Graph Convolutional Gaussian Processes

Ian Walker [1] Ben Glocker [1]

Abstract

We propose a novel Bayesian nonparametric method to learn translation-invariant relationships on non-Euclidean domains. The resulting graph convolutional Gaussian processes can be applied to problems in machine learning for which the input observations are functions with domains on general graphs. The structure of these models allows for high dimensional inputs while retaining expressibility, as is the case with convolutional neural networks. We present applications of graph convolutional Gaussian processes to images and triangular meshes, demonstrating their versatility and effectiveness, comparing favorably to existing methods, despite being relatively simple models.

1. Introduction

We present a new Gaussian process (GP) model called the graph convolutional GP (GCGP) model. This model learns translation-invariant relationships that mimic the behavior of a convolutional layer on images, but extends this property to general graphs. Since this is a GP model, the patch response function learned will be nonlinear and nonparametric, which may aid in expressibility and make more efficient use of small datasets. Additionally, since this is a Bayesian technique, it allows for the rigorous treatment of uncertainties for predictions, useful in many application domains, but difficult for existing convolutional models.

The main point of attempting to utilize general convolutions is to reduce the complexity of the model while maintaining expressibility. Indeed, standard GP models can suffer from the curse of dimensionality, making stable estimation difficult on high-dimensional inputs (Rasmussen, 2004). This work seeks to create a convolutional model for general graphs, such as images, social networks, or 3D meshes, that focuses attention on local patches to substantially reduce the dimensionality of the input in a principled manner, inspired by recent work in geometric deep learning (Bronstein et al., 2017). In this way, GP models can be applied to high-dimensional inputs that live on non-Euclidean domains. To the best of our knowledge, this is the first work to propose *spatial* graph convolutions with Gaussian processes.

We present applications of GCGPs to graphs which are Euclidean sampling grids (images) and further demonstrate the GCGP's performance when learning on non-Euclidean domains for classification tasks. We apply our method to triangular meshes and to an MNIST superpixel dataset, where each image is represented as a distinct graph. While graph convolutional GPs are shallow, though wide, the results are promising for such relatively terse models, and indicate that GCGPs can provide a simple and effective foundation for more complex models in the future.

[1]Department of Computing, Imperial College London, United Kingdom. Correspondence to: Ian Walker <ian.walker14@imperial.ac.uk>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

2. Background

A comprehensive overview of related graph kernel methods is given by Vishwanathan et al. (2010) with some recent work by Neumann et al. (2016) proposing kernels for graphs with node features and other work employing deep learning (Zhang et al., 2018; Duvenaud et al., 2015).

We focus on the task of classifying signals that live on general graphs, leaving regression tasks as a straightforward extension of the methodology. We further constrain our focus to domains which are the same size for each signal to be classified. Thus, our datasets will comprise of general graphs, which are composed of the same number of vertices. However, they do not necessarily have the same edge structures. This is an important difference to recent Graph GPs (Ng et al., 2018) and other approaches (Venkitaraman et al., 2018) operating in the spectral domain using the graph Laplacian which requires all observations to lie on the same graph (implying the same edge structure).

The dataset $\mathcal{D} = \{\mathcal{G}, \Psi, Y\}$ comprises a collection of signals Ψ with associated labels Y, which live on a corresponding domain in $\mathcal{G}$. Let $G \in \mathcal{G}$ be one such domain. This $G = \langle \mathcal{V}, \mathcal{E} \rangle$ is comprised of a set of $\mathcal{V}$ vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ edges. The corresponding signal $\psi \in \Psi$ is some function $\psi : \mathcal{V} \rightarrow \mathbb{R}^d$ on the vertices of the graph.

We wish to learn some function f which accepts ψ as input and produces an output which will be used for classification, for example passed through a sigmoidal function. It is this function f which we want to model using a Gaussian process prior. Initially, we might view the function of interest as $f : \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}$ and select a general covariance function such as a radial basis function (RBF) kernel for the GP prior on f. This produces an expressive model, but will suffer from two major drawbacks. First, it completely ignores the underlying graph structures of the inputs, which may contain valuable information for the classification task and may differ substantially between input signals. Second, as $|\mathcal{V}|$ or d increase, a kernel such as the RBF will become more difficult to estimate as it attempts to model the relationships between every input dimension and every vertex, and thus suffers from the curse of dimensionality, a problem common to many expressive models with high-dimensional inputs.

To reduce the complexity of the problem while maintaining expressibility, we will construct kernel functions that focus only on subsets of the input dimensions. These subsets will be defined to take into account the underlying structure of the input graph on which the signal is defined. In particular, we wish to model relationships between nearby vertices while ignoring distant relations, the same intuition that is the basis for convolutional models.

In the context of Gaussian processes, recent work by van der Wilk et al. (2017) provides a framework for estimating a convolutional GP for classification on Euclidean sampling grids such as images. This work relies fundamentally on additive GP models (Duvenaud et al., 2011). The basic insight of these models is that if Ω defines a set of subsets of the input dimensions, and we model our function of interest as the sum of functions on these subsets $f(x) = \sum_{\omega\in\Omega} g_\omega(\omega)$, where each g_ω defines some unique function for each ω modelled using a GP prior $g_\omega \sim \mathcal{GP}(0, k_\omega(\cdot,\cdot))$, then f has an induced GP prior with a covariance function defined by the sum of the respective covariance functions of the constituent g_ωs.

If we focus on the case where G is the same Euclidean sampling grid (image structure) for all observations and constrain Ω to consist of all n-neighborhoods on the interior of the image and g_ω to be the same g for all $\omega \in \Omega$, we obtain the convolutional Gaussian process model proposed by van der Wilk et al. (2017). To be more precise, the subsets in the convolutional GP model when applied to images are easily defined as the $m \times m$ pixel values around each pixel p. These are referred to by $\mathbf{x}^{[p]}$ in their paper with $|\mathbf{x}^{[p]}| = m^2$. Image borders can be ignored by only defining subgroups at pixels on the interior of the image, yielding $(w - m + 1) \cdot (h - m + 1)$ number of pixels p for which there is a $\mathbf{x}^{[p]} \subset \boldsymbol{x}$. We can restate this as follows: let the image be represented by the collection of random variables $\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{wh}\}$ arranged on an undirected graph with edge weights $\gamma : \mathcal{E} \to \mathbb{R}$ defined as $\gamma(e) = 1$ for all $e \in \mathcal{E}$. We can state the definition of $\mathbf{x}^{[p]}$ as:

$$\mathbf{x}^{[p]} = \{\boldsymbol{x}_j \in \boldsymbol{x} | d(\boldsymbol{x}_j, \boldsymbol{x}_p) \leq n\} \tag{1}$$

where $d : \mathcal{V}\times\mathcal{V} \to \mathbb{R}^+$ denotes the distance metric between two vertices defined as:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j) = \min_{\pi\in\mathcal{P}} \sum_{e\in\pi} \gamma(e) \tag{2}$$

where $\mathcal{P}$ is the set of all paths connecting $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and e is an edge in a given path.

If g is given a GP prior, a GP prior will be induced on f

$$\begin{aligned} g \sim \mathcal{GP}\left(0, k_g\left(\mathbf{t}, \mathbf{t}'\right)\right), \quad f(\mathbf{x}) = \sum_p g\left(\mathbf{x}^{[p]}\right), \\ \implies f \sim \mathcal{GP}\left(0, \sum_{p=1}^{P}\sum_{p'=1}^{P} k_g\left(\mathbf{x}^{[p]}, \mathbf{x}'^{[p']}\right)\right) \end{aligned} \tag{3}$$

where P is the set of pixels and $\mathbf{x}^{[p]}$ indicates the p^{th} patch of the vector $\mathbf{x}$. (In this literature, what we refer to as ψ is referred to as $\boldsymbol{x}$). Since the subsets are constrained to constitute the same neighborhoods across the image and g is constrained to be the same function on every patch, g becomes a nonlinear, nonparametric patch response function that is translation-invariant. It is this property that mimics the behavior of CNNs.

3. Graph Convolutional Gaussian Processes

We now generalize the convolutional GP model to general, non-regular graphs in the spatial domain, so that the model can be applied to a wider range of problems. As mentioned in the previous section, we restrict our focus to a set of graphs in $\mathcal{D}$ that have the same number of vertices. On an arbitrary graph, we can define the same method for grouping the random variables into subsets as set out in Eq. (1). The only difference between the settings is that now the underlying graphs do not have the same regular lattice structure as in an image grid. This is a problem because it means that there is no guarantee that $|\mathbf{x}^{[p]}|$ is the same for all p, indeed, there is no guarantee that $|\mathbf{x}^{[p]}| = |\mathbf{x}^{[p']}|$ for *any* $v, v' \in \mathcal{V}$ for an arbitrary undirected graph $\mathcal{G}$. An additive GP model could be applied to the set of subsets that are created

$$\begin{aligned} g_p \sim \mathcal{GP}\left(0, k_{g_p}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right) \forall p, \\ f(\mathbf{x}) = \sum_p g_p\left(\mathbf{x}^{[p]}\right) \end{aligned} \tag{4}$$

We would get a GP prior over f, but this would require a separate g_p for every p, and would thus not represent a terse, translation-invariant model. Hence, we need to devise a strategy to ensure that $|\mathbf{x}^{[p]}| = |\mathbf{x}^{[p']}|$ for all p, p'. One way of doing this is to throw away all p such that $|\mathbf{x}^{[p]}| \neq c$ for some c, but on a general graph this may result in throwing away almost all of the random variables. One could instead find the minimum connectivity of the graph, m, and use this to find the m-nearest neighbors for each p and then associate these neighbours, along with p itself, to p as $\mathbf{x}^{[p]}$. However, this would also mean throwing away information if the graph is sparsely connected in some areas, but densely connected in others.

To overcome this challenge, we appeal to the literature on graph convolutions (see Niepert et al. (2016), Kipf & Welling (2016) or Bronstein et al. (2017) for a review of recent work). Spatial-domain charting methods offer the most direct way forward, as they focus on producing a convolution operator, which, when applied to signals on the graph, produces a transformed output with the same dimensionality for every vertex in the graph independent of its connectedness. For a continuous manifold $\mathcal{X}$, the convolution operator can be defined as:

$$D_j(x)f = \int_{\mathcal{X}} f(x')u_j(x,x')\,dx' \qquad \forall x \in \mathcal{X}, \quad j = 1 \dots J \tag{5}$$

On a discrete graph this becomes:

$$D_j(v)f = \sum_{\mathcal{V}} f(v')u_j(v,v') \qquad \forall v \in \mathcal{V}, \quad j = 1 \dots J \tag{6}$$

where u_j is some weighting function and J plays the role of the number of bins selected manually. (In this literature, what we refer to as ψ is referred to as f). Several possible weighting functions u_j have been proposed in the literature (Bronstein et al., 2017). For example, the geodesic polar weighting function (Masci et al., 2015), which will be used in subsequent sections for demonstration purposes, is:

$$u_{j,k}(x,x') = e^{-\frac{(\rho(x')-\rho_k)^2}{2\sigma_\rho^2}} e^{-\frac{(\theta(x')-\theta_j)^2}{2\sigma_\theta^2}}, \qquad j = 1 \dots J, k = 1 \dots K \tag{7}$$

where $\rho(\cdot)$ measures the intrinsic radial distance and $\theta(\cdot)$ measures the intrinsic angular distance from x to x'. This collection of all u_{jk} forms a system of polar coordinate bins.

There are many potential forms for the weighting functions, however, such as a simpler diffusion function that solely tracks the intrinsic radial distance, anisotropic weighting functions that modify the spread of the weights in certain directions, or general mixtures of Gaussians (Monti et al., 2017). What these weighting functions all share is the use of some type of coordinates or pseudo-coordinates for the location of the vertices in the graph and some method for

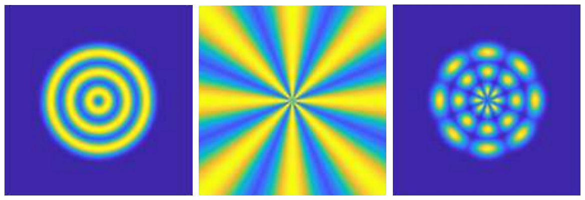

Figure 1. (R) The geodesic polar weighting functions in Eq. (7) on an Euclidean sampling grid, with the origin as x, and $J = 8, K = 3$. (L) shows the radial weights, equivalent to the diffusion weighting function. (M) shows the angular weights.

finding a distance between these coordinates that is used to create a pairwise importance weighting. How this is implemented is discussed further in the subsequent section.

The GCGP model can be applied to general graphs by appropriate selection of the weighting function u_j found in Eq. (6). Choice and design of this weighting function is one of the main research directions in geometric deep learning and some have been proposed for general graphs. For example, Monti et al. (2017) use $u_j = e^{-\frac{1}{2}(\mathbf{u}(x,y)-\mu_j)^\top \mathbf{\Sigma}_j^{-1}(\mathbf{u}(x,y)-\mu_j)}$ where, for a general graph, one can use pseudo-coordinates $\mathbf{u}(x,y) = \left(\frac{1}{\sqrt{\deg(x)}}, \frac{1}{\sqrt{\deg(y)}}\right)^\top$, where $\deg(\cdot)$ denotes the degree of a vertex. Selecting a different u_j is valid for the underlying machinery of the GP model presented here, provided that the transformation remains linear.

We can now state the GCGP model. Let $\psi, \psi' \in \mathcal{D}$ be $|\mathcal{V}| \times d$ matrices, and let their respective discrete G, G' have $|\mathcal{V}| = |\mathcal{V}'|$. A single $u_{j,k}(x,x')$ is a real value which is the weighting between two vertices, x and x', for a particular set of hyperparameter values, ρ_k and θ_j, of which there are K and J respectively. We then define the tensor U to be $(J \cdot K) \times |\mathcal{V}| \times |\mathcal{V}|$, where each element along the first dimension is a $|\mathcal{V}| \times |\mathcal{V}|$ pairwise weight matrix for a particular set of hyperparameter values. Each element of the weighting matrix is $u_{j,k}$ given some set of hyperparameters.

Since the convolution operator is a summation, we can rewrite it as a matrix of inner products we call the convolution patch matrix, Z. This will be $|\mathcal{V}| \times (JK \cdot d)$. To get a single element of the Z matrix, we take one element along the first dimension of U ($|\mathcal{V}| \times |\mathcal{V}|$), and then take one row of that ($1 \times |\mathcal{V}|$). From the signal matrix ψ ($|\mathcal{V}| \times d$), we take one column ($|\mathcal{V}| \times 1$). Taking the inner product of these gets us a 1×1, which is a single element of Z (cf. Fig. 2).

We organize the columns of Z first by dimension of the signal and then by hyperparameter values, so the first JK columns correspond to the first signal dimension, the next JK columns correspond to the second, and so on, appending d JK-sized blocks. The size of each column is $|\mathcal{V}|$.

Let $\mathbf{z}^{[i]}$ be the ith row of the matrix Z. This represents the

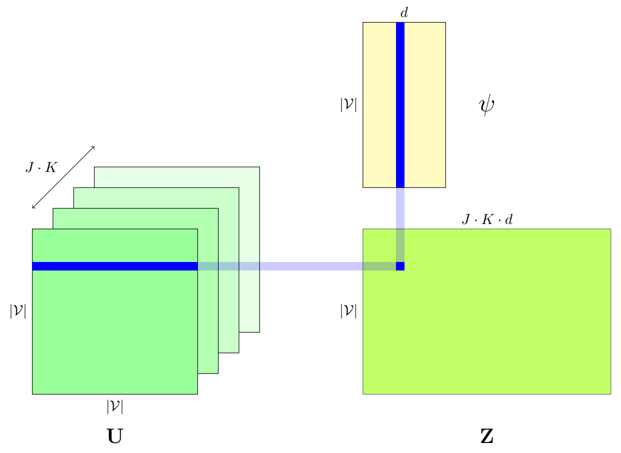

Figure 2. Diagram of the construction of patch matrix Z from signal matrix ψ and the weight tensor U.

local geodesic polar patch for vertex i of the d-dimensional signal ψ as a weighted average along each dimension at each of the JK bin locations.

The function we aim to learn is $f : \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}$, with input ψ, which will be used for classification. We can think of f as a composition of $h : \mathbb{R}^{|\mathcal{V}|\times(JKd)} \to \mathbb{R}$ and $\mathcal{Z} : \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}^{|\mathcal{V}|\times(JKd)}$, for which $\psi \mapsto Z$, thus $f(\psi) = h(\mathcal{Z}(\psi)) = h(Z)$. To reduce the dimensionality of the problem and to mimic the translation invariance property we desire, we can now model this h as the summation of a patch response function $g : \mathbb{R}^{JKd} \to \mathbb{R}$, which decomposes as the application of this g to each row of Z, i.e. $f(\psi) = h(Z) = \sum_{i\in\mathcal{V}} g(\mathbf{z}^{[i]})$. It is this g that we model using a GP prior with mean zero and some kernel function k_g, and thanks to the properties of additive GPs, this will induce a GP prior on f:

$$
\begin{aligned}
g &\sim \mathcal{GP}\left(\mathbf{0}, k_g\left(\mathbf{t}, \mathbf{t}'\right)\right) \\
\Rightarrow f &\sim \mathcal{GP}\left(\mathbf{0}, \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}'|} k_g\left(\mathbf{z}^{[i]}, \mathbf{z}'^{[j]}\right)\right)
\end{aligned} \tag{8}
$$

The k_g can be any valid kernel function, but for illustrative purposes an RBF kernel is used. However, this is an interesting choice point for the user since as the number of bins grows and the number of signal dimensions grows, this may suffer from the curse of dimensionality as well, so kernels that further decompose on the patches may become necessary. Since this formulation can be applied to signals on any graphs of the same number of vertices, and g is constrained to be the same patch response function on each extracted local patch, Eq. (8) can be thought of as a graph convolutional kernel. It is important to note that a similar kernel was proposed for use with support vector machines, but its application was limited to images (Mairal et al., 2014).

There are several benefits of our formulation. First, while there are numerous potential choices for the weighting functions, the convolution operator itself is linear in the variables that make up the input space. Since the convolution operator is a linear transformation of the collection of random variables, the machinery developed for efficient estimation of the convolutional GP can be applied in our setting as well. This efficient estimation technique relies on the use of interdomain GPs to place inducing points in the space of patches instead of in the input space (Lázaro-Gredilla & Figueiras-Vidal, 2009).

Second, the weighting function makes the assumptions about the shape of the convolution on the graph explicit. The standard convolution that we think of on an image is an indicator function that defines membership in a rectangular area the size of which is determined by the investigator. With more general weighting functions, shapes can include rings, ellipses, or more general shapes, making this another interesting choice to explore, as different problems may require different formulations even in the Euclidean domain.

Finally, because the application of the convolution operator is a transformation of the input prior to the application of a valid kernel function, the parameters of the weighting functions $u_{j,k}$ can be treated as hyperparameters of the kernel function. This is because the GCGP forms a manifold GP (Calandra et al., 2016), meaning that the hyperparameters can also be learned at training time, allowing the data to inform the shape of the convolution operator.

3.1. Estimation of GCGPs

In the experiments that follow, we estimate the GCGPs using the same estimation technique as used in van der Wilk et al. (2017). This method relies on the variational framework for approximation of GPs as proposed in Hensman et al. (2015) which has a computational complexity of $\mathcal{O}(NM^2)$, where N is the number of observations and M is the number of inducing points, and M is chosen such that $M \ll N$.

One may be concerned with the computational complexity of GP methods in general. However, as the overarching motivation is to learn effectively from few training examples, this is not a major practical problem for many interesting applications. Indeed, the often rather small size of datasets that are comprised of objects like meshes is one of the reasons why we think that Bayesian nonparametrics is a promising way forward, even more so when equipped with expressive convolutional feature learning.

4. Applications

In this section, we present applications of GCGPs to both regular domains (images) and non-regular domains (general graphs, meshes). In each application, we use an RBF kernel for the patch response function g and use the polar geodesic weighting function in the construction of the patches. The main challenge in applying GCGPs to different domains is defining the ρ and θ functions, which measure the radial and angular distance between vertices, respectively. The construction of these will be discussed in more detail in subsequent sections. All experiments were implemented using the GPFlow package (Matthews et al., 2017).

4.1. MNIST Classification

We first consider classification of the standard MNIST dataset in order to illustrate that while the proposed method is general and can handle non-Euclidean domain inputs, it can also be applied to standard datasets with regular sampling grids like images. An image from MNIST is a 28×28 sampling grid of greyscale pixel values. These pixel values play the role of the signal that lives on the vertices of the graph. We consider the graph structure of an image to be the standard eight-neighbour connectivity, with edge weights equal to 1 everywhere. We define the ρ function in Eq. (7) to be the same graph distance as in Eq. (2). We define the θ function in Eq. (7) to be the arctangent of the vertical difference between vertex v and vertex v' over the horizontal difference between the two. Defining these two functions is easy in this setting because we know the orientation of the images, and the orientation remains stable across images. The configuration of the pixels also does not change.

Our main intention here is to compare the performance of our GCGPs to the closest existing Gaussian process model in the literature, which is the convolutional GP model by van der Wilk et al. (2017). We follow the same general set-up as used in that paper. In their experiments on the MNIST dataset, they use a 5×5 patch operator, which produces a 25-dimensional patch response function, modeled using an RBF kernel. To keep comparisons as fair as possible, we use three radial bins and eight angular bins, for a total of a 24-dimensional patch response function. The input signals $\psi : \mathcal{V} \to \mathbb{R}$ are the pixel values that constitute an image. Note that the dimensionality of ψ here is one ($d = 1$) as the input is simply a greyscale image. To demonstrate the ability to learn the shape of the convolution, we treat the hyperparameters ρ_k and σ_ρ as hyperparameters of the kernel function, while leaving the θ_j and σ_θ fixed. The ρ_ks were initialized to $\{0, 1, 2\}$, such that the radial bins would be centered on a given pixel, plus the rings one and two pixels away. The σ_ρ was initialized to 1, so 68% of the weight is within one pixel distance from the centeral vertex.

During training, mini-batches of size 200 were used along with 750 inducing points and a learning rate of 0.001. The error rate converges to 1.7%. This compares favorably to the strictly translation-invariant convolutional GP, which attains an error rate of 2.1%. An RBF kernel, which models all 784 input dimensions, attains a 1.9% error rate (van der Wilk et al., 2017). It is important to note that both the convolutional GP and our GCGP are not hierarchical models, and to increase expressibility, van der Wilk et al. (2017) include learnable weights which multiply the patch response function for each patch. This increases classification accuracy but means that it is no longer strictly translation-invariant. Our GCGP model can similarly be extended, but this is left for future work. The hyperparameters ρ_k and σ_ρ converge to $\{1.312, 2.331, 4.034\}$ and 0.392.

Table 1. Error rates on MNIST classification

Method	Error rate
MNIST	
Conv. GP (25-dim) †	2.1%
RBF GP (784-dim) †	1.9%
GCGP (24-dim)	**1.7%**
MNIST Superpixel 75	
ChebNet (Defferrard et al., 2016)	24.4%
MoNet (Monti et al., 2017)	8.9%
GCGP	**4.2%**

† (van der Wilk et al., 2017)

4.2. MNIST Superpixels

To demonstrate the performance of GCGPs on general graphs, we apply the model to the 75-vertex MNIST superpixel dataset, following the methodology of Monti et al. (2017). This dataset decomposes each MNIST image into a 75-vertex graph with a unique edge structure. This dataset is constructed from the MNIST dataset as follows: for each image, pixels are divided into ‘background’ and ‘foreground’, denoting a pixel value of 0 and greater than 0 respectively. Then, for each group, k-mean clustering is performed on the pixel locations, the (x,y) index of the pixel, and pixel values. The parameter k is chosen for each group such that their sum is the desired number of superpixels, in this case 75, and so that the split is two thirds foreground and one third background. The resulting centroids give the location of the superpixel in Euclidean coordinates and the value of the superpixel. The value of the superpixels serve as the input signal $\psi : \mathcal{V} \to \mathbb{R}^d$, with $d = 1$, which is to be classified. Finally, the graph structure is determined by choosing a threshold value of the Euclidean distance between the centroids and adding an edge if the distance is below the threshold. This produces a unique graph structure for each image. The resulting dataset follows the same training and test set split as the standard MNIST dataset with 60,000 and 10,000 observations in each respectively. The GCGP is set up and trained in precisely the same way as in Section 4.1.

The GCGP converges to an error rate of 4.2% on this task, which is a substantial improvement over the ChebNet architecture proposed by Defferrard et al. (2016) that obtains

Table 2. Ablation study on Superpixel 75 error rates

Examples per class	100	500	1000
Error rate	13.7%	8.3%	6.3%

24.4%, and the MoNet architecture that obtains 8.9% on this task (Monti et al. (2017)). The improvement against ChebNet is to be expected, as ChebNet is based on learning filters in the spectral domain, which means it will struggle with a task that learns across graphs with different structure, while the GCGP is based on spatial convolutions that can handle such settings. The improvement over the MoNet architecture is more surprising. MoNet can be applied to such tasks and is a deep, hierarchical model in contrast to the GCGP, which is not hierarchical. This may demonstrate the advantage of using such a relatively terse model on a problem of this scale.

One reason we may prefer a relatively terse Bayesian model to a more complex model in applications to general graphs is we might expect it to perform well even with fewer observations in the training set. To investigate this, we performed an ablation study in which the size of the training set was reduced such that it was composed of 100, 500, or 1000 examples per class, randomly selected from the original training set, which produced new training sets of 1000, 5000, and 10000 examples respectively. The test set was left unchanged. The results of this study are presented in Table 2. Remarkably, our GCGP model using only a tenth of the training examples outperforms MoNet trained on the full training set. With only 5,000 training examples (500 per digit) GCGP achieves an error rate of 8.3% compared to MoNet yielding an error rate of 8.9% when trained on all 60,000 samples, demonstrating the data efficiency of GCGP.

4.3. Triangular 3D Mesh Classification

For our next test, we classify triangular meshes. As before, the number of vertices remains the same, but connectivity and embedding in three-dimensional space may be different for each graph.

The data is a collection of 100 meshes from the MPI Faust dataset[1] (Bogo et al., 2014). This includes ten different poses for each of ten different individuals. We use a training set of 70 meshes where we randomly select 7 poses for each individual. The remaining three poses for each individual comprise the test set (30 observations). Each mesh is comprised of 6800 vertices. These were downsampled to 2500, 1000, and 500 vertices using quadric edge collapse decimation in MeshLab (Cignoni et al., 2008) to test the sensitivity of the GCGP to the quality of the meshing.

[1]http://faust.is.tue.mpg.de/

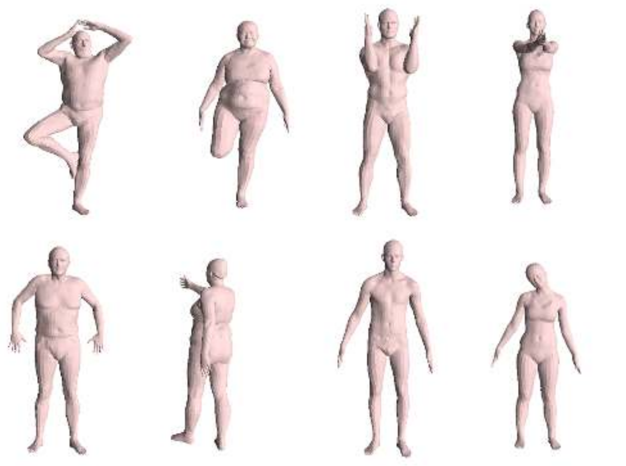
Figure 3. Example meshes from the MPI Faust dataset where different people are shown in different poses.

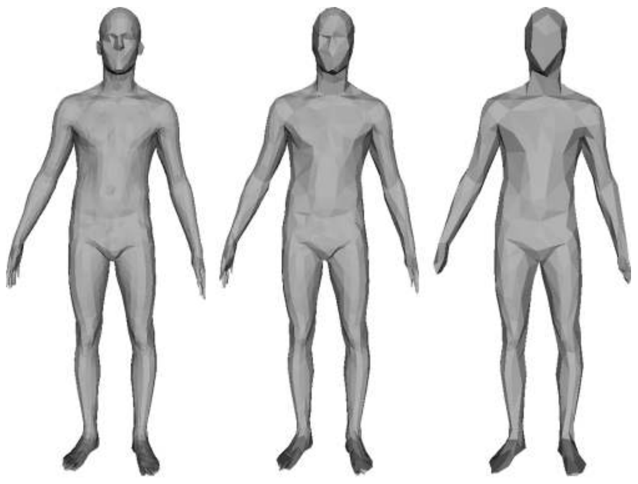
Figure 4. MPI Faust meshes resampled to 2500, 1000, and 500 vertices being used in our experiment in Sec. 4.3 to investigate the effect of resolution (cf. Tab. 3).

The goal is to learn the identity of the individual, making this a 10-class classification task. To the best of the authors' knowledge, Gaussian processes have not been employed in a similar task with such types of data.

The input data is a geometric descriptor vector that describes the intrinsic geometry of the mesh, as in Kokkinos et al. (2012). For the purposes of this example, we use only the first four descriptors. For each mesh we have an input signal $\psi : \mathcal{V} \to \mathbb{R}^4$ which is to be classified. We choose 16 angular and 5 radial bins for a total of 80 geodesic polar bins. When combined with the dimensionality of the input, this produces a 320-dimensional patch for each vertex.

We must define the $\rho(\cdot)$ and $\theta(\cdot)$ functions in Eq. (7). This is a more difficult problem than in the Euclidean domain above.

We begin by discussing how to compute the $\rho(\cdot)$ function, and subsequently discuss the construction of the $\theta(\cdot)$ function, which is more involved. Any graph pre-processing step in this section was implemented in C++ for efficiency.

4.3.1. Radial Distance Between Vertices

Computing the radial distance between two vertices on a triangular mesh is relatively straightforward, and we follow the same methodology laid out by Masci et al. (2015). We find the intrinsic distance between two vertices on the triangular mesh by using the Fast Marching Method on triangular domains (Sethian, 1996; Kimmel & Sethian, 1998). This method computes the time of arrival for a wave front traveling outwards from a group of initial conditions at a constant rate along the surface of the triangular mesh. By using each vertex as a starting point for the wave front separately, we are able to compute a measure of the intrinsic distance from each vertex to every other vertex, which produces a $|\mathcal{V}| \times |\mathcal{V}|$ matrix of intrinsic distances between the vertices. This can be precomputed for each mesh and does not represent too much of a computational overhead thanks to the $\mathcal{O}(n \log n)$ complexity of the Fast Marching Method. This matrix of distances for each mesh will be used to look up the radial component of the weighting function for each specific pair of vertices in Eq. (7).

4.3.2. Angular Distance Between Vertices

To compute $\theta(v, v')$, we must first compute for each vertex v a set of J geodesic rays emanating from v. By this we mean a set of rays starting at the central vertex of interest drawn along the surface of the mesh.

We begin by taking J rays emanating from a central point and equally dividing 2π radians between them. We then take these J rays and project them down onto the one-ring of triangles that surround the central vertex v. This is done by computing the total angle of the angles that are adjacent to the central vertex, aligning one of the geodesic rays with one of the existing edges of the mesh, and continuing around the central vertex by equally-spaced increments that are proportional to the total angle sum. In this way, each of the J rays will now be a ray lying on one of the faces adjacent to the central vertex. We will now focus on one specific ray and explain how to continue it along adjacent triangles.

We extend the ray to find its intersection point with the edge opposite the central vertex from which it emanates. We now begin a process known as unfolding. This is done by treating the original triangle as lying in a two-dimensional "unfolding plane". We then take the triangle from the mesh that shares the edge that the geodesic ray intersects and reconstruct a congruent triangle in the two-dimensional unfolding plane. An alternative way to visualize this is to rotate (around the shared edge) the adjacent triangle in three-dimensional space into the same plane as the starting triangle. We then extend the ray and find the edge of this unfolded triangle with which it intersects. We continue the process of extending the ray along the mesh, unfolding each successive triangle and keeping track of the vertex it passes closest to with each intersection. This process continues until it makes a complete circuit of the mesh, hits the mesh boundary, if one exists, or stops after a designated length.

Table 3. Error rates on MPI Faust mesh classification

Number of vertices	500	1000	2500
MoNet	40.00%	33.33%	33.33%
GCGP	**23.33%**	**10.00%**	**3.33%**

At this stage, for each of the J geodesic rays, there should be a list of vertices which are the vertices to which the geodesic passed the closest. To create the value of the θ function from each vertex to every other vertex, we use this list as the initial conditions for Fast Marching and compute the time of arrival for every vertex from this starting point. Once this process is complete, there will be a $J \times |\mathcal{V}| \times |\mathcal{V}|$ tensor that serves as a look-up table for the θ value for the jth angular bin from a given vertex to every other vertex.

If the θ_j parameters are fixed by the investigator, this can once again be precomputed for each mesh in the dataset. Not fixing the parameters presents an implementation difficulty because if the angular parameters are changed, then it is necessary to recompute all of the geodesics at every update step for the hyperparameters since the θ parameters control the angles at which the geodesics emanate from the central vertex. Note that this is not a problem for the radial bins since the ρ function, which is simply the intrinsic distances between the vertices on the mesh, remains fixed regardless of the settings of the ρ parameters, which control the midpoint of the radial bin along the surface of the mesh. In the presentation of our results we keep all hyperparameters of the weighting function fixed, which means that we can now use the computed ρ and θ functions to compute the weight tensors via Eq. (7). Using this, we can transform each signal into a pseudo-image that is $|\mathcal{V}| \times (JK \cdot d)$, which will then be used in the learning task.

4.3.3. Results

The results of these experiments are presented in Table 3. When performing a 10-way classification on the 2500-vertex task, the graph convolutional GP converges to an error of 3.33%, which represents one misclassification. As expected, reducing the resolution of the mesh decreases accuracy. It is difficult to compare these results to a standard GP model as these must be applied to high-dimensional inputs and few examples per class. Without taking into account any spatial information, these would have 10,000 input dimensions for 2500 vertices, which would be difficult for a standard RBF

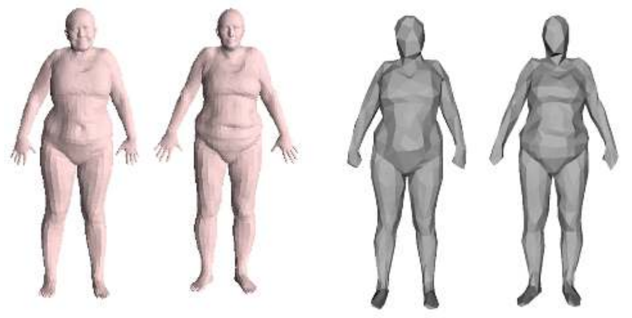

Figure 5. Individuals 4 and 8 from MPI Faust with full resolution (left) and 500 vertices (right). GCGP consistently confuses the identity of these two individuals at low resolution, which can be explained by their remarkably high visual similarity.

model to estimate. Indeed, an RBF modeling all 10,000 input dimensions fails to stably learn anything in this case. For these experiments we use a batch size of 30 along with 750 inducing points and a learning rate of 0.001. The results provide some useful, albeit qualitative, insights about the model. In particular, the GCGP model struggles to distinguish between individuals 4 and 8 (Figure 5), especially at the 500-vertex resolution, where it misclassifies every example of one as the other, which can be explained by the high visual similarity of these two subjects.

We also attempted to compare the GCGP performance on the mesh classification task with MoNet as a representative of recent geometric deep learning methods. To this end, we reimplemented MoNet to be able to directly apply it to the 3D mesh data. We initially tried to closely follow the setup as described in (Monti et al., 2017), i.e., using three graph convolutional layers but with the final layer replaced with a 10 dimensional softmax layer to classify the individual subjects. This model, however, overfitted immediately to the training data of 70 meshes for each of the 10 individuals, yielding a test accuracy close to random guessing. We also explored a less complex MoNet with only one graph convolutional layer for which the best performance we could obtain was an error rate of 33.33% on the 2500-vertex task. We believe that the small amount of training data is insufficient to properly train this deep learning method, but further investigation into this issue may be required.

5. Conclusion

Computer vision and imaging applications have led to significant progress with convolutional methods that excel when data lives on Euclidean domains. The drawback of these methods is that they require large amounts of training data and provide little means of capturing uncertainty when making predictions. Gaussian processes, on the other hand, are an attractive Bayesian approach that can learn from few data points on non-Euclidean domains and provide useful uncertainty estimates. So far, we have seen links between convolutional methods and GPs, leading to convolutional GPs. We have also seen approaches that bring convolutional methods to non-Euclidean domains in the form of graph CNNs, an area also known as geometric deep learning. Here, for the first time, we connect those three areas by combining convolutional methods with GPs on non-Euclidean domains. We believe that graph convolutional GPs provide a powerful new framework enabling exciting new applications and avenues for future research. We believe that the results of the experiments presented here support this, as they demonstrate the GCGP can improve accuracy and make more efficient use of smaller datasets relative to deep learning methods.

We envision augmenting GCGPs to become hierarchical models with recent work on deep GPs (Damianou & Lawrence, 2013) as a main directions of future research. This is because the strict translation invariance of the patch response function may not be enough in all situations, which may require combinations of responses across the graph. A simple step is to weigh each response with a set of learnable parameters as in (van der Wilk et al., 2017).

Another area for improving the model is relaxing the requirement for a fixed number of vertices. In the method presented here, the function that is learned, $f : \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}$, for classification, of the signals $\psi : \mathcal{V} \to \mathbb{R}^d$ is decomposed in a principled manner into the sum of the responses of a patch response function g across the graph. Modelling g with a GP prior induces a GP prior on f, but if $|\mathcal{V}|$ is different for different inputs this setup fails by definition. While there are many situations in which the inputs are signals on graphs with the same number of vertices with potentially different structure, other problems may require the analysis of graphs which have different structure *and* number of vertices, such as classification of molecules. We hope the method presented here can serve as a foundation for GP models that relax this requirement.

An interesting direction to explore further is the choice of the coordinates/pseudo-coordinates that need not to be the ones used in this paper, and many interesting alternatives exist for meshes and for graphs more generally that are worth investigating. Finally, a nice feature of the GCGP that allows it to be easily tailored to different applications is the selection for the kernel function that describes the patch response function g. This is mentioned in the discussion above, but not explored. For example, it may be the case that modelling the patch response function in Section 4.3 with a 320 dimensional RBF kernel is too expressive for the task and the number of observations resulting in suboptimal performance. Performance may therefore be improved by decomposing the patch response function to produce an additive kernel whose components only utilise subsets of the inputs, the same principle that underpins the GCGP.

Acknowledgements

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 757173, project MIRA, ERC-2017-STG). IW is supported by the Natural Environment Research Council (NERC).

References

Bogo, F., Romero, J., Loper, M., and Black, M. J. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017. ISSN 1053-5888. doi: 10.1109/MSP.2017.2693418.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold Gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 3338–3345, July 2016. doi: 10.1109/IJCNN.2016.7727626.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. MeshLab: an open-source mesh processing tool. In Scarano, V., Chiara, R. D., and Erra, U. (eds.), *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. ISBN 978-3-905673-68-5. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

Damianou, A. and Lawrence, N. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pp. 207–215, 2013.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3844–3852. Curran Associates, Inc., 2016.

Duvenaud, D., Nickisch, H., and Rasmussen, C. E. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pp. 226–234, 2011.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Hensman, J., Matthews, A., and Ghahramani, Z. Scalable Variational Gaussian Process Classification. In Lebanon, G. and Vishwanathan, S. V. N. (eds.), *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pp. 351–360, San Diego, California, USA, 09–12 May 2015. PMLR.

Kimmel, R. and Sethian, J. A. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 95:8431–8435, 1998.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kokkinos, I., Bronstein, M. M., Litman, R., and Bronstein, A. M. Intrinsic shape context descriptors for deformable shapes. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 159–166, June 2012. doi: 10.1109/CVPR.2012.6247671.

Lázaro-Gredilla, M. and Figueiras-Vidal, A. R. Inter-domain Gaussian processes for sparse inference using inducing features. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pp. 1087–1095, USA, 2009. Curran Associates Inc. ISBN 978-1-61567-911-9.

Mairal, J., Koniusz, P., Harchaoui, Z., and Schmid, C. Convolutional kernel networks. In *Advances in Neural Information Processing Systems 27*, pp. 2627–2635. Curran Associates, Inc., 2014.

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. Geodesic convolutional neural networks on Riemannian manifolds. In *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, ICCVW '15, pp. 832–840, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-9711-7. doi: 10.1109/ICCVW.2015.112.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

Monti, F., Boscaini, D., Masci, J., Rodol, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5425–5434, July 2017. doi: 10.1109/CVPR.2017.576.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245, 2016.

Ng, Y. C., Colombo, N., and Silva, R. Bayesian semi-supervised learning with graph gaussian processes. In *Advances in Neural Information Processing Systems*, pp. 1683–1694, 2018.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023, 2016.

Rasmussen, C. E. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pp. 63–71. Springer, 2004.

Sethian, J. A. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93 (4):1591–5, feb 1996.

van der Wilk, M., Rasmussen, C. E., and Hensman, J. Convolutional Gaussian processes. In *Advances in Neural Information Processing Systems 30*, pp. 2849–2858. Curran Associates, Inc., 2017.

Venkitaraman, A., Chatterjee, S., and Haendel, P. Gaussian processes over graphs. *arXiv preprint arXiv:1803.05776*, 2018.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.