

Sparse Orthogonal Variational Inference for Gaussian Processes

Jiaxin Shi
Tsinghua University

Michalis K. Titsias
DeepMind

Andriy Mnih
DeepMind

Abstract

We introduce a new interpretation of sparse variational approximations for Gaussian processes using inducing points, which can lead to more scalable algorithms than previous methods. It is based on decomposing a Gaussian process as a sum of two independent processes: one spanned by a finite basis of inducing points and the other capturing the remaining variation. We show that this formulation recovers existing approximations and at the same time allows to obtain tighter lower bounds on the marginal likelihood and new stochastic variational inference algorithms. We demonstrate the efficiency of these algorithms in several Gaussian process models ranging from standard regression to multi-class classification using (deep) convolutional Gaussian processes and report state-of-the-art results on CIFAR-10 among purely GP-based models.

1 INTRODUCTION

Gaussian processes (GP) (Rasmussen and Williams, 2006) are nonparametric models for representing distributions over functions, which can be seen as a generalization of multivariate Gaussian distributions to infinite dimensions. The simplicity and elegance of these models has led to their wide adoption in uncertainty estimation for machine learning, including supervised learning (Williams and Rasmussen, 1996; Williams and Barber, 1998), sequential decision making (Srinivas et al., 2010), model-based planning (Deisenroth and Rasmussen, 2011), and unsupervised data analysis (Lawrence, 2005; Damianou et al., 2016).

Despite the successful application of these models,

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

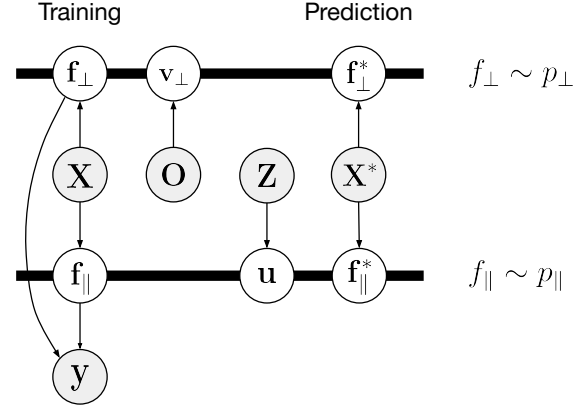


Figure 1: The graphical model of SOLVE-GP. The prior $f \sim \mathcal{GP}(0, k)$ is decomposed into two independent GPs (denoted by thick horizontal lines): $f_{\parallel} \sim p_{\parallel}$ and $f_{\perp} \sim p_{\perp}$. The variables connected by thick lines form a multivariate Gaussian. \mathbf{X}, \mathbf{y} denote the training data. \mathbf{X}^* are the test inputs. $\mathbf{f}_{\parallel} = f_{\parallel}(\mathbf{X})$, $\mathbf{f}_{\perp} = f_{\perp}(\mathbf{X})$. $\mathbf{u} = f_{\parallel}(\mathbf{Z})$ denote the inducing variables in standard SVGP methods. SOLVE-GP introduces another set of inducing variables $\mathbf{v}_{\perp} = f_{\perp}(\mathbf{O})$ to summarize p_{\perp} .

they suffer from $\mathcal{O}(N^3)$ computation and $\mathcal{O}(N^2)$ storage requirements given N training data points, which has motivated a large body of research on sparse GP methods (Csato and Opper, 2002; Lawrence et al., 2002; Seeger et al., 2003; Quiñonero-Candela and Rasmussen, 2005a; Titsias, 2009; Hensman et al., 2013; Bui et al., 2017). GPs have also been unfavourably compared to deep learning models for lacking representation learning capabilities.

Sparse variational GP (SVGP) methods (Titsias, 2009; Hensman et al., 2013, 2015a) based on variational learning of inducing points have shown promise in addressing these limitations. Such methods leave the prior distribution of the GP model unchanged and instead enforce sparse structures in the posterior approximation through variational inference. This gives $\mathcal{O}(M^2N + M^3)$ computation and $\mathcal{O}(MN + M^2)$ storage with M inducing points. Moreover, they allow us to perform mini-batch training by sub-sampling data

points. Successful application of SVGP allowed scalable GP models trained on billions of data points (Salimbeni and Deisenroth, 2017). These advances in inference methods have also led to more flexibility in model design. A recent convolutional GP model (van der Wilk et al., 2017) encodes translation invariance by summing over GPs that take image patches as inputs. The inducing points, which can be interpreted as image patches in this model, play a role similar to that of convolutional filters in neural networks. Their work showed that it is possible to implement representation learning in GP models. Further extensions of such models into deep hierarchies (Blomqvist et al., 2018; Dutordoir et al., 2019) significantly boosted the performance of GPs for natural images.

As these works suggest, currently the biggest challenge in this area still lies in scalable inference. The computational cost of the widely used SVGP methods scales cubically with the number of inducing points, making it difficult to improve the flexibility of posterior approximations (Shi et al., 2019). For example, state-of-the-art models like deep convolutional GPs use only 384 inducing points for inference in each layer to get a manageable running time (Dutordoir et al., 2019).

We introduce a new framework, called SOLVE-GP, which allows increasing the number of inducing points given a fixed computational budget. It is based on decomposing the GP prior as the sum of a low-rank approximation using inducing points, and a full-rank residual process. We observe that the standard SVGP methods can be reinterpreted under such decomposition. By introducing another set of inducing variables for the orthogonal complement, we can increase the number of inducing points at a much lower additional computational cost. With our method doubling the number of inducing points leads to a 2-fold increase in the cost of Cholesky decomposition, compared to the 8-fold increase for the original SVGP method. We show that SOLVE-GP is equivalent to a structured covariance approximation for SVGP defined over the union of the two sets of inducing points. Interestingly, under such interpretation our work can be seen as a generalization of the recently proposed decoupled-inducing-points method (Salimbeni et al., 2018). As the decoupled method often comes with a complex dual formation, our framework provides a simpler derivation and more intuitive understanding for it.

We conducted experiments on convolutional GPs and their deep variants. To the best of our knowledge, we are the first to train a purely GP-based model without any neural network components to achieve over 80% test accuracy on CIFAR-10. No data augmentation was used to obtain these results. Besides clas-

sification, we also evaluated our method on a range of regression datasets that range in size from tens of thousands to millions of data points. Our results show that SOLVE-GP is often competitive with the more expensive SVGP counterpart that uses the same number of inducing points, and outperforms SVGP when given the same computational budget.

2 BACKGROUND

Here, we briefly review Gaussian processes and sparse variational GP methods. A GP is an uncountable collection of random variables indexed by a real-valued vector \mathbf{x} taking values in $\mathcal{X} \subset \mathbb{R}^d$, of which any finite subset has a multivariate Gaussian distribution. A GP is defined by a mean function $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ and a covariance function $k(\mathbf{x}, \mathbf{x}') = \text{Cov}[f(\mathbf{x}), f(\mathbf{x}')] :$

$$f \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')).$$

Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times d}$ be (the matrix containing) the training data points and $\mathbf{f} = f(\mathbf{X}) \in \mathbb{R}^N$ denote the corresponding function values. Similarly we denote the test data points by \mathbf{X}^* and their function values by \mathbf{f}^* . Assuming a zero mean function, the joint distribution over \mathbf{f}, \mathbf{f}^* is given by:

$$p(\mathbf{f}, \mathbf{f}^*) := \mathcal{N} \left(\begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \middle| \mathbf{0}, \begin{bmatrix} \mathbf{K}_{\mathbf{ff}} & \mathbf{K}_{\mathbf{ff}^*} \\ \mathbf{K}_{\mathbf{f}^*\mathbf{f}} & \mathbf{K}_{\mathbf{f}^*\mathbf{f}^*} \end{bmatrix} \right),$$

where $\mathbf{K}_{\mathbf{ff}}$ is an $N \times N$ kernel matrix with its (i, j) th entry as $k(\mathbf{x}_i, \mathbf{x}_j)$, and similarly $[\mathbf{K}_{\mathbf{ff}^*}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j^*)$, $[\mathbf{K}_{\mathbf{f}^*\mathbf{f}}]_{ij} = k(\mathbf{x}_i^*, \mathbf{x}_j)$. In practice we often observe the training function values through some noisy measurements \mathbf{y} , generated by the likelihood function $p(\mathbf{y}|\mathbf{f})$. For regression, the likelihood usually models independent Gaussian observation noise: $y_n = f_n + \epsilon_n$, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$. In this situation the exact posterior distribution $p(\mathbf{f}^*|\mathbf{y})$ can be computed in closed form:

$$\mathbf{f}^*|\mathbf{y} \sim \mathcal{N}(\mathbf{K}_{\mathbf{f}^*\mathbf{f}}(\mathbf{K}_{\mathbf{ff}} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \mathbf{K}_{\mathbf{f}^*\mathbf{f}^*} - \mathbf{K}_{\mathbf{f}^*\mathbf{f}}(\mathbf{K}_{\mathbf{ff}} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{\mathbf{ff}^*}). \quad (1)$$

As seen from Eq. (1), exact prediction involves the inverse of matrix $\mathbf{K}_{\mathbf{ff}} + \sigma^2\mathbf{I}$, which requires $\mathcal{O}(N^3)$ computation. For large datasets, we need to avoid the cubic complexity by resorting to approximations.

Inducing points have played a central role in previous works on scalable GP inference. The general idea is to summarize \mathbf{f} with a small number of variables $\mathbf{u} = f(\mathbf{Z})$, where $\mathbf{Z} = [\mathbf{z}_1, \dots, \mathbf{z}_M]^\top \in \mathbb{R}^{M \times d}$ is a set of parameters, called inducing points, in the input space. The augmented joint distribution over $\mathbf{u}, \mathbf{f}, \mathbf{f}^*$ is $p(\mathbf{f}, \mathbf{f}^*|\mathbf{u})p(\mathbf{u})$, where $p(\mathbf{u}) = \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{uu}})$ and $\mathbf{K}_{\mathbf{uu}}$ denotes the kernel matrix of inducing points with the (i, j) th entry corresponding to $k(\mathbf{z}_i, \mathbf{z}_j)$. There is a

long history of developing sparse approximations for GPs by making different independence assumptions for the conditional distribution $p(\mathbf{f}, \mathbf{f}^* | \mathbf{u})$ to reduce the computational cost (Quiñero-Candela and Rasmussen, 2005b). However, these methods made modifications to the GP prior and tended to suffer from degeneracy and overfitting problems.

Sparse variational GP methods (SVGP), first proposed in Titsias (2009) and later extended for mini-batch training and non-conjugate likelihoods (Hensman et al., 2013, 2015a), provide an elegant solution to these problems. By reformulating the posterior inference problem as variational inference and restricting the variational distribution to be $q(\mathbf{f}, \mathbf{f}^*, \mathbf{u}) := q(\mathbf{u})p(\mathbf{f}, \mathbf{f}^* | \mathbf{u})$, the variational lower bound for minimizing $\text{KL}[q(\mathbf{f}, \mathbf{f}^*, \mathbf{u}) \| p(\mathbf{f}, \mathbf{f}^*, \mathbf{u} | \mathbf{y})]$ simplifies to:

$$\sum_{n=1}^N \mathbb{E}_{q(\mathbf{u})p(\mathbf{f}_n | \mathbf{u})} [\log p(y_n | \mathbf{f}_n)] - \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})]. \quad (2)$$

For GP regression the bound has a collapsed form obtained by solving for the optimal $q(\mathbf{u})$ and plugging it into (2) (Titsias, 2009):

$$\log \mathcal{N}(\mathbf{y} | \mathbf{0}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2 \mathbf{I}) - \frac{1}{2\sigma^2} \text{tr}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}}), \quad (3)$$

where $\mathbf{Q}_{\mathbf{ff}} = \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{K}_{\mathbf{uf}}$. Computing this objective requires $\mathcal{O}(M^2 N + M^3)$ operations, in contrast to the $\mathcal{O}(N^3)$ complexity of exact inference. The inducing points \mathbf{Z} can be learned as variational parameters by maximizing the lower bound. More generally, if we do not collapse $q(\mathbf{u})$ and let $q(\mathbf{u}) = \mathcal{N}(\mathbf{m}_{\mathbf{u}}, \mathbf{S}_{\mathbf{u}})$, where $\mathbf{m}_{\mathbf{u}}, \mathbf{S}_{\mathbf{u}}$ are trainable parameters, we can use the uncollapsed bound for mini-batch training and non-Gaussian likelihoods (Hensman et al., 2013, 2015a).

3 SOLVE-GP

Despite the success of SVGP methods, their $\mathcal{O}(M^3)$ complexity makes it difficult for the flexibility of posterior approximation to grow with the dataset size. We present a new framework called *Sparse Orthogonal Variational infErnce for Gaussian Processes* (SOLVE-GP), which allows the use of an additional set of inducing points at a lower computational cost than the standard SVGP methods.

3.1 Reinterpreting SVGP

We start by reinterpreting SVGP methods using a simple reparameterization, which will then lead us to possible ways of improving the approximation. First we notice that the covariance of the conditional distribution $p(\mathbf{f} | \mathbf{u}) = \mathcal{N}(\mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}, \mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})$ does not de-

pend on \mathbf{u} .¹ Therefore, samples from $p(\mathbf{f} | \mathbf{u})$ can be reparameterized as

$$\begin{aligned} \mathbf{f}_{\perp} &\sim p_{\perp}(\mathbf{f}_{\perp}) := \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}}), \\ \mathbf{f} &= \mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}. \end{aligned} \quad (4)$$

The reason for denoting the zero-mean component as \mathbf{f}_{\perp} shall become clear later. Now we can reparameterize the augmented prior distribution $p(\mathbf{f}, \mathbf{u})$ as

$$\mathbf{u} \sim p(\mathbf{u}), \quad \mathbf{f}_{\perp} \sim p_{\perp}(\mathbf{f}_{\perp}), \quad \mathbf{f} = \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u} + \mathbf{f}_{\perp}, \quad (5)$$

and the joint distribution of the GP model becomes

$$p(\mathbf{y}, \mathbf{u}, \mathbf{f}_{\perp}) = p(\mathbf{y} | \mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}) p(\mathbf{u}) p_{\perp}(\mathbf{f}_{\perp}). \quad (6)$$

Posterior inference for \mathbf{f} in the original model then turns into inference for \mathbf{u} and \mathbf{f}_{\perp} . If we approximate the above GP model by considering a factorised approximation $q(\mathbf{u})p_{\perp}(\mathbf{f}_{\perp})$, where $q(\mathbf{u})$ is a variational distribution and $p_{\perp}(\mathbf{f}_{\perp})$ is the prior distribution of \mathbf{f}_{\perp} that appears also in Eq. (6), we arrive at the standard SVGP method. To see this, note that minimizing $\text{KL}[q(\mathbf{u})p_{\perp}(\mathbf{f}_{\perp}) \| p(\mathbf{u}, \mathbf{f}_{\perp} | \mathbf{y})]$ is equivalent to maximizing the variational lower bound

$\mathbb{E}_{q(\mathbf{u})p_{\perp}(\mathbf{f}_{\perp})} \log p(\mathbf{y} | \mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}) - \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})]$, which is the SVGP objective (Eq. (2)) using the reparameterization in Eq. (4).

Under this interpretation of the standard SVGP method, it becomes clear that we can modify the form of the variational distribution $q(\mathbf{u})p_{\perp}(\mathbf{f}_{\perp})$ to improve the accuracy of the posterior approximation. There are two natural options: (i) keep $p_{\perp}(\mathbf{f}_{\perp})$ as part of the approximation and alter $q(\mathbf{u})$ so that it will have some dependence on \mathbf{f}_{\perp} , and (ii) keep $q(\mathbf{u})$ independent from \mathbf{f}_{\perp} , and replace $p_{\perp}(\mathbf{f}_{\perp})$ with a more structured variational distribution $q(\mathbf{f}_{\perp})$. While both options lead to new bounds and more accurate approximations than the standard method, we will defer the discussion of (i) to appendix A and focus on (ii) because it is amenable to large-scale training, as we will show next.

3.2 Orthogonal Decomposition

As suggested in section 3.1, we consider improving the variational distribution for \mathbf{f}_{\perp} . However, the complexity of inferring \mathbf{f}_{\perp} is the same as for \mathbf{f} and thus cubic. Resolving the problem requires a better understanding of the reparameterization we used in section 3.1.

The key observation here is that the reparameterization in Eq. (5) corresponds to an orthogonal decomposition in the function space. For simplicity,

¹Note that kernel matrices like $\mathbf{K}_{\mathbf{uu}}$ depend on \mathbf{Z} instead of \mathbf{u} ; the subscript only indicates that this is the covariance matrix of \mathbf{u} .

we first derive such decomposition in the Reproducing Kernel Hilbert Space (RKHS) induced by k , and then generalize the result to the GP sample space. The RKHS with kernel k is the closure of the space $\{\sum_{i=1}^{\ell} c_i k(\mathbf{x}'_i, \cdot), c_i \in \mathbb{R}, \ell \in \mathbb{N}^+, \mathbf{x}'_i \in \mathcal{X}\}$, with the inner product defined as $\langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} = f(\mathbf{x}), \forall f \in \mathcal{H}$. Let V denote the linear span of the kernel basis functions indexed by the inducing points: $V := \{\sum_{j=1}^M \alpha_j k(\mathbf{z}_j, \cdot), \boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_M]^\top \in \mathbb{R}^M\}$. For any function $f \in \mathcal{H}$, we can decompose it (Cheng and Boots, 2016) as

$$f = f_{\parallel} + f_{\perp}, \quad f_{\parallel} \in V \text{ and } f_{\perp} \perp V,$$

Assuming $f_{\parallel} = \sum_{j=1}^M \alpha'_j k(\mathbf{z}_j, \cdot)$, then we can solve for the coefficients (details in appendix B): $\boldsymbol{\alpha}' = k(\mathbf{Z}, \mathbf{Z})^{-1} f(\mathbf{Z})$, where $k(\mathbf{Z}, \mathbf{Z})$ denotes the kernel matrix of \mathbf{Z} . Therefore,

$$f_{\parallel}(\mathbf{x}) = k(\mathbf{x}, \mathbf{Z}) k(\mathbf{Z}, \mathbf{Z})^{-1} f(\mathbf{Z}), \quad f_{\perp} = f - f_{\parallel}. \quad (7)$$

Here $k(\mathbf{x}, \mathbf{Z}) := [k(\mathbf{z}_1, \mathbf{x}), \dots, k(\mathbf{z}_M, \mathbf{x})]$. Although Eq. (7) is derived by assuming $f \in \mathcal{H}$, it motivates us to study the same decomposition for $f \sim \mathcal{GP}(0, k)$. Then f_{\parallel} becomes $k(\cdot, \mathbf{Z}) \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}$. Interestingly, we can verify that this is a sample from a GP with a zero mean function and covariance function $\text{Cov}[f_{\parallel}(\mathbf{x}), f_{\parallel}(\mathbf{x}')] = k(\mathbf{x}, \mathbf{Z}) \mathbf{K}_{\mathbf{uu}}^{-1} k(\mathbf{Z}, \mathbf{x}')$. Similarly we can show that f_{\perp} is a sample from another GP and we denote these two independent GPs as p_{\parallel} and p_{\perp} (Hensman et al., 2017):

$$\begin{aligned} f_{\parallel} &\sim p_{\parallel} \equiv \mathcal{GP}(0, k(\mathbf{x}, \mathbf{Z}) \mathbf{K}_{\mathbf{uu}}^{-1} k(\mathbf{Z}, \mathbf{x}')), \\ f_{\perp} &\sim p_{\perp} \equiv \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{Z}) \mathbf{K}_{\mathbf{uu}}^{-1} k(\mathbf{Z}, \mathbf{x}')). \end{aligned}$$

Marginalizing out the GPs at the training points \mathbf{X} , it is easy to show that

$$\begin{aligned} \mathbf{f}_{\parallel} &= f_{\parallel}(\mathbf{X}) = \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{K}_{\mathbf{uf}}), \\ \mathbf{f}_{\perp} &= f_{\perp}(\mathbf{X}) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\mathbf{ff}} - \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{K}_{\mathbf{uf}}). \end{aligned}$$

This is exactly the decomposition we used in section 3.1, and the meaning of \mathbf{f}_{\perp} becomes clear.

3.3 SOLVE-GP Lower Bound

The decomposition described in the previous section gives new insights for improving the variational distribution for \mathbf{f}_{\perp} . Specifically, we can introduce a second set of inducing variables $\mathbf{v}_{\perp} := f_{\perp}(\mathbf{O})$ to approximate p_{\perp} , as illustrated in Fig. 1. We call this second set $\mathbf{O} = [\mathbf{o}_1, \dots, \mathbf{o}_{M_2}]^\top \in \mathbb{R}^{M_2 \times d}$ the *orthogonal* inducing points. The joint model distribution is then

$$p(\mathbf{y} | \mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}) p(\mathbf{u}) p_{\perp}(\mathbf{f}_{\perp} | \mathbf{v}_{\perp}) p_{\perp}(\mathbf{v}_{\perp}).$$

First notice that the standard SVGP methods correspond to using the variational distribution

$q(\mathbf{u}) p_{\perp}(\mathbf{v}_{\perp}) p_{\perp}(\mathbf{f}_{\perp} | \mathbf{v}_{\perp})$. To obtain better approximations we can replace the prior factor $p_{\perp}(\mathbf{v}_{\perp})$ with a tunable variational factor $q(\mathbf{v}_{\perp}) := \mathcal{N}(\mathbf{m}_{\mathbf{v}}, \mathbf{S}_{\mathbf{v}})$:

$$q(\mathbf{u}, \mathbf{f}_{\perp}, \mathbf{v}_{\perp}) = q(\mathbf{u}) q(\mathbf{v}_{\perp}) p_{\perp}(\mathbf{f}_{\perp} | \mathbf{v}_{\perp}).$$

This gives the SOLVE-GP variational lower bound:

$$\begin{aligned} &\mathbb{E}_{q(\mathbf{u}) q_{\perp}(\mathbf{f}_{\perp})} [\log p(\mathbf{y} | \mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u})] \\ &- \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})] - \text{KL}[q(\mathbf{v}_{\perp}) \| p_{\perp}(\mathbf{v}_{\perp})], \quad (8) \end{aligned}$$

where $q_{\perp}(\cdot) := \int p_{\perp}(\cdot | \mathbf{v}_{\perp}) q(\mathbf{v}_{\perp}) d\mathbf{v}_{\perp}$ is the variational predictive distribution for p_{\perp} . Simple computations show that $q_{\perp}(\mathbf{f}_{\perp}) = \mathcal{N}(\mathbf{C}_{\mathbf{fv}} \mathbf{C}_{\mathbf{vv}}^{-1} \mathbf{m}_{\mathbf{v}}, \mathbf{S}_{\mathbf{f}_{\perp}})$, where $\mathbf{S}_{\mathbf{f}_{\perp}} = \mathbf{C}_{\mathbf{ff}} + \mathbf{C}_{\mathbf{fv}} \mathbf{C}_{\mathbf{vv}}^{-1} (\mathbf{S}_{\mathbf{v}} - \mathbf{C}_{\mathbf{vv}}) \mathbf{C}_{\mathbf{vv}}^{-1} \mathbf{C}_{\mathbf{vf}}$. Here $\mathbf{C}_{\mathbf{ff}} := \mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}}$ is the covariance matrix of p_{\perp} on the training inputs and similarly for the other matrices. Because the likelihood factorizes given \mathbf{f} (i.e., $\mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{fu}} \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u}$), the first term of Eq. (8) simplifies to $\sum_{n=1}^N \mathbb{E}_{q(\mathbf{u}) q_{\perp}(\mathbf{f}_{\perp}(\mathbf{x}_n))} [\log p(y_n | \mathbf{f}_{\perp}(\mathbf{x}_n) + k(\mathbf{x}_n, \mathbf{Z}) \mathbf{K}_{\mathbf{uu}}^{-1} \mathbf{u})]$. Therefore, we only need to compute marginals of $q_{\perp}(\mathbf{f}_{\perp})$ at individual data points. In the general setting, the SOLVE-GP lower bound can be maximized in $\mathcal{O}(N \bar{M}^2 + \bar{M}^3)$ time per gradient update, where $\bar{M} = \max(M, M_2)$. In mini-batch training N is replaced by the batch size. The predictive density at test data points can be found in appendix D.

To intuitively understand the improvement over the standard SVGP methods, we derive a collapsed bound for GP regression using (8) and compare it to the Titsias (2009) bound. Plugging in the optimal $q(\mathbf{u})$, and simplifying (see appendix C), gives the bound

$$\begin{aligned} &\log \mathcal{N}(\mathbf{y} | \mathbf{C}_{\mathbf{fv}} \mathbf{C}_{\mathbf{vv}}^{-1} \mathbf{m}_{\mathbf{v}}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2 \mathbf{I}) - \frac{1}{2\sigma^2} \text{tr}(\mathbf{S}_{\mathbf{f}_{\perp}}) \\ &- \text{KL}[\mathcal{N}(\mathbf{m}_{\mathbf{v}}, \mathbf{S}_{\mathbf{v}}) \| \mathcal{N}(\mathbf{0}, \mathbf{C}_{\mathbf{vv}})]. \quad (9) \end{aligned}$$

With an appropriate choice of $q(\mathbf{v}_{\perp})$ this bound can be tighter than the Titsias (2009) bound. For example, notice that when $q(\mathbf{v}_{\perp})$ is equal to the prior $p_{\perp}(\mathbf{v}_{\perp})$, i.e., $\mathbf{m}_{\mathbf{v}} = \mathbf{0}$ and $\mathbf{S}_{\mathbf{v}} = \mathbf{C}_{\mathbf{vv}}$, the bound in (9) reduces to the one in (3). Another interesting special case arises when the variational distribution has the same covariance matrix as the prior (i.e., $\mathbf{S}_{\mathbf{v}} = \mathbf{C}_{\mathbf{vv}}$), while the mean $\mathbf{m}_{\mathbf{v}}$ is learnable. Then the bound becomes

$$\begin{aligned} &\log \mathcal{N}(\mathbf{y} | \mathbf{C}_{\mathbf{fv}} \mathbf{C}_{\mathbf{vv}}^{-1} \mathbf{m}_{\mathbf{v}}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2 \mathbf{I}) \\ &- \frac{1}{2\sigma^2} \text{tr}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}}) - \frac{1}{2} \mathbf{m}_{\mathbf{v}}^\top \mathbf{C}_{\mathbf{vv}}^{-1} \mathbf{m}_{\mathbf{v}}. \quad (10) \end{aligned}$$

Here we see that the second set of inducing variables \mathbf{v}_{\perp} mostly determines the mean prediction over \mathbf{y} , which is zero in the Titsias (2009) bound (Eq. (3)).

Our method introduces another set of inducing points to improve the variational approximation. One natural question to ask is, how does this compare to the

standard SVGP algorithm with the inducing points chosen to be union of the two sets? We answer it as follows: 1) Given the same number of inducing points, SOLVE-GP is more computationally efficient than the standard SVGP method; 2) SOLVE-GP can be interpreted as using a structured covariance in the variational approximation for SVGP.

Computational Benefits. For a quick comparison, we analyze the cost of the Cholesky decomposition in both methods. We assume the time complexity of decomposing an $M \times M$ matrix is cM^3 , where c is constant w.r.t. M . For SOLVE-GP, to compute the inverse and the determinant of $\mathbf{K}_{\mathbf{u}\mathbf{u}}$ and $\mathbf{C}_{\mathbf{v}\mathbf{v}}$, we need the Cholesky factors of them, which cost $c(M^3 + M_2^3)$. For SVGP with M inducing points, we need the Cholesky factor of $\mathbf{K}_{\mathbf{u}\mathbf{u}}$, which costs cM^3 . Adding another M inducing points in SVGP leads to an 8-fold increase (i.e., from cM^3 to $8cM^3$) in the cost of the Cholesky decomposition, compared to the 2-fold increase if we switch to SOLVE-GP with $M_2 = M$ orthogonal inducing points. A more rigorous analysis is given in appendix D, where we enumerate all the cubic-cost operations needed when we compute the bound.

Structured Covariance. We can express our variational approximation w.r.t. the original GP. Let $\mathbf{v} = f(\mathbf{O})$ denote the function outputs at the orthogonal inducing points. We then have the following relationship between \mathbf{u}, \mathbf{v} and $\mathbf{u}, \mathbf{v}_\perp$:

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v}_\perp \end{bmatrix}.$$

Therefore, the joint variational distribution over \mathbf{u} and \mathbf{v} that corresponds to the factorized $q(\mathbf{u})q(\mathbf{v}_\perp)$ is also Gaussian. By change-of-variable we can express it as $q(\mathbf{u}, \mathbf{v}) = \mathcal{N}(\mathbf{m}_{\mathbf{u},\mathbf{v}}, \mathbf{S}_{\mathbf{u},\mathbf{v}})$, where $\mathbf{m}_{\mathbf{u},\mathbf{v}} = [\mathbf{m}_{\mathbf{u}}, \mathbf{m}_{\mathbf{v}} + \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{m}_{\mathbf{u}}]^\top$ and

$$\mathbf{S}_{\mathbf{u},\mathbf{v}} = \begin{bmatrix} \mathbf{S}_{\mathbf{u}} & \mathbf{S}_{\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u}\mathbf{v}} \\ \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{S}_{\mathbf{u}} & \mathbf{S}_{\mathbf{v}} + \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{S}_{\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u}\mathbf{v}} \end{bmatrix}.$$

From $\mathbf{S}_{\mathbf{u},\mathbf{v}}$ we can see that our approach is different from making the mean-field assumption $q(\mathbf{u}, \mathbf{v}) = q(\mathbf{u})q(\mathbf{v})$, instead it captures the covariance between \mathbf{u}, \mathbf{v} through a structured parameterization.

4 EXTENSIONS

One direct extension of SOLVE-GP involves using more than two sets of inducing points by repeatedly applying the decomposition. However, this adds more complexity to the implementation. Below we show that the SOLVE-GP framework can be easily extended to different GP models where the standard SVGP method applies.

Inter-domain and Convolutional GPs. Similar to SVGP methods, SOLVE-GP can deal with inter-domain inducing points (Lázaro-Gredilla and Figueiras-Vidal, 2009) which lie in a different domain from the input space. The inducing variables \mathbf{u} , which we used to represent outputs of the GP at the inducing points, are now defined as $\mathbf{u} = g(\mathbf{Z}) := [g(\mathbf{z}_1), \dots, g(\mathbf{z}_M)]^\top$, where g is a different function from f that takes inputs in the domain of inducing points. In convolutional GPs (van der Wilk et al., 2017), the input domain is the space of images, while the inducing points are in the space of image patches. The convolutional GP function is defined as $f(\mathbf{x}) = \sum_p w_p g(\mathbf{x}^{[p]})$, where $g \sim \mathcal{GP}(0, k_g)$, $\mathbf{x}^{[p]}$ is the p th patch in \mathbf{x} , and $\mathbf{w} = [w_1, \dots, w_P]^\top$ are the assigned weights for different patches. In SOLVE-GP, we can choose either \mathbf{Z}, \mathbf{O} , or both to be inter-domain as long as we can compute the covariance between \mathbf{u}, \mathbf{v} and \mathbf{f} . For convolutional GPs, we let \mathbf{Z} and \mathbf{O} both be collections of image patches. Examples of the covariance matrices we need for this model include $\mathbf{K}_{\mathbf{v}\mathbf{f}}$ and $\mathbf{K}_{\mathbf{v}\mathbf{u}}$ (used for $\mathbf{C}_{\mathbf{v}\mathbf{v}}$). They can be computed as

$$[\mathbf{K}_{\mathbf{v}\mathbf{f}}]_{ij} = \text{Cov}[g(\mathbf{o}_i), f(\mathbf{x}_j)] = \sum_p w_p k_g(\mathbf{o}_i, \mathbf{x}_j^{[p]}),$$

$$[\mathbf{K}_{\mathbf{v}\mathbf{u}}]_{ij} = \text{Cov}[g(\mathbf{o}_i), g(\mathbf{z}_j)] = k_g(\mathbf{o}_i, \mathbf{z}_j).$$

Deep GPs. We show that we can integrate SOLVE-GP with popular doubly stochastic variational inference algorithms for deep GPs (Salimbeni and Deisenroth, 2017). The joint distribution of a deep GP model with inducing variables in all layers is

$$p(\mathbf{y}, \mathbf{f}^{1:L}, \mathbf{u}^{1:L}) = p(\mathbf{y}|\mathbf{f}^L) \prod_{\ell=1}^L [p(\mathbf{f}^\ell|\mathbf{u}^\ell, \mathbf{f}^{\ell-1})p(\mathbf{u}^\ell)],$$

where we define $\mathbf{f}^0 = \mathbf{X}$ and \mathbf{f}^ℓ is the output of the ℓ th-layer GP. The doubly stochastic algorithm applies SVGP methods to each layer conditioned on samples from the variational distribution in the previous layer. The variational distribution over $\mathbf{u}^{1:L}, \mathbf{f}^{1:L}$ is $q(\mathbf{f}^{1:L}, \mathbf{u}^{1:L}) = \prod_{\ell=1}^L [p(\mathbf{f}^\ell|\mathbf{u}^\ell, \mathbf{f}^{\ell-1})q(\mathbf{u}^\ell)]$. This gives a similar objective as in the single layer case (Eq. (2)): $\mathbb{E}_{q(\mathbf{f}^L)} [\log p(\mathbf{y}|\mathbf{f}^L)] - \sum_{\ell=1}^L \text{KL}[q(\mathbf{u}^\ell)||p(\mathbf{u}^\ell)]$, where $q(\mathbf{f}^L) = \int \prod_{\ell=1}^L [p(\mathbf{f}^\ell|\mathbf{u}^\ell, \mathbf{f}^{\ell-1})q(\mathbf{u}^\ell)d\mathbf{u}^\ell] d\mathbf{f}^{1:L-1}$. Extending this using SOLVE-GP is straightforward: we simply introduce orthogonal inducing variables $\mathbf{v}_\perp^{1:L}$ for all layers, which yields the lower bound:

$$\mathbb{E}_{q(\mathbf{u}^L, \mathbf{f}_\perp^L)} [\log p(\mathbf{y}|\mathbf{f}_\perp^L + \mathbf{K}_{\mathbf{f}\mathbf{u}}^L (\mathbf{K}_{\mathbf{u}\mathbf{u}}^L)^{-1} \mathbf{u}^L)] - \sum_{\ell=1}^L \{ \text{KL}[q(\mathbf{u}^\ell)||p(\mathbf{u}^\ell)] + \text{KL}[q(\mathbf{v}_\perp^\ell)||p_\perp(\mathbf{v}_\perp^\ell)] \}. \quad (11)$$

The expression for $q(\mathbf{u}^L, \mathbf{f}_\perp^L)$ is given in appendix E.

5 RELATED WORK

Many approximate algorithms have been proposed to overcome the computational limitations of GPs. The simplest of these are based on subsampling, such as the subset-of-data training (Rasmussen and Williams, 2006) and the Nyström approximation (Williams and Seeger, 2001). Better approximations can be constructed by learning a set of inducing points to summarize the dataset. As mentioned in section 2, these works can be divided into approximations to the GP prior (SoR, DTC, FITC, etc.; Quiñero-Candela and Rasmussen, 2005b), and sparse variational methods (Titsias, 2009; Hensman et al., 2013, 2015a).

Recently there have been many attempts to reduce the computational cost of using a large set of inducing points. A notable line of work (Wilson and Nickisch, 2015; Evans and Nair, 2018; Gardner et al., 2018) involves imposing grid structures on the locations of \mathbf{Z} to perform fast structure-exploiting computations. However, to get such benefits \mathbf{Z} need to be fixed due to the structure constraints, which often suffers from curse of dimensionality in the input space.

Another direction for allowing the use of more inducing points is the decoupled method (Cheng and Boots, 2017), where two different sets of inducing points are used for modeling the mean and the covariance function. This gives linear complexity in the number of mean inducing points which allows using many more of them. Despite the increasing interest in decoupled inducing points (Havasi et al., 2018; Salimbeni et al., 2018), the method has not been well understood due to its complexity. We found that SOLVE-GP is closely related to a recent development of decoupled methods: the orthogonally decoupled variational GP (ODVGP, Salimbeni et al., 2018), as explained next.

Connection with Decoupled Inducing Points.

If we set the β and γ inducing points in ODVGP (Salimbeni et al., 2018) to be \mathbf{Z} and \mathbf{O} , their approach becomes equivalent to using the variational distribution $q'(\mathbf{u}, \mathbf{v}) = \mathcal{N}(\mathbf{m}'_{\mathbf{u}, \mathbf{v}}, \mathbf{S}'_{\mathbf{u}, \mathbf{v}})$, where

$$\mathbf{m}'_{\mathbf{u}, \mathbf{v}} = \begin{bmatrix} \mathbf{m}_{\mathbf{u}} \\ \mathbf{m}_{\mathbf{v}} + \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{m}_{\mathbf{u}} \end{bmatrix}, \quad \mathbf{S}'_{\mathbf{u}, \mathbf{v}} = \begin{bmatrix} \mathbf{S}_{\mathbf{u}} & \mathbf{S}_{\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u}\mathbf{v}} \\ \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{S}_{\mathbf{u}} & \mathbf{K}_{\mathbf{v}\mathbf{v}} + \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}(\mathbf{S}_{\mathbf{u}} - \mathbf{K}_{\mathbf{u}\mathbf{u}})\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u}\mathbf{v}} \end{bmatrix}.$$

By comparing $\mathbf{S}_{\mathbf{u}, \mathbf{v}}$ to $\mathbf{S}'_{\mathbf{u}, \mathbf{v}}$, we can see that we generalize their method by introducing $\mathbf{S}_{\mathbf{v}}$, which replaces the original residual $\mathbf{K}_{\mathbf{v}\mathbf{v}} - \mathbf{K}_{\mathbf{v}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u}\mathbf{v}}$ (or $\mathbf{C}_{\mathbf{v}\mathbf{v}}$), so that we allow more flexible covariance modeling while still keeping the block structure. Thus ODVGP is a special case of SOLVE-GP where $q(\mathbf{v}_{\perp})$ is restricted to have the same covariance $\mathbf{C}_{\mathbf{v}\mathbf{v}}$ as the prior.

6 EXPERIMENTS

Since ODVGP is a special case of SOLVE-GP, we use M, M_2 to refer to $|\beta|$ and $|\gamma|$ in their algorithm, respectively.

6.1 1D Regression

We begin by illustrating our method on Snelson’s 1D regression problem (Snelson and Ghahramani, 2006) with 100 training points and mini-batch size 20. We compare the following methods: SVGP with 5 and 10 inducing points, ODVGP ($M = 5, M_2 = 100$), and SOLVE-GP ($M = 5, M_2 = 5$).

The results are plotted in Fig. 2. First we can see that 5 inducing points are insufficient to summarize the training set: SVGP ($M = 5$) cannot fit data well and underestimates the variance in regions beyond the training data. Increasing M to 10 fixes the issues, but requires 8x more computation for the Cholesky decomposition than using 5 inducing points². The decoupled formulation provides a cheaper alternative and we have tried ODVGP ($M = 5, M_2 = 100$), which has 100 additional inducing points for modeling the mean function. Comparing Fig. 2a and Fig. 2b, we can see that this results in a much better fit for the mean function. However, the model still overestimates the predictive variance. As ODVGP is a special case of the SOLVE-GP framework, we can improve on it in terms of covariance modeling. As seen in Fig. 2c, adding 5 orthogonal inducing points can closely approximate the results of SVGP ($M = 10$), with only a 2-fold increase in the cost of the Cholesky decomposition relative to SVGP ($M = 5$).

6.2 Convolutional GP Models

One class of applications that benefit from the SOLVE-GP framework is the training of large, hierarchical GP models where the true posterior distribution is difficult to approximate with a small number of inducing points. Convolutional GPs (van der Wilk et al., 2017) and their deep variants (Blomqvist et al., 2018; Dutordoir et al., 2019) are such models. There inducing points are feature detectors just like CNN filters, which play a critical role in predictive performance. As explained in section 4, it is straightforward to apply SOLVE-GP to these models.

Convolutional GPs. We train convolutional GPs on the CIFAR-10 dataset, using GPs with TICK kernels (Dutordoir et al., 2019) to define the patch response functions. Table 1 shows the results for SVGP

²In practice the cost is negligible in this toy problem but we are analyzing the theoretical complexity.

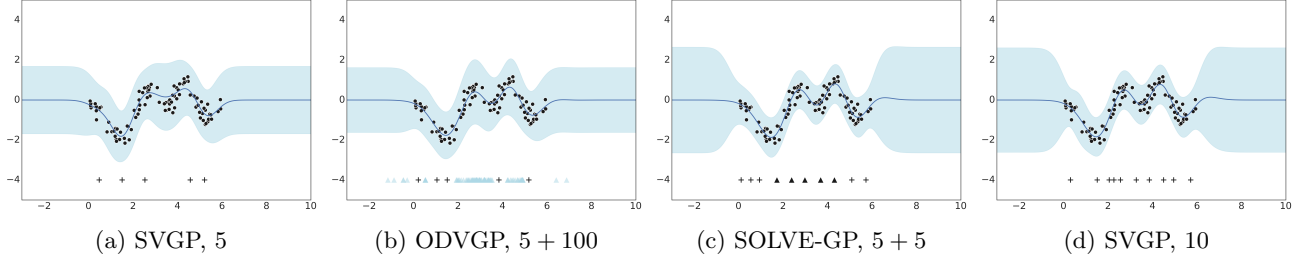


Figure 2: Posterior processes on the Snelson dataset, where shaded bands correspond to intervals of ± 3 standard deviations. The learned inducing locations are shown at the bottom of each figure, where $+$ correspond to \mathbf{Z} ; blue and dark triangles correspond to \mathbf{O} in ODVGP and SOLVE-GP, respectively.

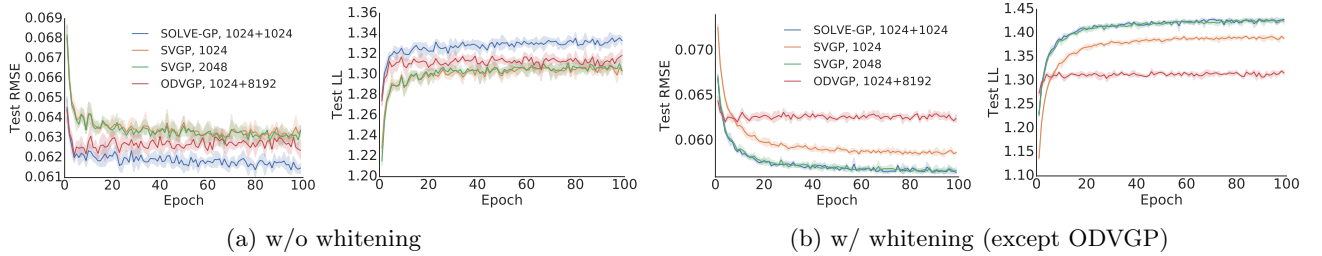


Figure 3: Test RMSE and predictive log-likelihoods during training on HouseElectric.

with 1K and 2K inducing points, SOLVE-GP ($M = 1\text{K}, M_2 = 1\text{K}$), and SVGP ($M = 1.6\text{K}$) that has a similar running time on GPU as SOLVE-GP. Clearly SOLVE-GP outperforms SVGP ($M = 1\text{K}$). It also outperforms SVGP ($M = 1.6\text{K}$), which has the same running time, and performs on par with the more expensive SVGP ($M = 2\text{K}$), which is very encouraging. This suggests that the structured covariance approximation is fairly accurate even for this large, non-conjugate model.

Deep Convolutional GPs. We further extend SOLVE-GP to deep convolutional GPs using the techniques described in section 4. We experiment with 2-layer and 3-layer models that have 1K inducing points in the output layer and 384 inducing points in other layers. The results are summarized in Table 3. These models are already quite slow to train on a single GPU, as indicated by the time per iteration. SOLVE-GP allows to double the number of inducing points in each layer with only a 2-fold increase in computation. This gives superior performance on both accuracy and test predictive likelihoods. The double-size SVGP takes a week to run and is only for comparison purpose.

As shown above, on both single layer and deep convolutional GPs, we improve the state-of-the-art results of CIFAR-10 classification by 3-4 percentage points. This leads to more than 80% accuracy on CIFAR-10 with a purely GP-based model, without any neural network components, closing the gap between GP/kernel regression and CNN baselines presented in Novak et al.

Table 1: Convolutional GPs for CIFAR-10 classification. Previous SOTA is 64.6% by SVGP with 1K inducing points (van der Wilk et al., 2017).

	$M(+M_2)$	Test Acc	Test LL	Time
SVGP	1K	66.07%	-1.59	0.241 s/iter
	1.6K	67.18%	-1.54	0.380 s/iter
SOLVE-GP	1K + 1K	68.19%	-1.51	0.370 s/iter
SVGP	2K	68.06%	-1.48	0.474 s/iter

(2019); Arora et al. (2019). Note that all the results are obtained without data augmentation.

6.3 Regression Benchmarks

Besides classification experiments, we evaluate our method on 10 regression datasets, with size ranging from tens of thousands to millions. The settings are followed from Wang et al. (2019) and described in detail in appendix G. We implemented SVGP with $M = 1024$ & 2048 inducing points, ODVGP and SOLVE-GP ($M = 1024, M_2 = 1024$), as well as SVGP with $M = 1536$ inducing points, which has roughly the same training time per iteration on GPU as the SOLVE-GP objective. An attractive property of ODVGP is that by restricting the covariance of $q(\mathbf{v}_\perp)$ to be the same as the prior covariance $\mathbf{C}_{\mathbf{v}\mathbf{v}}$, it can use far larger M_2 , because the complexity is linear with M_2 by sub-sampling the columns of $\mathbf{K}_{\mathbf{v}\mathbf{v}}$ for each gradient update. Thus for a fair comparison, we also include

Table 2: Test log-likelihood values for the regression datasets. The numbers in parentheses are standard errors. Best mean values are highlighted, and asterisks indicate statistical significance.

		Kin40k	Protein	KeggDirected	KEGGU	3dRoad	Song	Buzz	HouseElectric
	N	25,600	29,267	31,248	40,708	278,319	329,820	373,280	1,311,539
	d	8	9	20	27	3	90	77	9
SVGP	1024	0.094(0.003)	-0.963(0.006)	0.967(0.005)	0.678(0.004)	-0.698(0.002)	-1.193(0.001)	-0.079(0.002)	1.304(0.002)
	1536	0.129(0.003)	-0.949(0.005)	0.944(0.006)	0.673(0.004)	-0.674(0.003)	-1.193(0.001)	-0.079(0.002)	1.304(0.003)
ODVGP	1024 + 1024	0.137(0.003)	-0.956(0.005)	-0.199(0.067)	0.105(0.033)	-0.664(0.003)	-1.193(0.001)	-0.078(0.001)	1.317(0.002)
	1024 + 8096	0.144(0.002)	-0.946(0.005)	-0.136(0.063)	0.109(0.033)	-0.657 (0.003)	-1.193(0.001)	-0.079(0.001)	1.319(0.004)
SOLVE-GP	1024 + 1024	*0.187 (0.002)	-0.943(0.005)	0.973 (0.003)	0.680 (0.003)	-0.659(0.002)	-1.192 (0.001)	*-0.071 (0.001)	*1.333 (0.003)
SVGP	2048	0.137(0.003)	-0.940 (0.005)	0.907(0.003)	0.665(0.004)	-0.669(0.002)	-1.192 (0.001)	-0.079(0.002)	1.304(0.003)

Table 3: Deep convolutional GPs for CIFAR-10 classification. Previous SOTA is 76.17% by a 3-layer model with 384 inducing points in all layers (Dutordoir et al., 2019).

(a) 2-layer model

	SVGP	SOLVE-GP	SVGP
$M(+M_2)$	384, 1K	384 + 384, 1K + 1K	768, 2K
Test Acc	76.35%	77.80%	77.46%
Test LL	-1.04	-0.98	-0.98
Time	0.392 s/iter	0.657 s/iter	1.104 s/iter

(b) 3-layer model

	SVGP	SOLVE-GP	SVGP
$M(+M_2)$	384, 384, 1K	384 + 384, 384 + 384, 1K + 1K	768, 768, 2K
Test Acc	78.76%	80.30%	80.33%
Test LL	-0.88	-0.79	-0.82
Time	0.418 s/iter	0.752 s/iter	1.246 s/iter

ODVGP ($M_2 = 8096$), where in each iteration 1024 columns of $\mathbf{K}_{\mathbf{v}\mathbf{v}}$ are sampled to estimate the gradient. Other experimental details are given in appendix G.

We report the predictive log-likelihoods on test data in Table 2. For space reasons, we provide the results on two small datasets (Elevators, Bike) in appendix H. We can see that performance of SOLVE-GP is competitive with SVGP ($M = 2048$) that involves 4x more expensive Cholesky decomposition. Perhaps surprisingly, despite using a less flexible covariance in the variational distribution, SOLVE-GP often outperforms SVGP ($M = 2048$). We believe this is due to the optimization difficulties introduced by the 2048×2048 covariance matrix and will test hypothesis on the HouseElectric dataset below. On most datasets, using a large number of additional inducing points for modeling the mean function did improve the performance, as shown by the comparison between ODVGP ($M_2 = 1024$) and ODVGP ($M_2 = 8096$). However, more flex-

ible covariance modeling seems to be more important, as SOLVE-GP outperforms ODVGP ($M_2 = 8096$) on all datasets except for 3dRoad.

In Fig. 3a we plot the evolution of test RMSE and test log-likelihoods during training on HouseElectric. Interestingly, ODVGP ($M_2 = 8096$) performs on par with SOLVE-GP early in training before falling behind it substantially. The beginning stage is likely where the additional inducing points give good predictions but are not in the best configuration for maximizing the training lower bounds. This phenomenon is also observed on Protein, Elevators, and Kin40k. We believe such mismatch between the training lower bound and predictive performance is caused by fixing the covariance matrix of $q(\mathbf{v}_\perp)$ to the prior covariance. SVGP ($M = 2048$) does not improve over SVGP ($M = 1024$) and is outperformed by SOLVE-GP. Suggested above, this might be due to the difficulty of optimising large covariance matrices. To verify this, we tried the “whitening” trick (Murray and Adams, 2010; Hensman et al., 2015b), described in appendix F, which is often used to make optimization easier by reducing the correlation in the posterior distributions. As shown in Fig. 3b, the performance of SVGP ($M = 2048$) and SOLVE-GP becomes similar with whitening. We did not use whitening in ODVGP because it has a slightly different parameterization to allow sub-sampling $\mathbf{K}_{\mathbf{v}\mathbf{v}}$.

7 CONCLUSION

We proposed SOLVE-GP, a new variational inference framework for GPs using inducing points, that unifies and generalizes previous sparse variational methods. This increases the number of inducing points we can use for a fixed computational budget, which allows to improve performance of large, hierarchical GP models at a manageable computational cost. Future work includes experiments on challenging datasets like ImageNet and investigating other ways to improve the variational distribution, as mentioned in section 3.1.

Acknowledgements

We thank Alex Matthews and Yutian Chen for helpful suggestions on improving the paper.

References

- Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. *arXiv preprint arXiv:1904.11955*, 2019.
- Kenneth Blomqvist, Samuel Kaski, and Markus Heinonen. Deep convolutional Gaussian processes. *arXiv preprint arXiv:1810.03052*, 2018.
- Thang D. Bui, Josiah Yan, and Richard E. Turner. A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation. *Journal of Machine Learning Research*, 18(104):1–72, 2017.
- Ching-An Cheng and Byron Boots. Incremental variational sparse Gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 4410–4418, 2016.
- Ching-An Cheng and Byron Boots. Variational inference for Gaussian process models with linear complexity. In *Advances in Neural Information Processing Systems*, pages 5184–5194, 2017.
- L. Csato and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14:641–668, 2002.
- Andreas C. Damianou, Michalis K. Titsias, and Neil D. Lawrence. Variational inference for latent variables and uncertain inputs in Gaussian processes. *Journal of Machine Learning Research*, 17(42):1–62, 2016.
- Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472, 2011.
- Vincent Dutoit, Mark van der Wilk, Artem Artemev, Marcin Tomczak, and James Hensman. Translation insensitivity for deep convolutional Gaussian processes. *arXiv preprint arXiv:1902.05888*, 2019.
- Trefor Evans and Prasanth Nair. Scalable Gaussian processes with grid-structured eigenfunctions (GP-GRIEF). In *International Conference on Machine Learning*, pages 1416–1425, 2018.
- Jacob Gardner, Geoff Pleiss, Ruihan Wu, Kilian Weinberger, and Andrew Wilson. Product kernel interpolation for scalable Gaussian processes. In *Artificial Intelligence and Statistics*, pages 1407–1416, 2018.
- Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Artificial Intelligence and Statistics*, pages 249–256, 2010.
- Marton Havasi, José Miguel Hernández-Lobato, and Juan José Murillo-Fuentes. Deep Gaussian processes with decoupled inducing inputs. *arXiv preprint arXiv:1801.02939*, 2018.
- James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.
- James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360, 2015a.
- James Hensman, Alexander G Matthews, Maurizio Filippone, and Zoubin Ghahramani. MCMC for variationally sparse Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1648–1656, 2015b.
- James Hensman, Nicolas Durrande, and Arno Solin. Variational Fourier features for Gaussian processes. *Journal of Machine Learning Research*, 18(151):1–151, 2017.
- Daniel Hernández-Lobato, José M Hernández-Lobato, and Pierre Dupont. Robust multi-class Gaussian process classification. In *Advances in Neural Information Processing Systems*, pages 280–288, 2011.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: the informative vector machine. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.
- Neil Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6(Nov):1783–1816, 2005.
- Miguel Lázaro-Gredilla and Anibal Figueiras-Vidal. Inter-domain Gaussian processes for sparse inference using inducing features. In *Advances in Neural Information Processing Systems*, pages 1087–1095, 2009.
- Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems*, pages 1732–1740, 2010.
- Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *International Conference on Learning Representations*, 2019.
- J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process

- regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005a.
- Joaquin Quiñero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005b.
- Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.
- Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4588–4599, 2017.
- Hugh Salimbeni, Ching-An Cheng, Byron Boots, and Marc Deisenroth. Orthogonally decoupled variational Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 8711–8720, 2018.
- M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Ninth International Workshop on Artificial Intelligence*. MIT Press, 2003.
- Jiaxin Shi, Mohammad Emtiyaz Khan, and Jun Zhu. Scalable training of inference networks for Gaussian-process models. In *International Conference on Machine Learning*, pages 5758–5768, 2019.
- Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2006.
- Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*, pages 1015–1022, 2010.
- Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.
- Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979, 2014.
- Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 2849–2858, 2017.
- Ke Alexander Wang, Geoff Pleiss, Jacob R Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact Gaussian processes on a million data points. *arXiv preprint arXiv:1903.08114*, 2019.
- Christopher KI Williams and David Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.
- Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, pages 514–520, 1996.
- Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688, 2001.
- Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784, 2015.

A Tighter Sparse Variational Bounds for GP Regression

As mentioned in section 3.1, another way to improve the variational distribution $q(\mathbf{u})p_{\perp}(\mathbf{f}_{\perp})$ in SVGP is to make \mathbf{u} and \mathbf{f}_{\perp} dependent. The best possible approximation of this type is obtained by the setting $q(\mathbf{u})$ to the optimal exact posterior conditional $q^*(\mathbf{u}) = p(\mathbf{u}|\mathbf{f}_{\perp}, \mathbf{y})$. The corresponding collapsed bound for GP regression can be derived by analytically marginalising out \mathbf{u} from the joint model in Eq. (6),

$$\begin{aligned} p(\mathbf{y}|\mathbf{f}_{\perp}) &= \int p(\mathbf{y}|\mathbf{f}_{\perp} + \mathbf{K}_{\mathbf{f}\mathbf{u}}\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{u})p(\mathbf{u}) d\mathbf{u} \\ &= \mathcal{N}(\mathbf{y}|\mathbf{f}_{\perp}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I}), \end{aligned} \quad (12)$$

and then forcing the approximation $p_{\perp}(\mathbf{f}_{\perp})$:

$$\mathbb{E}_{p_{\perp}(\mathbf{f}_{\perp})} \log \mathcal{N}(\mathbf{y}|\mathbf{f}_{\perp}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I}). \quad (13)$$

This bound has a closed-form as

$$\log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I}) - \frac{1}{2} \text{tr} [(\mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I})^{-1}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})], \quad (14)$$

Applying the matrix inversion lemma to $(\mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I})^{-1}$, we have an equivalent form that can be directly compared with Eq. (3):

$$\underbrace{\log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{Q}_{\mathbf{ff}} + \sigma^2\mathbf{I}) - \frac{1}{2\sigma^2} \text{tr}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})}_{=\text{Eq. (3)}} + \frac{1}{2\sigma^4} \text{tr} [\mathbf{K}_{\mathbf{f}\mathbf{u}}(\mathbf{K}_{\mathbf{u}\mathbf{u}} + \sigma^{-2}\mathbf{K}_{\mathbf{uf}}\mathbf{K}_{\mathbf{fu}})^{-1}\mathbf{K}_{\mathbf{uf}}(\mathbf{K}_{\mathbf{ff}} - \mathbf{Q}_{\mathbf{ff}})], \quad (15)$$

where the first two terms recover Eq. (3), suggesting this is a tighter bound than the Titsias (2009) bound. This bound is not amenable to large-scale datasets because of $O(N^2)$ storage and $O(MN^2)$ computation time (dominated by the matrix multiplication $\mathbf{K}_{\mathbf{uf}}\mathbf{K}_{\mathbf{ff}}$) requirements. However, it is still of theoretical interest and can be applied to medium-sized regression datasets, just like the SGPR algorithm using the Titsias (2009) bound.

B Details of Orthogonal Decomposition

In section 3.2 we described the following orthogonal decomposition for $f \in \mathcal{H}$, where \mathcal{H} is the RKHS induced by kernel k :

$$f = f_{\parallel} + f_{\perp}, \quad f_{\parallel} \in V \text{ and } f_{\perp} \perp V. \quad (16)$$

Here V is the subspace spanned by the inducing basis: $V = \{\sum_{j=1}^M \alpha_j k(\mathbf{z}_j, \cdot), \boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_M]^{\top} \in \mathbb{R}^M\}$. Since $f_{\parallel} \in V$, we let $f_{\parallel} = \sum_{j=1}^M \alpha'_j k(\mathbf{z}_j, \cdot)$. According to the properties of orthogonal projection, we have

$$\langle f, g \rangle_{\mathcal{H}} = \langle f_{\parallel}, g \rangle_{\mathcal{H}}, \quad \forall g \in V, \quad (17)$$

where $\langle \cdot \rangle_{\mathcal{H}}$ is the RKHS inner product that satisfies the reproducing property: $\langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} = f(\mathbf{x})$. Similarly let $g = \sum_{j=1}^M \beta_j k(\mathbf{z}_j, \cdot)$. Then $\langle f, g \rangle_{\mathcal{H}} = \sum_{j=1}^M \beta_j f(\mathbf{z}_j)$, $\langle f_{\parallel}, g \rangle_{\mathcal{H}} = \sum_{i=1}^M \sum_{j=1}^M \alpha'_i \beta_j k(\mathbf{z}_i, \mathbf{z}_j)$. Plugging into Eq. (17) and rearranging the terms, we have

$$\boldsymbol{\beta}^{\top} (f(\mathbf{Z}) - k(\mathbf{Z}, \mathbf{Z})\boldsymbol{\alpha}') = 0, \quad \forall \boldsymbol{\beta} \in \mathbb{R}^M, \quad (18)$$

where $f(\mathbf{Z}) = [f(\mathbf{z}_1), \dots, f(\mathbf{z}_M)]^{\top}$, $k(\mathbf{Z}, \mathbf{Z})$ is a matrix with the ij -th term as $k(\mathbf{z}_i, \mathbf{z}_j)$, and $\boldsymbol{\alpha}' = [\alpha'_1, \dots, \alpha'_M]^{\top}$. Therefore,

$$\boldsymbol{\alpha}' = k(\mathbf{Z}, \mathbf{Z})^{-1} f(\mathbf{Z}), \quad (19)$$

and it follows that $f_{\parallel}(\mathbf{x}) = k(\mathbf{x}, \mathbf{Z})k(\mathbf{Z}, \mathbf{Z})^{-1} f(\mathbf{Z})$, where $k(\mathbf{x}, \mathbf{Z}) = [k(\mathbf{z}_1, \mathbf{x}), \dots, k(\mathbf{z}_M, \mathbf{x})]$. The above analysis arrives at the decomposition:

$$f_{\parallel} = k(\cdot, \mathbf{Z})k(\mathbf{Z}, \mathbf{Z})^{-1} f(\mathbf{Z}), \quad f_{\perp} = f - f_{\parallel}. \quad (20)$$

Although the derivation from Eq. (16) to (19) relies on the fact $f \in \mathcal{H}$, such that the inner product is well-defined, the decomposition in Eq. (20) is valid for any function f on \mathcal{X} . This motivates us to study it for $f \sim \mathcal{GP}(0, k)$. Substituting \mathbf{u} for $f(\mathbf{Z})$ and $\mathbf{K}_{\mathbf{u}\mathbf{u}}$ for $k(\mathbf{Z}, \mathbf{Z})$, we have for $f \sim \mathcal{GP}(0, k)$:

$$f_{\parallel} = k(\cdot, \mathbf{Z})\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}\mathbf{u} \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{Z})\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}k(\mathbf{Z}, \mathbf{x}')), \quad (21)$$

$$f_{\perp} \sim p_{\perp} \equiv \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{Z})\mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1}k(\mathbf{Z}, \mathbf{x}')). \quad (22)$$

C The Collapsed SOLVE-GP Lower Bound

We derive the collapsed SOLVE-GP lower bound in Eq. (9) by seeking the optimal $q(\mathbf{u})$ that is independent of \mathbf{f}_\perp . First we rearrange the terms in the uncollapsed SOLVE-GP bound (Eq. (8)) as

$$\mathbb{E}_{q(\mathbf{u})} \left\{ \mathbb{E}_{q_\perp(\mathbf{f}_\perp)} [\log \mathcal{N}(\mathbf{y} | \mathbf{f}_\perp + \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u}, \sigma^2 \mathbf{I})] \right\} - \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})] - \text{KL}[q(\mathbf{v}_\perp) \| p_\perp(\mathbf{v}_\perp)]. \quad (23)$$

where $q_\perp(\mathbf{f}_\perp) = \mathcal{N}(\mathbf{m}_{\mathbf{f}_\perp}, \mathbf{S}_{\mathbf{f}_\perp})$, and $\mathbf{m}_{\mathbf{f}_\perp} = \mathbf{C}_{\mathbf{f}\mathbf{v}} \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} \mathbf{m}_{\mathbf{v}}$, $\mathbf{S}_{\mathbf{f}_\perp} = \mathbf{C}_{\mathbf{f}\mathbf{f}} + \mathbf{C}_{\mathbf{f}\mathbf{v}} \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} (\mathbf{S}_{\mathbf{v}} - \mathbf{C}_{\mathbf{v}\mathbf{v}}) \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} \mathbf{C}_{\mathbf{v}\mathbf{f}}$. In the first term we can simplify the expectation over \mathbf{f}_\perp as:

$$\begin{aligned} & \mathbb{E}_{q_\perp(\mathbf{f}_\perp)} \log \mathcal{N}(\mathbf{y} | \mathbf{f}_\perp + \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u}, \sigma^2 \mathbf{I}) \\ &= \mathbb{E}_{q_\perp(\mathbf{f}_\perp)} \left[-\frac{N}{2} \log 2\pi - \frac{N}{2} \log \sigma^2 - \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{f}_\perp - \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u})^\top (\mathbf{y} - \mathbf{f}_\perp - \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u}) \right] \\ &= \left[-\frac{N}{2} \log 2\pi - \frac{N}{2} \log \sigma^2 - \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{m}_{\mathbf{f}_\perp} - \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u})^\top (\mathbf{y} - \mathbf{m}_{\mathbf{f}_\perp} - \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u}) \right] \\ & \quad - \mathbb{E}_{q_\perp(\mathbf{f}_\perp)} \left[\frac{1}{2\sigma^2} (\mathbf{f}_\perp - \mathbf{m}_{\mathbf{f}_\perp})^\top (\mathbf{f}_\perp - \mathbf{m}_{\mathbf{f}_\perp}) \right] \\ &= \log \mathcal{N}(\mathbf{y} | \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u} + \mathbf{m}_{\mathbf{f}_\perp}, \sigma^2 \mathbf{I}) - \frac{1}{2\sigma^2} \text{tr}(\mathbf{S}_{\mathbf{f}_\perp}). \end{aligned} \quad (24)$$

Plugging into Eq. (23) and rearranging the terms, we have

$$\underbrace{\mathbb{E}_{q(\mathbf{u})} [\log \mathcal{N}(\mathbf{y} | \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u} + \mathbf{m}_{\mathbf{f}_\perp}, \sigma^2 \mathbf{I})] - \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})]}_{\leq \log \int \mathcal{N}(\mathbf{y} | \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u} + \mathbf{m}_{\mathbf{f}_\perp}, \sigma^2 \mathbf{I}) p(\mathbf{u}) d\mathbf{u}} - \frac{1}{2\sigma^2} \text{tr}(\mathbf{S}_{\mathbf{f}_\perp}) - \text{KL}[q(\mathbf{v}_\perp) \| p_\perp(\mathbf{v}_\perp)]. \quad (25)$$

Clearly the leading two terms form a variational lower bound of the joint distribution $\mathcal{N}(\mathbf{y} | \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u} + \mathbf{m}_{\mathbf{f}_\perp}, \sigma^2 \mathbf{I}) p(\mathbf{u})$. The optimal $q(\mathbf{u})$ will turn it into the log marginal likelihood:

$$\log \int \mathcal{N}(\mathbf{y} | \mathbf{K}_{\mathbf{f}\mathbf{u}} \mathbf{K}_{\mathbf{u}\mathbf{u}}^{-1} \mathbf{u} + \mathbf{m}_{\mathbf{f}_\perp}, \sigma^2 \mathbf{I}) p(\mathbf{u}) d\mathbf{u} = \log \mathcal{N}(\mathbf{y} | \mathbf{m}_{\mathbf{f}_\perp}, \mathbf{Q}_{\mathbf{f}\mathbf{f}} + \sigma^2 \mathbf{I}). \quad (26)$$

Plugging this back, we have the collapsed SOLVE-GP bound in Eq. (9):

$$\log \mathcal{N}(\mathbf{y} | \mathbf{C}_{\mathbf{f}\mathbf{v}} \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} \mathbf{m}_{\mathbf{v}}, \mathbf{Q}_{\mathbf{f}\mathbf{f}} + \sigma^2 \mathbf{I}) - \frac{1}{2\sigma^2} \text{tr}(\mathbf{S}_{\mathbf{f}_\perp}) - \text{KL}[\mathcal{N}(\mathbf{m}_{\mathbf{v}}, \mathbf{S}_{\mathbf{v}}) \| \mathcal{N}(\mathbf{0}, \mathbf{C}_{\mathbf{v}\mathbf{v}})], \quad (27)$$

Moreover, we could find the optimal $q^*(\mathbf{v}) = \mathcal{N}(\mathbf{m}_{\mathbf{v}}^*, \mathbf{S}_{\mathbf{v}}^*)$ by setting the derivatives w.r.t. $\mathbf{m}_{\mathbf{v}}$ and $\mathbf{S}_{\mathbf{v}}$ to be zeros:

$$\mathbf{m}_{\mathbf{v}}^* = \mathbf{C}_{\mathbf{v}\mathbf{v}} [\mathbf{C}_{\mathbf{v}\mathbf{v}} + \mathbf{C}_{\mathbf{v}\mathbf{f}} \mathbf{A}^{-1} \mathbf{C}_{\mathbf{f}\mathbf{v}}]^{-1} \mathbf{C}_{\mathbf{v}\mathbf{f}} \mathbf{A}^{-1} \mathbf{y}, \quad (28)$$

$$\mathbf{S}_{\mathbf{v}}^* = \mathbf{C}_{\mathbf{v}\mathbf{v}} [\mathbf{C}_{\mathbf{v}\mathbf{v}} + \sigma^{-2} \mathbf{C}_{\mathbf{v}\mathbf{f}} \mathbf{C}_{\mathbf{f}\mathbf{v}}]^{-1} \mathbf{C}_{\mathbf{v}\mathbf{v}}, \quad (29)$$

where $\mathbf{A} = \mathbf{Q}_{\mathbf{f}\mathbf{f}} + \sigma^2 \mathbf{I}$. Then the collapsed bound with the optimal $q(\mathbf{v}_\perp)$ is

$$\log \mathcal{N}(\mathbf{y} | \mathbf{C}_{\mathbf{f}\mathbf{v}} \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} \mathbf{m}_{\mathbf{v}}^*, \mathbf{A}) - \frac{1}{2\sigma^2} \text{tr}[\mathbf{C}_{\mathbf{f}\mathbf{f}} - \mathbf{B}(\mathbf{B} + \sigma^2 \mathbf{I})^{-1} \mathbf{B}] - \text{KL}[\mathcal{N}(\mathbf{m}_{\mathbf{v}}^*, \mathbf{S}_{\mathbf{v}}^*) \| \mathcal{N}(\mathbf{0}, \mathbf{C}_{\mathbf{v}\mathbf{v}})], \quad (30)$$

where $\mathbf{B} = \mathbf{C}_{\mathbf{f}\mathbf{v}} \mathbf{C}_{\mathbf{v}\mathbf{v}}^{-1} \mathbf{C}_{\mathbf{v}\mathbf{f}}$.

D Computational Details

D.1 Training

To compute the lower bound in Eq. (8), we write it as

$$\sum_{n=1}^N \mathbb{E}_{q(f(\mathbf{x}_n); \Theta)} [\log p(y_n | f(\mathbf{x}_n))] - \text{KL}[q(\mathbf{u}) \| p(\mathbf{u})] - \text{KL}[q(\mathbf{v}_\perp) \| p_\perp(\mathbf{v}_\perp)], \quad (31)$$

Algorithm 1 The SOLVE-GP lower bound via Cholesky decomposition. We parameterize the variational covariance matrices with their Cholesky factors $\mathbf{S}_u = \mathbf{L}_u \mathbf{L}_u^\top$, $\mathbf{S}_v = \mathbf{L}_v \mathbf{L}_v^\top$. $\mathbf{A} = \mathbf{L}_u^0 \setminus \mathbf{K}_{uv}$ denotes the solution of $\mathbf{L}_u^0 \mathbf{A} = \mathbf{K}_{uv}$. \odot denotes elementwise multiplication. The differences from SVGP are shown in blue.

Input: \mathbf{X} (training inputs), \mathbf{y} (targets), \mathbf{Z}, \mathbf{O} (inducing points), $\mathbf{m}_u, \mathbf{L}_u, \mathbf{m}_v, \mathbf{L}_v$ (variational parameters)

- 1: $\mathbf{K}_{uu} = k(\mathbf{Z}, \mathbf{Z})$, $\mathbf{K}_{vv} = k(\mathbf{O}, \mathbf{O})$
- 2: $\mathbf{L}_u^0 = \text{Cholesky}(\mathbf{K}_{uu})$, $\mathbf{K}_{uv} = k(\mathbf{Z}, \mathbf{O})$, $\mathbf{A} := \mathbf{L}_u^0 \setminus \mathbf{K}_{uv}$, $\mathbf{C}_{vv} = \mathbf{K}_{vv} - \mathbf{A}^\top \mathbf{A}$, $\mathbf{L}_v^0 = \text{Cholesky}(\mathbf{C}_{vv})$
- 3: $\mathbf{K}_{uf} = k(\mathbf{Z}, \mathbf{X})$, $\mathbf{K}_{vf} = k(\mathbf{O}, \mathbf{X})$
- 4: $\mathbf{B} := \mathbf{L}_u^0 \setminus \mathbf{K}_{uf}$, $\mathbf{C}_{vf} = \mathbf{K}_{vf} - \mathbf{A}^\top \mathbf{B}$, $\mathbf{D} := \mathbf{L}_v^0 \setminus \mathbf{C}_{vf}$
- 5: $\mathbf{E} := \mathbf{L}_u^0 \setminus \mathbf{B}$, $\mathbf{F} := \mathbf{L}_u^\top \mathbf{E}$, $\mathbf{G} := \mathbf{L}_v^0 \setminus \mathbf{D}$, $\mathbf{H} := \mathbf{L}_v^\top \mathbf{G}$
- 6: $\boldsymbol{\mu}(\mathbf{X}) = \mathbf{E}^\top \mathbf{m}_u + \mathbf{G}^\top \mathbf{m}_v$
- 7: $\boldsymbol{\sigma}^2(\mathbf{X}) = \text{diag}(\mathbf{K}_{ff}) + (\mathbf{F} \odot \mathbf{F})^\top \mathbf{1} - (\mathbf{B} \odot \mathbf{B})^\top \mathbf{1} + (\mathbf{H} \odot \mathbf{H})^\top \mathbf{1} - (\mathbf{D} \odot \mathbf{D})^\top \mathbf{1}$
- 8: Compute LLD = $\sum_{n=1}^N \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}(\mathbf{x}_n), \boldsymbol{\sigma}^2(\mathbf{x}_n))} \log p(y_n | f(\mathbf{x}_n))$ in closed form or using quadrature/Monte Carlo.
- 9: **function** COMPUTE_KL($\mathbf{m}, \mathbf{L}, \mathbf{L}^0$)
- 10: $\mathbf{P} = \mathbf{L}^0 \setminus \mathbf{L}$, $\mathbf{a} = \mathbf{L}^0 \setminus \mathbf{m}$
- 11: **return** $\log(\text{diag}(\mathbf{L}^0))^\top \mathbf{1} - \log(\text{diag}(\mathbf{L}))^\top \mathbf{1} + 1/2((\mathbf{P} \odot \mathbf{P})^\top \mathbf{1} + \mathbf{a}^\top \mathbf{a} - M)$
- 12: **end function**
- 13: $\text{KL}_u = \text{COMPUTE_KL}(\mathbf{m}_u, \mathbf{L}_u, \mathbf{L}_u^0)$, $\text{KL}_v = \text{COMPUTE_KL}(\mathbf{m}_v, \mathbf{L}_v, \mathbf{L}_v^0)$
- 14: **return** LLD - KL_u - KL_v

where $\Theta := \{\mathbf{m}_u, \mathbf{S}_u, \mathbf{m}_v, \mathbf{S}_v, \mathbf{Z}, \mathbf{O}\}$ and $q(f(\mathbf{x}_n); \Theta)$ defines the marginal distribution of $\mathbf{f} = \mathbf{f}_\perp + \mathbf{K}_{fu} \mathbf{K}_{uu}^{-1} \mathbf{u}$ for the n -th data point given $\mathbf{u} \sim q(\mathbf{u})$ and $\mathbf{f}_\perp \sim q_\perp(\mathbf{f}_\perp)$. We can write $q(f(\mathbf{x}_n); \Theta)$ as

$$q(f(\mathbf{x}_n); \Theta) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}_n), \boldsymbol{\sigma}^2(\mathbf{x}_n)), \quad (32)$$

where

$$\boldsymbol{\mu}(\mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{Z}) \mathbf{K}_{uu}^{-1} \mathbf{m}_u + c(\mathbf{x}_n, \mathbf{O}) \mathbf{C}_{vv}^{-1} \mathbf{m}_v, \quad (33)$$

$$\boldsymbol{\sigma}^2(\mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{Z}) \mathbf{K}_{uu}^{-1} \mathbf{S}_u \mathbf{K}_{uu}^{-1} k(\mathbf{Z}, \mathbf{x}_n) + c(\mathbf{x}_n, \mathbf{x}_n) + c(\mathbf{x}_n, \mathbf{O}) \mathbf{C}_{vv}^{-1} (\mathbf{S}_v - \mathbf{C}_{vv}) \mathbf{C}_{vv}^{-1} c(\mathbf{O}, \mathbf{x}_n). \quad (34)$$

Here $c(\mathbf{x}, \mathbf{x}') := k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{Z}) \mathbf{K}_{uu}^{-1} k(\mathbf{Z}, \mathbf{x}')$ denotes the covariance function of p_\perp . The univariate expectation of $\log p(y_n | f(\mathbf{x}_n))$ under $q(f(\mathbf{x}_n); \Theta)$ can be computed in closed form (e.g., for Gaussian likelihoods) or using quadrature (Hensman et al., 2015b). It can also be estimated by Monte Carlo with the reparameterization trick (Kingma and Welling, 2013; Titsias and Lázaro-Gredilla, 2014; Rezende et al., 2014) to propagate gradients. For large datasets, an unbiased estimate of the sum can be used for mini-batch training: $\frac{N}{|B|} \sum_{(\mathbf{x}, y) \in B} \mathbb{E}_{q(f(\mathbf{x}); \Theta)} [\log p(y | f(\mathbf{x}))]$, where B denotes a small batch of data points.

Besides the log-likelihood term, we need to compute the two KL divergence terms:

$$\text{KL}[q(\mathbf{u}) \| p(\mathbf{u})] = \frac{1}{2} [\log \det \mathbf{K}_{uu} - \log \det \mathbf{S}_u - M + \text{tr}(\mathbf{K}_{uu}^{-1} \mathbf{S}_u) + \mathbf{m}_u^\top \mathbf{K}_{uu}^{-1} \mathbf{m}_u], \quad (35)$$

$$\text{KL}[q(\mathbf{v}_\perp) \| p_\perp(\mathbf{v}_\perp)] = \frac{1}{2} [\log \det \mathbf{C}_{vv} - \log \det \mathbf{S}_v - M + \text{tr}(\mathbf{C}_{vv}^{-1} \mathbf{S}_v) + \mathbf{m}_v^\top \mathbf{C}_{vv}^{-1} \mathbf{m}_v]. \quad (36)$$

We note that if the blue parts in Eqs. (31) to (34) are removed, then we recover the SVGP lower bound in Eq. (2). An implementation of the above computations using the Cholesky decomposition is shown in algorithm 1.

D.2 Prediction

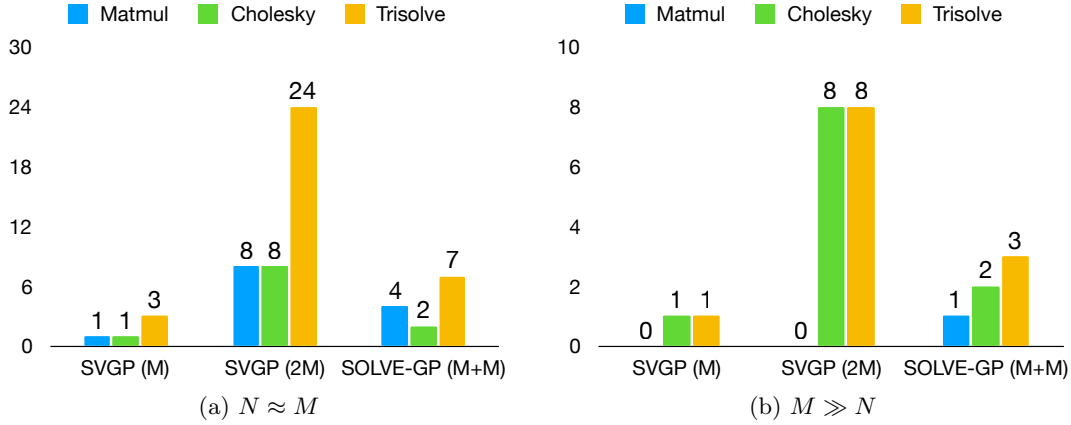
We can predict the function value at a test point \mathbf{x}^* with the approximate posterior by substituting \mathbf{x}^* for \mathbf{x}_n in Eq. (32). For multiple test points \mathbf{X}^* , we denote the joint predictive density by $\mathcal{N}(\mathbf{f}^* | \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$, where the predicted mean and covariance are

$$\boldsymbol{\mu}^* = \mathbf{K}_{*u} \mathbf{K}_{uu}^{-1} \mathbf{m}_u + \mathbf{C}_{*v} \mathbf{C}_{vv}^{-1} \mathbf{m}_v, \quad (37)$$

$$\boldsymbol{\Sigma}^* = \mathbf{K}_{*u} \mathbf{K}_{uu}^{-1} \mathbf{S}_u \mathbf{K}_{uu}^{-1} \mathbf{K}_{u*} + \mathbf{C}_{**} - \mathbf{C}_{*v} \mathbf{C}_{vv}^{-1} (\mathbf{C}_{vv} - \mathbf{S}_v) \mathbf{C}_{vv}^{-1} \mathbf{C}_{v*}. \quad (38)$$

Table 4: Cubic-cost operations in SOLVE-GP and SVGP, following the implementation in algorithm 1.

	SVGP	SOLVE-GP
Matrix multiplication	$\mathcal{O}(NM^2) \times 1$	$\mathcal{O}(NM^2) \times 1$
		$\mathcal{O}(NM_2^2) \times 1$
		$\mathcal{O}(NMM_2) \times 1$
		$\mathcal{O}(MM_2^2) \times 1$
Cholesky	$\mathcal{O}(M^3) \times 1$	$\mathcal{O}(M^3) \times 1$
		$\mathcal{O}(M_2^3) \times 1$
Solving triangular matrix equations	$\mathcal{O}(M^3) \times 1$ $\mathcal{O}(NM^2) \times 2$	$\mathcal{O}(M^3) \times 1$
		$\mathcal{O}(NM^2) \times 2$
		$\mathcal{O}(M_2^3) \times 1$
		$\mathcal{O}(NM_2^2) \times 2$
		$\mathcal{O}(M_2M^2) \times 1$


 Figure 4: Comparison of computational cost for SVGP and SOLVE-GP. For each method and each type of cubic-cost operation, we plot the factor of increase in cost compared to a single operation on $M \times M$ matrices.

D.3 Computational Complexity

As mentioned in section 3.3, the time complexity of SOLVE-GP is $\mathcal{O}(NM^2 + \bar{M}^3)$ per gradient update, where $\bar{M} = \max(M, M_2)$ and N is the batch size. Here we provide a more fine-grained analysis by counting cubic-cost operations and compare to the standard SVGP method. We underlined all the cubic-cost operations in algorithm 1, including matrix multiplication, Cholesky decomposition, and solving triangular matrix equations. We count them for SVGP and SOLVE-GP. The results are summarized in Table 4.

For comparison purposes, we study two cases of mini-batch training: (i) $N \approx M$ and (ii) $M \gg N$. We consider SOLVE-GP with $M_2 = M$, which has $2M$ inducing points in total, and then compare to SVGP with M and $2M$ inducing points. For each method and each type of operation, we plot the factor of increase in cost compared to a single operation on $M \times M$ matrices. For instance, when $N \approx M$ (Fig. 4a), SVGP with M inducing points requires solving three triangular matrix equations for $M \times M$ matrices. Doubling the number of inducing points in SVGP increases the cost by a factor of 8, plotted as 24 for SVGP (2M). In contrast, in SOLVE-GP with M orthogonal inducing points we only need to solve 7 triangular matrix equations for $M \times M$ matrices. The comparison under the case of $M \gg N$ is shown in Fig. 4b. In this case SOLVE-GP additionally introduces one $\mathcal{O}(M^3)$ matrix multiplication operation, but overall the algorithm is still much faster than SVGP (2M) given the speed-up in Cholesky decomposition and solving matrix equations.

E Details of Eq. (11)

The variational distribution in Eq. (11) is defined as:

$$q(\mathbf{u}^L, \mathbf{f}_\perp^L) = \int \prod_{\ell=1}^L [p_\perp(\mathbf{f}_\perp^\ell | \mathbf{v}_\perp^\ell, \mathbf{f}_\perp^{\ell-1}, \mathbf{u}^{\ell-1}) q(\mathbf{v}_\perp^\ell) q(\mathbf{u}^\ell) d\mathbf{u}^\ell d\mathbf{v}_\perp^\ell] \prod_{\ell=1}^{L-1} d\mathbf{f}_\perp^\ell. \quad (39)$$

F Whitening

Similar to the practice in SVGP methods, we can apply the “whitening” trick (Murray and Adams, 2010; Hensman et al., 2015b) to SOLVE-GP. The goal is to improve the optimization of variational approximations by reducing correlation in the posterior distributions. Specifically, we could “whiten” \mathbf{u} by using $\mathbf{u}' = \mathbf{K}_{\mathbf{uu}}^{-1/2} \mathbf{u}$, where $\mathbf{K}_{\mathbf{uu}}^{-1/2}$ denotes the Cholesky factor of the prior covariance $\mathbf{K}_{\mathbf{uu}}$. Then posterior inference for \mathbf{u} turns into inference for \mathbf{u}' , which has an isotropic Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Then we parameterize the variational distribution w.r.t. \mathbf{u}' : $q(\mathbf{u}') = \mathcal{N}(\mathbf{m}_u, \mathbf{S}_u)$. Whitening $q(\mathbf{v}_\perp)$ is similar to whitening $q(\mathbf{u})$, i.e., we parameterize the variational distribution w.r.t. $\mathbf{v}'_\perp = \mathbf{C}_{\mathbf{vv}}^{-1/2} \mathbf{v}_\perp$ and set $q(\mathbf{v}'_\perp) = \mathcal{N}(\mathbf{m}_v, \mathbf{S}_v)$. The algorithm can be derived by replacing $\mathbf{m}_u, \mathbf{m}_v$ with $\mathbf{L}_u^0 \mathbf{m}_u, \mathbf{L}_v^0 \mathbf{m}_v$, and $\mathbf{S}_u, \mathbf{S}_v$ with $\mathbf{L}_u^0 \mathbf{S}_u \mathbf{L}_u^{0\top}, \mathbf{L}_v^0 \mathbf{S}_v \mathbf{L}_v^{0\top}$ in algorithm 1 and removing the canceled terms.

G Experiment Details

For all experiments, we use kernels with a shared lengthscale across dimensions. All model hyperparameters, including kernel parameters, patch weights in convolutional GP models, and observation variances in regression experiments, are optimized jointly with variational parameters using ADAM. The variational distributions $q(\mathbf{u})$ and $q(\mathbf{v}_\perp)$ are by default initialized to the prior distributions. Unless stated otherwise, no “whitening” trick (Murray and Adams, 2010; Hensman et al., 2015b) is used for SVGP or SOLVE-GP.

G.1 1D Regression

We randomly sample 100 training data points from Snelson’s dataset (Snelson and Ghahramani, 2006) as the training data. All models use Gaussian RBF kernels and are trained for 10K iterations with learning rate 0.01 and mini-batch size 20. The GP kernel is initialized with lengthscale 1 and variance 1. The Gaussian likelihood is initialized with variance 0.1.

G.2 Convolutional GP Models

All models are trained for 300K iteration with learning rate 0.003 and batch size 64. The learning rate is annealed by 0.25 every 50K iterations to ensure convergence. We use a zero mean function and the robust multi-class classification likelihood (Hernández-Lobato et al., 2011). The Gaussian RBF kernels for the patch response GPs in all layers are initialized with lengthscale 5 and variance 5. We used the TICK kernel (Dutordoir et al., 2019) for the output layer GP, for which we use a Matérn32 kernel between patch locations with lengthscale initialized to 3. We initialize the inducing patch locations to random values in $[0, H] \times [0, W]$, where $[H, W]$ is the shape of the output feature map in patch extraction.

Convolutional GPs We set patch size to 5×5 and stride to 1. We use the whitening trick in all single-layer experiments for \mathbf{u} (and \mathbf{v}_\perp) since we find it consistently improves the performance. Inducing points are initialized by cluster centers which are generated from running K-means on $M \times 100$ (for SVGP) or $(M + M_2) \times 100$ (for SOLVE-GP) image patches. The image patches are randomly sampled from 1K images randomly selected from the dataset.

Deep Convolutional GPs The detailed model configurations are summarized in Table 5. No whitening trick is used for multi-layer experiments because we find it hurts performance. Inducing points in the input layer are initialized in the same way as in the single-layer model. In Blomqvist et al. (2018); Dutordoir et al. (2019), three-layer models were initialized with the trained values of a two-layer model to avoid getting stuck in bad

local minima. Here we design an initialization scheme that allows training deeper models without the need of pretraining. We initialize the inducing points in deeper layers by running K-means on $M \times 100$ (for SVGP) or $(M + M_2) \times 100$ (for SOLVE-GP) image patches which are randomly sampled from the projections of 1K images to these layers. The projections are done by using a convolution operation with random filters generated using Glorot uniform (Glorot and Bengio, 2010). We also note that when implementing the forward sampling for approximating the log-likelihood term, we follow the previous practice (Dutordoir et al., 2019) to ignore the correlations between outputs of different patches to get faster sampling, which works well in practice. While it is also possible to take into account the correlation when sampling as this only increases the computation cost by a constant factor, doing this might require multi-GPU training due to the additional memory requirements.

Table 5: Model configurations of deep convolutional GPs.

	2-layer	3-layer
Layer 0	patch size 5×5 , stride 1, out channel 10,	patch size 5×5 , stride 1, out channel 10
Layer 1	patch size 4×4 , stride 2	patch size 4×4 , stride 2, out channel 10
Layer 2	-	patch size 5×5 , stride 1

G.3 Regression Benchmarks

The experiment settings are followed from Wang et al. (2019), where we used GPs with Matérn32 kernels and 80% / 20% training / test splits. A 20% subset of the training set is used for validation. We repeat each experiment 5 times with random splits and report the mean and standard error of the performance metrics. For all datasets we train for 100 epochs with learning rate 0.01 and mini-batch size 1024.

H Additional Results

H.1 Regression Benchmarks

Due to space limitations in the main text, we include the Root Mean Squared Error (RMSE) on test data in Table 6. The results on Elevators and Bike are shown in Table 7.

Table 6: Test RMSE values of regression datasets. The numbers in parentheses are standard errors. Best mean values are highlighted, and asterisks indicate statistical significance.

		Kin40k	Protein	KeggDirected	KEGGU	3dRoad	Song	Buzz	HouseElectric
	N	25,600	29,267	31,248	40,708	278,319	329,820	373,280	1,311,539
	d	8	9	20	27	3	90	77	9
SVGP	1024	0.193(0.001)	0.630(0.004)	0.098(0.003)	0.123 (0.001)	0.482(0.001)	0.797(0.001)	0.263(0.001)	0.063(0.000)
	1536	0.182(0.001)	0.621(0.004)	0.098(0.002)	0.123 (0.001)	0.470(0.001)	0.797(0.001)	0.263(0.001)	0.063(0.000)
ODVGP	1024 + 1024	0.183(0.001)	0.625(0.004)	0.176(0.012)	0.156(0.004)	0.467(0.001)	0.797(0.001)	0.263(0.001)	0.062(0.000)
	1024 + 8096	0.180(0.001)	0.618(0.004)	0.157(0.009)	0.157(0.004)	0.462 (0.002)	0.797(0.001)	0.263(0.001)	0.062(0.000)
SOLVE-GP	1024 + 1024	*0.172 (0.001)	0.618(0.004)	0.095 (0.002)	0.123 (0.001)	0.464(0.001)	0.796 (0.001)	0.261 (0.001)	*0.061 (0.000)
SVGP	2048	0.177(0.001)	0.615 (0.004)	0.100(0.003)	0.124(0.001)	0.467(0.001)	0.796 (0.001)	0.263(0.000)	0.063(0.000)

H.2 Convolutional GP Models

We include here the full tables for CIFAR-10 classification, where we also report the accuracies and predictive log-likelihoods on the training data. Table 8 contains the results by convolutional GPs. Table 9 and Table 10 include the results of 2/3-layer deep convolutional GPs.

Table 7: Regression results on Elevators and Bike. Best mean values are highlighted.

		Elevators $N = 10,623, d = 18$		Bike $N = 11,122, d = 17$	
		Test LL	RMSE	Test LL	RMSE
SVGP	1024	-0.516(0.006)	0.398(0.004)	-0.218(0.006)	0.283(0.003)
	1536	-0.511(0.007)	0.396(0.004)	-0.203(0.006)	0.279(0.003)
ODVGP	1024 + 1024	-0.518(0.006)	0.397(0.004)	-0.191(0.006)	0.272(0.003)
	1024 + 8096	-0.523(0.006)	0.399(0.004)	-0.186 (0.006)	0.270 (0.003)
SOLVE-GP	1024 + 1024	-0.509(0.007)	0.395 (0.004)	-0.189(0.006)	0.272(0.003)
SVGP	2048	-0.507 (0.007)	0.395 (0.004)	-0.193(0.006)	0.276(0.003)

Table 8: Convolutional GPs for CIFAR-10 classification.

		Train Acc	Train LL	Test Acc	Test LL	Time
SVGP	1000	77.81%	-1.36	66.07%	-1.59	0.241 s/iter
	1600	78.44%	-1.26	67.18%	-1.54	0.380 s/iter
SOLVE-GP	1000 + 1000	79.32%	-1.20	68.19%	-1.51	0.370 s/iter
SVGP	2000	79.46%	-1.22	68.06%	-1.48	0.474 s/iter

Table 9: 2-layer deep convolutional GPs for CIFAR-10 classification.

		Inducing Points	Train Acc	Train LL	Test Acc	Test LL	Time
SVGP		384, 1K	84.86%	-0.82	76.35%	-1.04	0.392 s/iter
SOLVE-GP		384 + 384, 1K + 1K	87.59%	-0.72	77.80%	-0.98	0.657 s/iter
SVGP		768, 2K	87.25%	-0.74	77.46%	-0.98	1.104 s/iter

Table 10: 3-layer deep convolutional GPs for CIFAR-10 classification.

		Inducing Points	Train Acc	Train LL	Test Acc	Test LL	Time
SVGP		384, 384, 1K	87.70%	-0.67	78.76%	-0.88	0.418 s/iter
SOLVE-GP		(384 + 384) \times 2, 1K + 1K	89.88%	-0.57	80.30%	-0.79	0.752 s/iter
SVGP		768, 768, 2K	90.01%	-0.58	80.33%	-0.82	1.246 s/iter