Deep Gaussian Processes and Applications

Florent Rambaud
Department of Mathematics
ENS Paris-Saclay
flo.rambaud@gmail.com

Narek Arakelyan
Department of Mathematics
ENS Paris-Saclay
araqelyannarek@gmail.com

Pierre Osselin
Department of Mathematics
ENS Paris-Saclay
pierre.osselin@gmail.com

Abstract

Deep Neural Networks have revolutionized the world of machine learning as they embody very powerful models. However, they do not display any uncertainty quantification and are very weak when data is scarce. In this paper we study the deep Gaussian process (GP) model introduced in [9]. The deep GP is a multi-layer belief network based on composition of Gaussian Processes. The intuition is that the construction of a deep architecture would result in a larger class of generative models. With deep GP models we require fewer data and train through variational approaches for approximating the marginal likelihood. The model can be used both for supervised and unsupervised learning and for Bayesian optimization problems among others. After motivating the introduction of deep GPs through Gaussian Processes and recent insights into Deep Bayesian Neural Networks, we will make experimentations on the mentionned tasks and highlight the strength of the Deep GPs for particular contexts. Our code is available on the Github repository [1].

1 Introduction

Deep architecture models for machine learning are well studied and widely used. There is a large range of problems in different fields that can be solved with deep learning. However, deeper architectures needs more data for training. On the other hand, for humans these tasks are possible by having a few examples.

We know that GP is equivalent to a neural network with infinity neurons [1, 10], but there are problems where the GP representations are not relevant or where we have to choose a complex kernel to fit the data. We wonder if it is possible to learn deep complex totally probabilistic generative models that work on scarce data. An answer is given in [9] where they introduce deep GP models and bring two principal contributions: first they exploit recent advances in variational inference to derive a rigorous lower bound on the marginal likelihood, then they use this lower bound to apply deep models even when data is scarce. We will implement some experiments seen in the paper for supervised and unsupervised learning examples on new data. In addition we will implement the Bayesian optimisation using deep Gaussian process with one hidden layer. We use the code for deep GP models available on GitHub [2].

In the next section we detail the framework of Gaussian processes with their applications, then we move to sparse Gaussian processes, and lastly explain the link with Bayesian deep learning models.

[1]https://github.com/pierreosselin/Deep_Gaussian_Process_Project
[2]https://github.com/SheffieldML/PyDeepGP

In the last section we introduce deep Gaussian processes and highlight some of its intrinsic properties through various tasks and experiments.

2 Gaussian Processes

2.1 Model

Gaussian processes belong to the family of *Bayesian Nonparametric approach.* In this framework we treat a function $f : \mathcal{X} \to \mathcal{Y}$ as the random variable taking values in an infinite-dimensional space of functions. Then, we model a prior over the set of functions $\mathcal{F}$ that we denote $p(f)$ and compute a posterior over function $p(f|\mathcal{D})$. Given an index set $\mathcal{X}$, the set $(A_x)_{x\in\mathcal{X}}$ is a Gaussian process if

$$\forall(x_1, ..., x_n) \in \mathcal{X} : [A_{x_1}, ..., A_{x_n}]^T \sim \mathcal{N}([\mu(x_i)]_i^T, [k(x_i, x_j)]_{i,j})$$

where $\mu(x)$ and $k(x, x^{'})$ are the mean and covariance functions, which fully specify the Gaussian process:

$$\mu(x) = \mathbb{E}[f(x)]$$
$$k(x, x^{'}) = \mathbb{E}[(f(x) - \mu(x))(f(x^{'}) - \mu(x^{'}))]$$

Here the expectation is taken according to the source of randomness in the function f. The mean function $\mu : \mathcal{X} \to \mathbb{R}$ can be anything but is often taken to be zero in practice, and the covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ has to be positive definite.
The specification of the prior is important for making predictions. This prior defines the set of prior belief that we have about the shape of our functions, different types of kernels are thus able to capture different prior properties of our function. For example the Matérn kernel allows us to specify functions with a certain degree of differentiability:

$$C_\nu(x, x^{'}) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \Bigg(\sqrt{2}\frac{||x - x^{'}||_2}{\rho}\Bigg)^\nu K_\nu\Bigg(\sqrt{2\nu}\frac{||x - x^{'}||_2}{\rho}\Bigg)$$

Where $\Gamma(.)$ is the gamma function, K_ν the modified Bessel function of the second kind and ρ and ν are non-negative parameters of the covariance. In particular, a Gaussian process with this covariance function is $\lceil\nu\rceil - 1$ times differentiable in the mean-square sense.
Another classic kernel used to model infinitely-differentiable functions is the RBF kernel:

$$K(x, x^{'}) = \exp\Bigg(-\frac{||x - x^{'}||_2^2}{2\sigma^2}\Bigg)$$

We plot some samples following a Gaussian processes according to these kernels with different hyperparameters in Annex A.1

2.2 Inference

Regression: In the Gaussian process regression model, we assume the following likelihood function:

$$\boldsymbol{f} \sim \mathcal{GP}(\mu(.), K(., .))$$
$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{f}, \sigma^2\boldsymbol{I})$$

Then we obtain a closed-form posterior predictive distribution $\boldsymbol{f}^{'}$ on unknown points $\boldsymbol{x}^{'}$ knowing $\boldsymbol{y}$:

$$\boldsymbol{f}^{'}|\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{K}_{x^{'}x}(\boldsymbol{K}_{xx} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}, \boldsymbol{K}_{x^{'}x^{'}} - \boldsymbol{K}_{x^{'}x}(\boldsymbol{K}_{xx} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{K}_{x^{'}x})$$

We plot some regression examples in the Annex A.2 to highlight the different predictions made by a Gaussian process with different kernels.

Classification: In the Gaussian process classification model, we assume the following likelihood function:

$$\boldsymbol{f} \sim \mathcal{GP}(\mu(.), K(., .))$$
$$p\Big(y_i = +1|f(x_i) = \sigma(f(x_i)\Big) = \frac{1}{1 + \exp(-f(x_i))}$$

Then the posterior is intractable, but can be approximated using Laplace approximation [8].

2.3 Unsupervised Learning

A building block of the model we will introduce in the next section is the Gaussian process used as a model for unsupervised learning (GP-LVM) [7]. If we suppose we have data $Y \in \mathbb{R}^{N\times D}$ where N is the number of data points and D the dimension of the data, we can perform dimensionality reduction via latent variables $X \in \mathbb{R}^{N\times Q}$. We can suppose that the data are generated via Gaussian processes:

$$p(Y|X) = \prod_{d=1}^{D} p(\boldsymbol{y}_d|X) \text{ (The likelihood factorizes accross dimensions)}$$

$$p(\boldsymbol{y}_d|X) = \mathcal{N}(\boldsymbol{y}_d|\mathbf{0}, \boldsymbol{K}_{NN} + \beta^{-1}\boldsymbol{I}) \text{ (Gaussian process with noise)}$$

$$p(X) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\mathbf{0}, \boldsymbol{I}_Q) \text{ (Normal Prior on the latent space)}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-\frac{1}{2}\sum_{q=1}^{Q} \alpha_q (x_q - x_q')^2) \text{ (ARD covariance function)}$$

The kernel is chosen to be the ARD squared exponential form in order to perform automatic model selection. The hyperparameters in this model are the kernel parameters as well as β. In this model, the dimensions of the data are supposed to be drawn independently from a Gaussian process over a latent space, that we suppose a priori to have a simple shape. Learning is performed via variational approximation where we approximate the posterior distribution $P(X|Y)$ by a simple distribution $q(X) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_n, \boldsymbol{S}_n)$ which allows us to compute a lower bound on the marginal likelihood $p(Y)$ by applying variational sparse GP regression. The bound can then be jointly maximized over the variational parameters $(\{\boldsymbol{\mu}_n, \boldsymbol{S}_n\}_{n=1}^{N}, Z)$, Z being the inducing points (see next section), and the model hyperparameters $(\boldsymbol{\theta}, \beta)$ by applying gradient-based optimization techniques.

2.4 Sparse GP

Gaussian processes are very powerful models but also very computationally expensive, since it is a nonparametric model, it has a $\mathcal{O}(n^2)$ memory complexity to keep track of the covariance matrix and to keep track of data. Furthermore, the posterior requires the inversion of covariance matrices of size n which require a $\mathcal{O}(n^3)$ computational complexity. This is very limiting in practice and justifies the use of approximation techniques to limit this cost. These approximation techniques will also be used in our deep Gaussian process model. Multiple methods have been developped, most of them relies on the following matrix factorization. If we suppose that the covariance matrix can be approximated as $\boldsymbol{K}_{xx} \approx QQ^T$, where $Q \in \mathbb{R}^{n\times m}$ then:

$$\left(QQ^T + \sigma^2\boldsymbol{I}\right)^{-1} = \sigma^{-2}\boldsymbol{I} - \sigma^{-2}Q\left(\sigma^2\boldsymbol{I} + Q^TQ\right)^{-1}Q^T$$

Which only requires the inversion of a $m \times m$ matrix. The optimal low-rank approximation requires to perform a eigendecomposition of $\boldsymbol{K}_{xx}$ which has a complexity of $\mathcal{O}(n^3)$.
Instead the *Nyström approximation* only select a set of inducing points $\{z_i\}_{1,...,m}$ and make the approximation $\tilde{\boldsymbol{K}}_{xx} = \boldsymbol{K}_{xz}\boldsymbol{K}_{zz}^{-1}\boldsymbol{K}_{zx}$. While the most ancient method consists of using greedy subset of regressors [2] of a reparameterized formulation of the Gaussian process regression model, more modern methods use the Nyström approximation by only keeping a subset of the data (the "active" set) chosen greedily according to some criterion [3]. Methods using pseudo-input points and not part of the data have also been developed in [4]. The pseudo-inputs are found by maximizing the marginal likelihood of the data predicted with the pseudo-inputs. The most recent methods, one of which we will describe quickly, use variational approximation.
One such method [5] approximates the posterior GP distribution:

$$p(\boldsymbol{z}|\boldsymbol{y}) = \int p(\boldsymbol{z}|\boldsymbol{f}_m, \boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{f}_m, \boldsymbol{y})p(\boldsymbol{f}_m|\boldsymbol{y})d\boldsymbol{f}_m d\boldsymbol{f}$$

where $\boldsymbol{f}_m$ are drawn from the GP prior given inducing points X_m. If we suppose that $\boldsymbol{f} \perp\!\!\!\perp \boldsymbol{z}|\boldsymbol{f}_m$ (i.e $\boldsymbol{f}_m$ are sufficient statistics for $\boldsymbol{f}$ and $\boldsymbol{z}$) then we end up with the approximate posterior:

$$q(\boldsymbol{z}) = \int p(\boldsymbol{z}|\boldsymbol{f}_m)\phi(\boldsymbol{f}_m)d\boldsymbol{f}_m$$

where $\phi(\boldsymbol{f}_m) = p(\boldsymbol{f}_m|\boldsymbol{y})$. This variational method then approximates $\phi(\boldsymbol{f}_m)$ by a multivariate normal distribution and optimizes the Kullback-leibler divergence $KL(q(\boldsymbol{f}, \boldsymbol{f}_m)||p(\boldsymbol{f}, \boldsymbol{f}_m|\boldsymbol{y}))$ according to (X_m, ϕ). This technique is the one used for the previous unsupervised model as well as for the model we will introduce in the next part.

2.5 Link to Deep Neural Networks

Gaussian processes are very strong and adaptive models that draw a lot of attention. Another key characteristic of the power of the Gaussian processes relies on their strong link with neural networks. The first paper written by Neal [1] shows that, thanks to the central limit theorem, an infinitely wide Bayesian neural network with one hidden layer is equivalent to a Gaussian process with a certain kernel:

$$f_k(x) = b_k + \sum_{j=1}^{H} v_{jk}\sigma(a_j + \sum_{i=1}^{I} u_{ij}x_i)$$

Where we have K output units, H hidden units, a bounded (to apply the CLT) activation function σ and where every parameters of the neural networks have a prior following a i.i.d centered Gaussian distribution. If we have a Gaussian prior distribution whose variance is equal to $\sigma_v = \omega_v H^{\frac{1}{2}}$ and the variance of the bias is σ_b then the k^{th} output of the neural network tends to a Gaussian process with zero mean and kernel:

$$k(x, x^{'}) = \sigma_b^2 + \omega_v^2 C(x, x^{'}) \text{ where } C(x, x^{'}) = \mathbb{E}_{a_j, u_{ij}}[\sigma(a_j + \sum_{i=1}^{I} u_{ij}x_i)\sigma(a_j + \sum_{i=1}^{I} u_{ij}x_i^{'})]$$

A very interesting remark from Neal is that the outputs of such system are independent, if the weights over the hidden layers are independent. Hence it is not possible to define a Gaussian-based prior expressing the idea that two outputs might show a large change in the same input region, the location of this region being unknown a priori, without also fixing whether the changes in the two outputs have the same or opposite sign.
In the case of deep neural networks, a similar result has been shown very recently [10]. The authors applied the previous result sequentially from the bottom hidden layers to the upper layers. Suppose:

$$z_i^l = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l x_j^l(x),\, x_j^l(x) = \sigma(z_j^{l-1}(x))$$

where we place ourself at the l^{th} layer and, by induction, z_j^{l-1} is a GP identical and independant over j. With $N_l \to +\infty$ we obtain a new GP over $z_i^l \sim \mathcal{GP}(0, K^l)$ with a covariance function :

$$K^l(x, x^{'}) = \mathbb{E}[z_i^l(x)z_i^l(x^{'})] = \sigma_b^2 + \sigma_w^2 \mathbb{E}_{z_i^{l-1} \sim \mathcal{GP}(0, K^{l-1})}[\sigma(z_i^{l-1}(x))\sigma(z_i^{l-1}(x^{'}))] \tag{1}$$

$$K^l(x, x^{'}) = \sigma_b^2 + \sigma_w^2 F_\sigma(K^{l-1}(x, x^{'}), K^{l-1}(x, x), K^{l-1}(x^{'}, x^{'}))$$

where F_σ is a deterministic function depending on σ. Some work has been carried out to assess the rate of convergence of finite Bayesian neural networks towards their corresponding Gaussian processes, like in [12] where the authors used particular non-linear activation functions that allow for a simple closed form corresponding kernel for the Gaussian process. They also showed that the asymptotic convergence is independent of the shape of the neural network architecture as long as the sequence of neural networks has a strictly increasing number of hidden units on every one of their hidden layers. These results are truly beautiful in the sense that it allows us to translate "local" priors over weights of neural network into an asymptotic functional prior. These studies allow us to understand which space of functions we approximate by putting certain priors in a neural network. Reversely, it allows us to translate functional prior into local prior in Bayesian neural network to approximate the spaces of functions that we want. Equation 1 is crucial because it embodies the fundamental link between Gaussian processes and the deep neural network, together with the influence of its depth and activation function.

3 Deep Gaussian Processes

3.1 Motivation

We have seen that a neural network with infinitely wide hidden layers gives rise to a Gaussian process with a complex composition of kernels governed by Equation 1. Capturing complex expressiveness thus requires to either build complex kernels plugged into a Gaussian process or to derive a deep and wide neural network of "simple" atomic operations. In this work published by Damianou et al. [9], another approach is taken where the model is described by a hierarchy of stacked Gaussian processes with simple kernels. In this structure, every Gaussian process transforms its input via a Gaussian process and feed it into the input of the next layer. This can interpreted as a hierarchy of prior distributions over the input.

3.2 Model

The Deep Gaussian process model extends the GP-LVM model introduced previously by "Stacking" Gaussian processes. This can be interpreted as hierarchically modelling the input prior of each Gaussian process by a Gaussian process. More formally, given a leaf node $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$ of observed data, we introduce intermediate layers of latent nodes $\boldsymbol{X}_h \in \mathbb{R}^{N \times Q_h}$, $h = 1, ..., H-1$ where H is the number of hidden layers, and the parent node $\boldsymbol{Z} = \boldsymbol{X}_H \in \mathbb{R}^{N \times Q_Z}$. The parent can be either a latent space (unsupervised scenario) where we put a desired structure on it through the elicitation of a convenient prior, or an observed input (supervised scenario). The graphical model of such model is shown in Figure 1. In order to cope with the addtion of model parameters introduced by $\boldsymbol{X}_h$ the original paper aims at variationally marginalizing the latent space to perform an efficient training as well as an automatic Occam's Razor. Similarly to the GP-LVM model, the covariance function of the Gaussian processes will be taken to be Automatic Relevance Determination covariance functions (ARD):

$$k(\boldsymbol{x}, \boldsymbol{x}^{'}) = \sigma_{ard}^2 \exp(-\frac{1}{2}\sum_{q=1}^{Q} w_q (x_q - x_q^{'})^2)$$

Figure 14 in the Annex shows that the modeled functional space can be complex, even for two layers of simple (RBF) Gaussian processes.

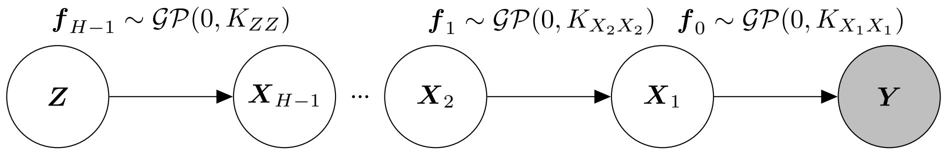
$\boldsymbol{f}_{H-1} \sim \mathcal{GP}(0, K_{ZZ})$
$\boldsymbol{f}_1 \sim \mathcal{GP}(0, K_{X_2X_2})$
$\boldsymbol{f}_0 \sim \mathcal{GP}(0, K_{X_1X_1})$
$\boldsymbol{Z}$
$\boldsymbol{X}_{H-1}$
...
$\boldsymbol{X}_2$
$\boldsymbol{X}_1$
$\boldsymbol{Y}$

Figure 1: Deep GPs graphical model

3.3 Training

The complexity of the model comes with its training complexity. For simple Gaussian regression models, the hyperparameter tuning is tractable via the optimization of the marginalized distribution of the observations. For Deep GPs we have:

$$p(\boldsymbol{Y}) = \int_{\boldsymbol{X}_{1:H}} p(\boldsymbol{Y}|\boldsymbol{X}_1)p(\boldsymbol{X}_1|\boldsymbol{X}_2)...p(\boldsymbol{X}_{H-1}|\boldsymbol{Z})p(\boldsymbol{Z})d\boldsymbol{X}_{1:H}$$

Which is intractable (nonlinear composition of the inverse of covariance matrices). To circumvent this problem many variational approximations has been developed [7] [10] including the one we introduced which is the classical approach of the paper. Given a variational distribution $q(\boldsymbol{X}_{1:H})$:

$$log(p(\boldsymbol{Y})) \geq \mathbb{E}_q[log(p(\boldsymbol{Y}|\boldsymbol{X}_{1:H}))] - KL(q(\boldsymbol{X}_{1:H})||p(\boldsymbol{X}_{1:H})) \tag{2}$$

By introducing inducing variables at each layer $Z_h = [\boldsymbol{z}_h^1, ..., \boldsymbol{z}_h^{m_h}]^T$, $\boldsymbol{z}^{(i)} \in \mathbb{R}^{Q_h}$ and $U_h = \boldsymbol{f}_h(Z_h) \in \mathbb{R}^{n \times Q_h}$. We can approximate the posterior of the joint distribution of the latent variables $p(\boldsymbol{X}_{1:H}, \boldsymbol{U}_{1:H}|\boldsymbol{Y},)$ by a variational distribution $q(\boldsymbol{X}_{1:H}, \boldsymbol{U}_{1:H})$ with an assumption of independence across layers:

$$q(\boldsymbol{X}_{1:H}, \boldsymbol{U}_{1:H}) = \prod_{h=1}^{H} q(\boldsymbol{X}_h)q(\boldsymbol{U}_h)$$

If the variational distributions are also supposed to belong to the exponential family then results from the sparse variational litterature (see [5]) show that a tractable lower bound can be found for Equation

2. The parameters optimized become the kernel parameters $\mathbf{\Theta}_{1:H}$, the inducing points $Z_{1:H}$ and the variational distribution parameters (mean and covariance matrix) $(\bar{\boldsymbol{X}}_h, \mathbf{\Sigma}_h)_{h=1}^{H}$. A graphical model summarizing the approximation is displayed Figure 2.

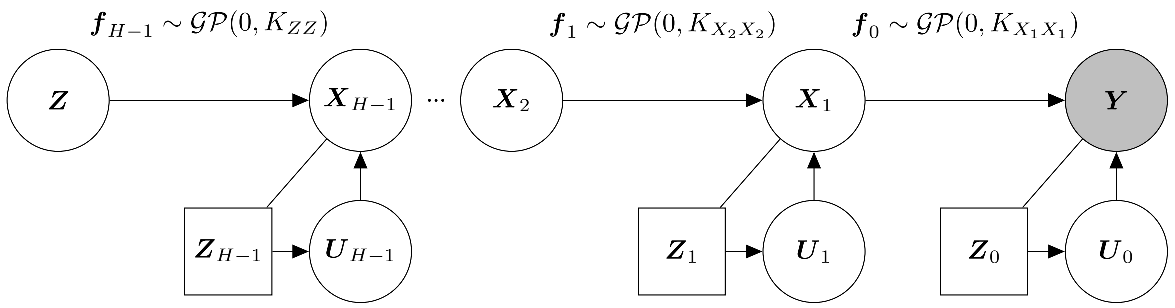

Figure 2: Deep GPs variational approximation

3.4 Direct Application

Supervised Learning: We applied the deep Gaussian process to a simple regression task where we aim at reconstructing a non-stationary function from a sample of points. Here we took the function $f(x) = \sin(\frac{2}{0.1+x})$ which is rough around $x = 0$ and becomes smoother when x diverge. A plot of the function is displayed Figure 3. We can clearly understand why a classical Gaussian process with a simple kernel would not work. For instance, for the RBF kernel, the optimization of the hyperparameter lengthscale would result in either a lack of representativity where the function is rough of where it is smooth.

We took a deep Gaussian process with one hidden layer and 20 inducing points corresponding to the number of data points. We also chose to have a hidden layer of dimension 2 for a more complex model as well as to allow us to plot it. We compared it with a sparse Gaussian process from the python package *GPy* with the same number of inducing points. The results are shown Figure 3.

As expected the classical optimized Gaussian process with an RBF kernel fails to learn the function. As it can only capture stationary functions with an uniform degree of smoothness, it learns a very small length scale and thus believe that very nearby points are almost independent and predicts that their evaluations is drawn from the prior, hence zero. The Deep Gaussian process, on the other hand,learns the function almost perfectly, as the third point is not attained and the rightmost part of the function seems to be too rough. This can be seen on the latent space (Figure 4) where the ending points seem to start behaving like the first points where the function is rough by increasing its second dimension.

Unsupervised Learning: We implemented the Deep GP latent variable model (DGP-LVM) thanks to the package *PyDeepGP* of the paper [9] on the dataset *Fashion MNIST*. This dataset is made of clothes pictures of size 28×28. We subsampled the dataset to keep only four different classes (Figure 5) with 200 samples per class. We used the following architecture: five layers with respective size $\{30, 24, 18, 12, 8\}$. We used 100 inducing points at each layer and optimized for 1000 iterations. The ARD kernel automatically selected only two dimensions four the first layer as we can see in Figure 6. Then we could use this model to generate new samples by injecting new values at the first layer as detailed in Figure 7. We can see that this generative model, that is trained in an unsupervised way, is a powerful tool for either clustering, dimension reduction or sampling new data.

3.5 Bayesian Optimization

The strength of the Deep GPs highlighted in the previous subsection can be leveraged to disrupt the state of the art of the bayesian optimization algorithm. Their ability to capture non-stationary function allows us to have a local and tunable uncertainty quantification that enables a more efficient exploration of the function to minimize.

In the Bayesian Optimization setting, one aims at minimizing an expensive objective function as efficiently as possible. To do this the function is modeled with a probabilistic function and points are

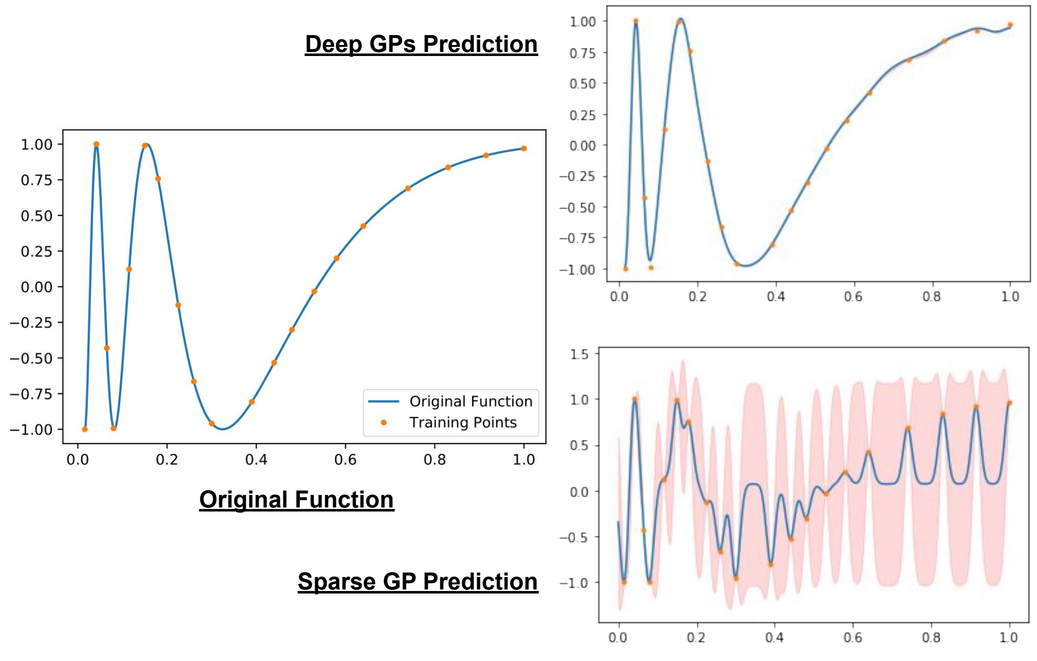

Figure 3: Comparison predictive function for a deep Gp and a sparse Gp

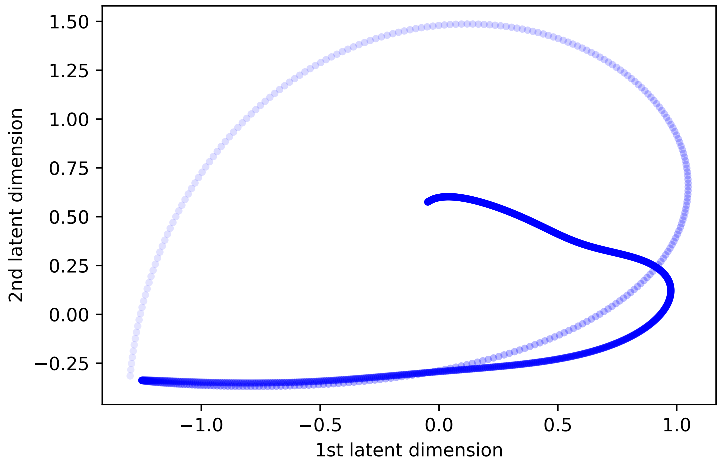

Figure 4: Resulting Latent space, from opaque (x = 0) to bold (x = 1)

explored sequentially. One criterion for choosing points is the Expected Improvement (EI):

$$EI(\boldsymbol{x}) = \mathbb{E}[I(x)] = \int_{\mathbb{R}} \max(0, y_{min} - t) p(t|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{y})$$

In our case $p(t|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{y})$ will be our predictive distribution according to our deep GP model. This integral is not available in closed for our model, contrary to a classic Gaussian distribution. To compute we will sample from our predictive distribution and compute this integral via Monte Carlo:

$$EI_n(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} \max(0, y_{min} - t_i) \text{ with } t_i \sim p(t|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{y})$$

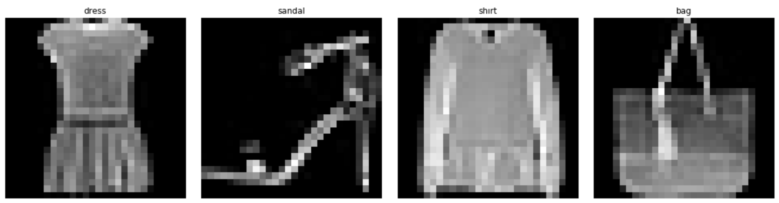

Figure 5: Four classes of *Fashion MNIST* dataset

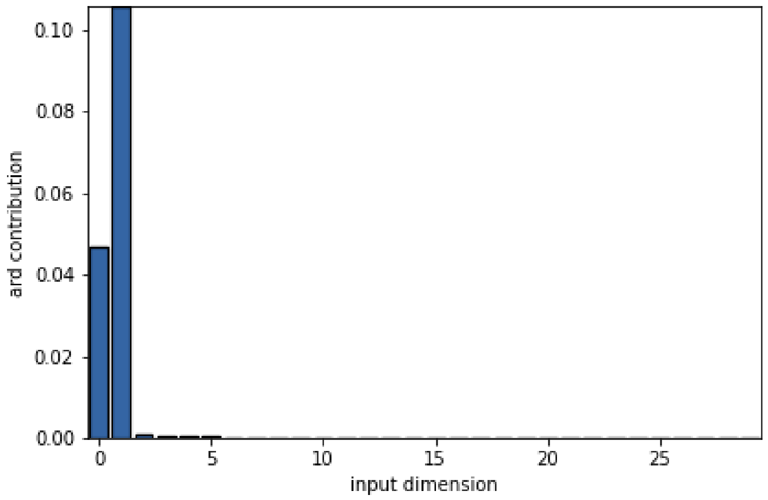

Figure 6: ARD coefficients for the first layer

We implemented a working code on our Github repository that performs Bayesian Optimization with the *DeepGP* package. We used the same modified sinus function used for our supervised experiments. We started with 4 random points on the function and used a Deep Gaussian Process model with one hidden layer of dimension two and the same number of inducing points as the number of data. We used the Expected Improvement criterion for choosing our next points. The result is given Figure 8. We can see on the figure the locality of the uncertainty quantification.

4 Conclusion

We have introduced the framework of Gaussian and Sparse Gaussian processes to derive the framework of deep Gaussian processes, and motivated their introduction with recent results form the BNN litterature. We saw how DGPs can be trained through sparse variational methods and how they can be used in practise in various examples. Like GPs, DGPs suffer from a high computational cost, that we faced during our implementations. We also faced important instabilities during our experiments, depending on the choice of the numerous hyperparameters (number of layers, dimension of the layers, inducing points per layer, kernel, ...). This raises the question of how elicitating good default parameters depending on the data and training procedure. However DGPs come with a greater representational power than GP (as seen in Figure 3) and the ability to perform even when data is scarce (unlike deep neural networks). One main challenge of DGPs remains their scalability to large real life problems due to their complexity: indeed with the *Fashion MNIST* dataset the model took several hours to train on only 800 images of size 28×28. However a better approach to treat images was proposed in [11] with the use of convolutional Gaussian processes. Moreover, as we noticed in our unsupervised experiments, the properties of the uncovered latent spaces learnt directly from data are hard to grasp, and the question of their interpretability is as challenging as important to understand and develop this model.

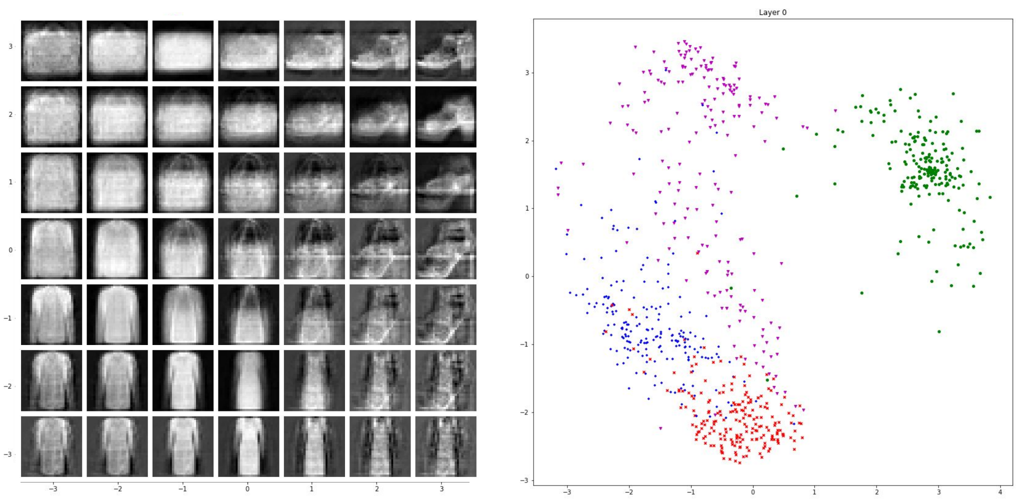

Figure 7: Generated samples depending on the latent space (two first dimensions) values

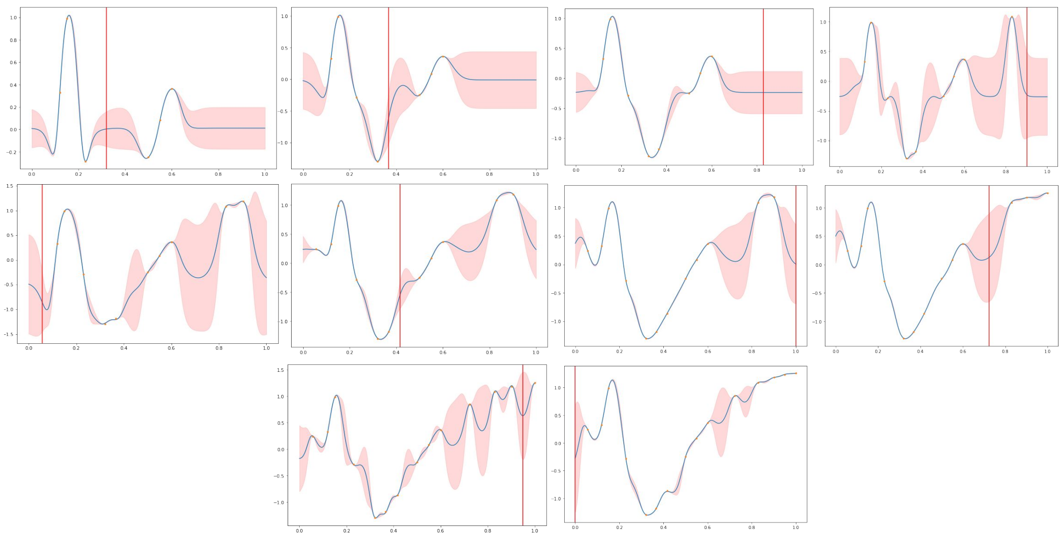

Figure 8: Example of Bayesian Optimization instantiation with Deep GPs

References

[1] Radford M Neal. "Priors for infinite networks". In: *Bayesian Learning for Neural Networks*. Springer, 1996, pp. 29–53.

[2] Alex J Smola and Peter L Bartlett. "Sparse greedy Gaussian process regression". In: *Advances in neural information processing systems*. 2001, pp. 619–625.

[3] Ralf Herbrich, Neil D Lawrence, and Matthias Seeger. "Fast sparse Gaussian process methods: The informative vector machine". In: *Advances in neural information processing systems*. 2003, pp. 625–632.

[4] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems*. 2006, pp. 1257–1264.

[5] Michalis Titsias. "Variational learning of inducing variables in sparse Gaussian processes". In: *Artificial Intelligence and Statistics*. 2009, pp. 567–574.

[6] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: (2010).

[7] Michalis Titsias and Neil D Lawrence. “Bayesian Gaussian process latent variable model”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 844–851.

[8] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[9] Andreas Damianou and Neil Lawrence. “Deep gaussian processes”. In: *Artificial Intelligence and Statistics*. 2013, pp. 207–215.

[10] Jaehoon Lee et al. “Deep neural networks as gaussian processes”. In: *arXiv preprint arXiv:1711.00165* (2017).

[11] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. “Convolutional gaussian processes”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 2849–2858.

[12] Alexander G de G Matthews et al. “Gaussian process behaviour in wide deep neural networks”. In: *arXiv preprint arXiv:1804.11271* (2018).

A Gaussian Process Illustration

A.1 Gaussian Process priors

In figure 9 and 10 we samples examples of functions drawn from the prior Gaussian process and plot it. We can clearly outline the influence of the lengthscale and the kernel over the smoothness of the function.

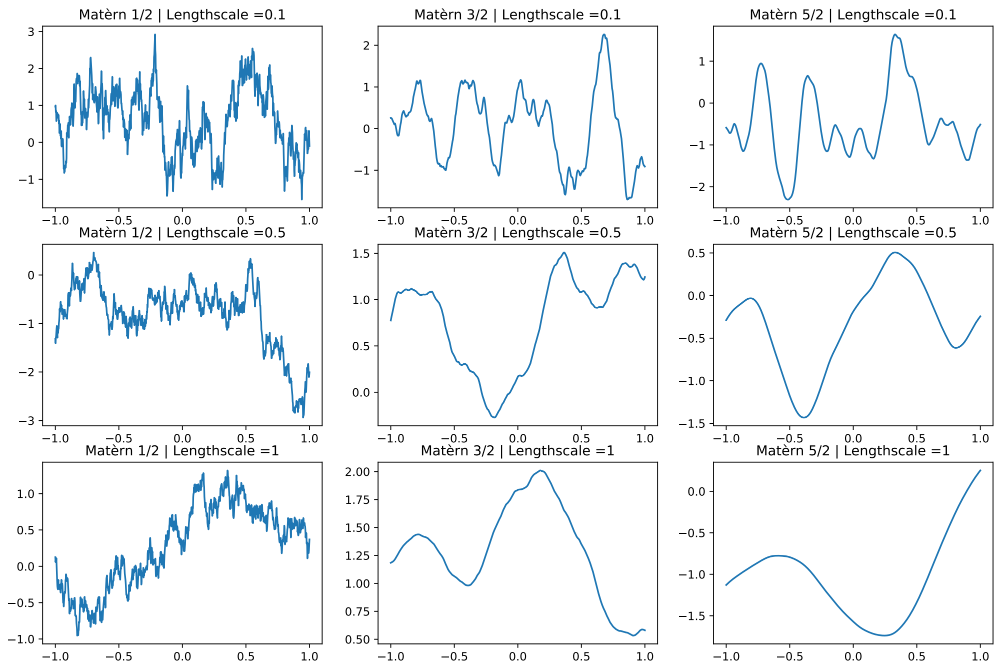

Figure 9: Samples from the Matérn prior for different lengthscale

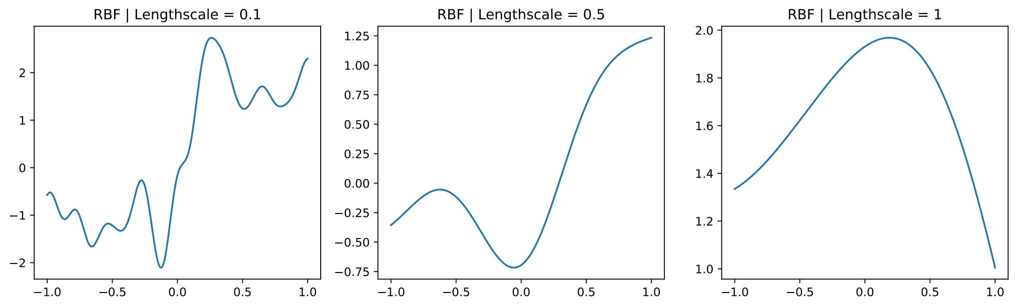

Figure 10: Samples from the RBF prior for different lengthscale

A.2 Gaussian Process Regression

In this section we performed multiple regression tasks with different kernels and hyperparameters. We first fixed the lengthscales of the kernels used and then optimized them according to the parginal likelihood of the observed data. Figure 11 shows some regression according to our studied kernels. Figure 12 shows the results of the regression while optimizing the hyper-parameters (σ and l), we used the scipy L-BFGS-B algorithm. We can notice that the initialization of our model is important for the optimization, in the case of the RBF, we initialized the lengthscale to 0.5 and the optimization process found a local optima for a large lengthscale and large observation noise, which obviously fits the data.

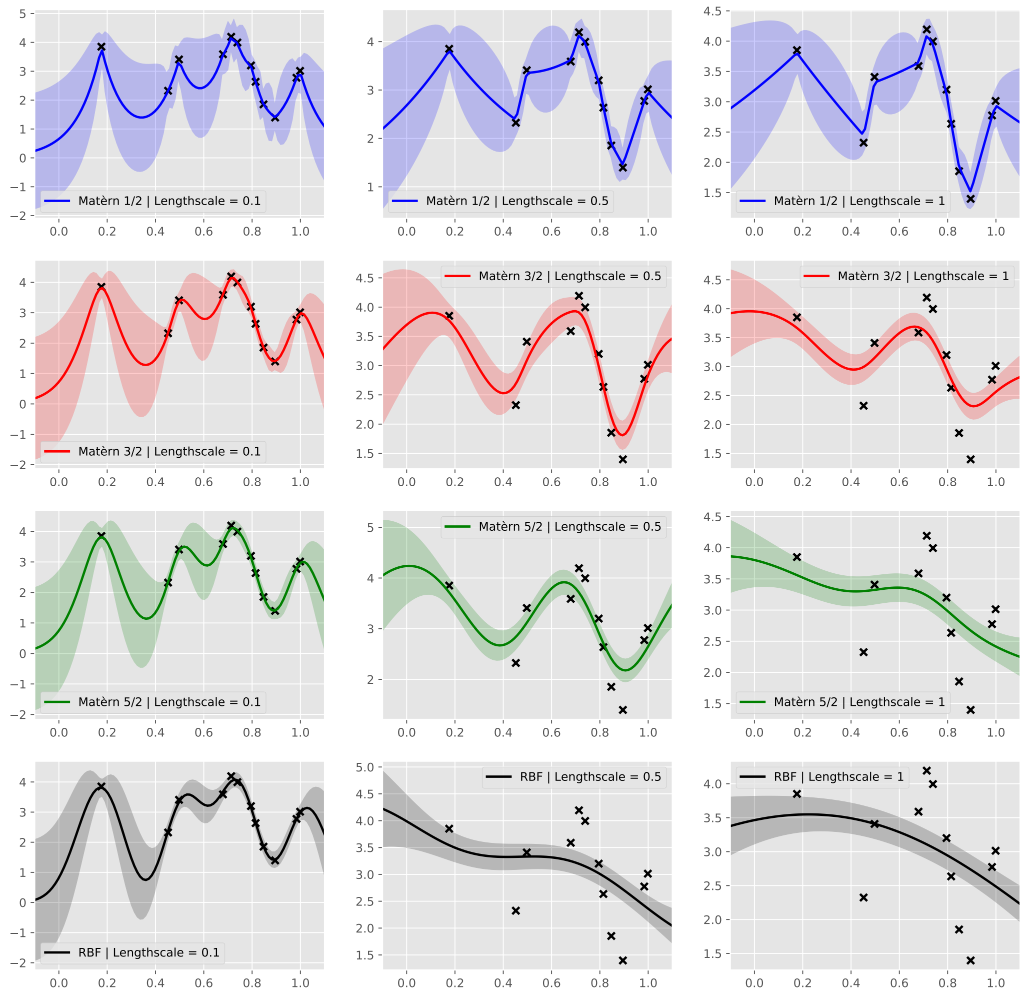

Figure 11: Kernel Regression

A.3 Gaussian Process Unsupervised Learning with GP-LVM

In this subsection, we experimented the GP-LVM model on a subset of the MNIST dataset by Yann Lecun [6]. We took 3 classes, the digit 1, 5 and 9 and 700 random instances of each of these classes. Then we applied the GP-LVM model to perform probabilistic reduction of dimension with 25 inducing points, 2 latent variables and optimized over 200 BFGS steps. We obtained the Figure 13. On this figure, we can see that the labels are clearly separated which means the model successfully captured the relevant digit characteristics. Although the latent space is relevant, it does not seem that

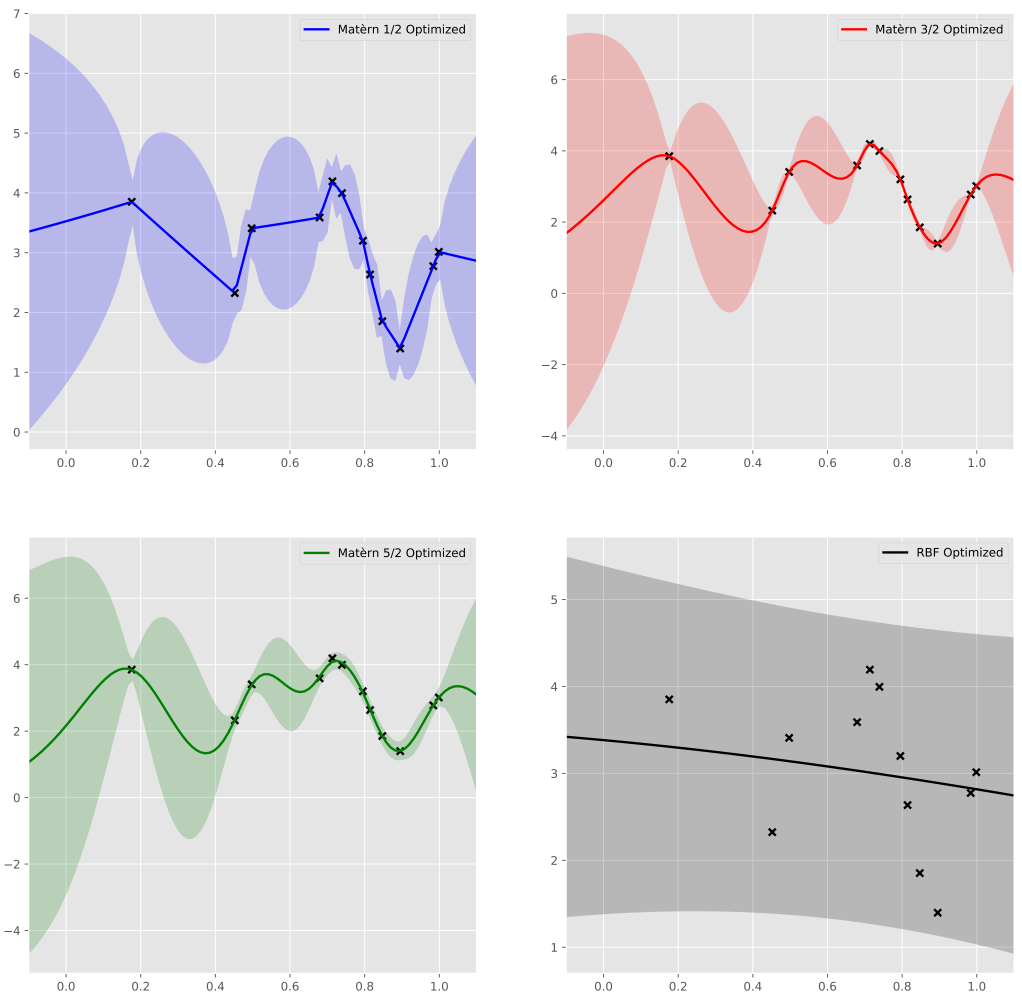

Figure 12: Optimized Kernel Regression

it encapsulates interpretable underlying mechanism, the x-axis seems to be somehow responsible for rotation, size and background color shape while the y-axis seems for responsible for contour shape.

B Deep Gaussian Process Experiments

B.1 Deep Gaussian Process priors

In this experiment we sampled functions from a two-layer deep Gaussian process where the dimension of every layer is one. For both Gaussian processes, we took a RBF covariance function with lengthscale $l = 0.2$. We first sampled the first layer and then sampled 10 functions from the output layers. We plotted both functions with regards to the input $[-1, 1]$. The results is shown Figure 14. From the output plot we can see that the 2-layer GP is able to generate function with complex structure. Taking the analogy with a RBF Gaussian process, the modeled functions seems to have a varying lengthscale accross the output and present some non-trivial long-term dependency contrary to a simple Gaussian process.

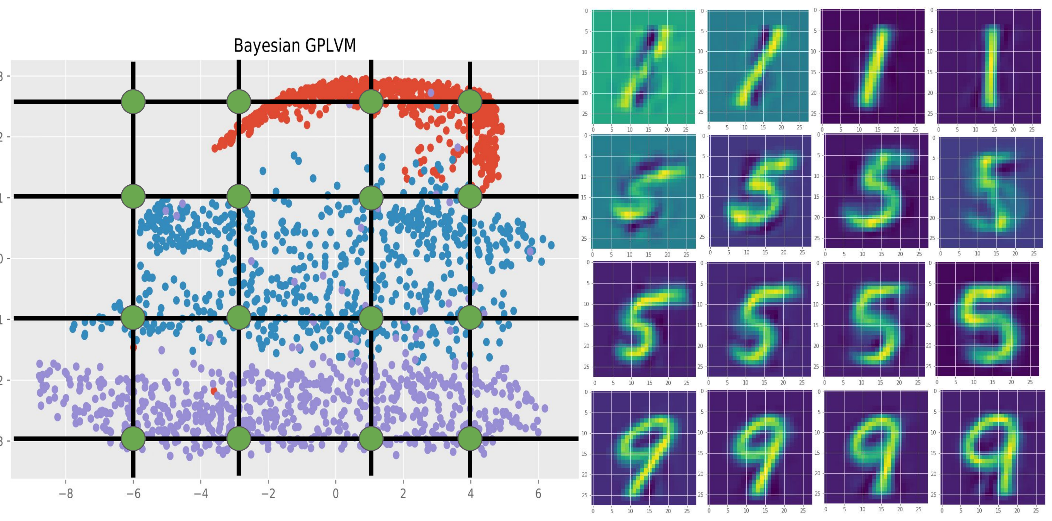

Figure 13: GP-LVM unsupervised learning and samples drawn from the latent space

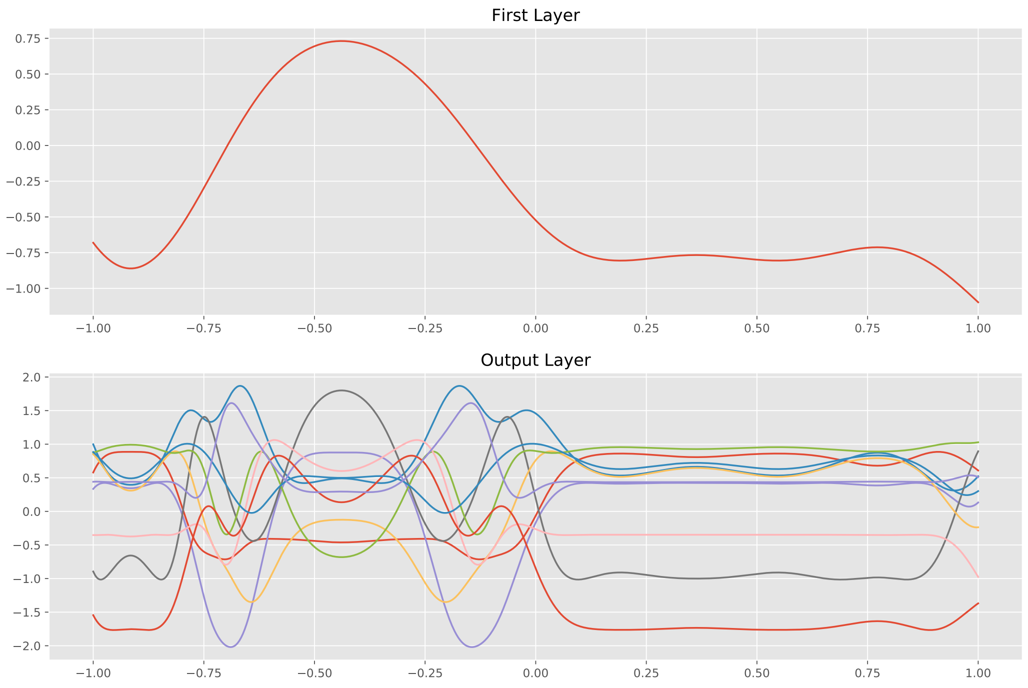

Figure 14: Samples from a 2-layers Deep-Gaussian process