Scalable Deep Gaussian Markov Random Fields for General Graphs

Joel Oskarsson [1] **Per Sidén** [1 2] **Fredrik Lindsten** [1]

Abstract

Machine learning methods on graphs have proven useful in many applications due to their ability to handle generally structured data. The framework of Gaussian Markov Random Fields (GMRFs) provides a principled way to define Gaussian models on graphs by utilizing their sparsity structure. We propose a flexible GMRF model for general graphs built on the multi-layer structure of Deep GMRFs, originally proposed for lattice graphs only. By designing a new type of layer we enable the model to scale to large graphs. The layer is constructed to allow for efficient training using variational inference and existing software frameworks for Graph Neural Networks. For a Gaussian likelihood, close to exact Bayesian inference is available for the latent field. This allows for making predictions with accompanying uncertainty estimates. The usefulness of the proposed model is verified by experiments on a number of synthetic and real world datasets, where it compares favorably to other both Bayesian and deep learning methods.

1. Introduction

Graphs are immensely important constructs in scientific modeling. They show up as natural representations of technological networks, such as computer and electricity networks, but also of biological and social networks (Stanković et al., 2020a; 2019). As massive amounts of data are collected about these networks, the area of machine learning on graphs becomes increasingly relevant. In this field we find probabilistic graphical models, that allow for drawing statistically sound inferences about the quantities in the graph (Koller & Friedman, 2009). Another, more recent, family of graph-based models are Graph Neural Networks (GNNs) (Wu et al., 2020; Kipf & Welling, 2017). These bring the flexibility and scalability of deep learning to graph-structured data. Unifying statistically sound methods with deep learning is an important goal in the field of graph-based methods and in modern machine learning as a whole.

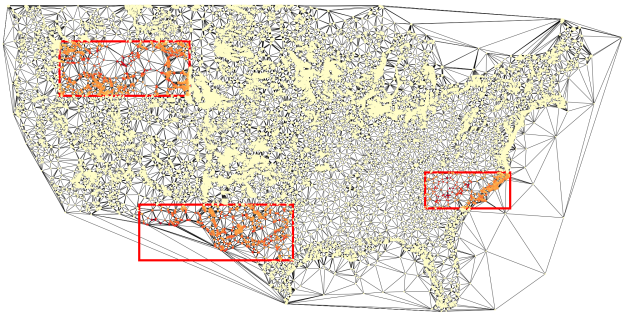


Figure 1. Predictive uncertainties over a large graph. Nodes inside the rectangles are unobserved, resulting in higher uncertainty (more red). Values are marginal standard deviations, taken from the experiment with wind speed data in section 4.4.

There is a need for methods with a strong statistical basis, but with the scalability of GNNs. While some graphs, such as those describing molecules, are comparatively small, massive graphs can emerge for example from social and traffic networks. We need efficient methods that scale also to these. In many situations it is additionally desirable not just to produce accurate predictions, but also some measure of the uncertainty about the predictions. An example of this is shown in Figure 1. We are here concerned with the node-wise regression setting, where a single graph is considered and nodes are associated with real-valued targets. We assume the full graph structure to be known. The main application of interest is prediction for a subset of unobserved nodes. Such problems can for example arise when information is missing about some individuals in a social network or due to partial outages in technological networks.

Bayesian methods provide a principled way to obtain predictive uncertainty estimates, that properly account for the uncertainty in latent variables. One type of Bayesian model that utilizes the sparsity of graphs are Gaussian Markov Random Fields (GMRFs) (Rue & Held, 2005). The Deep GMRF (DGMRF) framework of Sidén & Lindsten (2020) combines

[1]Division of Statistics and Machine Learning, Department of Computer and Information Science, Linköping University, Linköping, Sweden [2]Arriver Software AB. Correspondence to: Joel Oskarsson <joel.oskarsson@liu.se>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

GMRFs with Convolutional Neural Networks (CNNs) for the special case of image-structured data. DGMRFs can be trained efficiently and keep all useful properties of GMRFs, such as exact Bayesian inference for the latent field. In this paper we extend and generalize the DGMRF framework to the general graph setting. This requires us to design a new layer construct for DGMRFs based on local operations over node neighborhoods. Without making any assumptions on the graph structure we propose methods that allow the model training to scale to massive graphs.

Our main contributions are: 1) We extend the DGMRF framework to general graphs by designing a new type of layer construct based on GNNs. 2) We adapt the DGMRF training to this new setting, making use of an improved variational distribution. 3) We propose scalable methods for performing the log-determinant computations central to the training. 4) We demonstrate properties of the resulting model on synthetic data. 5) We experiment on multiple real-world datasets, for which our model outperforms existing methods.

2. Background

2.1. GMRFs

In graphical models a set of random variables are associated with the nodes of a graph (Koller & Friedman, 2009; Bishop, 2006). GMRFs are undirected graphical models where the nodes jointly follow a Gaussian distribution. More specifically, let $\mathcal{G}$ be an undirected graph with N nodes concatenated in the random vector $\boldsymbol{x} \in \mathbb{R}^N$. We say that $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, Q^{-1})$ is a GMRF with mean $\boldsymbol{\mu}$ and precision matrix Q w.r.t. the graph $\mathcal{G}$ iff $Q_{i,j} \neq 0 \Leftrightarrow j \in n(i), \forall i \neq j$, where $n(i)$ is the exclusive neighborhood of node i ($i \notin n(i)$). A GMRF is thus a multivariate Gaussian with a precision matrix as sparse as the graph. Note however that the covariance matrix can still be fully dense, enabling dependencies between all nodes in the graph.

Consider now the common situation where we observe $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 I)$. A GMRF prior on $\boldsymbol{x}$ is conjugate to this Gaussian likelihood. We will mainly consider the application of GMRFs to problems where $\boldsymbol{y}$ is observed only for some nodes. Let $\boldsymbol{m} \in \{0,1\}^N$ be a mask vector with ones in positions corresponding to the observed nodes, $\boldsymbol{y}_m = \boldsymbol{y} \odot \boldsymbol{m}$ and $I_m = \mathrm{diag}(\boldsymbol{m})$. The posterior for $\boldsymbol{x}$ in this setting is then given by $\boldsymbol{x}|\boldsymbol{y}_m \sim \mathcal{N}\left(\tilde{\boldsymbol{\mu}}, \tilde{Q}^{-1}\right)$, where

$$\tilde{Q} = Q + \frac{1}{\sigma^2} I_m \tag{1a}$$

$$\tilde{\boldsymbol{\mu}} = \tilde{Q}^{-1}\left(Q\boldsymbol{\mu} + \frac{1}{\sigma^2}\boldsymbol{y}_m\right). \tag{1b}$$

While the posterior is analytically tractable, and again a GMRF, computing the involved entities explicitly can be a significant computational challenge for large N.

2.2. DGMRFs

Sidén & Lindsten (2020) note that for an affine map $g\colon \mathbb{R}^N \to \mathbb{R}^N$ a GMRF $\boldsymbol{x}$ can be defined by

$$\boldsymbol{z} = g(\boldsymbol{x}) = G\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, I) \tag{2}$$

where $G \in \mathbb{R}^{N \times N}$ is a matrix and $\boldsymbol{b}$ some offset vector. This results in a GMRF with mean $\boldsymbol{\mu} = -G^{-1}\boldsymbol{b}$ and precision matrix $Q = G^\intercal G$. Note how the direction of the mapping in Eq. 2 makes $\boldsymbol{x}$ implicitly defined, a different setup from other generative models mapping Gaussian noise to data. The affine map g can in turn be defined as a combination of L simpler layers as $g = g^{(L)} \circ g^{(L-1)} \circ \cdots \circ g^{(1)}$, adding the depth to the Deep GMRF. The value of considering the layers separately is that they can be implemented implicitly, using some operation that is known to be affine. Thus, multiple such operations can be chained without performing the expensive matrix multiplications to create G.

Sidén & Lindsten (2020) consider the special case where the entries of $\boldsymbol{x}$ are associated with pixels in an image. They then define each $g^{(l)}$ as a 2-dimensional convolution with a filter containing trainable parameters. Such a DGMRF is a GMRF w.r.t. a lattice graph (Rue & Held, 2005), a graph where each pixel is connected to neighboring pixels within a window determined by the filter size. The resulting model shares much of its structure with CNNs. This allows for utilizing existing deep learning frameworks for efficient convolution computations, automatic differentiation, and GPU support.

After observing some data $\boldsymbol{y}_m$ inference for the latent field $\boldsymbol{x}$ follows from Eq. 1. To avoid inverting $\tilde{Q}$, the posterior mean $\tilde{\boldsymbol{\mu}}$ can be computed using the Conjugate Gradient (CG) method (Hestenes & Stiefel, 1952; Shewchuk, 1994). Often, also the posterior marginal variances $\mathrm{Var}[x_i|\boldsymbol{y}_m]$ are of interest. To avoid computing the covariance matrix explicitly, an alternative is to use a Monte Carlo estimate based on a set of samples from the posterior. It is possible to use the CG method also for efficiently drawing posterior samples (Papandreou & Yuille, 2010).

3. DGMRFs on Graphs

We extend the DGMRF framework of Sidén & Lindsten (2020) to general graphs, removing the assumption of a lattice graph to match the general definition of a GMRF. To achieve this we design a new type of layer $g^{(l)}$ without any assumptions on the graph structure. We then propose a way to train this new type of DGMRF using scalable log-determinant computations and a new, more flexible, variational distribution. An overview of our model is shown in Figure 2.

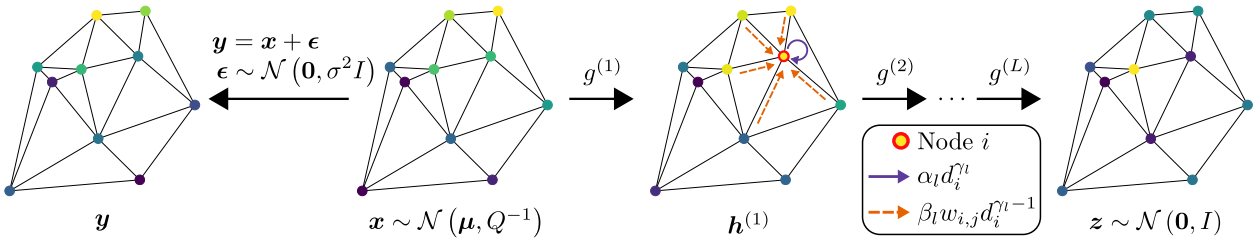


Figure 2. Overview of the graph DGMRF model. The latent field $\boldsymbol{x}$ is transformed to $\boldsymbol{z}$ through L affine maps $g^{(1)}, \ldots, g^{(L)}$. The data $\boldsymbol{y}$ is a noisy observation of $\boldsymbol{x}$. In $\boldsymbol{h}^{(1)}$ we illustrate Eq. 5 for a single node i. The node itself is weighted with $\alpha_l d_i^{\gamma_l}$ (solid purple arrow). Each node j in the neighborhood is weighted with $\beta_l w_{i,j} d_i^{\gamma_l - 1}$ (dashed orange arrows). The value at node i after the layer is the sum of all these contributions plus the bias term b_l.

Let $\mathcal{G}$ be an undirected connected graph with N nodes and adjacency matrix $A \in \mathbb{R}^{N \times N}$. In general we consider weighted graphs with $A_{i,j} = w_{i,j} \mathbb{I}_{\{j \in n(i)\}}$, where $w_{i,j} = w_{j,i} > 0$ is the weight of the edge between nodes i and j. For unweighted graphs $w_{i,j} = 1 \, \forall i, j$. We denote the degree of node i as $d_i = \sum_j A_{i,j}$ $(= |n(i)|$ in the unweighted case) and arrange all node degrees in the degree matrix $D = \text{diag}([d_1, d_2, \ldots, d_N]^\intercal)$.

3.1. Graph layer

Generalizing the CNN-based DGMRF layers of Sidén & Lindsten (2020) to general graphs requires some special considerations. It is integral to take into account the varying node degrees in the graph. To achieve this we look to the GNN framework and consider a linear version of a message passing neural network (Gilmer et al., 2017). Let $\boldsymbol{h}^{(l)} \in \mathbb{R}^N$ be the node values after layer l, with $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ and $\boldsymbol{h}^{(L)} = \boldsymbol{z}$. An intuitive way to define the operation of $g^{(l)}$ would be to sum over the node neighborhood and weight the center node by its degree,

$$h_i^{(l)} = b_l + \alpha_l d_i h_i^{(l-1)} + \beta_l \sum_{j \in n(i)} h_j^{(l-1)}, \tag{3}$$

where α_l, β_l and b_l are layer-specific trainable parameters. This operation can be viewed as a parametrized version of the graph Laplacian (Stanković et al., 2020a), a central construct in graph-based machine learning (Stanković et al., 2020b; Kipf & Welling, 2017). As a special case of Eq. 3 we also find the commonly used Intrinsic GMRF (IGMRF) model (Rue & Held, 2005). A limitation of Eq. 3 is however that there are no parameter values that reduce the layer to an identity mapping. If the model would consist of a single layer, this would not have been an issue. However, when stacking multiple layers in a deep architecture this inability introduces undesirable restrictions, in the sense that the range of attainable models is not strictly increasing as we add more layers. To avoid this shortcoming we also consider an alternative way to define $g^{(l)}$ by instead taking the mean over the neighborhood,

$$h_i^{(l)} = b_l + \alpha_l h_i^{(l-1)} + \beta_l \frac{1}{d_i} \sum_{j \in n(i)} h_j^{(l-1)}. \tag{4}$$

Unlike Eq. 3, this operation includes the identity mapping as a special case.

Finally, we propose a layer structure that generalizes these two ideas, making it possible for the model to learn which is the better choice for the data at hand. Our proposed layer is defined by

$$h_i^{(l)} = b_l + \alpha_l d_i^{\gamma_l} h_i^{(l-1)} + \beta_l d_i^{\gamma_l - 1} \sum_{j \in n(i)} w_{i,j} h_j^{(l-1)}, \tag{5}$$

where we introduce another parameter $\gamma_l \in]0, 1[$ and the optional edge weights $w_{i,j}$. Eq. 3 and 4 are special cases as γ_l tends to its limits. We empirically verify the usefulness of this layer construct in Appendix A.1. Note also that when $\alpha_l = 1$, $\beta_l = 0$, $b_l = 0$ and $\gamma_l \to 0$ Eq. 5 reduces to an identity mapping. Another motivation for this specific layer construct is that it will enable scalable methods for computing the log-determinant of G, as will be explained in section 3.3. We additionaly reparametrize the model so that $\alpha_l > 0$ and $|\beta_l| < |\alpha_l|$. This avoids degenerate, indefinite solutions (see Appendix B for details) and will enable our most scalable method for log-determinant computations.

If we consider the entire vector $\boldsymbol{h}^{(l)}$, the layer corresponds to $\boldsymbol{h}^{(l)} = g^{(l)}(\boldsymbol{h}^{(l-1)}) = G^{(l)} \boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$ with

$$G^{(l)} = \alpha_l D^{\gamma_l} + \beta_l D^{\gamma_l - 1} A \tag{6}$$

and $\boldsymbol{b}^{(l)} = b_l \boldsymbol{1}$, where $\boldsymbol{1} \in \mathbb{R}^N$ is a vector with all ones. As the operation $g^{(l)}$ corresponds to a layer of a GNN we can rely on existing software libraries for the model implementation. Such libraries come with a number of useful properties, most importantly automatic differentiation and GPU-acceleration (Fey & Lenssen, 2019; Grattarola & Alippi, 2020).

3.2. Variational training

The parameters of a DGMRF can be trained by maximizing the log marginal likelihood $\log p(\boldsymbol{y}_m|\theta)$. This is however often infeasible as it requires computing the determinant of the posterior precision matrix $\tilde{Q}$ (Sidén & Lindsten, 2020). For large N one can instead resort to variational inference, maximizing the Evidence Lower Bound (ELBO),

$$\begin{aligned}\text{ELBO}(\theta,\phi) &= \mathbb{E}_{q(\boldsymbol{x}|\phi)}[\log p(\boldsymbol{y}_m,\boldsymbol{x}|\theta)] + \text{H}[q(\boldsymbol{x}|\phi)] \\ &\leq \log p(\boldsymbol{y}_m|\theta)\end{aligned} \tag{7}$$

where q is a variational distribution with parameters ϕ and $\text{H}[\cdot]$ refers to differential entropy. For a DGMRF with a Gaussian likelihood the first term of the ELBO is

$$\begin{aligned}&\mathbb{E}_{q(\boldsymbol{x}|\phi)}[\log p(\boldsymbol{y}_m,\boldsymbol{x}|\theta)] = \\ &-\frac{1}{2}\mathbb{E}_{q(\boldsymbol{x}|\boldsymbol{\phi})}\left[g(\boldsymbol{x})^\intercal g(\boldsymbol{x}) + \frac{1}{\sigma^2}(\boldsymbol{y}_m-\boldsymbol{x})^\intercal I_m(\boldsymbol{y}_m-\boldsymbol{x})\right] \\ &+\log|\det(G)| - M\log\sigma + \text{const.}\end{aligned} \tag{8}$$

where $M = \sum_{i=1}^N m_i$ is the number of observed nodes. The expectation in Eq. 8 can be estimated using a set of samples drawn from q. As $G = G^{(L)}G^{(L-1)}\dots G^{(1)}$, the log-(absolute)-determinant is given by

$$\log|\det(G)| = \sum_{l=1}^{L}\log\left|\det\left(G^{(l)}\right)\right|. \tag{9}$$

Computing this efficiently is one of the major challenges with the general graph setting, as will be discussed further in section 3.3.

The full set of model parameters θ are the trainable parameters of each layer and the noise standard deviation σ. Maximizing the ELBO w.r.t. θ and ϕ can be done using gradient-based stochastic optimization.

3.2.1. Variational Distribution

A natural and useful way to choose the variational distribution is as another Gaussian $q(\boldsymbol{x}|\phi) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\nu}, SS^\intercal)$. This corresponds to defining q by another affine transformation in the opposite direction of the DGMRF,

$$\boldsymbol{x} = S\boldsymbol{r} + \boldsymbol{\nu}, \quad \boldsymbol{r}\sim\mathcal{N}(\boldsymbol{0}, I). \tag{10}$$

Note the difference to Eq. 2 as we here parametrize the covariance matrix instead of the precision matrix. This parametrization additionally allows for computing gradients through the sampling process, by the use of the reparametrization trick (Kingma & Welling, 2014).

Sidén & Lindsten (2020) use a simple mean field approximation with a diagonal S, making all components of $\boldsymbol{x}$ independent (Bishop, 2006). However, we propose a more flexible q by choosing

$$S = \text{diag}(\boldsymbol{\xi})\tilde{G}\,\text{diag}(\boldsymbol{\tau}) \tag{11}$$

where $\boldsymbol{\xi}, \boldsymbol{\tau} \in \mathbb{R}^N$ are vectors containing positive parameters and $\tilde{G}$ is defined in the same way as the DGMRF layer in Eq. 6. Including the matrix $\tilde{G}$ in S introduces off-diagonal elements in the covariance matrix of q, alleviating the independence assumption between nodes. Multiple such layers can also be used, introducing longer dependencies between nodes in the graph. The full set of variational parameters ϕ is then $\boldsymbol{\nu}, \boldsymbol{\xi}, \boldsymbol{\tau}$ and all trainable parameters from the layer(s) $\tilde{G}$. In Appendix A.2 we empirically show that DGMRFs trained using our more flexible variational distribution consistently outperforms those trained using the simple mean field approximation.

With this choice of S the entropy term of the ELBO is

$$\begin{aligned}\text{H}[q(\boldsymbol{x}|\phi)] &= \log|\det(S)| + \text{const.} \\ &= \log\left|\det\left(\tilde{G}\right)\right| + \sum_{i=1}^{N}\log\xi_i + \log\tau_i + \text{const.}\end{aligned} \tag{12}$$

Re-using the DGMRF layer construct in q has the added benefit that the techniques we develop for the log-determinant computation readily extend also to computing $\text{H}[q(\boldsymbol{x}|\phi)]$.

3.3. Computing the log-determinant

Computing the necessary log-determinants in Eq. 9 and 12 efficiently is a major challenge with the general graph setting. The CNN-based DGMRF was defined on a lattice graph, which creates a special structure in G and allows for finding efficient closed-form expressions for the log-determinants (Sidén & Lindsten, 2020). As we do not make any such assumptions on the graph structure we here propose new scalable methods to compute the log-determinants.

3.3.1. Eigenvalue method

One way to compute the log-determinant is based on the eigenvalues of the matrix. As the determinant is given by the product of all eigenvalues,

$$\begin{aligned}\log\left|\det\left(G^{(l)}\right)\right| &= \log|\det(D^{\gamma_l})| \\ &\quad + \log\left|\det\left(\alpha_l I + \beta_l D^{-1}A\right)\right| \\ &= \sum_{i=1}^{N}\gamma_l\log(d_i) + \log|\lambda_i|\end{aligned} \tag{13}$$

where $\{\lambda_i\}_{i=1}^N$ are the eigenvalues of $\alpha_l I + \beta_l D^{-1}A$. It can be shown[1] that $\lambda_i = \alpha_l + \beta_l\lambda_i'$ with λ_i' being the i:th

[1]For an eigenvector $\boldsymbol{v}_i$ of $D^{-1}A$ with eigenvalue λ_i', $D^{-1}A\boldsymbol{v}_i = \lambda_i'\boldsymbol{v}_i \Rightarrow (\alpha_l I + \beta_l D^{-1}A)\boldsymbol{v}_i = (\alpha_l + \beta_l\lambda_i')\boldsymbol{v}_i$.

eigenvalue of the matrix $D^{-1}A$. Since $D^{-1}A$ only depends on the graph structure, it does not change during training. This means that we can compute the eigenvalues $\{\lambda_i'\}_{i=1}^N$ as a pre-processing step and store these for computing Eq. 13 during training. Note that this is enabled by the specific layer construct that we propose. By choosing our layer as described in section 3.1 we manage to shift the main computational burden of log-determinant computations from the iterative training to a pre-processing step that only has to be performed once.

This method is feasible for moderate N, but computing all eigenvalues in pre-processing can be unreasonably slow for very large graphs. To guarantee that the training scales to massive graphs we propose an alternative method for the log-determinant computation.

3.3.2. Power series method

Another way to rewrite the log-determinant is

$$\begin{aligned}\log\left|\det\left(G^{(l)}\right)\right| =& N\log(\alpha_l) + \sum_{i=1}^{N}\gamma_l\log(d_i) \\ &+ \log\left|\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right|\end{aligned} \tag{14}$$

where we define $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. For computing the last term in Eq. 14 we follow the approach of Behrmann et al. (2019), where an approximation of the log-determinant is constructed based on a power series,

$$\log\left|\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right| = \sum_{k=1}^{\infty} -\frac{1}{k}\left(-\frac{\beta_l}{\alpha_l}\right)^k \mathrm{Tr}\left(\tilde{A}^k\right). \tag{15}$$

During training of the model we truncate the series at some large value $k = K$, resulting in an approximation. We note that $\mathrm{Tr}\left(\tilde{A}^k\right)$ only depends on the graph structure, not the model parameters. This means that the traces for $k \in \{1, \ldots, K\}$ can be computed as a pre-processing step. In Appendix C we give further details on this pre-processing step, show that the series converges and bound the truncation error.

3.4. Scalability

The graph DGMRF is defined using the graph $\mathcal{G}$, but it can be noted that the model is in fact not a GMRF w.r.t. this graph. Instead, an L-layer DGMRF is a GMRF w.r.t. the $2L$-hop graph $\mathcal{G}^{2L}$, which is defined by connecting all nodes that are at a distance $\leq 2L$ from each other in $\mathcal{G}$ (ignoring edge weights). Equivalently, the resulting precision matrix Q shares its zero-pattern with $(A+I)^{2L}$.[2]

We consider now how the model training scales w.r.t. the number of layers L and the graph structure. With the proposed methods for the log-determinants these can be computed in $\mathcal{O}(NL)$. The computational complexity is instead dominated by the application of G, as this operation scales with the number of edges in the graph. To simplify the discussion, assume $d_i = d\ \forall i$, resulting in a graph with $\mathcal{O}(Nd)$ edges. Applying L DGMRF layers to this graph is then $\mathcal{O}(NdL)$. This can be compared to explicitly creating $\mathcal{G}^{2L}$ in a pre-processing step, resulting in a graph with $\mathcal{O}\left(Nd^{2L}\right)$ edges. The layer structure is thus central to the scalability of the model. Using multiple layers additionally adds flexibility in the form of more parameters and longer node dependencies in the mapping g.

4. Experiments

The graph DGMRF was implemented[3] using PyTorch and the PyTorch Geometric GNN library (Fey & Lenssen, 2019). We evaluate the proposed model on a number of synthetic and real world datasets. All DGMRF training is repeated for 5 different random seeds and the DGMRF results reported as the mean across these. Details on the datasets and setup of each experiment are described in Appendix E.

We consider graphs where only a fraction of the nodes are observed. The model parameters are trained by maximizing the ELBO using the observed nodes. We generally treat 50% of nodes as unobserved, chosen uniformly at random. Unless stated otherwise we use the eigenvalue method for the log-determinant computations, as this is feasible for the majority of the considered graphs. While the power series method could also be used in these cases, this would incur unnecessary approximations. The trained model is then used to perform inference according to Eq. 1, utilizing the CG method. We do not compute the full posterior covariance matrix, but instead draw 100 posterior samples to estimate the marginal variance of each node. Based on the posterior mean and marginal variances different evaluation metrics can then be computed for the unobserved nodes. Using a consumer-grade GPU the full pre-processing, training, and inference procedure for a single DGMRF model takes around 10 minutes for smaller graphs and up to one hour for the largest and most dense graphs.

[2] Note that $G^{(l)}$ has zeros in the same positions as $I + A$. Since $Q = G^{(1)\intercal}\ldots G^{(L)\intercal}G^{(L)}\ldots G^{(1)}$ it has the same zero-pattern as $(I+A)^{2L} = \sum_{k=0}^{2L}\binom{2L}{k}A^k$. The value of $[A^k]_{i,j}$ is the number of graph walks between nodes i and j of length k. So if i and j are at a distance $k' \leq 2L$ in $\mathcal{G}$, then $[A^{k'}]_{i,j} \neq 0$ and $Q_{i,j} \neq 0$. This implies that Q defines a GMRF w.r.t. the $2L$-hop graph.

[3] Our code is available at https://github.com/joeloskarsson/graph-dgmrf.

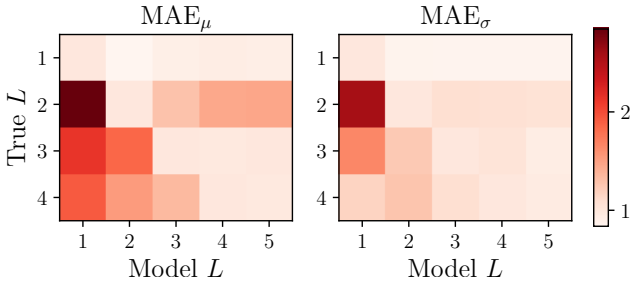


Figure 3. MAE between posterior means (left) and between posterior standard deviations (right). The difference is computed between the true posterior of $\boldsymbol{x}$ and the posterior of each trained model. Rows represents synthetic data sampled from DGMRFs with different number of layers. Columns represent DGMRFs with different number of layers, trained on the synthetic data. For easy comparison each row has been divided by the error of the model with L matching the true DGMRF, which makes the diagonal equal to 1.

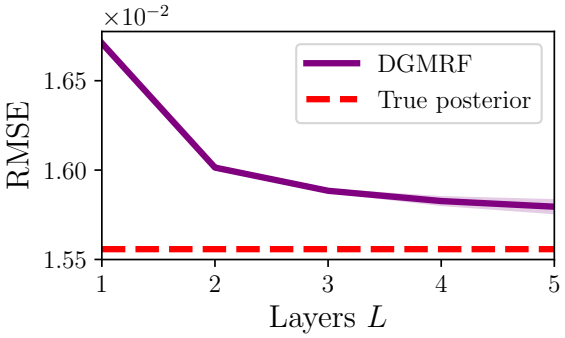


Figure 4. RMSE of DGMRF models with 1–5 layers evaluated on the Mix synthetic dataset. The small shaded area corresponds to 95% confidence interval across 5 random seeds. The RMSE for the true posterior mean is also shown.

4.1. Synthetic data

We start by considering synthetic data sampled from a model in the same class as our DGMRF. Based on a random graph with 3000 nodes we define DGMRF models with 1–4 layers and some fixed, known parameters (details in Appendix E.1). A sample of $\boldsymbol{x}$ is then drawn from each such DGMRF and Gaussian noise added to construct $\boldsymbol{y}$. For this experiments we treat 25% of the nodes as unobserved.

Using the synthetic data we train 1–5 layer DGMRF models. As the graph is reasonably small and we know the exact parameters of the true model we can here compute the true posterior. We evaluate the trained models by comparing their posteriors to the known true posterior for the unobserved nodes. The metrics used are Mean Absolute Error (MAE) between the posterior means and MAE between the posterior marginal standard deviations. Results are presented in Figure 3.

The pattern in Figure 3 illustrates how a model specified with too few layers fails to learn the true posterior, as represented by the high error in the bottom left of the figure. Without enough layers the precision matrix becomes too sparse, resulting in the model not capturing complex dependencies between nodes in the graph. On the other hand we note how the error generally does not increase much when unnecessarily many layers are used in the model. This indicates that deeper architectures do not impose unwanted restrictions on the model class.

We consider an additional synthetic graph datasets sampled from a GMRF that lies outside the model class of our DGMRF. The precision matrix of this GMRF is specified by taking a random linear combination of some other simple precision matrices (details in Appendix E.1). We refer to this as the Mix dataset. Here we compare the Root Mean Squared Error (RMSE) of the posterior mean to the value of $\boldsymbol{y}$ for the unobserved nodes. The results for 1–5 layer DGMRFs are shown in Figure 4. It is clear from the plot that DGMRFs with more layers perform better. More layers results in a more flexible model and a more dense precision matrix, better matching the data at hand. An additional experiment for synthetic data constructed from another GMRF can be found in Appendix A.3.

4.2. Wikipedia graphs

Next, we conduct experiments with three graphs based on Wikipedia pages related to different animals (Rozemberczki et al., 2021). The three graphs vary in sparsity and number of nodes. In these graphs the nodes represent Wikipedia pages and edges mutual links between them. The target attribute $\boldsymbol{y}$ is the log average monthly traffic of each page.

As before we evaluate model predictions by RMSE. To also take the predictive uncertainty into account we additionally consider the probabilistic metric Continuous Ranked Probability Score (CRPS) (Gneiting & Raftery, 2007). Let F_i be the cumulative distribution function of the predictive distribution for node i. The CRPS is then defined as

$$\text{CRPS}(F_i, y_i) = -\int_{-\infty}^{\infty} \left(F_i(t) - \mathbb{I}_{\{t \geq y_i\}}\right)^2 dt \qquad (16)$$

which can be interpreted as the difference between F_i and a unit step located at y_i. The integral in Eq. 16 has a closed form solution when the predictive distribution is Gaussian. We use the negative CRPS, meaning that lower values are better. The final metric is then computed as the mean negative CRPS over the unobserved nodes. For baseline models that do not directly yield a predictive distribution we instead create a crude uncertainty estimate by training an ensemble of 10 models. The mean and standard deviation across the

Table 1. Results on the three Wikipedia datasets. The best value of each metric is marked with bold and second best underlined. The last column denotes the number of trainable parameters in each model, not including variational parameters.

	CHAMELEON		SQUIRREL		CROCODILE		
MODEL	RMSE	CRPS	RMSE	CRPS	RMSE	CRPS	# PARAMETERS
GRAPH GP	2.115	1.216	1.772	1.001	2.169	1.251	4
LP	2.102	-	1.930	-	2.014	-	-
IGMRF	1.805	1.030	1.718	0.986	1.526	0.939	2
DGP ($\times 10$)	1.613	1.067	<u>1.400</u>	0.984	1.308	0.786	$\approx 10^4$
DGMRF, $L = 1$	1.589	0.883	1.643	0.924	1.311	0.704	5
DGMRF, $L = 3$	<u>1.511</u>	<u>0.835</u>	1.493	<u>0.837</u>	<u>1.228</u>	<u>0.652</u>	13
DGMRF, $L = 5$	**1.465**	**0.804**	**1.374**	**0.765**	**1.169**	**0.614**	21

ensemble is then used for computing the metrics. We denote such ensembles with ($\times 10$) in figures and tables.

In addition to the DGMRF we consider a number of baseline models. We use a regression version of Label Propagation (LP) (Zhu et al., 2003; Jia & Benson, 2020), a first order IGMRF (Rue & Held, 2005) and the Graph Gaussian Process (GP) model of Borovitskiy et al. (2021). It would be of interest to include a GNN baseline, but since we do not use node features this requires special consideration. Inspired by the CNN-based Deep Image Prior model of Ulyanov et al. (2018) we use a GNN with random noise as input. We denote this baseline as Deep Graph Prior (DGP). Detailed descriptions of the different baselines can be found in Appendix D.

The results of DGMRFs and baseline models on the Wikipedia graphs are presented in Table 1 (standard deviations in Appendix A.4). In terms of RMSE the 1-layer DGMRF performs similar or better to the baseline models. Adding more layers also seems beneficial, as the best performance is reached at $L = 5$. It can be noted that the diameters of the considered graphs are ≈ 10, meaning that a 5-layer DGMRF corresponds to an almost fully dense precision matrix. Note that our framework allows us to define such a model without ever working with that large matrix explicitly. With close to exact posterior inference the DGMRF model also gives principled uncertainty estimates, which is reflected in the favorable CRPS values.

4.3. California housing data

We here experiment on the classical California housing dataset (Kelley Pace & Barry, 1997). This dataset contains median house values of 20 640 housing blocks located in California. Based on their spatial coordinates we create a sparse graph by Delaunay triangulation (De Loera et al., 2010). We also weight the graph by the inverse distances between nodes, here showcasing the ability of our model to work with a weighted adjacency matrix. The California housing dataset additionally contains socio-economic features associated with the housing blocks. To handle these node features in the DGMRF we use an auxiliary linear model, similarly to Sidén & Lindsten (2020). The linear model is also given a Bayesian treatment with its own variational distribution trained jointly with the rest of the model.

Table 2. Results on the California housing dataset. All metrics are listed multiplied by a factor 100. When using no features only the graph or spatial coordinates are utilized.

	NO FEATURES		FEATURES	
MODEL	RMSE	CRPS	RMSE	CRPS
BAYES LR	13.319	7.521	8.872	4.834
MLP ($\times 10$)	11.086	7.915	7.094	4.525
GCN ($\times 10$)	8.760	5.683	6.837	4.273
GAT ($\times 10$)	9.166	6.049	6.788	4.348
SVGP	10.172	5.689	7.287	3.930
GRAPH GP	11.202	6.350	-	
LP	6.989	-	-	
IGMRF	6.989	3.841	-	
DGMRF, $L = 1$	<u>6.909</u>	3.665	5.894	3.078
DGMRF, $L = 2$	**6.853**	**3.651**	<u>5.810</u>	<u>3.041</u>
DGMRF, $L = 3$	**6.853**	<u>3.656</u>	**5.804**	**3.039**

For this experiment additional deep learning baselines are used. We use a standard Multilayer Perceptron (MLP) and two GNNs, Graph Convolutional Network (GCN) (Kipf & Welling, 2017) and Graph Attention Network (GAT) (Veličković et al., 2018). A Bayesian Linear Regression (LR) baseline is also included. To compare the graph-based methods to a direct spatial approach we also consider a Sparse Variational Gaussian Process (SVGP) model (Hensman et al., 2015).

In Table 2 we report results with and without using the node features. The results demonstrate how utilizing both the graph structure and node features allows our DGMRF to accurately model the data. Even without node features the probabilistic predictions of our DGMRF result in the lowest CRPS. While the main point of this experiment is to compare purely graph-based methods, it also serves as

Table 3. Results on the wind speed dataset. In *Spatial Mask* three rectangular areas were masked out, with nodes inside these treated as unobserved. For *Random Mask* half of the nodes were treated as unobserved, chosen uniformly at random.

	SPATIAL MASK		RANDOM MASK	
MODEL	RMSE	CRPS	RMSE	CRPS
BAYES LR	1.167	0.677	0.948	0.524
MLP (×10)	1.176	0.815	0.653	0.396
GCN (×10)	1.116	0.719	0.606	0.357
GAT (×10)	1.082	0.650	0.622	0.362
LP	**0.979**	-	0.311	-
IGMRF	**0.980**	**0.610**	0.311	0.160
DGMRF, $L = 1$	1.246	0.700	**0.272**	**0.123**
DGMRF, $L = 2$	1.083	0.616	**0.273**	**0.123**
DGMRF, $L = 3$	1.075	0.612	**0.272**	**0.123**

an example of how our DGMRF can be applied to spatial data. It should however be noted that a graph based on spatial positions is an approximation. If only the graph is used by the model, some spatial information is being lost. Methods working directly with the continuous coordinates can in some cases be more suitable for this type of problem, but often come with computational challenges (Hensman et al., 2015).

4.4. Wind speed data

To test the scalability of our model we experiment on a large dataset containing wind speeds. The dataset contains the average wind speed for 126 652 sites around the US, based on a state-of-the-art meteorological simulation (Draxl et al., 2015). Similar pre-processing and experiment setup is used as for the California housing data.

Due to the large size of the graph we here use the power series method for log-determinant computations. Both a random and spatial observation mask were considered. The spatial mask can be seen in Figure 1. Results for the wind speed dataset are presented in Table 3. The experiment showcases the ability of our model to scale to large graphs without sacrificing predictive performance.

4.5. Impact of percentage of observed nodes

In the next experiment we investigate how the model performance is impacted by the percentage of observed nodes. We repeat the experiment on the Crocodile Wikipedia graph with 5%–95% of the nodes observed and a 3-layer DGMRF. The RMSE of the DGMRF and a number of baseline models are presented in Figure 5. Similar plots for CRPS and for the synthetic Mix dataset can be found in Appendix A.6. From Figure 5 we note how the the DGMRF performs well at 20% observed nodes and improves somewhat as even more are observed. When only 5% of the nodes are observed the resulting RMSE is substantially higher and the model seems to become more sensitive to the random seed used.

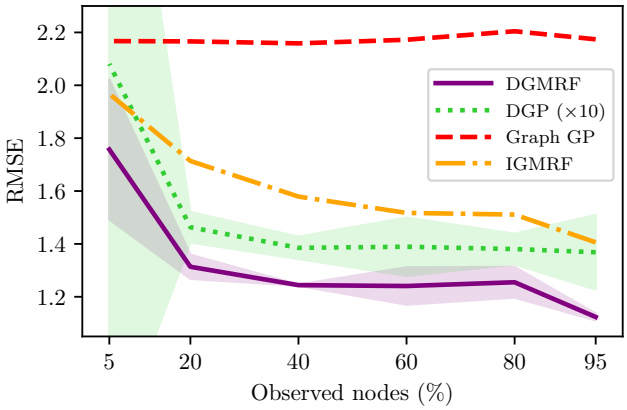


Figure 5. RMSE on the Crocodile Wikipedia graph for different percentages of observed nodes. The shaded area corresponds to 95% confidence interval, evaluated across different random seeds.

5. Related Work

Using sparse linear algebra computations, including the sparse Cholesky decomposition, GMRFs can be scaled to large graphs (Rue & Held, 2005, Section 2.3–2.4). These methods do however rely on Q being highly sparse, which is not always the case for multi-layer DGMRFs. Our GNN-based framework gives us additional benefits in the form of automatic differentiation and GPU-acceleration. Another approach to automatic differentiation for GMRFs is that of Durrande et al. (2019), where the sparse Cholesky operator is endowed with a method for reverse-mode differentiation.

GMRFs have close connections to GPs and multiple attempts have been made to also define GPs on graphs. Venkitaraman et al. (2020) define a version of GPs on comparably small graphs, but as with regular GPs scaling to large datasets is a challenge. Borovitskiy et al. (2021) propose a more scalable graph GP by making use of variational inference and stochastic optimization. Their method does however require computing eigenvalues of the graph Laplacian. For our DGMRF this can be circumvented by the power series method discussed in section 3.3.2, but this technique does not readily extend to the graph GP case. In our experiments this graph GP also seems to suffer in predictive performance due to the approximations necessary to make it scale to very large graphs. Li et al. (2020) define deep GPs (Salimbeni & Deisenroth, 2017) on graphs. They do however consider a different setting, where multiple signals are available for the same graph. GPs on graphs can also be defined as solutions to stochastic partial differential equations. This approach is used by Nikitin et al. (2022)

to construct spatio-temporal graph GPs and by Bolin et al. (2022) to define a GP on both nodes and edges of the graph. Also noteworthy is that GMRFs can be viewed as approximations to GPs in both the euclidean and graph settings (Lindgren et al., 2011; Borovitskiy et al., 2021).

GNNs are more scalable graph models based on deep learning (Wu et al., 2020; Kipf & Welling, 2017). While GNNs have proven to give accurate predictions in many settings they lack the probabilistic interpretation of GMRFs and GPs. Our model utilizes the computational benefits of GNNs without sacrificing the rigorous statistical basis of GMRFs. Attempts have also been made to combine GNNs with probabilistic approaches, for example by modeling the correlation of node residuals as jointly Gaussian (Jia & Benson, 2020).

One interpretation of DGMRFs is as a linear version of a normalizing flow model (Sidén & Lindsten, 2020; Dinh et al., 2017). Normalizing flows have been defined for graphs, but mainly for generating the graph structure (Liu et al., 2019; Kaushalya et al., 2019). Sidén & Lindsten (2020) define a non-linear version of DGMRF (a normalizing flow) in the CNN-setting, but demonstrate no convincing results on its usefulness. Preliminary experiments on a non-linear version of our graph DGMRF indicate similar poor results.

6. Conclusions

In this paper we have presented DGMRFs for general graphs. Through the computational framework of GNNs and scalable log-determinant computations the model can be applied to large graphs. In experiments the proposed model compares favorably to other both Bayesian and deep learning methods. We have only considered Gaussian likelihoods, but extensions to non-conjugate settings could be of interest (for example node classification). Other directions for future work involve better ways to make use of node features and methods for jointly learning also the graph structure.

Acknowledgements

This research is financially supported by the Swedish Research Council via the project *Handling Uncertainty in Machine Learning Systems* (contract number: 2020-04122), the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the Excellence Center at Linköping–Lund in Information Technology (ELLIIT).

References

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 573–582, 09–15 Jun 2019.

Bishop, C. M. *Pattern recognition and machine learning*. Information science and statistics. Springer, 2006. ISBN 978-0-387-31073-2.

Bolin, D., Simas, A. B., and Wallin, J. Gaussian Whittle-Matérn fields on metric graphs. *arXiv preprint arXiv:2205.06163*, 2022.

Borovitskiy, V., Azangulov, I., Terenin, A., Mostowsky, P., Deisenroth, M., and Durrande, N. Matérn Gaussian processes on graphs. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 2593–2601, 2021.

De Loera, J. A., Rambau, J., and Santos, F. *Triangulations*, volume 25 of *Algorithms and Computation in Mathematics*. Springer, 2010. ISBN 978-3-642-12970-4.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. In *International Conference on Learning Representations (ICLR)*, 2017.

Draxl, C., Clifton, A., Hodge, B.-M., and McCaa, J. The wind integration national dataset (WIND) toolkit. *Applied Energy*, 151:355–366, 2015.

Durrande, N., Adam, V., Bordeaux, L., Eleftheriadis, S., and Hensman, J. Banded matrix operators for gaussian markov models in the automatic differentiation era. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2780–2789, 2019.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272, 2017.

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

Grattarola, D. and Alippi, C. Graph neural networks in tensorflow and keras with spektral. In *ICLR Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

Hensman, J., Matthews, A. G. d. G., and Ghahramani, Z. Scalable variational gaussian process classification. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, pp. 351–360, 2015.

Hestenes, M. R. and Stiefel, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.

Jia, J. and Benson, A. R. Residual correlation in graph neural network regression. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 588–598, 2020.

Kaushalya, M., Katushiko, I., Kosuke, N., and Motoki, A. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Kelley Pace, R. and Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

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

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. Adaptive computation and machine learning. MIT Press, 2009. ISBN 978-0-262-01319-2.

Li, N., Li, W., Sun, J., Gao, Y., Jiang, Y., and Xia, S. Stochastic deep Gaussian processes over graphs. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Lindgren, F., Rue, H., and Lindström, J. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Nikitin, A. V., John, S., Solin, A., and Kaski, S. Nonseparable spatio-temporal graph kernels via SPDEs. In *International Conference on Artificial Intelligence and Statistics*, pp. 10640–10660. PMLR, 2022.

Papandreou, G. and Yuille, A. L. Gaussian sampling by local perturbations. In *Advances in Neural Information Processing Systems*, volume 23, pp. 1858–1866, 2010.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2), 2021.

Rue, H. and Held, L. *Gaussian Markov random fields: theory and applications*. Number 104 in Monographs on statistics and applied probability. Chapman & Hall/CRC, 2005. ISBN 978-1-58488-432-3.

Salimbeni, H. and Deisenroth, M. Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, 1994.

Sidén, P. and Lindsten, F. Deep Gaussian Markov random fields. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 8916–8926, 2020.

Stanković, L., Mandic, D. P., Daković, M., Kisil, I., Sejdić, E., and Constantinides, A. G. Understanding the basis of graph signal processing via an intuitive example-driven approach. *IEEE Signal Processing Magazine*, 36(6):133–145, 2019.

Stanković, L., Mandic, D., Daković, M., Brajović, M., Scalzo, B., Li, S., and Constantinides, A. G. Data analytics on graphs part I: Graphs and spectra on graphs. *Foundations and Trends in Machine Learning*, 13(1):1–157, 2020a.

Stanković, L., Mandic, D., Daković, M., Brajović, M., Scalzo, B., Li, S., and Constantinides, A. G. Data analytics on graphs part III: Machine learning on graphs, from graph topology to applications. *Foundations and Trends in Machine Learning*, 13(4):332–530, 2020b.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Venkitaraman, A., Chatterjee, S., and Handel, P. Gaussian processes over graphs. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5640–5644, 2020.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

Zhu, X., Ghahramani, Z., and Lafferty, J. D. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, pp. 912–919, 2003.

A. Additional Results

In this appendix we present additional experiments that verify properties of our model. We also present some supplementary results from the different experiments in section 4.

A.1. Trainable γ_l

In section 3.1 we propose a new DGMRF layer that generalizes Eq. 3 and 4 by introducing the γ_l-parameter. To verify the usefulness of this parametrization, we perform a number of experiments with fixed $\gamma_l = 1$ (Eq. 3), $\gamma_l = 0$ (Eq. 4), and with γ_l being trainable (Eq. 5, our proposed layer). We train and evaluate DGMRFs with 1, 3 and 5 layers on multiple datasets. To decouple the parametrization of the DGMRF layers from any layers $\tilde{G}$ used in our variational distribution we here only use a mean field approximation. Apart from this we use the same experiment setups as described in section 4 and Appendix E for each dataset.

Results from the experiments are reported in Table 4. It is clear that DGMRFs with a trainable γ_l-parameter generally models the data more accurately. In practice it can also be observed that γ_l often takes values that are not close to neither 0 nor 1, further indicating the usefulness of the added flexibility. Comparing the models with fixed γ_l, the version with $\gamma_l = 0$ generally performs better, in particular in multi-layer cases. We have observed that with $\gamma_l = 1$ the training easily becomes unstable. For the Chameleon data the training of the multi-layer models does not even converge, so these results are left out. In Table 4 it is also noteworthy that the DGMRF with $L = 3$ and $\gamma_l = 1$ corresponds to the true model for the Synthetic DGMRF data. Despite this the DGMRF with trainable γ_l ends up closer to the true posterior. We believe the reason for this is the problematic training of the $\gamma_l = 1$ models, preventing the trained DGMRF from reaching parameter values close to the true ones. On the synthetic DGMRF data the $\gamma_l = 1$ model especially seems to suffer from the independence assumptions in the mean field approximation. If trained using our more flexible variational distribution the error of this model is reduced substantially[4].

A.2. Variational distribution

To evaluate our improved variational distribution we conduct additional experiments on a few of our considered datasets. We train DGMRF models with 1, 3 and 5 layers on the synthetic 3-layer DGMRF data, the Chameleon Wikipedia data and the California housing data without features. We train each DGMRF model using three different variational distributions:

- Mean Field (MF) approximation (used by Sidén & Lindsten (2020)),
- $\tilde{G}$ corresponding to 1 DGMRF layer (ours, see Eq. 11),
- $\tilde{G}$ corresponding to 2 DGMRF layers.

We then use the standard experiment setup for each dataset. As we have seen no use for more than 3-layer models on the California housing dataset we there exclude the 5-layer DGMRF.

Table 5 and 6 list results for the different variational distributions. Our more flexible distribution shows a clear advantage over the simple mean field approximation. We also report the converged ELBO values of each model. The addition of DGMRF layers also in the variational distribution results in higher ELBOs overall. This indicates that the learned variational distribution is closer to the true posterior in terms of Kullback–Leibler divergence. The higher ELBO can also be explained by a higher log marginal likelihood for the learned model parameters, which would be in line with the improved metric values. While the use of 1 layer in the variational distribution is an improvement over the mean field approximation, there is no pronounced benefit of 2 layers. This motivates our choice to use 1 layer in remaining experiments.

A.3. Synthetic data

In section 4.1 DGMRF models were trained and evaluated on the synthetic Mix dataset. We present additional results from this experiment in Figure 6, where we evaluate the CRPS of the model predictions.

We consider another synthetic dataset to evaluate how well multi-layer DGMRFs can model data from a GMRF with a more dense precision matrix. Based on a random graph $\mathcal{G}$ we create the 3-hop graph $\mathcal{G}^3$. We then draw a sample from a 1-layer

[4]We re-trained the 3-layer model on the synthetic DGMRF data using our variational distribution including 2 DGMRF layers (with trainable γ_l). The corresponding MAE_μ is then 0.266 ± 0.016 for $\gamma_l = 1$, 0.395 ± 0.000 for $\gamma_l = 0$ and 0.149 ± 0.003 for γ_l trainable.

Table 4. Results comparing a fixed and trainable γ_l-parameter. DGMRF models with $L \in \{1, 3, 5\}$ layers have been considered. The datasets used are: synthetic data sampled from a 3-layer DGMRF (earlier described in section 4.1), the synthetic Mix dataset and the Chameleon Wikipedia data. For the first of these we compare the model to the true posterior using MAE of posterior mean and marginal standard deviations. All metrics are reported as mean±standard deviation across 5 random seeds. For readability some metrics are listed multiplied by a factor 100.

		SYNTH. DGMRF ($L = 3$)		MIX		CHAMELEON	
DGMRF	γ_l	MAE_μ ($\times 100$)	MAE_σ ($\times 100$)	RMSE ($\times 100$)	CRPS ($\times 100$)	RMSE	CRPS
$L = 1$	$\gamma_l = 1$	**0.335±0.000**	**0.355±0.004**	1.752±0.001	0.988±0.000	1.841±0.001	1.065±0.001
	$\gamma_l = 0$	0.441±0.000	0.440±0.005	1.689±0.000	0.955±0.000	**1.617±0.001**	**0.899±0.001**
	TRAINABLE	**0.335±0.000**	**0.355±0.006**	**1.687±0.000**	**0.954±0.000**	**1.617±0.001**	**0.899±0.001**
$L = 3$	$\gamma_l = 1$	0.370±0.008	0.446±0.024	1.945±0.002	1.050±0.001	-	-
	$\gamma_l = 0$	0.404±0.000	0.372±0.010	1.587±0.000	0.896±0.000	1.634±0.001	0.904±0.001
	TRAINABLE	**0.163±0.002**	**0.240±0.010**	**1.584±0.001**	**0.894±0.000**	**1.527±0.002**	**0.845±0.001**
$L = 5$	$\gamma_l = 1$	0.935±0.056	0.904±0.034	5.188±0.698	1.753±0.035	-	-
	$\gamma_l = 0$	0.414±0.000	0.384±0.005	1.581±0.000	0.892±0.000	1.584±0.001	0.861±0.000
	TRAINABLE	**0.184±0.005**	**0.229±0.002**	**1.578±0.001**	**0.890±0.001**	**1.489±0.063**	**0.813±0.039**

Table 5. Results comparing models trained using different variational distributions. DGMRF models with $L \in \{1, 3, 5\}$ layers have been trained on synthetic data sampled from a 3-layer DGMRF. We compare the models to the true posterior using MAE of posterior mean and marginal standard deviations. All values are reported as mean±standard deviation across 5 random seeds. Note that higher ELBO values are better.

		SYNTH. DGMRF ($L = 3$)		
DGMRF	**VI LAYERS**	MAE_μ ($\times 100$)	MAE_σ ($\times 100$)	ELBO
$L = 1$	0 (MF)	0.335±0.000	**0.355±0.006**	2.056±0.000
	1	**0.323±0.001**	0.366±0.007	**2.080±0.000**
	2	**0.323±0.000**	0.364±0.004	**2.080±0.000**
$L = 3$	0 (MF)	0.163±0.002	0.240±0.010	2.107±0.000
	1	0.151±0.005	**0.221±0.007**	**2.129±0.000**
	2	**0.149±0.003**	0.226±0.013	**2.129±0.000**
$L = 5$	0 (MF)	0.184±0.005	0.229±0.002	2.115±0.000
	1	**0.151±0.007**	**0.206±0.006**	2.132±0.000
	2	0.153±0.005	0.212±0.005	**2.133±0.000**

Table 6. Results comparing models trained using different variational distributions. DGMRF models with $L \in \{1, 3, 5\}$ layers have been trained on the Chameleon Wikipedia data and the California housing data (without features). All values are reported as mean±standard deviation across 5 random seeds. Note that higher ELBO values are better.

		CHAMELEON			CALIFORNIA HOUSING, NO FEATURES		
DGMRF	**VI LAYERS**	RMSE	CRPS	ELBO	RMSE ($\times 100$)	CRPS ($\times 100$)	ELBO
$L = 1$	0 (MF)	1.617±0.001	0.899±0.001	-1.026±0.001	7.074±0.001	3.743±0.001	0.438±0.000
	1	**1.589±0.000**	**0.883±0.000**	-1.000±0.000	6.909±0.000	3.665±0.000	0.488±0.000
	2	**1.589±0.001**	**0.883±0.000**	**-0.999±0.000**	**6.897±0.002**	**3.661±0.001**	**0.490±0.000**
$L = 3$	0 (MF)	1.527±0.002	0.845±0.001	-1.002±0.001	6.914±0.001	3.702±0.002	0.491±0.000
	1	**1.511±0.006**	**0.835±0.003**	**-0.981±0.008**	**6.853±0.000**	3.656±0.001	**0.500±0.000**
	2	1.515±0.014	0.837±0.007	-0.986±0.010	**6.853±0.001**	**3.654±0.001**	**0.500±0.000**
$L = 5$	0 (MF)	1.489±0.063	0.813±0.039	-0.987±0.044		-	
	1	1.465±0.033	0.804±0.023	-0.963±0.011		-	
	2	**1.453±0.020**	**0.795±0.015**	**-0.961±0.009**		-	

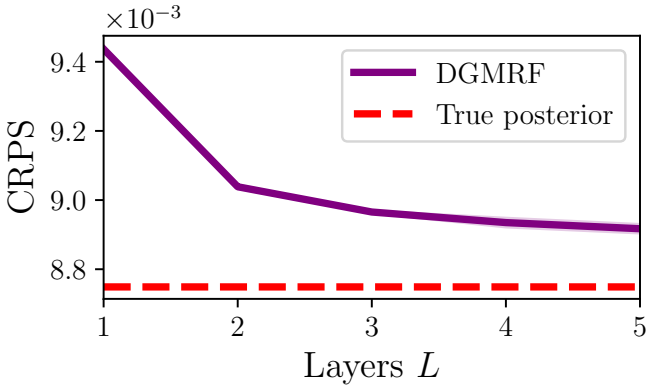
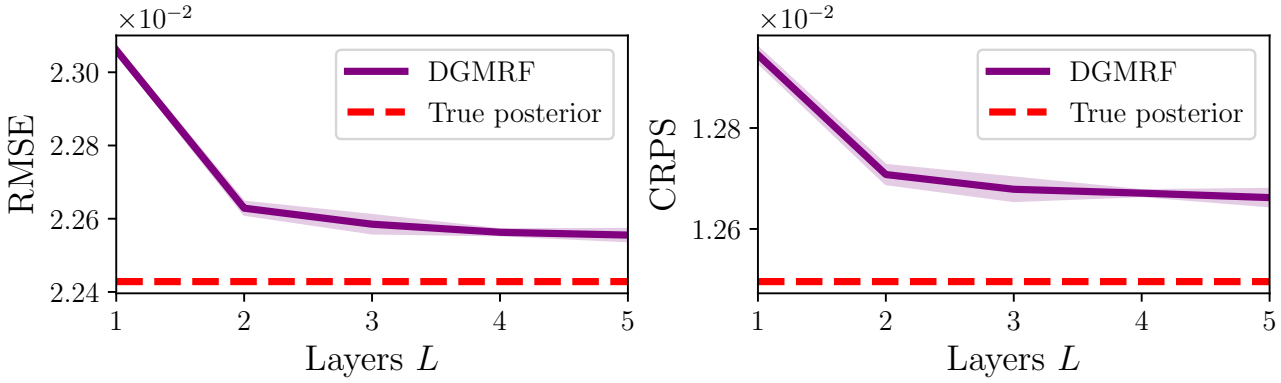


Figure 6. CRPS of DGMRF models with 1–5 layers evaluated on the Mix synthetic dataset. The shaded area corresponds to 95% confidence interval across 5 random seeds. The CRPS for the true posterior is also shown.



(a) RMSE (b) CRPS

Figure 7. RMSE and CRPS of DGMRF models with 1–5 layers evaluated on the Dense synthetic datasets. The shaded area corresponds to 95% confidence interval across 5 random seeds. The metric values for the true posterior is also shown.

DGMRF defined w.r.t. $\mathcal{G}^3$. Gaussian noise is then added and 25% of the nodes treated as unobserved. We call this the Dense synthetic dataset. Similar to the experiment on the Mix data we train 1–5 layer DGMRFs. Note that the models we train are defined on $\mathcal{G}$ rather than $\mathcal{G}^3$. The true model for the data also here lies outside the model class of DGMRFs defined on $\mathcal{G}$. Results for the Dense dataset are presented in Figure 7. With more layers the DGMRF can come close to the true posterior, even when defined on a more sparse graph. The performance of the DGMRF models clearly improves up until 3 layers, at which point the sparsity of the precision matrix matches that of the true model.

A.4. Wikipedia graphs

For the experiments on Wikipedia graphs (section 4.2) we list the standard deviations across 5 random seeds in Table 7. The random seeds only affect the model training and the set of unobserved nodes is kept fixed.

A.5. California housing and wind speed data

For the experiments on the California housing and wind speed data (section 4.3 and 4.4) we list the standard deviations across 5 random seeds in Table 8.

A.6. Impact of percentage of observed nodes

Here we present additional results for the experiment in section 4.5, where model performance for different percentages of observed nodes is investigated. Figure 8 shows for the Crocodile Wikipedia graph how the CRPS changes with the

Table 7. Standard deviations of DGMRF results for the Wikipedia experiment. Each standard deviation is computed across the results of 5 models trained using different random seeds.

	CHAMELEON		SQUIRREL		CROCODILE	
DGMRF	RMSE	CRPS	RMSE	CRPS	RMSE	CRPS
$L = 1$	0.000	0.000	0.000	0.000	0.000	0.000
$L = 3$	0.006	0.003	0.034	0.021	0.031	0.018
$L = 5$	0.033	0.023	0.022	0.013	0.030	0.019

Table 8. Standard deviations of DGMRF results for the the California housing and wind speed data. Each standard deviation is computed across the results of 5 models trained using different random seeds. Values for the California housing data are multiplied by a factor 100 to match the scale in Table 2.

	CAL., NO FEATURES		CAL., FEATURES		WIND, SPATIAL MASK		WIND, RANDOM MASK	
DGMRF	RMSE	CRPS	RMSE	CRPS	RMSE	CRPS	RMSE	CRPS
$L = 1$	0.000	0.000	0.001	0.001	0.013	0.008	0.000	0.000
$L = 2$	0.000	0.001	0.001	0.001	0.011	0.007	0.001	0.000
$L = 3$	0.000	0.001	0.001	0.001	0.017	0.011	0.001	0.000

percentage of observed nodes. The same experiment was also repeated using the synthetic Mix dataset and the results from this can be found in Figure 9.

B. Reparametrization

When parametrizing the model we want to avoid the situation where $|\alpha_l| = |\beta_l|$, as this would lead to a $G^{(l)}$ with determinant 0 and a non-definite precision matrix Q. Even if this exact situation would likely be avoided during optimization, this represents an unstable area of the parameter space as the log-determinant goes towards $-\infty$. We solve this problem by enforcing the inequality $|\alpha_l| > |\beta_l| \Leftrightarrow \left|\frac{\beta_l}{\alpha_l}\right| < 1$. We choose this direction of the inequality as it will guarantee the convergence of the power series that we use in one of our methods for the log-determinant computation (see Appendix C). It also makes intuitive sense that the central node should have a larger influence in each layer than it's neighbors. We also parametrize the model so that $\alpha_l > 0$, which is no restriction on the model class. See this for example in how Eq. 2 is invariant to the sign of the right hand side.

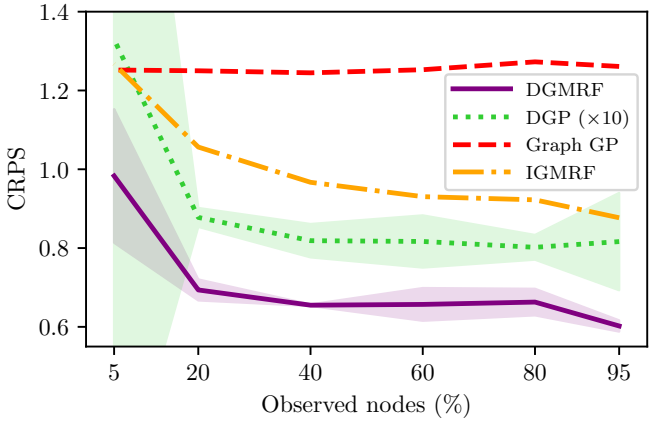
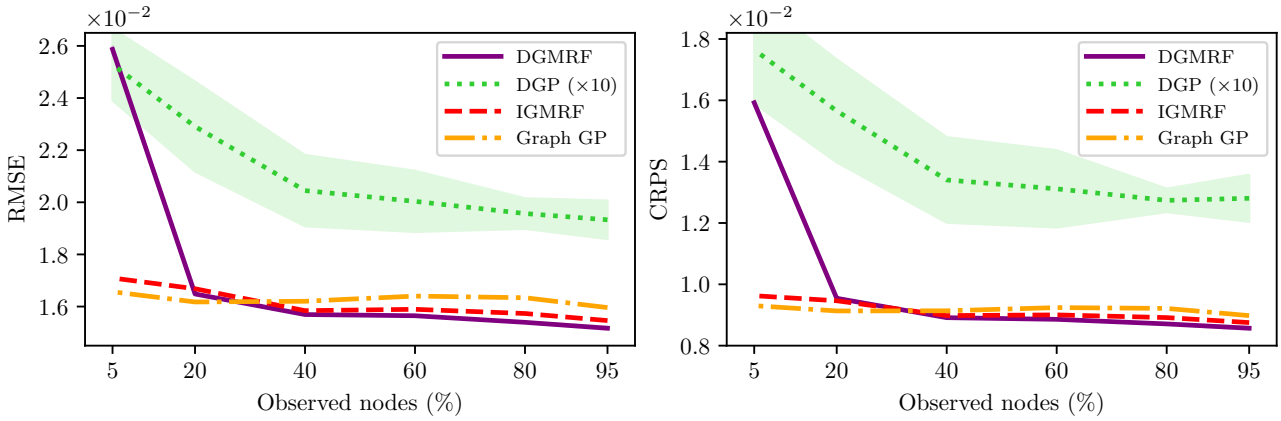


Figure 8. CRPS on the Crocodile Wikipedia graph for different percentages of observed nodes. The shaded area corresponds to 95% confidence interval, evaluated across different random seeds.



(a) RMSE (b) CRPS

Figure 9. RMSE and CRPS on the Mix synthetic graph for different percentages of observed nodes. The shaded area corresponds to 95% confidence interval, evaluated across different random seeds.

To achieve these inequalities, as well as $\gamma_l \in]0, 1[$, we reparametrize the model as

$$\alpha_l = \exp\left(\theta_1^{(l)}\right) \tag{17}$$

$$\beta_l = \alpha_l \tanh\left(\theta_2^{(l)}\right) \tag{18}$$

$$\gamma_l = \text{sigmoid}\left(\theta_3^{(l)}\right) \tag{19}$$

where $\theta_1^{(l)}, \theta_2^{(l)}, \theta_3^{(l)} \in \mathbb{R}$ are free parameters. This allows us to use unconstrained optimization while preserving the desired restrictions on $G^{(l)}$. In a similar way we reparametrize $\sigma = \exp(\theta_\sigma) > 0$ with a free parameter $\theta_\sigma \in \mathbb{R}$.

C. Details on the Power Series Log-Determinant Computation

To make our graph DGMRF scale to large graphs the computation of $\log\left|\det\left(G^{(l)}\right)\right|$ has to be both efficient and differentiable w.r.t. all layer parameters. We follow the approach of Behrmann et al. (2019), where a stochastic approximation of the log-determinant is constructed based on a power series.

We start by rewriting $G^{(l)}$ as

$$\begin{aligned} G^{(l)} &= D^{\gamma_l}(\alpha_l I + \beta_l D^{-1} A) \\ &= D^{\gamma_l - \frac{1}{2}}(\alpha_l D^{\frac{1}{2}} D^{-\frac{1}{2}} + \beta_l D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) D^{\frac{1}{2}} \\ &= \alpha_l D^{\gamma_l - \frac{1}{2}} \left(I + \frac{\beta_l}{\alpha_l} \tilde{A}\right) D^{\frac{1}{2}} \end{aligned} \tag{20}$$

where we again use the definition $\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Then

$$\begin{aligned} \log\left|\det\left(G^{(l)}\right)\right| &= \log\left(\left|\det\left(\alpha_l D^{\gamma_l - \frac{1}{2}} \left(I + \frac{\beta_l}{\alpha_l} \tilde{A}\right) D^{\frac{1}{2}}\right)\right|\right) \\ &= \log\left(\left|\alpha_l^N \det(D)^{\gamma_l - \frac{1}{2}} \det\left(I + \frac{\beta_l}{\alpha_l} \tilde{A}\right) \det(D)^{\frac{1}{2}}\right|\right) \\ &= N \log(\alpha_l) + \gamma_l \log(\det(D)) + \log\left|\det\left(I + \frac{\beta_l}{\alpha_l} \tilde{A}\right)\right| \end{aligned} \tag{21}$$

where we have used that $\alpha_l > 0$ and $\det(D) > 0$ as well as a number of properties of determinants (see for example Petersen & Pedersen (2012)). All involved quantities are simple and efficient to compute, except for $\log\left|\det\left(I + \frac{\beta_l}{\alpha_l} \tilde{A}\right)\right|$

which involves the adjacency matrix of the graph. To tackle this challenge, we will formulate this quantity as a power series and apply a stochastic approximation.

We first note a few things about the matrix $\tilde{A}$. Firstly, the eigenvalues of $\tilde{A}$ lie in $[-1, 1]$. This follows for example from the fact that the normalized graph Laplacian $I - \tilde{A}$ has eigenvalues in $[0, 2]$ (Stanković et al., 2020a; Bolla & Tusnády, 1994). Secondly, since $\tilde{A}$ is symmetric we can also conclude that its singular values are given by the absolute values of its eigenvalues[5]. In particular, the maximum singular value $\sigma_{\max}(\tilde{A}) \leq 1$.

To construct our approximation, we first show that $\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right) > 0$. Let $\tilde{\lambda}_i \in [-1, 1]$ be an eigenvalue of $\tilde{A}$ with eigenvector $\boldsymbol{w}_i$. Then

$$\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\boldsymbol{w}_i = \boldsymbol{w}_i + \frac{\beta_l}{\alpha_l}\tilde{A}\boldsymbol{w}_i = \left(1 + \frac{\beta_l}{\alpha_l}\tilde{\lambda}_i\right)\boldsymbol{w}_i, \tag{22}$$

so $\psi_i = 1 + \frac{\beta_l}{\alpha_l}\tilde{\lambda}_i$ is an eigenvalue to $I + \frac{\beta_l}{\alpha_l}\tilde{A}$. By our parametrization $\frac{\beta_l}{\alpha_l} \in]-1, 1[$, which implies that $\psi_i > 0 \, \forall i$. Hence

$$\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right) = \prod_{i=1}^{N}\psi_i > 0 \Rightarrow \log\left|\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right| = \log\left(\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right). \tag{23}$$

Similarly to Behrmann et al. (2019) we next formulate the log-determinant in terms of the power series

$$\log\left(\det\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right) = \mathrm{Tr}\left(\log\left(I + \frac{\beta_l}{\alpha_l}\tilde{A}\right)\right) = \sum_{k=1}^{\infty}(-1)^{k+1}\frac{\mathrm{Tr}\left(\left(\frac{\beta_l}{\alpha_l}\tilde{A}\right)^k\right)}{k} = \sum_{k=1}^{\infty} -\frac{1}{k}\left(-\frac{\beta_l}{\alpha_l}\right)^k \mathrm{Tr}\left(\tilde{A}^k\right). \tag{24}$$

Using the properties of $\tilde{A}$ discussed earlier we can show that

$$\left\|\frac{\beta_l}{\alpha_l}\tilde{A}\right\|_2 = \left|\frac{\beta_l}{\alpha_l}\right|\left\|\tilde{A}\right\|_2 = \left|\frac{\beta_l}{\alpha_l}\right|\sigma_{\max}\left(\tilde{A}\right) < 1 \tag{25}$$

which implies that the power series in Eq. 24 converges. We can then truncate this series at some large $k = K$ to get an approximation, as discussed in section 3.3.2.

Using the same reasoning as Behrmann et al. (2019), the truncation error of the series can be bounded. We note that

$$\left|\mathrm{Tr}\left(\tilde{A}^k\right)\right| = \left|\sum_{i=1}^{N}\tilde{\lambda}_i^k\right| \leq \sum_{i=1}^{N}\left|\tilde{\lambda}_i^k\right| \leq N \tag{26}$$

where we have used that each $\tilde{\lambda}_i^k$ is an eigenvalue to $\tilde{A}^k$ and that $\left|\tilde{\lambda}_i^k\right| \leq 1$. The absoute error E_K of the series in Eq. 24 truncated at K can then be bounded as

$$E_K \leq N\left(-\log\left(1 - \left|\frac{\beta_l}{\alpha_l}\right|\right) - \sum_{k=1}^{K}\frac{1}{k}\left|\frac{\beta_l}{\alpha_l}\right|^k\right). \tag{27}$$

We refer to Behrmann et al. (2019) for a proof of this bound. Our case is fully analogues and can be derived by replacing their Lipschitz constant by $\left|\frac{\beta_l}{\alpha_l}\right|$. For the majority of parameter values this bound decreases very quickly with K, as shown in Figure 10. It is only when $\left|\frac{\beta_l}{\alpha_l}\right|$ is close to 1 that the rate of decrease slows down.

As the traces in Eq. 24 only depend on the graph structure, their values can be pre-computed and stored for use during the model training. Computing $\mathrm{Tr}\left(\tilde{A}^k\right)$ can for example be done using sparse linear algebra, multiplying together k sparse matrices. For large values of k these matrix products do however become increasingly dense and the computation inefficient. An alternative approach is to use the fact that

$$\mathrm{Tr}\left(\tilde{A}^k\right) = \mathbb{E}_{p(\boldsymbol{u})}\left[\boldsymbol{u}^\intercal\tilde{A}^k\boldsymbol{u}\right] \tag{28}$$

[5]Let $\boldsymbol{v}_i$ and λ_i be the i:th eigenvector and eigenvalue of a symmetric matrix B. Then $B^\intercal B\boldsymbol{v}_i = B^2\boldsymbol{v}_i = \lambda_i B\boldsymbol{v}_i = \lambda_i^2\boldsymbol{v}_i$ and $\sqrt{\lambda_i^2} = |\lambda_i|$ is a singular value of B.

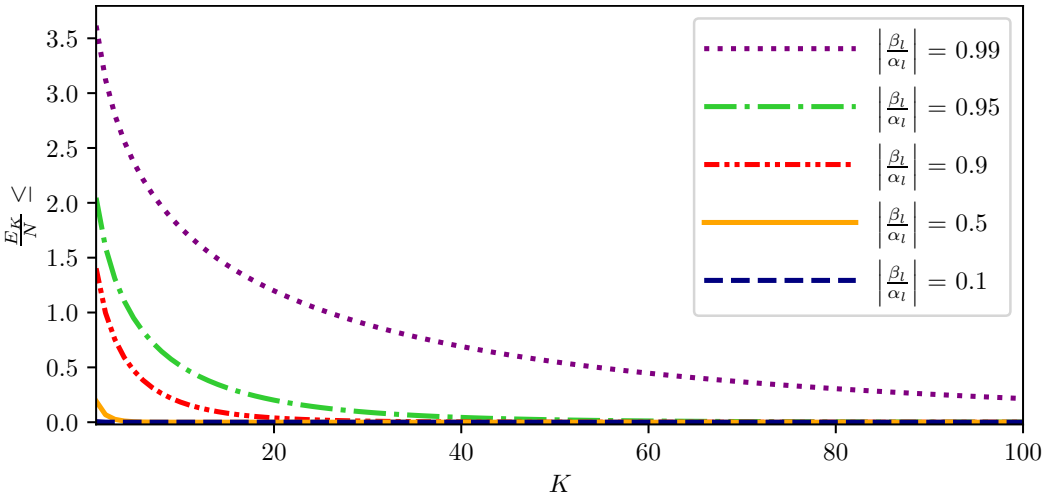


Figure 10. The quantity $-\log\left(1-\left|\frac{\beta_l}{\alpha_l}\right|\right) - \sum_{k=1}^{K} \frac{1}{k}\left|\frac{\beta_l}{\alpha_l}\right|^k$ from the bound in Eq. 27, plotted as a function of K for different parameter values.

for any distribution $p(\boldsymbol{u})$ with zero mean and covariance matrix I (Hutchinson, 1990). Taking a Monte Carlo (MC) estimate of Eq. 28 yields the Hutchinson's trace estimator. We note that while the estimator is unbiased, we here only compute its value during pre-processing and then reuse it during training. This gives an approximation of the trace, but since we do not have to re-compute it during training we can use a high value for K and many samples for the MC estimate. We choose $p(\boldsymbol{u})$ as the uniform distribution over $\{-1, 1\}^N$, resulting in a minimum variance estimator. Computing this estimate only requires applying $\tilde{A}$ to vectors sampled from $p(\boldsymbol{u})$. As the multiplication with $\tilde{A}$ corresponds to a single GNN layer we implement this using the same GNN framework as the rest of the model.

D. Baseline Models

In this appendix we describe the baseline models used in our experiments.

D.1. Label Propagation (LP)

The label propagation baseline follows the basic framework of (Zhu et al., 2003), but using real-valued labels (Jia & Benson, 2020). This approach boils down to finding a joint solution to the equation

$$y_i = \frac{1}{d_i} \sum_{j \in n(i)} y_j \tag{29}$$

for all i corresponding to unobserved nodes. Such a solution exists, but requires the inversion of a potentially very large matrix. We utilize the CG method for this.

The LP baseline does not give a predictive distribution that can be used for computing CRPS. As there is no randomness in the method we can not apply the ensemble method for uncertainty estimation here.

D.2. Intrinsic GMRF (IGMRF)

The IGMRF baseline is a GMRF with mean $\boldsymbol{\mu} = \mathbf{0}$ and precision matrix $Q = \kappa(D - A) + \epsilon I$. Here κ is a parameter of the model and ϵ a small value added to the diagonal in order to ensure positive definiteness and numerical stability. We use $\epsilon = 10^{-4}$ for the Wikipedia graphs and $\epsilon = 10^{-6}$ for all other experiments. Similarly to our DGMRF, this GMRF defines a prior over $\boldsymbol{x}$ and we then use a Gaussian likelihood with noise variance σ^2. The parameters κ and σ are optimized by maximizing the log marginal likelihood $\log p(\boldsymbol{y}_m)$. For this we use a simple grid-based optimization where $\log p(\boldsymbol{y}_m)$ is evaluated for each combination of $\sigma \in \{0.001, 0.01, 0.1, 1\}$ and 20 log-spaced values of κ in the range $[0.01, 1000]$. To

compute the log marginal likelihood we use the fact that

$$p(\boldsymbol{y}_m) = \left. \frac{p(\boldsymbol{y}_m|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{y}_m)} \right|_{\boldsymbol{x}=\boldsymbol{x}'} \tag{30}$$

holds for any $\boldsymbol{x}'$. We then utilize sparse linear algebra computations, such as the sparse Cholesky decomposition, to compute all quantities involved (Rue & Held, 2005). In contrast to our DGMRF this is feasible here since Q is very sparse.

D.3. Graph GP

For the Graph GP baseline we use the reference implementation made available[6] by the authors (Borovitskiy et al., 2021). In order to make the model scale to large graphs we use a number of approximation techniques, all described in the original paper. For training the model we use doubly stochastic variational inference with a batch size of 128 and the proposed interdomain inducing variables. The kernel matrix is approximated using the 500 smallest eigenvalues of the graph Laplacian and corresponding eigenvectors. When applicable we use the weighted adjacency matrix also for the Graph GP baseline.

D.4. Sparse Variational Gaussian Process (SVGP)

The SVGP (Hensman et al., 2015) implementation is from the GPyTorch library (Gardner et al., 2018). We use a Matérn kernel with $\nu = 3/2$, automatic relevance determination and a batch size of 256. The positions of 500 inducing points are learned jointly with the model parameters. A Gaussian variational distribution is used, where we parametrize the Cholesky factor of the covariance matrix directly. When more features than the spatial coordinates are included, the features and coordinates are concatenated and together used as kernel input.

D.5. Bayesian Linear Regression (Bayes LR)

The Bayesian linear regression model is a default Bayesian Ridge model from the scikit-learn library (Pedregosa et al., 2011). It uses a Gaussian prior on the regression coefficients and uninformative gamma priors on regularization parameters.

D.6. Multilayer Perceptron (MLP)

In comparison to other baseline models, the deep learning baselines require some tuning of the network architectures. To achieve this we reserve 20% of the training data (observed nodes) as a validation set. No such validation set is used for DGMRFs or other baseline models. The validation set is used to decide the network architecture and for regularization by early stopping.

For the MLP baseline we consider the layer configurations $(\text{number of hidden layers} \times \text{dimensionality}) \in \{1 \times 128, 1 \times 512, 2 \times 128, 2 \times 512\}$. We use an MSE loss and the Adam optimizer (Kingma & Ba, 2015) to train one model of each configuration. The layer configuration resulting in the lowest validation loss is then used for the ensemble.

D.7. GNNs (Graph Convolutional Network (GCN) and Graph Attention Network (GAT))

The GNN models are trained in the same way as the MLP, also using 20% of the observed nodes for validation. The full graph is fed to the GNN at each iteration and the MSE loss computed for all nodes designated for training. We here consider the layer configurations $(\text{number of hidden layers} \times \text{dimensionality}) \in \{1 \times 32, 1 \times 64, 3 \times 32, 3 \times 64, 3 \times 128, 5 \times 32, 5 \times 64, 5 \times 128, 7 \times 128\}$. We experiment with two GNN baselines, the Graph Convolutional Network (GCN) of Kipf & Welling (2017) and the Graph Attention Network (GAT) of Veličković et al. (2018). For the GAT model we use single-head attention.

D.8. Deep Graph Prior (DGP)

As earlier noted the DGMRF model can be related to the message passing formulation of GNNs. An interesting question arises about how much of the performance of these types of models can be attributed to any inductive biases of the GNN layer formulation. We ask, does the GNN architecture in itself restrict the model class to a set of models that are highly useful for common types of signals on graphs? Ulyanov et al. (2018) investigate a similar question for CNNs and natural images in their Deep Image Prior model. For the in-painting setting, random noise is fed to the network and the model

[6] https://github.com/spbu-math-cs/Graph-Gaussian-Processes

Table 9. Properties of the different graph datasets. The density of an undirected graph with N nodes and N_e edges is computed as $2N_e/N(N-1)$. This is equal to the fraction of possible edges present, compared to the complete graph. The (unweighted) diameter of a graph is the longest shortest path between any two nodes in the graph.

GRAPH	NODES	EDGES	DENSITY	DIAMETER	UNOBSERVED NODES
SYNTHETIC DGMRF	3 000	8 977	0.0020	32	25%
DENSE	3 000	8 974	0.0020	32	25%
MIX	5 000	14 970	0.0012	41	50%
WIKIPEDIA CHAMELEON	2 277	31 371	0.0121	11	50%
WIKIPEDIA SQUIRREL	5 201	198 353	0.0147	10	50%
WIKIPEDIA CROCODILE	11 631	170 773	0.0025	11	50%
CALIFORNIA HOUSING	20 536	61 585	2.92×10^{-4}	57	50%
WIND SPEED, SPATIAL MASK	126 652	379 933	4.74×10^{-5}	101	9.12%
WIND SPEED, RANDOM MASK	126 652	379 933	4.74×10^{-5}	101	50%

then trained on the observed pixels of a single picture. The idea is that any inductive biases in the model architecture can still steer the training to good solutions. As the model is massively overparametrized the training should not be allowed to converge, but stopped at some earlier point that still results in a good solution. We hypothesize that GNNs can exhibit similar properties as the CNNs investigated by Ulyanov et al. (2018) and implement a graph version that we denote DGP.

For the DGP model we use the same setup for training GNNs as described above. We use GCN layers, tune the layer configuration, and use early stopping in the same way as before. A single Gaussian sample is drawn and used as the model input throughout the whole training.

The DGP model allows for applying GNNs to graphs without input features. As DGP has proven useful in some of our experiments this indicates that our earlier stated hypothesis has some ground to it. Exploring this type of model and the inductive biases of GNNs in general is an interesting direction for further research.

E. Details on Experiments and Datasets

In the interest of reproducibility we present additional details about the experiment setups and datasets in this appendix. For training our DGMRF we use a learning rate of 0.01 and the Adam optimizer (Kingma & Ba, 2015) in all experiments. The model has not been observed to be sensitive to these choices so no extensive tuning has been done. Note that overfitting is not a considerable problem here. If necessary the learning rate can be tuned to make the ELBO converge, just using the training data (observed nodes). On synthetic data we train the model for 50 000 iterations, on the Wikipedia and California housing datasets 80 000 iterations (150 000 for 5-layer DGMRFs) and for the wind speed data 150 000 iterations. These numbers are large enough for the ELBO to converge and often unnecessarily high, meaning that runtimes could be slightly reduced with a more carfeful choice. In all experiments we use one DGMRF layer for $\tilde{G}$ in the variational distribution q (see Eq. 11). At each iteration of training we draw 10 samples from q to estimate the expectation in the ELBO.

An overview of the different graphs used in experiments is presented in Table 9.

E.1. Synthetic data

E.1.1. SYNTHETIC DGMRF DATA

The synthetic data used for the experiment in Figure 3 was sampled from 1–4 layer DGMRFs based on a synthetic generated graph. A graph with 3000 nodes was generated by first sampling node positions uniformly over $[0, 1]^2$ and then forming the graph using a Delaunay triangulation (De Loera et al., 2010). The positions of the nodes were only used to construct the graph and for visualization purposes. DGMRFs with parameters $\alpha_l = 1.2$, $\beta_l = -1.0$, $\gamma_l = 1$, and $b_l = 0$ for all layers were then defined on this graph. The final data $\boldsymbol{y}$ was constructed by drawing a single sample $\boldsymbol{x}$ from a DGMRF and adding Gaussian noise with $\sigma = 0.01$. Observation masks were additionally created by randomly setting 25% of nodes as unobserved.

E.1.2. DENSE

For the synthetic Dense data (discussed in Appendix A.3), a random graph was generated in a similar way as above. Based on this random graph the corresponding 3-hop graph was then created. A sample $\boldsymbol{x}$ was drawn from a 1-layer DGMRF with $\alpha_l = 1.2$, $\beta_l = -1.0$, $\gamma_l = 1$, and $b_l = 0$ defined on this 3-hop graph. Gaussian noise with $\sigma = 0.01$ was again added.

E.1.3. MIX

For the Mix data a graph with 5000 nodes was created in the same way as above. A GMRF was defined on this graph with mean $\boldsymbol{\mu} = \mathbf{0}$ and precision matrix $Q = \sum_{i=1}^{4} G_i^\intercal G_i$, where each G_i was defined as a DGMRF layer according to Eq. 6. We defined each G_i not directly on the generated graph, but on the i-hop graph $\mathcal{G}^i$ (in the same way as the Dense dataset). The parameters of each G_i were sampled randomly as $\alpha_l \sim \mathcal{U}([0.5, 1.5])$, $\beta_l \sim \mathcal{U}([-1.1, -0.1])$ and γ_l fixed to 1. The data was then sampled in the same way as above.

E.2. Wikipedia graphs

The Wikipedia graphs were created and made available[7] by Rozemberczki et al. (2021). In our experiments we do not use the accompanying node features and we use the logarithm of target values. We pre-processed the data by removing duplicate edges and self-loops (edges with the same node as both endpoints).

After tuning the DGP model the architecture used for the ensemble was 5×128 on the Crocodile data and 7×128 on the Chameleon and Squirrel data.

E.3. California housing data

The California housing dataset was loaded directly through the scikit-learn library[8] (Pedregosa et al., 2011). We used the logarithm of target values and performed the same feature transformation as described in the original paper (Kelley Pace & Barry, 1997). Outliers in the data were removed by the elliptic envelope method, determining the contamination degree by visual inspection. Features and targets were finally normalized to $[0, 1]$.

An equirectangular projection was used on the latitude and longitude of each housing block to create the node positions. Based on these positions the graph was created as a Delaunay triangulation. The graph creation was done using the built in Delaunay computation in PyTorch Geometric (Fey & Lenssen, 2019), which in turn relies on the Qhull library[9]. We note that many choices exist for how to create such a spatial graph, but the Delaunay method has in our experiments shown to work well while maintaining a highly sparse graph. Considering different algorithms for graph creation is outside the scope of this work. The graph was weighted with $w_{i,j} = (\rho_{i,j} + \epsilon)^{-1}$ based on the euclidean distance $\rho_{i,j}$ between nodes i and j. A small ϵ is included to prevent division by zero for nodes assigned to the same position.

For handling features in the DGMRF model we follow the approach of Sidén & Lindsten (2020) and use an auxiliary Bayesian linear regression model. An uninformative $\mathcal{N}(\mathbf{0}, 10^8 I)$ prior and a mean-field variational distribution are used for the coefficients of the linear model. For posterior inference the coefficients can be integrated out.

When no features are used the baseline models in Table 2 only utilize the node positions as inputs. When features are used both the node positions and the pre-processed socio-economic features are used. The Graph GP, LP and IGMRF baselines only utilize the graph and no node positions or other features. There is no obvious way to adapt these methods to also use node features, so this was deemed outside the scope of this paper.

The resulting layer configurations after tuning for the deep learning baselines are listed in Table 10 together with the number of trainable parameters. These numbers can be compared to the significantly smaller $4L + 1$ trainable parameters in an L-layer DGMRF.

[7] https://github.com/benedekrozemberczki/MUSAE

[8] https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

[9] http://www.qhull.org/

Table 10. Layer configurations (number of hidden layers $\times$ dimensionality) and corresponding number of trainable parameters of the deep learning models. Each layer configuration is the final one decided on from hyperparameter tuning on the validation data.

	CAL., NO FEATURES		CAL., FEATURES		WIND, SPATIAL MASK		WIND, RANDOM MASK	
MODEL	LAYERS	# PARAM.	LAYERS	# PARAM.	LAYERS	# PARAM.	LAYERS	# PARAM.
MLP	2×512	264 705	2×512	268 801	2×512	266 241	2×512	266 241
GCN	7×128	99 585	3×64	9 089	7×128	99 969	7×128	99 969
GAT	5×128	67 841	3×128	35 329	5×32	4 769	5×64	17 729

E.4. Wind speed data

The wind speed data originates from the Wind Integration National Dataset Toolkit[10]. The mean wind speed in the summary statistics for the different sites was used as our target. To somewhat limit ourselves to continental US we only included sites at a latitude $\geq 24.4°$. Also here an equirectangular projection was used and the graph then created in the same way as for the California housing data.

The LP and IGMRF baseline models only utilize the graph in this experiment. All other baselines (not the DGMRFs) use the node positions as features, but here expanded to second degree polynomial features in order to capture some non-linear trends. Deep learning architectures are listed in Table 10. We were not able to apply the Graph GP to this dataset as the required pre-computation of eigenvalues and eigenvectors does not scale to a graph of this size.

For the DGMRF we here use the power series method of section 3.3.2 for the log-determinant computations. We truncate the power series in Eq. 15 at $K = 50$ terms and use the Hutchinson's trace estimator with 1000 MC samples (see Appendix C).

E.5. Impact of percentage of observed nodes

In this experiment we only changed the mask specifying which nodes were observed in the Mix and Wikipedia Crocodile graphs. Such masks were created for 5%, 20%, 40%, 60%, 80% and 95% of nodes being observed. Each set of observed nodes was chosen to be a superset of the previous one, with the additional nodes sampled uniformly at random.

In this experiment we use 5 random seeds also for the DGP baseline, in order to compute confidence intervals. This means that the whole ensemble of 10 models is re-trained for each random seed, in total creating 50 models for each observation mask. To make this feasible we fix the GNN architecture used. Based on a preliminary tuning experiment we found good choices to be the 7×128 architecture for the Mix data and the 5×32 architecture for the Wikipedia Crocodile data.

Appendix References

Bolla, M. and Tusnády, G. Spectra and optimal partitions of weighted graphs. *Discrete Mathematics*, 128(1):1–20, 1994.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Hutchinson, M. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2015.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Petersen, K. B. and Pedersen, M. S. The matrix cookbook, nov 2012.

[10] https://www.nrel.gov/grid/wind-toolkit.html