencoder* (VGAE) [Kipf and Welling, 2016] is a latent variable model for unsupervised learning on data that admits a graph (or graph-like) structure. It is essentially an extension of the VAE to graphs where instead of wanting to learn the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$ of random variables on $\mathbb{R}^D \times \mathbb{R}^M$, we are interested in learning a joint distribution on graph representations; although we do have more flexibility as compared to VAEs. For example, in link prediction tasks (see author's example below), we might want to learn the model $p_{\boldsymbol{\theta}}(\mathbf{A}, \mathbf{Z}|\mathbf{X})$ on the space $\mathbb{R}^{N\times N} \times \mathbb{R}^{N\times M}$ since we are more concerned with the reconstruction of the adjacency matrix $\mathbf{A}$. On the other hand, we would consider the model $p_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{Z}|\mathbf{A})$ on $\mathbb{R}^{N\times D} \times \mathbb{R}^{N\times M}$ for node regression or classification tasks instead since we might be interested in the reconstruction of the feature matrix $\mathbf{X}$. In both cases, we again are free to choose what kind of parametrization we want for the encoder and decoder depending on the task at hand, just like the VAE.

The original paper by Kipf and Welling [2016] was interested in a link prediction task, i.e., learning a joint distribution on $(\mathbf{A}, \mathbf{Z})$. The generative model was chosen to be a simple, non-parametric *inner product decoder*

$$p(\mathbf{A}|\mathbf{Z}, \mathbf{X}) = p(\mathbf{A}|\mathbf{Z}) = \prod_{i=1}^{N}\prod_{j=1}^{N} p_{\boldsymbol{\theta}}(A_{ij} = 1|\mathbf{z}_i\mathbf{z}_j); \text{ with } p(A_{ij} = 1|\mathbf{z}_i\mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j),$$

where $\mathbf{z}_i$ is the latent representation of node $v_i \in \mathcal{V}$ and corresponds to the i-th row in the $\mathbf{Z}$ matrix, A_{ij} is the (i, j) entry of the adjacency matrix $\mathbf{A}$ and σ is the sigmoid function $\sigma(z) = e^z/(e^z + 1)$. The encoder $q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X}, \mathbf{A})$, on the other hand, was chosen to be parameterized by a two-layer GCN which assumes a factorized Gaussian as the approximate posterior:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \mathsf{GCN}_{\boldsymbol{\phi}}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathsf{ReLU}(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)}, \tag{2.32}$$

$$q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i|\mathbf{X}, \mathbf{A}); \text{ with } q(\mathbf{z}_i|\mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i), \tag{2.33}$$

where $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is obtained using the *renormalization trick*, ReLU is the

activation function and $\boldsymbol{\phi} = (\mathbf{W}^{(0)}, \mathbf{W}^{(1)})$ are trainable weight parameters for each of the GCN layer. Note that $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{N \times M}$ here are matrices and not vectors. As in VAE, the optimization objective of the VGAE is to maximize the ELBO as in equation (2.19):

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{A}|\mathbf{Z})\right] - \mathsf{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{Z}|\mathbf{X},\mathbf{A}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{Z})\right),$$

where the prior $p(\mathbf{Z}) = \prod_i p(\mathbf{z}_i)$ is chosen to be a Gaussian $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i; \mathbf{0}, \mathbf{I})$ to ensure parsimony in the latent space.

2.3.3 Symmetric Graph Convolutional Autoencoders

The VGAE architecture proposed in [Kipf and Welling, 2016] was designed specifically to learn a joint distribution $p(\mathbf{A}, \mathbf{Z})$ instead of a joint distribution $p(\mathbf{X}, \mathbf{Z})$ since they were interested in reconstructing the adjacency matrix $\mathbf{A}$ instead of the feature matrix $\mathbf{X}$. The inner product decoder used is not learnable and so the node features are not even propagated through the decoder. As a consequence, the learning capability for the VGAE to reconstruct adjacency matrices can be limited. As such, we cannot just naively take the proposed VGAE architecture if we are interested in node regression or classification problems which involves reconstruction of the feature matrix $\mathbf{X}$.

The straightforward architecture for node feature reconstruction is to build a decoder that mirrors the encoder. For example, if the encoder is parameterized by a three-layer GCN with 6-, 4- and 2-dimensional hidden sizes, then we can use a three-layer GCN with 2-, 4- and 6-dimensional hidden sizes for the decoder. However, this approach, while sensible, has its setbacks that can be seen if we go back to the mathematical foundations of a GCN.

Li et al. [2018] has shown in their seminal *oversmoothing* paper that GCN is inherently a special kind of *Laplacian smoothing* [Taubin, 1995]. Hence, repeatedly applying GCN layers can be detrimental, not only because it now takes more computational power to train, but also because node features become indistinguishable after several layers of aggregation. For node classification tasks, a suitable amount of Laplacian smoothing would help make nodes in the same class have highly similar features which is good (and explains why [Kipf and Welling, 2017] had excellent performance in their node classification task). However, for autoencoders, applying more GCN layers at the decoding stage may result in a node feature matrix reconstruction $\hat{\mathbf{X}}$ that is oversmoothed, especially for highly complex datasets that require

VGAEs with larger capacity.

To mitigate the issue of oversmoothing for autoencoder reconstruction, Park et al. [2019] suggests a counterpart graph neural network layer that does *Laplacian sharpening* as opposed to Laplacian smoothing. They call this approach *Graph convolutional Autoencoder using LAplacian smoothing and sharpening* (GALA). Instead of making node embeddings closer, Laplacian sharpening makes node embeddings move away from its neighbourhood centroid. In the context of an autoencoder, reconstruction is therefore accelerated as node embeddings are "encouraged" to move away from its latent state.

For any undirected graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, a single Laplacian sharpening GCN layer is given by the propagation rule

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right) \tag{2.34}$$

where $\bar{\mathbf{A}} = 2\mathbf{I} - \mathbf{A}$ is the negated adjacency matrix with doubly added self-loops, $\bar{\mathbf{D}}_{ii} = \sum_j \bar{\mathbf{A}}_{ij}$ is the diagonal node degree matrix of $\bar{\mathbf{A}}$ and everything else follows the same notation as in (2.30). Similar to what we have discussed regarding GCNs, the renormalization

$$\hat{\mathbf{A}}_{\text{sharpen}} = \bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-\frac{1}{2}} \tag{2.35}$$

is used to ensure that repeated application of the Laplacian sharpening GCN is numerically stable and does not yield exploding/vanishing gradients. Since we now have two different GCN models, we shall denote the Laplacian sharpening GCN as SharpenGCN and change the notation of Laplacian smoothing GCN from GCN to SmoothGCN instead.

2.4 PriorVAE Preliminaries

2.4.1 Latent Gaussian models

Suppose we collect outcome data $\{y_i\}_{i=1}^N$ over some *domain* of interest $\mathcal{B} = \bigcup_{i=1}^N B_i$. Here, $\{B_i\}_{i=1}^N$ defines a partition of the domain that we can choose depending on the problem we want to solve. For example, in a real-world setting, the partition could be defined by administrative borders (such as counties within the UK, states within the US, or countries within planet Earth). The dataset $\{y_i\}$ that we are interested in are typically based in epidemiology and so can vary from disease prevalence data (continuous) to number of infected cases in an area (discrete). As such, Bayesian

generalized linear models are favoured to be used for modelling since it captures all these kinds of data under a single framework:

$$\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta}), \tag{2.36}$$

$$\boldsymbol{f}|\boldsymbol{\theta} \sim \mathsf{GP}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}}), \tag{2.37}$$

$$\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{f}, \tag{2.38}$$

$$\mathbf{y}|\boldsymbol{\eta} \sim \pi(u^{-1}(\boldsymbol{\eta}), \boldsymbol{\theta}). \tag{2.39}$$

Here $\boldsymbol{\theta}$ is a hyperparameter, $\boldsymbol{f}$ is a Gaussian random field defined by mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$. $\boldsymbol{\beta}$ is the fixed effects, $\mathbf{X}$ is the (fixed-effects) design matrix and $\boldsymbol{\eta}$ is a linear predictor combining the fixed effects and random effects. Finally, u is a link function which characterizes the mean of the distribution.

We shall consider a GP prior over $\boldsymbol{f}$, which subsequently implies that any finite realizations $\boldsymbol{f}_{\mathsf{GP}} = (f_1, \ldots, f_t)$ are jointly normally distributed with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Furthermore, without loss of generality, we may assume that $\boldsymbol{\mu} = \mathbf{0}$ for any finite realizations $\boldsymbol{f}_{\mathsf{GP}}$ since it is an additive term in the linear predictor $\boldsymbol{\eta}$. The covariance matrix $\boldsymbol{\Sigma}$, on the other hand, is a choice we can make depending on the spatial structure. Typically, we use a kernel function such as an RBF (radial basis function) to define the covariance matrix. We can then learn from the observed data by computing the linear predictor,

$$u\left(\mathbb{E}[\mathbf{y}|\boldsymbol{f}_{\mathsf{GP}}]\right) = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{f}_{\mathsf{GP}}$$

followed by a computation of the likelihood. Assuming that the random effect is non-trivial (in which case the covariance matrix $\boldsymbol{\Sigma}$ is not the identity matrix), computing $\boldsymbol{f}_{\mathsf{GP}}$ can be challenging as it involves performing matrix inversion and computing the determinant covariance matrix Σ. These are $\mathcal{O}(N^3)$ operations which hinders the scalability of the GP approach.

2.4.2 PriorVAE: encoding spatial priors with VAE

PriorVAE [Semenova et al., 2022] is a generative modelling technique proposed to mitigate the challenge that arose when computing $\boldsymbol{f}_{\mathsf{GP}}$. Instead of dealing with cubic matrix inversions at inference time, the authors suggest the use of variational autoencoders to replace the $\boldsymbol{f}_{\mathsf{GP}}$ realizations in the linear predictor with another prior. To obtain this prior is a two-step process. First, a VAE is trained on a broad

class of GP priors where a "broad class" here means, for example, using varying lengthscales and variances in an RBF kernel. Once the VAE has been trained, we sample from the VAE latent prior and propagate this sample through the learnt decoder, yielding priors $\boldsymbol{f}_{\mathsf{VAE}}$ which can be used for spatial inference.

Observe that this method require only training the VAE once for any fixed domain $\mathcal{B}$. As such, an expert with a huge computational resource can train the VAE and share the learnt decoder parameters with other researchers, for example, via GitHub. Furthermore, because we use GP priors as training data instead of observed data collected over the domain $\mathcal{B}$, we can train the VAE *a priori* on an infinite amount of data until the VAE parameters are truly optimal (with respect to our chosen architecture). In fact, since we can always generate new training GP data, we do not need to save any training dataset locally, but only the very few parameters that determines the class of GP priors we want to generate. Moreover, there are also no issues with data quality, since we can simulate exact GP priors that are noise-free which is not true when we use real world observations.

This novel approach for doing spatial inference does, however, come with limitations. Before obtaining a learnt decoder for performing inference, the VAE has to be trained which can be computationally extensive, although as we have argued this training step is only required once. Furthermore, looking deeper into the architecture, we observe that PriorVAE does not truly take the spatial structure into account as the spatial observations $\{y_i\}_{i=1}^N$, where each y_i corresponds to an area B_i in a partition of $\mathcal{B}$, are flattened and passed into an MLP in the encoding phase, and also during the decoding phase for reconstruction. As such, without any learning, the VAE point of view is that all areas B_i share a border; and as a consequence, the VAE approach does not truly propagate values locally, but in a sense globally across the structure.

Using layers of MLP in the VAE approach can also be problematic as the number of parameters increases dramatically as the number of layers increases. In essence, the number of parameters in an n-layer MLP equals the product of the number of perceptrons in each layer. Since it can be a natural choice to choose deeper MLPs for more complex problems, and more perceptrons when the partition $\{B_i\}$ of $\mathcal{B}$ gets finer, a pure VAE approach with only MLP layers can be computationally challenging to train. As such, using an approach that allows for shared parameters is favourable to increase computational efficiency.

3 Enhancing PriorVAE with Graph Convolutional Networks

3.1 Overview

As discussed in Chapter 2, the PriorVAE does not truly take the spatial structure into account as it applies a flattening layer to the input data. Moreover, it can be computationally challenging to train when the number of areal units increases significantly since then the number of parameters to be learnt also increases dramatically.

We now present a novel architecture, called **PriorVGAE**, to encode spatial priors (such as GP) which leverages the VGAE architecture, making use of GCN layers in both the encoding and decoding phase. This architecture truly considers the local neighbourhood structure as information is propagated by "borrowing" information from neighbouring areal units. Furthermore, a single parameter matrix is used across the whole spatial structure for each GCN layer instance. This favourably reduces the number of parameters to be learnt which makes training computationally cheap unlike the aforementioned approach.

3.2 Local vs Global Learning

The main purpose of introducing a GCN layer for encoding spatial priors is to address the issue of learning in PriorVAE which inherently ignores the local neighbourhood structure. The reason why we feel this is a key issue to address is because spatial priors, as its name suggests, is strongly dependent on the spatial structure, especially "local" ones such as the number of (connected) neighbouring units and the presence of a cycle within neighbouring units (e.g., three neighbouring units could form a triangle). So a mechanism which enables local neighbourhood information aggregation like the GCN would definitely lead to a more effective learning of the desired spatial priors.

Interestingly, however, aggregating **only** local neighbourhood information is not

enough for encoding the spatial priors that we have in mind. For example, GP priors are essentially defined by its kernel which determines its shape; and the kernel induces a covariance matrix that takes into account the interaction of every pair of locations in the domain of interest. As another example, *conditional auto-regressive* (CAR) priors [Besag, 1974] which encourage neighbouring units to be spatially autocorrelated, are inherently defined by its precision matrix; which in turn is dependent on the adjacency and degree matrix of the underlying spatial structure. The main takeaway here is that units generated using these priors, in fact, have access to a certain degree of global information. Therefore, we hypothesize that using a model that strictly considers only local neighbourhood information would limit our performance instead of boosting it.

To this end, we consider several mechanisms that would help learning global information in our model. Of course, the naive (and impractical) way is to just consider many graph layers since K GCN layers would allow information to be propagated K-hops away relative to a node. Unfortunately, this approach would not work due to the oversmoothing phenomenon [Li et al., 2018] where node embeddings average out and results in nodes becoming indistinguishable from one another. It may also not even be computationally feasible since adding GCN layers corresponds to more parameters to be learnt. A more feasible approach was considered by Gilmer et al. [2017] where they suggested introducing *virtual graph elements* as part of the data preprocessing step. In particular, they introduced adding virtual edges between any pair of neighbouring units that are not adjacent by default which allows information to pass beyond local neighbourhoods. It turns out that this simple technique was quite effective. They also introduced adding a so-called master node that connects to all other nodes in the graph, and this also turns out to be helpful for learning. Alon and Yahav [2021] suggested adding a *fully-adjacent* (FA) *layer* which is a similar virtual edge scheme as the one considered by Gilmer et al. [2017], but the only difference is that the FA layer is applied only at the final graph layer. That is, aggregation at the final layer is performed over the entire graph instead over just local neighbourhoods which introduce global information passing that we desired. However, we should note that the authors introduced this scheme to break *bottlenecks* in graph instead of wanting to propagate global knowledge. Nevertheless, it seems like a good idea to explore as this technique does not introduce more learnable parameters, although it does create a dense adjacency matrix which could affect training speed.

We argue, however, that none of these approaches are suitable for learning spatial

priors that we have in mind. This is because spatial priors, such as GP and CAR, not only makes neighbouring units highly spatially autocorrelated but also makes any pair of units that are sufficiently far away from each other having the opposite effect. As such, utilizing a GCN layer for learning, which aggregates information via a **mean** computation, together with a virtual edge scheme that connects non-neighbouring units would lead to a zero out effect. For example, if a unit B_i has the value $y_i = -0.1$ and a neighbouring unit B_{i+1} has the value $y_{i+1} = 0.2$, then this averages out nicely to $y = 0.05$. But if we consider a unit B_j far away from B_i with the value $y_j = 1.5$, then the average in this case amounts to $y = 0.7$. It is then not hard to see that sufficiently many iterations of this mean computation across the whole domain (since we add virtual edges) would lead to a constant value $y = c$ for **all** units in the domain. That is, we will learn priors that comes from a family of constants which is not a desired effect.

So what can we do to enable global information learning in our model? We propose a local-to-global passing scheme by replacing the GCN output layer in the decoder (not the encoder) with an MLP which will consolidate all the local information across the whole domain. After local information is propagated and shared throughout the domain using GCN layers, the output hidden matrix is flattened and is then passed into a single layer MLP. Because MLPs are universal approximators [Cybenko, 1989], this output layer MLP will be able to learn which of the units in the domain that require extra information from relatively remote units in a way that ensure reconstruction loss is minimal. Remarkably, this technique helps us to learn spatial priors successfully as we shall see in Chapter 4.

3.3 Speeding Up Inference

The graph autoencoder architectures we have seen so far uses only GCN layers for learning, especially in the encoder. Such an encoder thus outputs a matrix $\mathbf{Z} \in \mathbb{R}^{N \times M}$ where N is the number of nodes and M is the latent space dimension. For the link prediction task using VGAE in [Kipf and Welling, 2016], a large dimension M should not be problematic as the number of latent samples required is linear in the number of graphs to predict. In particular, if we want to make link predictions on a single graph $\mathcal{G}$, then we require at most one latent sample at test time. This runtime complexity is the same for the node classification and link prediction task using GALA in [Park et al., 2019] (although in this case, we are not doing any sampling per se as GALA is not a *variational* autoencoder).

On the other hand, the task we are ultimately interested in is to perform spatial inference using the learnt decoder, which involves exploring the uncorrelated latent space until a desired number of samples is attained. More precisely, if we have a single graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$, we now have to sample from the latent space $b + n$ times, where b is the number of burn-in samples and n is the number of desired samples after burn-in (here we assumed a thinning factor of 1). The problem now is the following. Since the nodes $v_i \in \mathcal{V}$ assumes their own latent representation $\mathbf{z}_i$, each with dimension M, then the uncorrelated latent space we want to explore has dimension MN which can be relatively large depending on the input dimension. In our particular use case, each input node will typically have a single feature (where then the first GCN layer performs a feature engineering step on top of local information aggregation). Therefore, we have to explore a latent space that is M times bigger than the input space! For example, if $N = 400$ and $M = 10$, we have to explore a latent space with 4000 dimensions which is one order of magnitude higher than the input space. This can lead to an unfavourable reduction in MCMC inference speed, especially if the latent space that is learnt by the VGAE is not strongly uncorrelated.

Inspired by the *graph isomorphism network* architecture [Xu et al., 2018], we suggest to mitigate this issue by replacing the final K-th GCN layer in the encoder (which determines the mean and log standard deviation) with an MLP which compresses the output feature matrix of the $(K-1)$-th GCN layer into a low-dimensional latent vector. By using an MLP, we are able to effectively reduce the latent dimension from MN to a number M' such that $M' \ll MN$, typically at least an order of magnitude lower. As we shall see in Chapter 4, we often choose M' to be less than the number of nodes N since we want to learn good low-dimensional representations of the graph, which is also the case in the existing PriorVAE approach. As a consequence, we are still able to enjoy the same inference runtime speed as PriorVAE on top of now abling to perform effective local information aggregation during training.

3.4 Theoretical Formulation

This section discusses our proposed architecture from the mathematical point of view; and the following section will look at how this architecture can be implemented in practice.

From a probabilistic perspective, our goal is to learn a latent variable model $p_{\boldsymbol{\theta}}(\mathbf{y}, \mathbf{z})$ over $\mathbb{R}^N \times \mathbb{R}^M$ using a GCN layer, where $\mathbf{y}$ is the observations over our domain of

interest. Note that we can equivalently write $\mathbf{y}$ as a (node feature) matrix $\mathbf{Y} \in \mathbb{R}^{N\times 1}$ here whenever we want to be consistent with the graph neural networks literature presented in Chapter 2, although it should be clear that $\mathbf{y} = \mathbf{Y}$ as vectors are always viewed as column vectors in this paper. As discussed in Chapter 2, the VAE approach suggests that we can learn such a model $p_{\boldsymbol{\theta}}(\mathbf{y}, \mathbf{z})$ using a coupled network — an encoder that squeezes the input space into a low-dimensional latent space and a decoder which performs reconstruction of the input space — and the VGAE and GALA approach tells us that we can inject graph layers into the autoencoder scheme to enable the learning of low-level graph representations. Building on the discussion presented in Sections 3.3 and 3.2, we consider a "hybrid" VGAE to work best for learning spatial priors. The encoder and decoder used in this VGAE is given as follows.

Decoder. Let $\mathbf{A}$ be the adjacency matrix induced by the domain. Since we are concerned with real-valued spatial priors, we let the decoder $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z}, \mathbf{A})$ be a multivariate Gaussian parametrized by an MLP-GCN-MLP neural network:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{3.1}$$

$$\mathbf{h} = \mathsf{MLP}(\mathbf{z}), \tag{3.2}$$

$$\mathbf{H} = \mathsf{SharpenGCN}\left(\mathsf{Reshape}\left(\mathbf{h}\right), \mathbf{A}\right), \tag{3.3}$$

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z}, \mathbf{A}) = \mathsf{MLP}(\mathsf{Flatten}\,\mathbf{H}), \tag{3.4}$$

$$\mathbf{y}|\mathbf{z} \sim p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z}, \mathbf{A}). \tag{3.5}$$

The first MLP is to decompress the latent vector $\mathbf{z}$, the GCN is where local information aggregation is performed and the final layer MLP consolidates these local information for learning over the whole graph defined by $\mathbf{A}$.

Encoder. For the encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{X}, \mathbf{A})$, we shall use a Laplacian smoothing GCN for local information aggregation followed by an MLP to approximate the true posterior with a factorized Gaussian, yielding the following encoder:

$$\mathbf{H} = \mathsf{SmoothGCN}(\mathbf{Y}, \mathbf{A}), \tag{3.6}$$

$$(\boldsymbol{\mu}_{\boldsymbol{\phi}}, \log \boldsymbol{\sigma}_{\boldsymbol{\phi}}) = \mathsf{MLP}(\mathsf{Flatten}(\mathbf{H})), \tag{3.7}$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y}, \mathbf{A}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}_{\boldsymbol{\phi}}), \tag{3.8}$$

$$\mathbf{z}|\mathbf{y} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y}, \mathbf{A}). \tag{3.9}$$

As in PriorVAE, we approximate the true posterior using a factorized Gaussian which

leads to an uncorrelated latent space. This simple assumption allows us to perform MCMC inference on spatial priors efficiently unlike traditional methods which suffers from slow chain mixing due to high spatial autocorrelations.

Model learning. For learning, we maximize the ELBO using the formulation given by equation (2.19) with a small modification:

$$\mathcal{L}_{\phi,\boldsymbol{\theta}}(\mathbf{y}) = \xi \cdot \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y},\mathbf{A})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z},\mathbf{A})\right] - \mathsf{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y},\mathbf{A}) \,\|\, p(\mathbf{z})\right), \tag{3.10}$$

where $\xi \in \mathbb{R}^+$ is a heuristic that captures the variance of the Gaussian decoder (see below). Here, we enfore a Gaussian prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ in the KL-divergence to ensure parsimony in the latent space, which as a consequence ensures the latent variables are uncorrelated — a feature that we want for efficient MCMC sampling. Furthermore, the reconstruction term $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y},\mathbf{A})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z},\mathbf{A})\right]$ that we want to maximize in this ELBO can be replaced by the (negative of the) mean square error $\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y},\mathbf{A})}\left[-\|\mathbf{y} - \hat{\mathbf{y}}\|^2\right]$. This is possible because we assumed the likelihood to be a multivariate Gaussian. To see this in detail, assume an isotropic Gaussian likelihood and define

$$p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z},\mathbf{A}) = \mathcal{N}(\mathbf{y} \,|\, \hat{\mathbf{y}}(\mathbf{z},\mathbf{A}), s^2\mathbf{I}),$$

where the function $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{z},\mathbf{A})$ determines the mean of the Gaussian and is parameterized by the decoder. On the other hand, the variance s^2 is a fixed choice. We can then expand this Gaussian likelihood which yields:

$$\begin{aligned}
p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z},\mathbf{A}) &= \mathcal{N}(\mathbf{y} \,|\, \hat{\mathbf{y}}, s^2\mathbf{I}) && (3.11)\\
&= (\det 2\pi s^2\mathbf{I})^{-1/2} \exp\left\{-\frac{1}{2}(\mathbf{y}-\hat{\mathbf{y}})^\top \left(s^2\mathbf{I}\right)^{-1}(\mathbf{y}-\hat{\mathbf{y}})\right\} && (3.12)\\
&= (2\pi)^{-N/2}\, s^{-N} \exp\left\{-\frac{\|\mathbf{y}-\hat{\mathbf{y}}\|^2}{2s^2}\right\} && (3.13)\\
&\propto s^{-N} \exp\left\{-\frac{\|\mathbf{y}-\hat{\mathbf{y}}\|^2}{2s^2}\right\}. && (3.14)
\end{aligned}$$

Taking logarithms further gives us

$$\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z},\mathbf{A}) \propto \log\left(s^{-N} \exp\left\{-\frac{\|\mathbf{y}-\hat{\mathbf{y}}\|^2}{2s^2}\right\}\right) \propto -\left(2N\log s + \frac{\|\mathbf{y}-\hat{\mathbf{y}}\|^2}{s^2}\right). \tag{3.15}$$

And if we further impose $s^2 = 1$, then we ultimately end up with

$$\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z}, \mathbf{A}) \propto -\left(2N \log 1 + \|\mathbf{y} - \hat{\mathbf{y}}\|^2\right) = -\|\mathbf{y} - \hat{\mathbf{y}}\|^2,$$

which is the desired result. Thus, we see that it is sensible to replace the Gaussian log-likelihood with the mean squared error in the ELBO since maximizing the former is equivalent to minimizing the latter (or equivalently, maximizing the negative of the latter). Note, however, that we made a choice to fix $s^2 = 1$ and by doing so we put a restriction on the shape of the Gaussian decoder. Instead of fixing the variance, we can consider the full Gaussian log-likelihood by computing the logarithm as given in equation (3.12). However, in this case, determining the variance s^2 becomes a question of how. For example, one can compute the sample variance across a whole batch or across the whole dataset. One can also treat s^2 as a hyperparameter to be tuned, but [Rybkin et al., 2020] argued that such an approach is not very efficient and they further proposed an automatic way of calculating s^2.

For us, we decided to use a heuristic by introducing a multiplicative hyperparameter ξ in the ELBO (3.10) as we find already great success in this simple approach. This hyperparameter would, in theory, still help increase the spread of the Gaussian decoder as it acts as a heuristic approximation to the factors involving s^2. This can be seen by rewriting equation (3.15) in the following way:

$$\log \tilde{p}_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{z}, \mathbf{A}) = -\left(2N \log s + \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{s^2}\right) = -\left(\frac{2N \log s}{\|\mathbf{y} - \hat{\mathbf{y}}\|^2} + \frac{1}{s^2}\right) \|\mathbf{y} - \hat{\mathbf{y}}\|^2,$$

where $\tilde{p}_{\boldsymbol{\theta}}$ denotes $p_{\boldsymbol{\theta}}$ up to a constant. So we introduce ξ such that

$$\xi \approx \frac{2N \log s}{\|\mathbf{y} - \hat{\mathbf{y}}\|^2} + \frac{1}{s^2}.$$

Immediately, we see that this heuristic hyperparameter should not be optimal regardless of our choice as it is dependent on the observations $\mathbf{y}$. Therefore, if training does not converge as expected, one should consider the aforementioned approaches that involves using the whole Gaussian log-likelihood. Fortunately, we do not see any problems during training time using our heuristic. The final ELBO to be computed is thus given as follows

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\mathbf{y}) = -\xi \cdot \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{y},\mathbf{A})}\left[\|\mathbf{y} - \hat{\mathbf{y}}\|^2\right] - \mathsf{KL}\left(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}, \boldsymbol{\sigma}_{\boldsymbol{\phi}}) \,\|\, \mathcal{N}(\mathbf{0}, \mathbf{I})\right). \tag{3.16}$$

Spatial inference. Once the VGAE has learnt how to encode the spatial priors,

we can then apply the same spatial inference procedure as we did in PriorVAE as discussed in Section 2.4. We sample from the uncorrelated latent prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and pass it through the learnt decoder, yielding the VGAE learnt priors $\boldsymbol{f}_{\mathsf{VGAE}}$. These learnt priors can then be used for spatial MCMC inference by replacing the GP priors evaluation $\boldsymbol{f}_{\mathsf{GP}}$ in (2.38) resulting in the linear predictor

$$u(\mathbb{E}[\mathbf{y}|\boldsymbol{f}_{\mathsf{VGAE}}]) = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{f}_{\mathsf{VGAE}},$$

followed by "observing" data via the likelihood function. A similar setup is done for other spatial priors.

3.5 Model Implementation

We now give a detailed description of how the VGAE is trained end-to-end from input data to their reconstructions. One can view this section as the "neural network formulation" of our architecture, in contrast to the previous section which gave the "Bayesian formulation". The hope is that this section would enable anyone to understand the model with full transparency, and implement it across whatever languages or libraries they prefer.

3.5.1 Encoder Architecture

Input data. The collected dataset is typically a vector $\mathbf{y} \in \mathbb{R}^N$ which is areal data over N points located on a fixed known spatial structure defined by the domain. Such a point is called a *location* (or a *unit*) on the domain. The spatial structure gives rise to an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ that captures the adjacency between locations. Since the first stage of the encoder will be a GCN layer, the encoding phase begins by reshaping the vector $\mathbf{y}$ into a node feature matrix $\mathbf{Y} \in \mathbb{R}^{N \times 1}$ for compatibility with our graph layer.

Graph layer. The pair $(\mathbf{Y}, \mathbf{A})$ is passed through $K \geqslant 1$ layers of Laplacian smoothing GCNs:

$$\mathbf{H}^{(\ell+1)} = \mathsf{SmoothGCN}^{(\ell)}(\mathbf{Y}, \mathbf{A}) = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right),$$

where K is a design choice made at training time based on the complexity of the problem and $\mathbf{H}^{(0)} = \mathbf{Y}$. Here, $\hat{\mathbf{A}}$ is the (Laplacian smoothing) normalized adjacency matrix as given in (2.31), $\mathbf{W}^{(\ell)}$ is a learnable parameter and σ is an activation function

applied element-wise for nonlinearity. Typically, we choose σ to be the ELU activation function. The GCN layers yield node hidden matrices $\mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(K)}$ with hidden dimension $F_1, \ldots, F_K$ respectively. That is, each of these matrix $\mathbf{H}^{(\ell)}$ has dimension $N \times F_\ell$. The hidden dimensions F_ℓ is a design choice to be made before training. Naturally, the hidden dimensions are chosen such that $F_1 \geqslant F_2 \geqslant \ldots \geqslant F_K$ since we want embed the node feature matrix into a lower-dimensional space. However, this is not strictly necessary especially at the intermediary GCN layers (i.e. $\ell < K$) as these layer are used as a feature engineering step for our initial single feature matrix. Furthermore, the actual encoding is done by an MLP which we shall discuss next.

MLP latent layer. The node hidden matrix $\mathbf{H}^{(K)} \in \mathbb{R}^{N \times F_K}$ is flattened by first applying the transpose operator to its rows and then stacking the resulting columns into a hidden vector $\mathbf{h} \in \mathbb{R}^{NF_K}$. This hidden vector is then passed through two independent single layer MLPs which yields the mean vector $\boldsymbol{\mu} \in \mathbb{R}^M$ and the logarithm of the diagonal standard deviation $\log \boldsymbol{\sigma} \in \mathbb{R}^M$. These are the mean and standard deviation which determines our factorized Gaussian approximate posterior. In particular, we obtain the latent vector $\mathbf{z}$ by using the reparametrization trick $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The latent dimension M is a hyperparameter and is typically chosen such that $M < NF_K$ to allow for learning good low-dimensional embeddings. The mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ are then used for computing the KL-divergence between the approximate posterior $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma})$ and the latent prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Note that we do not apply any activation function in the latent layer.

3.5.2 Decoder Architecture

MLP layer. To prepare for forward propagation via a GCN layer, we pass the latent vector $\mathbf{z} \in \mathbb{R}^M$ through a single layer MLP with hidden dimension NF_K, yielding a vector $\mathbf{h}' \in \mathbb{R}^{NF_K}$. Here, F_K is the hidden dimension of the final encoder GCN layer; and this design choice is made with the hope that this MLP layer would approximate a function that is inverse to that learnt by the encoding MLP, enabling effective decompression. An activation function like the ELU is then applied element-wise to the output vector for nonlinearity. The resulting nonlinearized vector $\mathbf{h}'$ is then reshaped into a node hidden matrix $\mathbf{H} \in \mathbb{R}^{N \times F_K}$. This is done by splitting $\mathbf{h}'$ into N columns each of dimension F_K, applying the transpose operator to each resulting rows and stacking them vertically into a matrix. We are now ready to local information propagation using GCN layers.

Graph layer. Using the adjacency matrix $\mathbf{A}$, the pair $(\mathbf{H}, \mathbf{A})$ is propagated

through K Laplacian sharpening GCNs:

$$\tilde{\mathbf{H}}^{(\ell+1)} = \mathsf{SharpenGCN}^{(\ell)}(\mathbf{H}, \mathbf{A}) = \sigma\left(\hat{\mathbf{A}}_{\text{sharpen}}\tilde{\mathbf{H}}^{(\ell)}\tilde{\mathbf{W}}^{(\ell)}\right),$$

where K here is the same as in the encoder, and $\tilde{\mathbf{H}}^{(0)} = \mathbf{H}$. Here, $\hat{\mathbf{A}}_{\text{sharpen}}$ is the Laplacian sharpening normalized adjacency matrix as given in (2.35), $\tilde{\mathbf{W}}^{(\ell)}$ is a parameter weight to be learnt and σ is the same activation function as before. The GCN layers now returns node hidden matrices $\tilde{\mathbf{H}}^{(1)}, \ldots, \tilde{\mathbf{H}}^{(K)}$ with hidden dimension $\tilde{F}_1, \ldots, \tilde{F}_K$ respectively. These hidden dimensions are chosen by simply putting $\tilde{F}_\ell = F_{K+1-\ell}$, which creates an ordering $\tilde{F}_1 \leqslant \tilde{F}_2 \leqslant \ldots \leqslant \tilde{F}_K$ that is reversed from that of the encoder. This choice of dimension is sensible because as we argued, we want to perform effective decompression here.

MLP output layer. Finally, we consolidate local information across the whole graph by passing the pair $(\tilde{\mathbf{H}}^{(K)}, \mathbf{A})$ through a single layer MLP with dimension N. This yields an output vector $\hat{\mathbf{y}} \in \mathbb{R}^N$ which is a reconstruction of the input $\mathbf{y}$. The reconstruction error can then be computed by evaluating the mean squared error $\|\mathbf{y} - \hat{\mathbf{y}}\|^2$ which forms the final ingredient in the ELBO optimization step.

4 Results

4.1 Technical Setup

All the model implementations were performed in Python using JAX [Bradbury et al., 2018], Haiku [Hennigan et al., 2020] and Jraph [Godwin* et al., 2020]; where the latter two libraries are also based on JAX. Choosing JAX (and JAX-based libraries) as our primary machine learning framework allows us to take advantage of the efficient JIT (just-in-time) compiler which enables highly efficient vectorization and automatic differentiation. Furthermore, the functional programming interface provided by JAX allows us to translate our model in its mathematical formulation to Python code with ease. This is due to the composability (as in any functional programming language) of functions which allows us to read mathematics and code almost interchangeably. Moreover, the source of randomness are all clearly parameterized in JAX via the use of predetermined RNG (random number generator) keys. Hence, reproducibility does not become an issue and any other practitioner should be able to run our same code on their machine and obtain the same results.

Our JAX implementation of the Laplacian sharpening GCN layer is based on the original paper [Park et al., 2019] and is built by modifying the (Laplacian smoothing) GCN layer implemented in Jraph. The original Jraph implementation is publicly available in the GitHub repository: `https://github.com/deepmind/jraph`. On the other hand, our GP and PriorVAE implementations (including visualizations) are based on the publicly available PriorVAE repository given here: `https://github.com/elizavetasemenova/PriorVAE`. The full code of all our implementations are given in the following repository: `https://github.com/salfaris/PriorVGAE`.

All the experiments in the forthcoming sections are executed on an Apple M1 Pro CPU machine with 16 GB memory.

4.2 One-dimensional Gaussian process over regular grids

This section presents the results obtained when performing inference using the VGAE on a Gaussian process over a one-dimensional regular grid.

Dataset generation. We first partition the closed interval $[0, 1]$ into $N = 400$ equispaced ordered points $(p_1, \ldots, p_N)$ which we call *locations*. These locations give rise to a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the nodes are given by the locations $\mathcal{V} = \{p_1, \ldots, p_N\}$ and the edges $\mathcal{E}$ are given by the multiset defined by

$$e_{jk} = \begin{cases} 1, & \text{if } |j - k| = 1, \\ 0, & \text{otherwise,} \end{cases}$$

where e_{jk} is an edge between node p_j and node p_k if it equals 1. Naturally, this graph induces an adjacency matrix $\mathbf{A}$. Over this graph, we generate training data $\mathbf{y} = (y_1, \ldots, y_N)$ by sampling from a GP with zero mean and covariance given by the RBF kernel

$$\mathcal{K}(p_i, p_j) = \sigma^2 \exp\left\{-\frac{(p_i - p_j)^2}{2\ell^2}\right\} \tag{4.1}$$

where here σ is the *variance* (or *scale*) of the GP from the zero mean and ℓ is the *lengthscale* which governs the smoothness of the covariance over the grid. This choice of the kernel is suitable for small-area estimation, especially when the covariance matrix is (inherently) based on the Euclidean distance. To ensure that the the VGAE will be trained on a broad class of GP priors, we do not want a fixed variance and lengthscale so we impose a log-normal hyperprior $\text{LogNormal}(0, 0.1)$ on the variance and an inverse gamma hyperprior $\text{InverseGamma}(4, 1)$ on the lengthscale. The full dataset generation process is thus given as follows:

$$\sigma^2 \sim \text{LogNormal}(0, 0.1), \tag{4.2}$$

$$\ell \sim \text{InverseGamma}(4, 1), \tag{4.3}$$

$$\mathcal{K}_{ij} = \sigma^2 \exp\left\{-\frac{(p_i - p_j)^2}{2\ell^2}\right\}, \tag{4.4}$$

$$\mathbf{y} \sim \mathsf{GP}(\mathbf{0}, \mathbf{K}), \text{ where } \mathbf{K} = (\mathcal{K}_{ij}). \tag{4.5}$$

GP evaluations $\boldsymbol{f}_{\mathsf{GP}} := \mathbf{y}$ can be seen in Figure 4.1a. Once we have generated training data $\mathcal{D} = \left\{\mathbf{y}^{(i)}\right\}_{i=1}^{L}$, we then convert each $\mathbf{y}^{(i)}$ into an $N \times 1$ node feature matrix

$\mathbf{Y}^{(i)}$, and the pair $(\mathbf{Y}^{(i)}, \mathbf{A})$ is propagated through the VGAE for model training.

4.2.1 Model Implementation

For the encoder, we use a two-layer Laplacian smoothing GCN with hidden layer dimensions 12 and 6, respectively, and apply the ELU activation function for non-linearity. For the bottleneck, we use a 10-dim latent layer. The latent dimension is chosen to match the baseline implementation (see below) to ensure that a fair comparison is being made. As discussed in Section 3.5, the first two stages of the decoder are symmetrical with respect to the encoder's implementation so no hyper-parameter specification is required. The final decoder layer is an MLP with output dimension $N = 400$. We train the model using the Adam optimizer on 10,000 graph batches over 10,000 epochs (i.e., each epoch sees a new GP batch). We use a batch size of 100, an initial learning rate of 0.001 and a ξ factor of 100. The learning rate is varied using an exponential schedule with a decay rate of 0.99 over 2000 transitions.

Baseline implementation. To compare our proposed architecture with the existing approach, we also train a VAE following the PriorVAE architecture with model specifications following exactly those used in the original paper [Semenova et al., 2022]. In particular, we use a two-layer MLP with dimensions 35 and 30; and a 10-dim bottleneck latent layer. The ReLU activation function is applied for nonlinearity in each hidden layer. The decoder has a symmetrical implementation so it uses the same hidden layer dimensions in reverse. We train the model as the authors did using the Adam optimizer on 50,000 GP batches with batch size 100 over 50,000 epochs. The learning rate that is used 0.001, without any scheduler being employed.

4.2.2 Experimental Results

Figure 4.1 shows the GP priors to be encoded, the PriorVAE priors and the priors learnt using the VGAE. We observe that the priors learnt using our method have the same shape and mean near zero as the desired GP priors. However, we see that the amount of uncertainty is slightly a bit lower than expected. To assess how well our VGAE have trained, we generate a ground truth data $\mathbf{y}^{(\text{truth})}$ using the process given in (4.2)-(4.5), and then add i.i.d noise following a half-normal distribution $\mathcal{N}^{+}(0.1)$ to yield a simulation of an observed data $\mathbf{y}^{(\text{observed})}$. We then try to use priors learnt by the VGAE to infer the true underlying data $\mathbf{y}^{(\text{truth})}$ based on varying amounts of missing locations in $\mathbf{y}^{(\text{observed})}$: with (i) 99.5% of locations are missing,

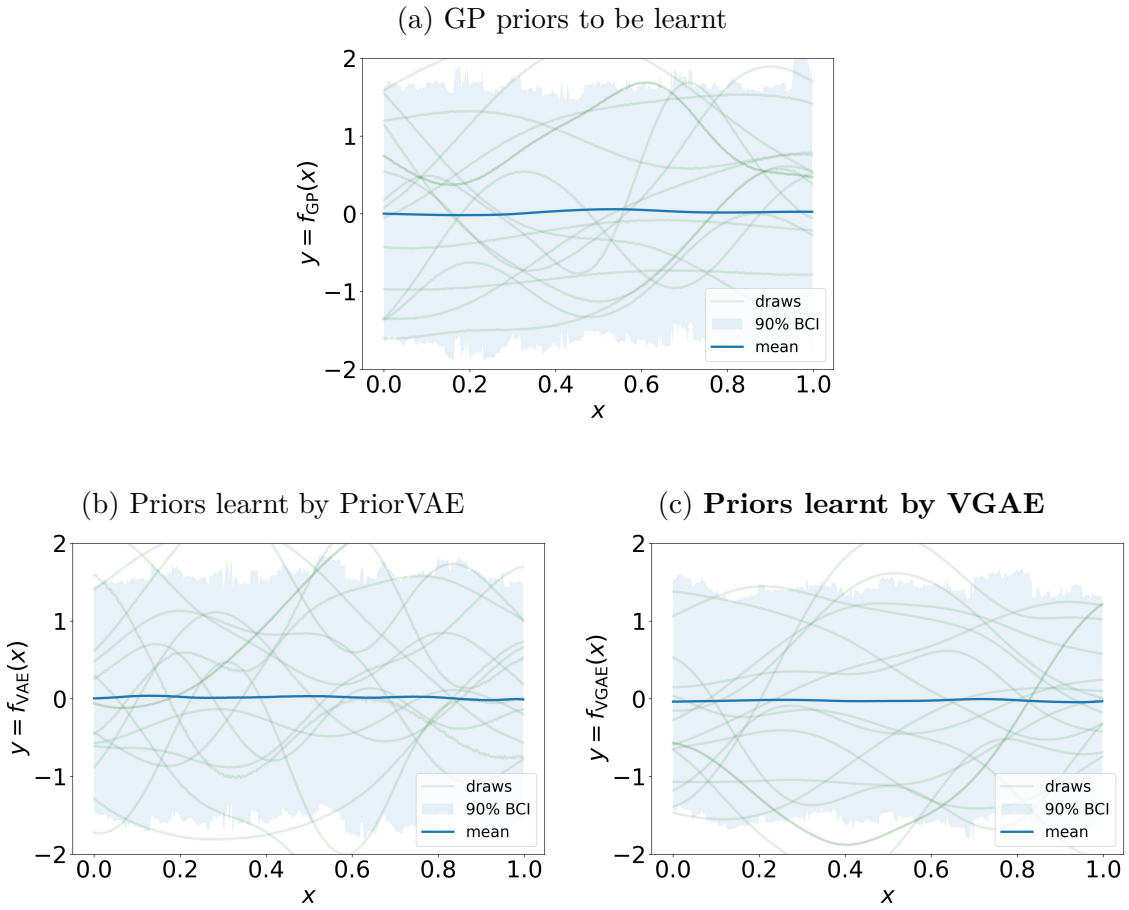
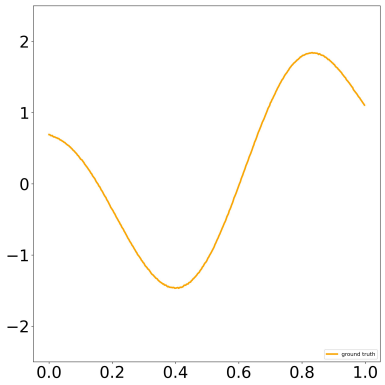


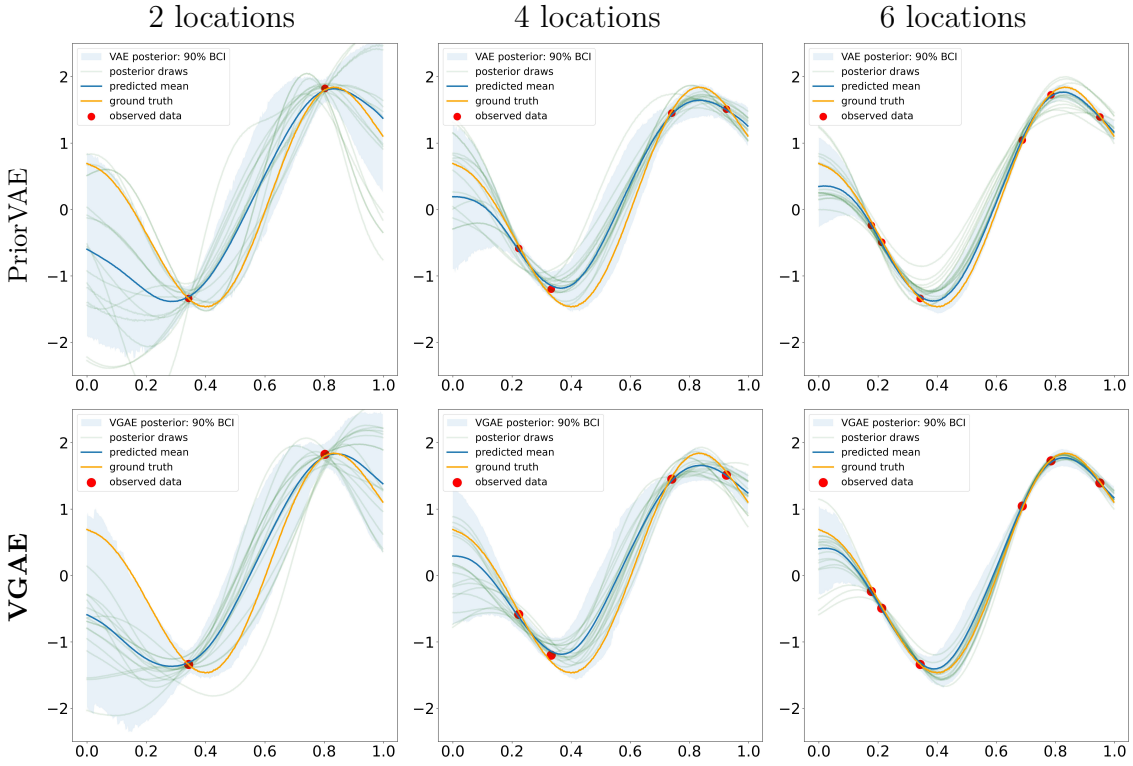
Figure 4.1: One-dimensional GP priors over a **regular** grid. The mean and 90% credible interval (blue coloured) are computed by drawing 1000 samples from the prior.

(ii) 99% of locations are missing and 98.5% locations are missing. We do this by using the model $\mathbf{y} \sim \mathcal{N}(\boldsymbol{f}_{\text{VGAE}}, s^2)$ where $\boldsymbol{f}_{\text{VGAE}}$ is sampled from the learnt VGAE decoder, and the noise s^2 admits a half-normal prior $s \sim \mathcal{N}^+(0.1)$. The result for this assessment is given in Figure 4.2 where we observe that the estimated mean gets closer to the ground truth $\mathbf{y}^{(\text{truth})}$ as the number of missing locations decreases. Moreover, the amount of uncertainty seems lowest within the proximity of observed locations and highest for those far away from observed locations; and both of these are expected and desired phenomena. To further verify that we have learnt good priors, we compute the empirical covariance matrix based on 1000 samples from the priors. Figure 4.3 shows that indeed the empirical covariance matrix of the VGAE admits a similar shape to that of the GP prior.

To compare the performance of our approach and the baseline PriorVAE model,

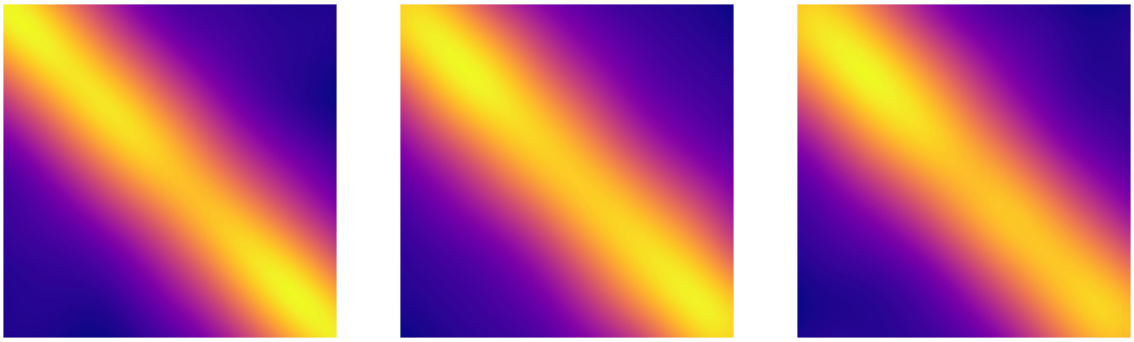


(a) Ground truth



(b) Inference using learnt priors

Figure 4.2: Fitting a noisy GP data on a **regular** grid using the PriorVAE priors (first row in (b)) and learnt VGAE priors (second row in (b)). The ground truth to be inferred is given in (a). We perform inference on varying percentages of missing locations: 99.5%, 99% and 98.5% out of 400 locations which corresponds to 2, 4 and 6 observed locations. These observations are indicated as a red dot (●) in the plots. The posterior mean of our learnt prior is plotted in green and a 90% credible interval is plotted in blue. The mean prediction improves with increasing amounts of observed locations.



(a) GP (b) PriorVAE (c) **VGAE**

Figure 4.3: Empirical covariance matrices of the one-dimensional GP priors on a **regular** grid. Each covariance matrix is computed based on 1000 draws from the respective priors.

we compute the *mean squared error* (MSE) between the ground truth data and the mean inferred by both models. This assessment is given in Table 4.1. We observe that our approach has a lower MSE than PriorVAE across all the missing location settings considered. However, the difference between the MSEs gets narrower as more points are observed which suggests that for large number of observations, the PriorVAE should perform just as good as the VGAE on the one-dimensional regular grid.

	2 locations	4 locations	6 locations
PriorVAE	0.421	0.082	0.014
VGAE	**0.409**	**0.075**	**0.011**

Table 4.1: MSE between the ground truth GP and the inferred mean in the one-dimensional **regular** grid for $n = 2, 4, 6$ noisy observations. The mean is computed using 1000 draws from the prior.

4.3 One-dimensional Gaussian process over irregular grids

We now consider the VGAE for inference on a Gaussian process over an irregular one-dimensional grid.

Dataset generation. To generate an irregular grid, we first generate a regular grid as in Section 4.2 and then draw a small subset of locations from this grid by uniform sampling. For our experiment, we choose a subset of $N = 32$ locations.

This results in an ordered set of locations $(p_1, \ldots, p_N)$ which can be turned into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by using the exact same description detailed in the one-dimensional regular grid experiment. Training data $\mathbf{y} = (y_1, \ldots, y_N)$ is then generated over this graph by using the same data generation process in the regular grid experiment, i.e., as given by eqs. (4.2)–(4.5). In particular, we again use hyperpriors for the variance and lengthscale to ensure that our VGAE will be trained on a broad class of GP priors.

4.3.1 Model Implementation

We use the same model specifications as in the one-dimensional regular grid GP except that we now use an initial learning rate of 0.01 and a ξ factor of 1000. An exponential schedule is again employed on the learning rate with a decay rate of 0.7 over 2000 epochs.

Baseline implementation. As for the baseline, we use the exact same VAE setup as in the regular grid case which is what the authors of the PriorVAE did.

4.3.2 Experimental Results

Figure 4.4 displays the GP realizations that we want to learn, the learnt PriorVAE priors and the priors learnt using our proposed approach. As in the regular grid experiment, we see that the priors learnt using the VGAE have the same mean and shape as the desired GP prior. Moreover, we again observe that the amount of uncertainty in the learnt VGAE priors is lower than expected, but it still matches the amount of uncertainty displayed by PriorVAE priors. To assess the performance of our proposed method, we repeat the same procedure as we did in the one-dimensional regular grid experiment. That is, we generate a ground data $\mathbf{y}^{(\text{truth})}$, simulate an observation by adding i.i.d half-normal noise to $\mathbf{y}^{(\text{truth})}$, and then try to infer $\mathbf{y}^{(\text{truth})}$ based on a limited number of observations. Again, the amount of missing locations used are 99.5%, 99% and 98.5% which corresponds to 2, 3 and 4 observations, respectively. The result for this experiment is given in Figure 4.5 where we again see that the quality of the inferred mean by he VGAE improves with decreasing number of missing locations. We also compute the empirical covariance matrices of each prior using 1000 sample draws to verify that the learnt VGAE priors admits the same shape as the GP prior; which is what we observe as seen in Figure 4.6.

We now compare the performance of our proposed approach with the baseline PriorVAE. First, we can visually observe in Figure 4.5 that our proposed VGAE are

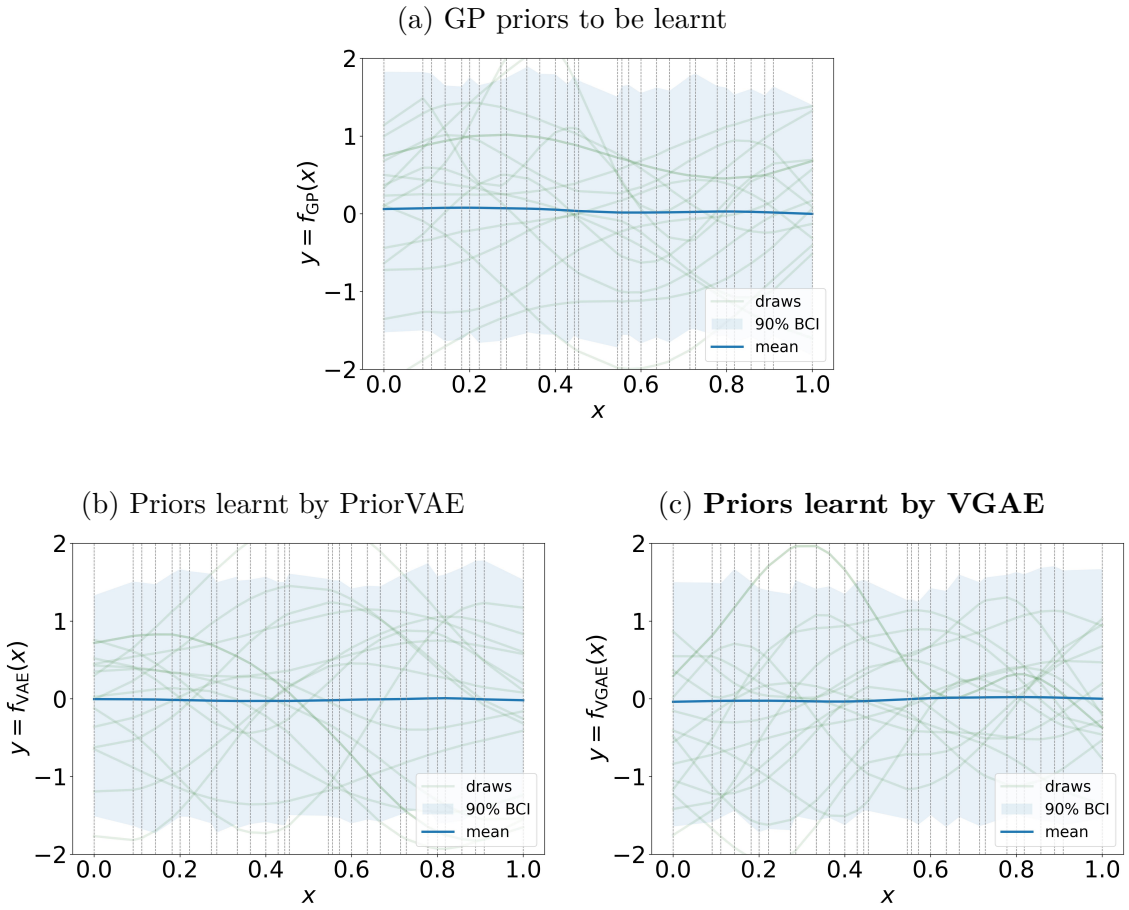


Figure 4.4: One-dimensional GP priors over an **irregular** grid. The black dashed lines indicates the locations on the irregular grid. The mean and 90% credible interval (blue coloured) are computed by drawing 1000 samples from the prior.

able to fit to the ground truth with more success than PriorVAE, especially in the case where 3 and 4 observations are given. In fact, PriorVAE was not able to fit the ground truth correctly at all even when 4 observations are given, evident from the high uncertainty displayed in the central regions. Now to make this observation concrete, we compute the MSE between the ground truth and the estimate of both approaches. This calculation is given in Table 4.2 where we observe that the MSE is significantly lower using our approach in all the considered settings.

To explain this observation, we first note that GP priors using the RBF kernel over an *irregular* grid implies that "neighbouring" locations could have significantly different covariance which was not true in the regular grid. To concretely see this, we consider the interval $[0, 10]$ and choose the three-point subset $\{p_1 = 0, p_2 = 0.1, p_3 = 10\} \subseteq [0, 10]$. Then by definition of our graph, there exist

edges (p_1, p_2) and (p_2, p_3) which implies that the points $\{p_1, p_3\}$ forms a neighbourhood of p_2. But then computing the RBF kernel (4.1) with fixed $\sigma = \ell = 1$ shows that

$$\mathcal{K}(p_1, p_2) = e^{-0.05^2} \gg e^{-4.95^2} = \mathcal{K}(p_3, p_2).$$

That is, the covariance of (p_1, p_2) is much higher than that of (p_2, p_3) despite both p_1 and p_3 being "neighbours" of the same location. In this particular case, $\mathcal{K}(p_3, p_2) \approx 0$ whereas $\mathcal{K}(p_1, p_2) \approx 1$, and so p_2 tends to "follow" p_1 more as compared to p_3. Now back to the discussion on VGAE versus PriorVAE. The difference in covariance between neighbouring locations implies that a location would be biased to follow one neighbour more than the other; this is a *local* property of the irregular grid GP priors. As such, a mechanism that allows for *explicit* local information aggregation would help improve fitting over such priors, as opposed to a mechanism that *has to learn* that there is (local) bias in the data. This explains why the VGAE was able to perform significantly better than PriorVAE; and thus the VGAE is preferred for irregular grids.

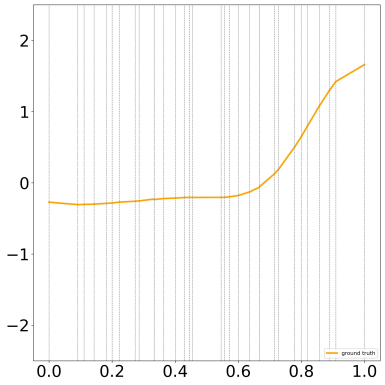
	2 locations	3 locations	4 locations
PriorVAE	0.166	0.054	0.049
VGAE	**0.129**	**0.035**	**0.003**

Table 4.2: MSE between the ground truth GP and the inferred mean in the one-dimensional **irregular** grid for $n = 2, 3, 4$ noisy observations. The mean is computed using 1000 draws from the prior.

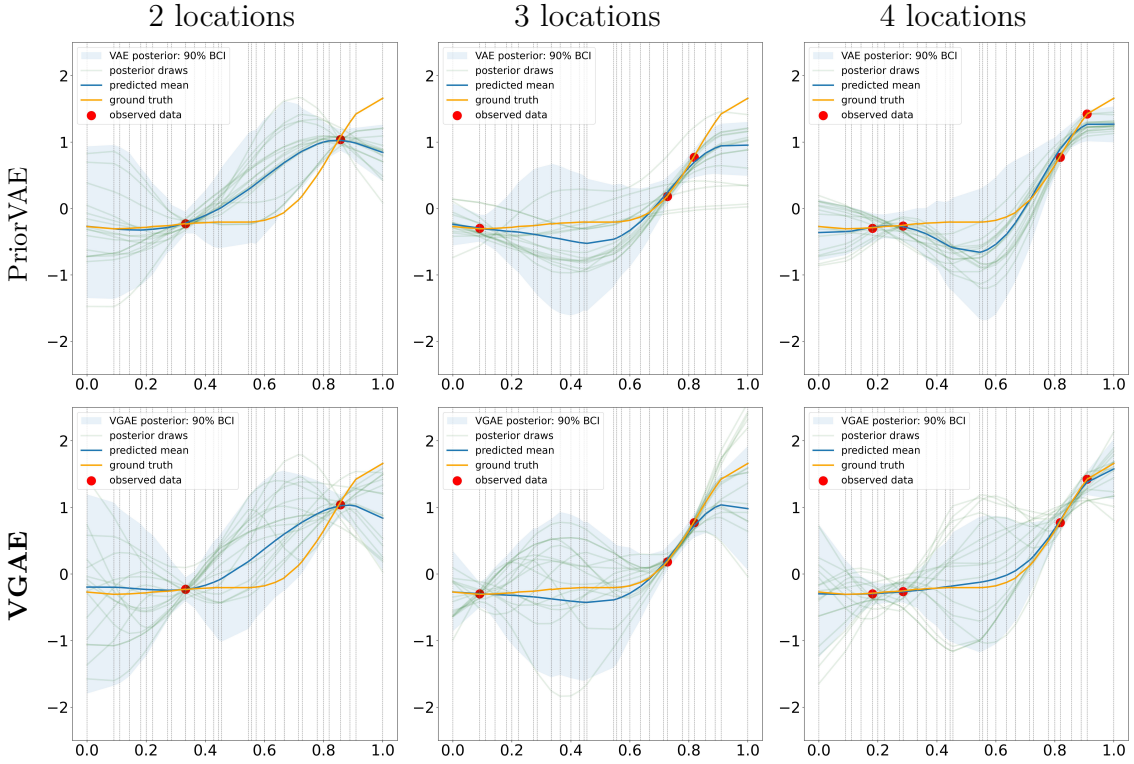
4.4 Two-dimensional Gaussian process

We now use the VGAE to perform inference on a Gaussian process over a two-dimensional regular grid.

Dataset generation. A 25×25 regular grid is first created by partitioning a unit square into $N = 625$ equally-sized locations. If we fix an ordering on the locations, we can further write these locations compactly as a vector $(\mathbf{p}_1, \ldots, \mathbf{p}_N) \in \mathbb{R}^{N \times 2}$ where $\mathbf{p}_i$ corresponds to the coordinate vector of location i. These locations of course has a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the nodes are given by $\mathcal{V} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ but the edges $\mathcal{E}$ are a bit complicated to define given how we did the partition. Partitioning a unit square into a 25×25 regular grid means that the locations are indexed at $\left\{0, \frac{1}{25}, \frac{2}{25}, \ldots, \frac{24}{25}\right\}$ along both axes. Such an indexing can be injectively mapped

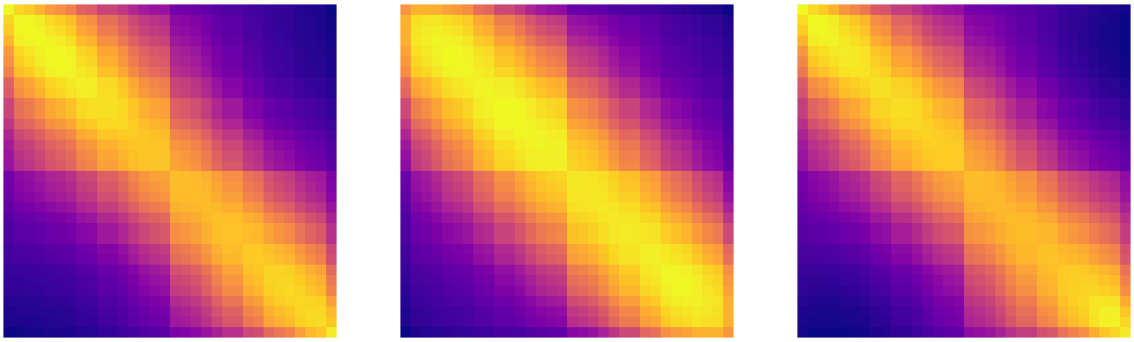


(a) Ground truth



(b) Inference using learnt priors

Figure 4.5: Fitting a noisy GP data over an **irregular** grid using the PriorVAE priors (first row in (b)) and learnt VGAE priors (second row in (b)). The black dashed lines indicates the locations on the irregular grid. The ground truth to be inferred is given in (a). We perform inference on varying percentages of missing locations: 99.5%, 99% and 98.5% out of 32 locations which corresponds to 2, 3 and 4 observed locations, respectively. These observations are indicated as a red dot (●) in the plots. The posterior mean of our learnt prior is plotted in green and a 90% credible interval is plotted in blue. The mean prediction improves with increasing amounts of observed locations.



(a) GP (b) PriorVAE (c) **VGAE**

Figure 4.6: Empirical covariance matrices of the one-dimensional GP priors on an **irregular** grid. Each covariance matrix is computed based on 1000 draws from the respective priors.

to the set $\{0, 1, 2, \ldots, 24\}$ via the map $x/25 \to x$, which implies that without loss of generality, a coordinate vector $\left(\frac{a}{25}, \frac{b}{25}\right)$ can be simply written as (a, b). In this formulation, the edges $\mathcal{E}$ can thus be defined by

$$e_{jk} = \begin{cases} 1, & \text{if } |a_j - a_k| = 1 \text{ or } |b_j - b_k| = 1, \\ 0, & \text{otherwise,} \end{cases}$$

where e_{jk} is an edge between node $\mathbf{p}_j = (a_j, b_j)$ and $\mathbf{p}_k = (a_k, b_k)$ if it equals 1. Training data $\mathbf{y} = (y_1, \ldots, y_N)$ can then be generated over this grid by using a similar approach to the one-dimensional case by realizing GPs with zero mean and covariance given by the RBF kernel

$$\mathcal{K}(\mathbf{p}_i, \mathbf{p}_j) = \sigma^2 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\ell^2}\right\}.$$

Also similar to the one-dimensional GP experiment, we want to train the VGAE on a broad class of GP prior. As such, we impose the same hyperpriors on the variance and the lengthscale. The full dataset generation process is thus given as follows

$$\sigma^2 \sim \text{LogNormal}(0, 0.1), \tag{4.6}$$

$$\ell \sim \text{InverseGamma}(4, 1), \tag{4.7}$$

$$\mathcal{K}_{ij} = \sigma^2 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\ell^2}\right\}, \tag{4.8}$$

$$\mathbf{y} \sim \mathsf{GP}(\mathbf{0}, \mathbf{K}), \text{ where } \mathbf{K} = (\mathcal{K}_{ij}). \tag{4.9}$$

Sampling L times using the above scheme yields the training data $\mathcal{D} = \left\{\mathbf{y}^{(i)}\right\}_{i=1}^{L}$ which we convert to $N \times 1$ node feature matrices $\left\{\mathbf{Y}^{(i)}\right\}_{i=1}^{L}$. The pair $(\mathbf{Y}^{(i)}, \mathbf{A})$ is then passed through the VGAE for training, where $\mathbf{A}$ here is the adjacency matrix associated to $\mathcal{G}$.

4.4.1 Model Implementation

For both the encoder and decoder, we use the exact same model specifications as described in the one-dimensional GP experiment. However, we now train the model using the Adam optimizer on 1000 GP graph batches with batch size 100 over 1000 epochs. We use an initial learning rate of 0.0001 and a ξ factor of 1000. An exponential schedule is again employed on the learning rate with a decay rate of 0.99 over 2000 training steps.

Baseline implementation. For the baseline, we follow the PriorVAE paper by training a VAE using the same model specification as described in the one-dimensional GP experiment.

4.4.2 Experimental Results

We generate ground truth data using the process as described in eqs. (4.6)–(4.9). To simulate observed data, we add i.i.d noise based on a LogNormal$(0, 0.1)$ distribution. As before, we hide some amount of locations and try to infer the underlying ground truth data using priors learnt using VGAE. For this experiment, we consider the case of 99%, 98% and 97% missing locations which corresponds to 6, 12 and 19 observed locations respectively. At the inference stage, we use the model $\mathbf{y} \sim \mathcal{N}(\boldsymbol{f}_{\mathsf{VGAE}}, s^2)$ where $s^2 \sim$ LogNormal$(0, 0.01)$. The result of this experiment is presented in Figure 4.7, where the mean are computed based on 1000 prior draws. We observe that as the percentage of missing locations decreases, the mean inferred using our technique gets closer to the ground truth. Moreover, the uncertainty of this estimate across all locations favourably decreases as well.

As before, we compare our model's performance with the existing PriorVAE technique by computing the MSE between the inferred mean and the ground truth data. The result of this experiment is present in Table 4.3. As in the the one-dimensional GP experiment, we observe that the VGAE approach has favourably smaller MSE across all the three missing location setups.

	6 locations	12 locations	19 locations
PriorVAE	0.478	0.268	0.127
VGAE	**0.437**	**0.200**	**0.120**

Table 4.3: Mean squared error (MSE) between the ground truth GP and the inferred mean (based on 1000 samples) in the two-dimensional regular grid for $n = 6, 12, 19$ noisy observations.

4.5 Synthetic conditional auto-regressive (CAR)

We now look at how we can use the VGAE to perform MCMC inference on spatial priors beyond the GP. In this section, we train a VGAE on the synthetic CAR dataset over a two-dimensional regular grid and try to perform inference using the learnt VGAE priors.

Dataset generation. We create a 15×10 regular grid by partitioning a unit rectangle into $N = 150$ equally-sized locations. This partition gives rise to a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ similar to what was described in the two-dimensional GP experiment; and this graph structure induces an adjacency matrix $\mathbf{A}$, which is a key component for generating data in the CAR model [Besag, 1974]. Training data $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_N)$ is generated over this grid using the following two-step process:

$$\bar{\mathbf{Q}}^{-1} = \mathbf{D} - \alpha\mathbf{A}, \tag{4.10}$$

$$\bar{\boldsymbol{\phi}} \sim \mathcal{N}(\mathbf{0}, \bar{\mathbf{Q}}^{-1}). \tag{4.11}$$

Here, $\mathbf{D}$ is the degree matrix of $\mathbf{A}$ given by $D_{ii} = \sum_j A_{ij}$, α is a parameter that determines the amount of spatial autocorrelation between locations, and $\mathbf{Q}$ is the precision matrix of the multivariate normal. To guarantee that the VGAE is trained on a sufficiently broad family of CAR priors, we impose a uniform hyperprior on α, where we assume that $\alpha \sim \mathsf{Uniform}(0.4, 1)$. The support $[0.4, 1]$ chosen in this hyperprior is not completely arbitrary. Instead, we are most interested in learning CAR priors that exhibit high spatial autocorrelation and larger values $\alpha > 0.4$ in CAR results in exactly such priors. Thus, the full dataset generation process we use

is given as follows:

$$\alpha \sim \mathsf{Uniform}(0.4, 1), \tag{4.12}$$

$$\bar{\mathbf{Q}}^{-1} = \mathbf{D} - \alpha\mathbf{A}, \tag{4.13}$$

$$\bar{\boldsymbol{\phi}} \sim \mathcal{N}(\mathbf{0}, \bar{\mathbf{Q}}^{-1}). \tag{4.14}$$

4.5.1 Model Implementation

To demonstrate the importance of local-to-global learning, we shall train two VGAEs on the CAR priors. The first VGAE will be the model we have been using so far which follows the architecture described in Section 3.5. From now on, this VGAE will be referred to as VGAE+L2G, where L2G denotes "local-to-global" learning as argued in Section 3.2. On the other hand, the second VGAE, which we will refer to as "base VGAE" or just VGAE, will use the same architecture as ours except at the final layer in the decoder, where instead of an MLP "pooling" layer, we use a Laplacian sharpening GCN. In particular, the second architecture uses only local information for reconstructing the input data.

The following model specification shall be the same for both of the considered VGAEs. For the encoder, we use a one-layer Laplacian smoothing GCN with a 5-dim hidden layer followed by ELU activation; and for the bottleneck, we use a latent layer with 130 dimensions. The dimension here is chosen so that it matches the baseline latent dimension exactly (see below). In VGAE+L2G, the final decoder layer is an MLP with dimension $N = 150$; and for the base VGAE, the final decoder layer is a Laplacian sharpening GCN with a 1-dim output layer. Both of these models are trained using the Adam optimizer on 10,000 graph batches over 10,000 epochs (again, each epoch sees a fresh batch), with a batch size of 100. We employ an initial learning rate of 0.01 and a ξ factor of 1000; and we use an exponential schedule for the learning rate with 0.99 decay rate done over 2000 training steps.

Baseline implementation. As a baseline, we employ a VAE based on the architecture given by the author. In particular we use a single layer MLP with hidden dimension 130 followed by the ELU activation function; and for the bottleneck, we use a latent layer with dimension 130.

4.5.2 Experimental Results

To assess the performance of our models, we generate a ground truth data that has high spatial autocorrelation by fixing $\alpha = 0.7$ and use the data generation process

	MSE	ESS of spatial random effect	MCMC elapsed time (s)
PriorVAE	0.241	3418	4
VGAE	0.165	2028	95
VGAE+L2G	**0.153**	**2714**	**38**
CAR	0.135	4104	159

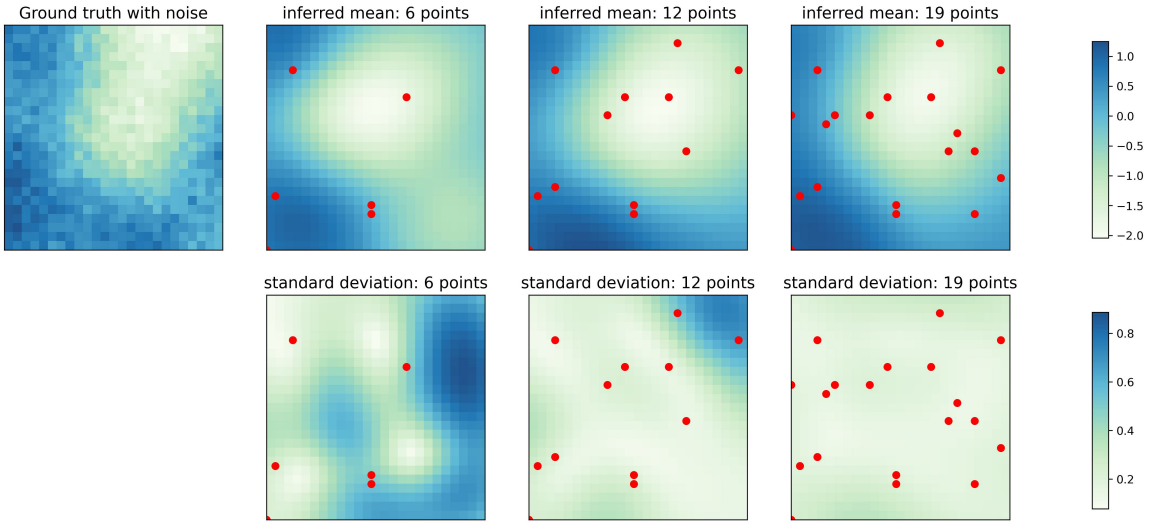
Table 4.4: Mean squared error (MSE) between the ground truth CAR data and the model mean predictions (based on 1000 samples) in the two-dimensional regular grid.

given in eqs. (4.13) and (4.14). To simulate an observation, we add i.i.d noise based on the $\mathcal{N}(0, 0.25)$ distribution to each location. We then fit our models to this observed data and compute the (mean squared) error between the model estimate and the underlying ground truth. As in the GP experiments, we use the model $\mathcal{N}(\boldsymbol{f}, s^2)$ for inference where $\boldsymbol{f}$ is the model prior and the noise s^2 is given a $\mathsf{Uniform}(0.01, 1)$ hyperprior. $\boldsymbol{f}$ here can either be the fitted CAR priors or the priors learnt using PriorVAE, VGAE or VGAE+L2G. The fitting process is done by running MCMC for 1000 burn-ins before drawing 2000 samples from the posterior predictive distribution. The ground truth, observed data and the fitted model predictions by the baseline and VGAEs are given in Figure 4.8. We observe that both VGAEs are able to estimate the ground truth quite well. In particular, they both are able to capture the values in the wider range $\phi > 1.0$ and $\phi < -1.0$ which is not true in PriorVAE whose estimate suffers from the VAE smoothing effect [Larsen et al., 2016; Dumoulin et al., 2017].

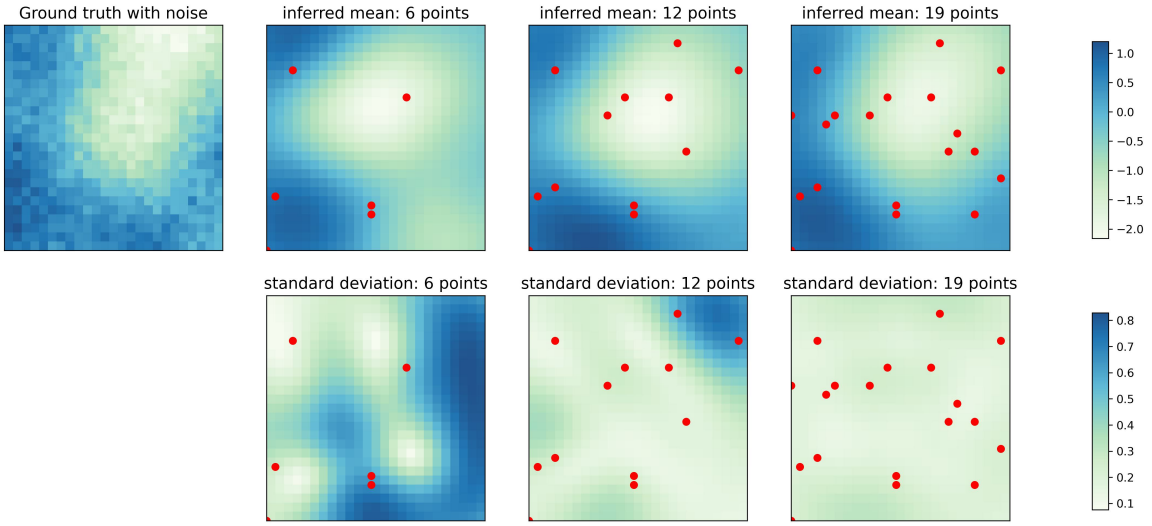
To compare the performance of our proposed approach with the baseline, we further compute the MSE between the ground truth CAR prior and the model estimates. We also compute the *effective sample size* (ESS) of the spatial random effects and the total elapsed time for fitting to the data using MCMC. These results are presented in Table 4.4. Naturally, the CAR estimate has the lowest MSE as we are fitting an observation that comes from the CAR model. This estimate is not expected to be beaten but rather serves as an upper bound on the performance that we can achieve. We observe that both VGAEs have a lower MSE as compared to the baseline PriorVAE which supports our previous qualitative discussion. However, the total MCMC elapsed time in PriorVAE is significantly lower as compared to both VGAE methods. We believe that this is due to the high ξ factor used in the VGAE ELBO (3.16) which leads to putting more weight in the reconstruction term instead of the regularizer KL-divergence term, making the latent space not as uncorrelated as

we desired. This also explains why both VGAEs have a much lesser ESS as compared to the PriorVAE.

We observe that the local-to-global scheme has a significant positive impact in learning the CAR priors. First, we see that VGAE+L2G approach estimates the ground truth better than the base VGAE, evident from the lower MSE computation. Secondly, we hypothesize that the local-to-global scheme leads to learning a more uncorrelated latent space which is a desired effect. This is evident from the higher ESS of spatial random effects that we get when using the local-to-global model as compared to the base VGAE which does not apply such a pooling scheme. Furthermore, this hypothesis is supported by the significantly lower MCMC elapsed time of 38 seconds as compared to 95 seconds in VGAE, suggesting that the latent space is much more simpler to be explored at inference time.



(a) **VGAE**



(b) PriorVAE

Figure 4.7: We perform inference on a GP data over a two-dimensional regular grid with added i.i.d log-normal noise using our learnt priors. Inference is performed on the data with 99%, 98% and 97% missing locations, which corresponds to 6, 12 and 19 observed locations. These locations (or points) are indicated as a red dot (●) in the plots. We see that the inferred mean improves with decreasing uncertainty as the number of observed locations increases.

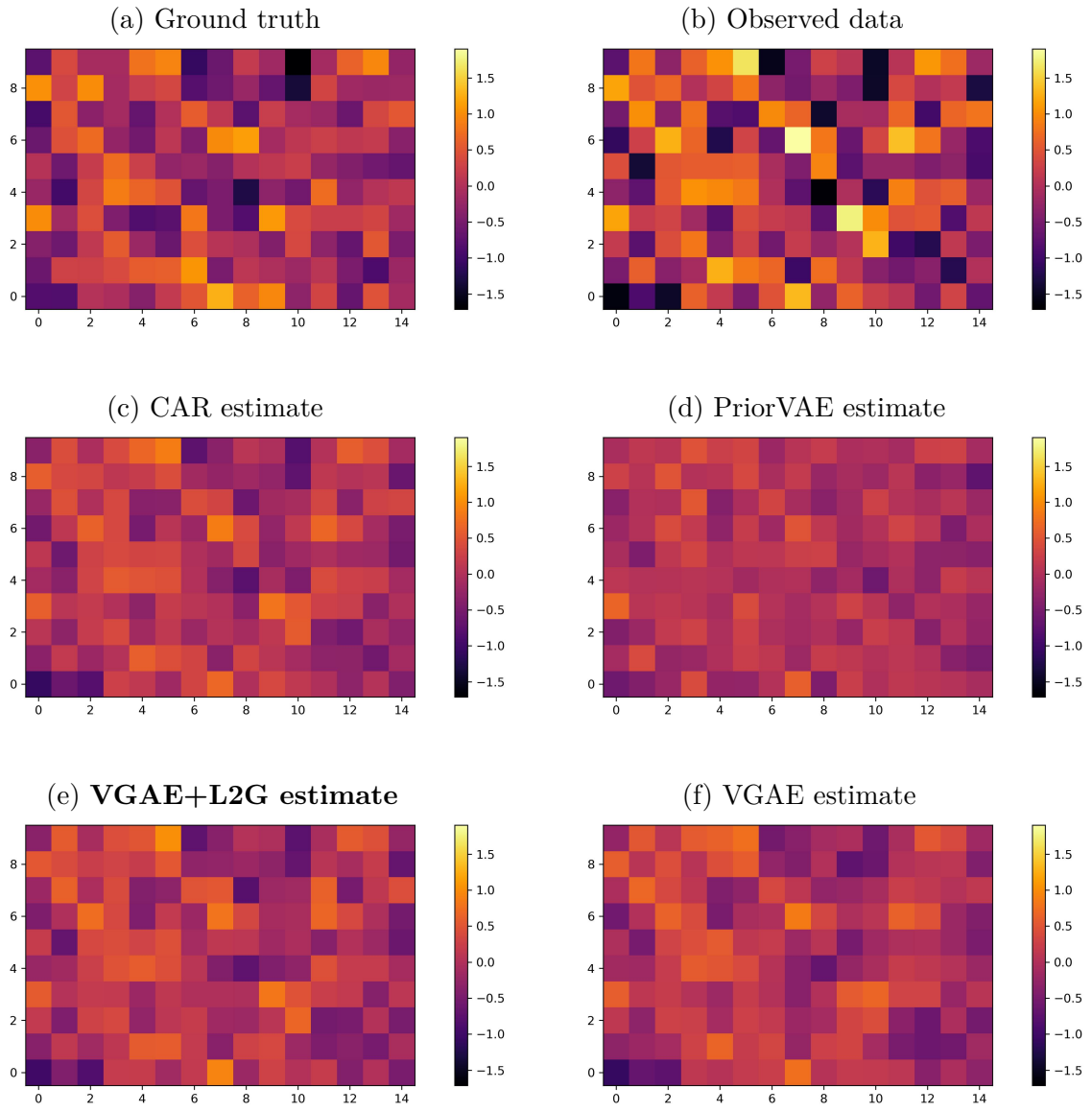


Figure 4.8: Inference results on the synthetic CAR example: (a) Ground truth data; (b) Observed data which is generated by adding a noise term to the ground truth; (c) CAR model estimate; (d) PriorVAE model estimate; (e) VGAE+L2G estimate which uses an MLP output layer (our method); and (f) VGAE estimate which uses a GCN output layer.

5 Discussion

In this report, we have extended the work of PriorVAE [Semenova et al., 2022] and introduced PriorVGAE, a novel graph-based architecture that allows for learning on spatial priors (such as GP and CAR) which then enables us to perform small-area estimation over arbitrary spatial structures. In particular, we have shown that leveraging GCNs in a VAE allows us to positively enhance the learning of spatial priors in a Bayesian modelling framework.

Our proposed approach has the key advantage of propagating local information, unlike the PriorVAE approach which attempts to learn the desired spatial priors whilst being entirely unaware of the underlying spatial structure. For highly complex spatial problems, the VGAE also has fewer learnable parameters as compared to a VAE since the GCN weights are shared across locations, resulting in computationally cheaper training. As a bonus addition, out-of-sample predictions now become more tractable as restarting training is relatively inexpensive as compared to PriorVAE. PriorVGAE also enjoys the same advantages as the PriorVAE. Since we are using the same clever PriorVAE trick of using the (fixed) learnt decoder after imposing a factorized Gaussian encoder, replacing the spatial GP priors with VGAE priors makes inference using MCMC super efficient (in the MCMC sense of having high acceptance rates, and in the computational sense) because the latent dimensions are uncorrelated. Furthermore, as we are using noise-free GP priors for training instead of a dataset collected in the real world, we can train and tune the model on an unlimited amount of data; and further test it on ground truth data that best reflects the problem at hand. Moreover, there are no memory issues as these priors can be generated at training time, and reproducibility of the training set can easily be handled using a predetermined RNG key.

5.1 Limitations and Future Work

Our model does, however, come with limitations despite having better performance than the PriorVAE approach which we shall discuss now.

The use of a GCN layer greatly reduces the number of learnable parameters. However, our architecture still employs MLP layers for encoding and for achieving effective local-to-global information passing at reconstruction time. Therefore, training using a VGAE can require a higher computational cost as compared to a VAE when the problem is not too complex (e.g., fitting CAR with $\alpha = 0.1$ where the spatial autocorrelation is really low). In such simple problems, we may still achieve stellar performance with only using a lightweight MLP in the encoder and decoder, the simplest of which would require a single hidden layer in the encoder, a single hidden layer in the decoder, and an output layer. In contrast, the simplest VGAE model based on our proposed architecture would require a single GCN encoder layer, a single encoder hidden layer, a single decoder hidden layer, a single GCN decoder layer and a final output layer. So in the simplest setting, the VAE approach is preferred over the VGAE. However, we should note that the (number of) layers in the MLPs used in VGAE are fixed. That is, with increasing problem complexity, only GCN layers would be added instead of more hidden layers which makes training significantly faster and cheaper than PriorVAE. Thus, for challenging spatial datasets, the VGAE approach is preferred. However, we believe that it would still be an interesting direction to consider approaches that would eliminate the use of an MLP in the VGAE completely. We have seen in Section 3.3 that using a GCN layer for the approximate posterior would result in an astronomical inference cost, so removing the MLP at the final encoder layer may not be possible. Thus, a beneficial direction is to either find a clever way to perform efficient inference using a (pure) GCN encoder or to explore new local-to-global schemes that do not use an MLP but still allow for good reconstructions of the input data.

While using the GCN layer allows for learning via propagating local information, it does not provide a mechanism for learning link-level data. For example, we are not able to incorporate data such as transmission rate of a disease between area B_i and B_j in the aggregation process which may be beneficial for learning. Such a mechanism was also not present in the PriorVAE approach. Fortunately, the architecture that we have proposed provides a framework that allows us to actually do so. This can be done by simply replacing the GCN layer with another graph neural network layer that allows for edge-level learning such as the *interaction network* [Battaglia et al., 2016] or the *graph network* [Battaglia et al., 2018]. In fact, the latter graph network is a generalization of the former and allows for graph-level learning which we think could be beneficial for doing spatial inference. For example, using the graph network on CAR priors means that it is now able to learn how to distinguish prior

realizations that have low, medium and high spatial correlations (governed by α). Note that this approach does not fit the definition of a local-to-global scheme as we have described but rather, it is a local-plus-global scheme, where we consider local and global information simultaneously.

It should again be noted that PriorVGAE as proposed can only be used for inference over the spatial structure it was trained on. This restriction was true in PriorVAE as well. As such, if we consider new areal units beyond the original structure — for example, new areas neighbouring an existing border unit, or new areas that exist because we consider a finer partition of an existing unit — then inference is not immediately possible. Instead, we have to restart the training process which can be computationally costly. However, because the model weights are shared across all locations, at least in the GCN stage, there are significantly less parameters to be learnt unlike a pure MLP approach. Therefore, a training restart using the VGAE might not be as expensive as a PriorVAE, especially for complex spatial datasets. A possible technique that avoids restarting completely is to apply a simple padding scheme on the nodes as follows. Let a, b, c, d be integers and suppose we are interested in performing inference on an $a \times b$ regular grid such that $a < c$ and $b < d$. Then we first learn spatial priors on the bigger $c \times d$ regular grid using a VGAE. And then at inference time, we can pad the desired $a \times b$ grid into a $c \times d$ grid and fit to the data. The predictions that we are interested in is then just the values that lies on the $a \times b$ grid. Observe that this padding scheme can be applied for both VGAE and the PriorVAE, although the latter might suffer from having to learn too many parameters. Essentially, inference using this padding scheme should not be any different than performing missing data interpolation like we did in Chapter 4, except that we now have a larger domain. Data fitting as in the CAR experiment in Section 4.5 can also be viewed as missing data interpolation under this scheme where the points beyond the initial grid are viewed as missing. Note that the initial spatial domain does **not** have to be a grid at all since any arbitrary connected graph can be padded to form a regular grid. We have to be careful, however, when the graph has disjoint components. In which case, padding into a *regular* grid (where all nodes are assumed to connect to neighbouring nodes) might result in poor inference performance due to unintended connections. As such, an exploration of padding schemes, especially for disjoint spatial structures, would be highly of interest as it enables inference over a larger family of spatial structures despite only needing to perform training once.

Bibliography

Uri Alon and Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. *ArXiv*, abs/2006.05205, 2021.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4509–4517, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. URL https://arxiv.org/pdf/1806.01261.pdf.

Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974. ISSN 00359246. URL http://www.jstor.org/stable/2984812.

Michael Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo, 2017.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv: Learning*, 2016.

George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially Learned Inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=B1ElR4cgg.

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2013.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. URL http://github.com/deepmind/jraph.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57:97–109, 1970.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 14(40):1303–1347, 2013.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Thomas Kipf and Max Welling. Variational Graph Auto-Encoders. *ArXiv*, abs/1611.07308, 2016.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond Pixels Using a Learned Similarity Metric. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1558–1566. JMLR.org, 2016.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Qimai Li, Zhichao Han, and Xiao-ming Wu. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11604.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017. URL http://jmlr.org/papers/v18/16-537.html.

N. Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, A. H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

Thomas P. Minka. Expectation Propagation for approximate Bayesian inference, 2013. URL https://arxiv.org/abs/1301.2294.

Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6518–6527, 2019.

J. N. K. Rao. *Small-Area Estimation*, pages 1–8. John Wiley & Sons, Ltd, 2017. ISBN 9781118445112. doi: https://doi.org/10.1002/9781118445112.stat03310.pub2.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. 2014. Appears In Proceedings of the 31st International Conference on Machine Learning (ICML), JMLR: W&CP volume 32, 2014.

Håvard Rue, Sara Martino, and Nicolas Chopin. Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(2):319–392, 2009. doi: https://doi.org/10.1111/j.1467-9868.2008.00700.x.

Oleh Rybkin, Kostas Daniilidis, and Sergey Levine. Simple and effective vae training with calibrated decoders, 2020.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham, 2018. Springer International Publishing. ISBN 978-3-319-93417-4.

Elizaveta Semenova, Yidan Xu, Adam Howes, Theo Rashid, Samir Bhatt, Swapnil Mishra, and Seth Flaxman. PriorVAE: Encoding spatial priors with variational autoencoders for small-area estimation. *J. R. Soc. Interface*, 19(191), June 2022.

William T. Stephenson, Soumya Ghosh, Tin D. Nguyen, Mikhail Yurochkin, Sameer Deshpande, and Tamara Broderick. Measuring the robustness of Gaussian processes to kernel choice . In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3308–3331. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/stephenson22a.html.

Gabriel Taubin. A signal processing approach to fair surface design. *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995.

Jakub Tomczak and Max Welling. VAE with a VampPrior. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1214–1223. PMLR, 09–11 Apr 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? *CoRR*, abs/1810.00826, 2018.

Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. Yes, but Did It Work?: Evaluating Variational Inference. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5581–5590. PMLR, 10–15 Jul 2018.