Learning to Learn with Gaussian Processes

Quoc Phong Nguyen[1] **Bryan Kian Hsiang Low**[1] **Patrick Jaillet**[2]

[1]Department of Computer Science, National University of Singapore, Republic of Singapore
[2]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA

Abstract

This paper presents *Gaussian process meta-learning* (GPML) for few-shot regression, which explicitly exploits the distance between regression problems/tasks using a novel *task kernel*. It contrasts sharply with the popular *metric-based meta-learning* approach which is based on the distance between data inputs or their embeddings in the few-shot learning literature. Apart from the superior predictive performance by capturing the diversity of different tasks, GPML offers a set of *representative tasks* that are useful for understanding the task distribution. We empirically demonstrate the performance and interpretability of GPML in several few-shot regression problems involving a multi-modal task distribution and real-world datasets.

1 INTRODUCTION

Though a state-of-the-art machine learning model can now be trained to reach an unprecedented level of predictive accuracy in a given learning task, this is often data-inefficient and time-consuming. It is therefore desirable to be able to exploit the learning experience from similar tasks for building a model that can rapidly adapt to a new task using only a limited quantity of its training data, which is the goal of *"learning to learn"*, also known as *meta-learning* [Vanschoren, 2018].

The seminal work of [Finn et al., 2017] proposes a *gradient-based model agnostic meta-learning* (MAML) algorithm: In essence, MAML searches for model parameters to serve as a *common* initializer for any new task such that the model using this initializer can be quickly adapted to the new task by requiring only a few *gradient descent* (GD) updates based on its training data. However, such an adaptation step (i.e., only a few GD updates from a common initializer) may no longer be adequate if there is considerable *diversity* among the new tasks. Therefore, apart from improving the time and/or quality of the adaptation step in MAML [Grant et al., 2018, Nichol et al., 2018, Rajeswaran et al., 2019, Yoon et al., 2018], it appears beneficial to consider multiple initializers and their diversity to match that among the tasks. Furthermore, the incapability of a common initializer to explain the diversity among tasks raises the need of generating a set of multiple *representative tasks* that are able to characterize this diversity.

Nonetheless, these desirable goals require us to overcome the nontrivial challenge of defining the distance between tasks in order to quantify the above diversity. Although the notion of a distance metric has been widely explored in few-shot classification for images [Allen et al., 2019, Koch et al., 2015, Snell et al., 2017, Vinyals et al., 2016], one cannot directly implement their ideas to explicitly model the distance between tasks. This is because they only consider the distance between the *embeddings* (latent representations) of images which are the training/test inputs to a task. Yet, in meta-learning, the inputs to a meta-learning algorithm are tasks represented by datasets. Hence, instead of the distance between the inputs of a task, a meta-learning algorithm should be based on the distance between tasks/datasets which is challenging to define explicitly. While introducing full context embeddings[1] can be viewed as an attempt to implicitly measure the distance between tasks, this approach does not outperform the prototypical networks based on the distance between the embeddings of images [Snell et al., 2017]. It is possibly due to the additional complexity of encoding the training set. Furthermore, it is difficult to interpret the distance between classification tasks as their training sets are transformed through the embedding function [Vinyals et al., 2016]. As a result, designing a distance metric between tasks remains an open question.

Our main contribution is to cast the few-shot regression prob-

[1]Full context embeddings include the whole training set (i.e., the support set) as inputs to the embedding function in the matching network approach [Vinyals et al., 2016].

lem as a *Gaussian process* (GP), which results in *Gaussian process meta-learning* (GPML) (Section 3). GPML belongs to both the gradient-based meta-learning approach, which searches for an initializer for a new task, and the metric-based meta-learning approach that is based on a distance metric. Different from existing approaches, the dependency between initializers and tasks is explicitly modelled via GP and the distance metric is between tasks.

It is not obvious that GP with a distance-based kernel is able to model a function whose inputs are tasks. Unlike the input space in the traditional GP regression that is endowed with the Euclidean distance, different tasks are defined by different datasets that are often generated from disjoint subsets of the input domain. This renders direct application of the Euclidean distance to datasets meaningless. While one can naively learn the underlying functions of tasks to measure a sensible distance between them, this approach is hardly feasible due to the limited amount of training data available to each task and the computational cost. Thus, efficiently measuring the distance between tasks without knowing their underlying functions is challenging.

By exploiting the structure of a sparse GP model [Quiñonero-Candela and Rasmussen, 2005], we manage to overcome this challenge. The main intuition is that a task can be defined by its distance to a number of "representative" tasks, which we call *inducing tasks* (i.e., inspired by the sparse GP literature) (Section 3.3). Though constructing a useful distance between arbitrary tasks is indeed daunting, restricting to the distance between a task and an inducing task makes the problem more palatable. This is because we have total control over the design of the inducing task, unlike a training task defined strictly by its dataset. Furthermore, the training of GPML naturally encourages the diversity and representativeness of these inducing tasks. It implies that given more computational resources, increasing the number of inducing tasks can improve the performance of GPML, as compared to MAML. The benefits are twofold: (a) GPML is likely to predict well for tasks in different modes of the distribution (especially when the number of inducing tasks is large), and (b) these inducing tasks are useful for the interpretation of the task distribution. We empirically evaluate GPML in synthetic problems: learning a random cosine function, learning a mixture of cosine and linear functions (i.e., a multimodal task distribution), and introduce new real-world applications of few-shot regression in collaborative filtering and recovering missing data.

2 BACKGROUND AND NOTATIONS

2.1 REGRESSION TASK

A regression task aims to construct a *regression model* that learns a function mapping an input to an output based on a set of *training examples*, i.e., *training set*. A training example is a pair of an input and its desired output. Let $y_t : \mathcal{D}_x \to \mathcal{D}_y$ denote the unknown underlying function of a task t where $\mathcal{D}_x$ and $\mathcal{D}_y$ are the domains of the input and the output, respectively. The training set of a task t is denoted as $(\mathcal{X}_t, y_t(\mathcal{X}_t))$ where $\mathcal{X}_t$ and $y_t(\mathcal{X}_t) \triangleq (y_t(\mathbf{x}))_{\mathbf{x}\in\mathcal{X}_t}$ are the training inputs and their desired training outputs, respectively. For simplicity, we assume that $\mathcal{D}_y$ is 1-dimensional. The distribution of the input of a task t is denoted as $\mathcal{P}_t$. Let the regression model parameterized by the *task parameters* $\boldsymbol{\theta}$ be denoted as $f(\mathbf{x}, \boldsymbol{\theta})$.

Given the above model $f(\mathbf{x}, \boldsymbol{\theta})$ of a task t, the goal of a regression algorithm is to minimize the expectation of a *loss function* over the input distribution of the task, i.e., $\mathbb{E}_{\mathbf{x}\sim\mathcal{P}_t} l(y_t(\mathbf{x}), \boldsymbol{\theta})$ where $l(y_t(\mathbf{x}), \boldsymbol{\theta})$ is a loss function measuring the dissimilarity between the predicted output $f(\mathbf{x}, \boldsymbol{\theta})$ and the desired output $y_t(\mathbf{x})$, e.g., the *mean squared error* (MSE): $l(y_t(\mathbf{x}), \boldsymbol{\theta}) = (y_t(\mathbf{x}) - f(\mathbf{x}, \boldsymbol{\theta}))^2$. As $y_t(\mathbf{x})$ is not available for all $\mathbf{x} \in \mathcal{D}_x$, the training of the model minimizes the loss on the training set, i.e., *training loss*: $l(y_t(\mathcal{X}_t), \boldsymbol{\theta}) \triangleq |\mathcal{X}_t|^{-1} \sum_{\mathbf{x}\in\mathcal{X}_t} l(y_t(\mathbf{x}), \boldsymbol{\theta})$. It results in the optimal parameters $\boldsymbol{\theta}^* \triangleq \operatorname{argmin}_{\boldsymbol{\theta}} l(y_t(\mathcal{X}_t), \boldsymbol{\theta})$. To evaluate the performance of a model with $\boldsymbol{\theta}^*$, a *test set* is set aside, denoted as $(\mathcal{X}_t^*, y_t(\mathcal{X}_t^*))$ where $\mathcal{X}_t^*$ are drawn from $\mathcal{P}_t$. Then, the performance measure, called *test loss*, is defined as $l(y_t(\mathcal{X}_t^*), \boldsymbol{\theta}^*) \triangleq |\mathcal{X}_t^*|^{-1} \sum_{\mathbf{x}\in\mathcal{X}_t^*} l(y_t(\mathbf{x}), \boldsymbol{\theta}^*)$.

2.2 FEW-SHOT REGRESSION AND MAML

The few-shot regression problem, in particular the k-shot regression problem, aims to construct a *meta model* that maps a regression task defined by a training set of size k (i.e., k input-output pairs) to the underlying regression function of the task, which is based on similar regression tasks (*training tasks*) [Finn et al., 2017]. Similar tasks are defined as tasks drawn from the same task distribution, e.g., a sinusoidal waves with known distributions of amplitude, phase, and frequency. A *training task* t is a tuple $((\mathcal{X}_t, y_t(\mathcal{X}_t), \mathcal{X}_t^*), y_t(\mathcal{X}_t^*))$ where $|\mathcal{X}_t| = k$. These training tasks constitute the *meta training set*, denoted as $\mathcal{T}$. To evaluate the performance of a meta model, a *meta test set* is set aside, denoted as $\mathcal{T}_*$. Each *test task* in $\mathcal{T}_*$ is defined in a similar way as a training task and is drawn from the same task distribution as that of $\mathcal{T}$. It is noted that $|\mathcal{X}_t^*|$ is not necessarily equal to k. However, we assume that $|\mathcal{X}_t^*| = k$ if $t \in \mathcal{T}$ to conform with the setting of MAML while we use a large test set $\mathcal{X}_t^*$ for $t \in \mathcal{T}_*$ to accurately measure the performance of the meta model.

In this paper, we follow the approach of MAML [Finn et al., 2017], which is to find an initializer of a task such that it can be quickly adapted to the task by performing a small number m of *gradient descent* (GD) updates. Let $\boldsymbol{\theta}_t^\circ$ denote the initializer of a task t, and $g_t^{(m)}(\boldsymbol{\theta}_t^\circ)$ denote a function equivalent to performing m GD updates on the training loss

$l(y_t(\mathcal{X}_t), \boldsymbol{\theta})$ starting from $\boldsymbol{\theta} = \boldsymbol{\theta}_t^\circ$, i.e.,

$$g_t^{(i)}(\boldsymbol{\theta}_t^\circ) \triangleq g_t^{(1)}(g_t^{(i-1)}(\boldsymbol{\theta}_t^\circ)) \quad \forall i \geq 2 \, ,$$
$$g_t^{(1)}(\boldsymbol{\theta}_t^\circ) \triangleq \boldsymbol{\theta}_t^\circ - \delta \nabla l_t(y_t(\mathcal{X}_t), \boldsymbol{\theta}_t^\circ)$$

where $\nabla l_t(y_t(\mathcal{X}_t), \boldsymbol{\theta}_t^\circ)$ is the gradient of $l_t(y_t(\mathcal{X}_t), \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ at $\boldsymbol{\theta} = \boldsymbol{\theta}_t^\circ$ and $\delta > 0$ is the step size of GD. The number m of GD updates is selected manually, which allows for a trade-off between the speed of the adaptation to a new task and the performance of the updated $g_t^{(m)}(\boldsymbol{\theta}_t^\circ)$.

Let $h(t, \boldsymbol{\phi})$ denote a mapping from a training set $(\mathcal{X}_t, y_t(\mathcal{X}_t))$ to an initializer $\boldsymbol{\theta}_t^\circ$ where $\boldsymbol{\phi}$ is the parameters of h, i.e., the *meta parameters*. Then, given the training set of a task t, the meta model predicts an initializer $\boldsymbol{\theta}_t^\circ = h(t, \boldsymbol{\phi})$ of the regression model f. Given the initializer, the adaptation via m GD updates produces a function $f(\mathbf{x}, g_t^{(m)}(h(t, \boldsymbol{\phi})))$ for the regression task. The goal of meta-learning is that this function $f(\mathbf{x}, g_t^{(m)}(h(t, \boldsymbol{\phi})))$ is close to the underlying function of the task t, which can be quantified by the test loss of the task t, i.e., $l(y_t(\mathcal{X}_t^*), g_t^{(m)}(h(t, \boldsymbol{\phi})))$. Therefore, to train of the meta model, i.e., the *meta training*, is to minimize the average of the test loss over tasks in the meta training set:

$$l(\mathcal{T}, \boldsymbol{\phi}) \triangleq |\mathcal{T}|^{-1} \textstyle\sum_{t \in \mathcal{T}} l\left(y_t(\mathcal{X}_t^*), g_t^{(m)}(h(t, \boldsymbol{\phi}))\right) \, .$$

Let the optimal meta parameters obtained by meta training be defined as $\boldsymbol{\phi}_* \triangleq \operatorname{argmin}_{\boldsymbol{\phi}} l(\mathcal{T}, \boldsymbol{\phi})$. To measure the performance of the meta-learning algorithm, the average of the test loss over tasks in the meta test set is computed, i.e., $l(\mathcal{T}_*, \boldsymbol{\phi}_*)$. Under the above procedure, MAML is an algorithm where h is a constant function, i.e., $h(t, \boldsymbol{\phi}) = \boldsymbol{\theta}$ for all t where $\boldsymbol{\theta}$ is the common initializer that does not depend on the task t. In other words, the meta parameters coincide with the task parameters of f, i.e., $\boldsymbol{\phi} = \boldsymbol{\theta}$. Although the adaptation step takes into account the training set of the task (i.e, $g_t^{(m)}(\boldsymbol{\theta})$), it is often the case that the number m of GD updates is small for fast adaptation. So, the influence of $y_t(\mathcal{X}_t)$ may be limited, especially when the task distribution is multimodal, as illustrated in our experiments. Therefore, in the next section, we introduce a meta model that is capable of expressing the relationship between the training set $(\mathcal{X}_t, y_t(\mathcal{X}_t))$ of a task t and its initializer $\boldsymbol{\theta}_t^\circ = h(t, \boldsymbol{\phi})$, i.e., learning a *task-dependent initializer*.

3 LEARNING TO LEARN WITH GAUSSIAN PROCESS

3.1 GAUSSIAN PROCESSES

In this section, we present the *Gaussian process* (GP) in the context of meta-learning. In other words, GP is used to model an unknown function mapping from the training set of a task t to its initializer $\boldsymbol{\theta}_t^\circ$, denoted as $h(t)$.[2] For simplicity, we only consider a 1-dimensional θ. In practice, $\boldsymbol{\theta}$ is a vector of n elements (e.g., n parameters of a neural network). Then, $h(t)$ is modelled with n GPs, each of which is described as below. Every finite subset of $\{h(t)\}_{t \in \mathcal{T} \cup \mathcal{T}_*}$ follows a multivariate Gaussian distribution. The GP is fully specified by its *prior* mean $\mathbb{E}[h(t)]$ and covariance $k(t, t') \triangleq \text{cov}[h(t), h(t')]$ for all $t, t' \in \mathcal{T} \cup \mathcal{T}_*$. In this work, we use a constant mean function $\mathbb{E}[h(t)] = \bar{\mu}$ for all $t \in \mathcal{T} \cup \mathcal{T}_*$. Suppose a column vector $\boldsymbol{\theta}_{\mathcal{T}'} \triangleq (h(t) + \epsilon)_{t \in \mathcal{T}'}^\top$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ is observed, the GP posterior $p(\theta_t^\circ | \boldsymbol{\theta}_{\mathcal{T}'}) = p(h(t) | \boldsymbol{\theta}_{\mathcal{T}'})$ for any task $t \in \mathcal{T} \cup \mathcal{T}_*$ is a Gaussian distribution with the following posterior mean and posterior variance [Rasmussen and Williams, 2006]:

$$\begin{aligned} \mu_{t|\mathcal{T}'} &\triangleq \mathbf{k}_{t\mathcal{T}'}(\mathbf{K}_{\mathcal{T}'\mathcal{T}'} + \sigma_n^2 \mathbf{I})^{-1}(\boldsymbol{\theta}_{\mathcal{T}'} - \bar{\mu}) + \bar{\mu} \, , \\ \sigma_{t|\mathcal{T}'}^2 &\triangleq k(t,t) - \mathbf{k}_{t\mathcal{T}'}(\mathbf{K}_{\mathcal{T}'\mathcal{T}'} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_{\mathcal{T}'t} \end{aligned} \tag{1}$$

where $\mathbf{k}_{t\mathcal{T}'}$ is a row vector whose elements are $k(t, t')$ for $t' \in \mathcal{T}'$, $\mathbf{k}_{\mathcal{T}'t} \triangleq \mathbf{k}_{t\mathcal{T}'}^\top$, $\mathbf{K}_{\mathcal{T}'\mathcal{T}'}$ is a matrix whose element at the i-th row and the j-th column is $k(t_i, t_j)$ such that t_i and t_j are, respectively, the i-th and j-th elements in $\mathcal{T}'$, and $\mathbf{I}$ is an identity matrix of the same size as $\mathbf{K}_{\mathcal{T}'\mathcal{T}'}$.

The covariance $\text{cov}[h(t), h(t')]$ is defined by a *kernel* (or covariance function) $k(t, t')$, which we call a *task kernel*. In other words, the task kernel measures the covariance between initializers θ_t° and $\theta_{t'}^\circ$ based on the distance between these tasks, i.e., between their training sets $(\mathcal{X}_t, y_t(\mathcal{X}_t))$ and $(\mathcal{X}_{t'}, y_{t'}(\mathcal{X}_{t'}))$. Hence, the design of the task kernel plays a crucial role in enhancing the predictive performance of GP with the correlation between initializers.

3.2 TASK KERNEL

This section presents a novel *stationary kernel* that is a function of the distance between $(\mathcal{X}_t, y_t(\mathcal{X}_t))$ and $(\mathcal{X}_{t'}, y_{t'}(\mathcal{X}_{t'}))$. Our fresh perspective is to view $y_t(\mathcal{X}_t)$ as a vector whose elements are $y_t(\mathbf{x})$ indexed by $\mathbf{x} \in \mathcal{X}_t$. However, the challenge is that the two index sets ($\mathcal{X}_t$ and $\mathcal{X}_{t'}$) are often not the same to evaluate the difference between $y_t(\mathcal{X}_t)$ and $y_t(\mathcal{X}_{t'})$. A naïve solution is to consider the intersection set $\mathcal{X}_t \cap \mathcal{X}_{t'}$, i.e., the largest set containing training inputs shared by these tasks. Then, inspired by the well-known *squared exponential* (SE) kernel, we define a task kernel as

$$k(t, t' | \mathcal{X}_t \cap \mathcal{X}_{t'}) \triangleq \sigma_s^2 \exp\left(\sum_{\mathbf{x} \in \mathcal{X}_t \cap \mathcal{X}_{t'}} \frac{-(y_t(\mathbf{x}) - y_{t'}(\mathbf{x}))^2}{2 |\mathcal{X}_t \cap \mathcal{X}_{t'}| \, l_{\mathbf{x}}^2} \right) \tag{2}$$

where σ_s^2 is the *signal variance* and $l_{\mathbf{x}}$ is the *length-scale*. Both of these parameters are known as the *hyperparameters* of the GP. A minor difference from the definition of

[2]As h is modelled with a GP, which is a nonparametric model, we write $h(t)$ without its parameters $\boldsymbol{\phi}$.

the SE kernel is the factor $1/|\mathcal{X}_t \cap \mathcal{X}_{t'}|$ which reduces the dependence between the kernel magnitude and $|\mathcal{X}_t \cap \mathcal{X}_{t'}|$.

An obvious drawback of the above task kernel is that the intersection set $\mathcal{X}_t \cap \mathcal{X}_{t'}$ may be small, or even empty. In such cases, the evaluation of the task kernel $k(t, t'|\mathcal{X}_t \cap \mathcal{X}_{t'})$ ignores most of the information in the training set (if $\mathcal{X}_t \cap \mathcal{X}_{t'}$ is small) or may even be impossible (if $\mathcal{X}_t \cap \mathcal{X}_{t'} = \emptyset$).

Remark 1. We would like to highlight that the requirements for the task kernel in few-shot regression problems in this section are different from those for the task kernel in the *multi-task and multi-output* GP (MT/MO GP) literature [Bonilla et al., 2008, Alvarez and Lawrence, 2011]. In particular, few-shot regression problems often involve a large number of training tasks (e.g., an unlimited number of training tasks in Sec. 4.1), which cannot be handled by MT/MO GP. Moreover, MT/MO GP requires the model of each task to be a GP while few-shot regression does not. In this work, we employ neural networks to model tasks, which makes constructing a task kernel nontrivial.

3.3 INDUCING TASK

To resolve the above issue, we introduce a set $\mathcal{T}_u$ of n_u *inducing tasks* such that $\mathcal{X}_t \cap \mathcal{X}_{t_u} = \mathcal{X}_t$ for all $t_u \in \mathcal{T}_u$ and $t \in \mathcal{T} \cup \mathcal{T}_*$. Since $\mathcal{X}_t$ can be any subset of the input domain $\mathcal{D}_x$, it follows that $\mathcal{X}_{t_u} = \mathcal{D}_x$ for all $t_u \in \mathcal{X}_u$. In other words, it is necessary that the training inputs of any inducing task constitute the whole input domain $\mathcal{D}_x$. Therefore, unlike tasks in $\mathcal{T} \cup \mathcal{T}_*$, an inducing task $t_u \in \mathcal{T}_u$ is defined by its optimal parameter, denoted as $\theta^*_{t_u}$. Given $\theta^*_{t_u}$, the output $y_{t_u}(\mathbf{x})$ at any input $\mathbf{x} \in \mathcal{D}_x$ can be easily computed as $f(\mathbf{x}, \theta^*_{t_u})$. As a result, the task kernel between a task t and an inducing task $t_u \in \mathcal{T}_u$ can be computed as

$$k(t, t_u|\mathcal{X}_t) \triangleq \sigma_s^2 \exp\left(\sum_{\mathbf{x}\in\mathcal{X}_t} \frac{-\left(y_t(\mathbf{x}) - f\left(\mathbf{x}, \theta^*_{t_u}\right)\right)^2}{2|\mathcal{X}_t|\; l^2_{\mathbf{x}}}\right)$$

where $y_t(\mathcal{X}_t)$ is fully utilized, unlike (2).

On the other hand, evaluating the task kernel between tasks $t, t' \in \mathcal{T} \cup \mathcal{T}_*$ is still difficult due to the possibility of $\mathcal{X}_t \cap \mathcal{X}_{t'} = \emptyset$. This is gracefully resolved by assuming the conditional independence among θ°_t for $t \in \mathcal{T} \cup \mathcal{T}_*$ given $\boldsymbol{\theta}^\circ_{\mathcal{T}_u} \triangleq (\theta^\circ_{t_u})^\top_{t_u\in\mathcal{T}_u}$ such that the inference of GP does not require the evaluation of $k(t, t')$ for any $t, t' \in \mathcal{T} \cup \mathcal{T}_*$. It implies that all the information in the meta training set can be summarized by $\boldsymbol{\theta}^\circ_{\mathcal{T}_u}$, which was introduced in the sparse GP literature [Quiñonero-Candela and Rasmussen, 2005].

Lastly, we need to evaluate the task kernel between inducing tasks. As the output of inducing tasks at any input can be obtained, we can define a set of *pivot inputs* $\mathcal{X}_p \subset \mathcal{D}_x$ to evaluate the task kernel between any $t_u, t'_u \in \mathcal{T}_u$ as

$$k(t_u, t'_u|\mathcal{X}_p) \triangleq \sigma_s^2 \exp\left(\sum_{\mathbf{x}\in\mathcal{X}_p} \frac{-\left(f(\mathbf{x}, \theta^*_{t_u}) - f(\mathbf{x}, \theta^*_{t'_u})\right)^2}{2|\mathcal{X}_p|\; l^2_{\mathbf{x}}}\right)$$

Intuitively, a good initializer of a task should be close to its optimal parameter so that the initializer can be quickly updated to the optimum through a few GD updates. We translate this intuition into an assumption: $\boldsymbol{\theta}^*_{\mathcal{T}_u}$ follows a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}^\circ_{\mathcal{T}_u}, \sigma_n^2\mathbf{I})$ where σ_n^2 is learned during meta training as a meta parameter. It follows that the posterior distribution of θ°_t for any $t \in \mathcal{T} \cup \mathcal{T}_*$ is a Gaussian distribution, i.e, $p(\theta^\circ_t|\boldsymbol{\theta}^*_{\mathcal{T}_u}) \triangleq \mathcal{N}(\tilde{\mu}_{t|\mathcal{T}_u}, \tilde{\sigma}^2_{t|\mathcal{T}_u})$ where $\tilde{\mu}_{t|\mathcal{T}_u}$ and $\tilde{\sigma}^2_{t|\mathcal{T}_u}$ are defined in a similar manner to (1):

$$\begin{aligned}\tilde{\mu}_{t|\mathcal{T}_u} &\triangleq \mathbf{k}_{t\mathcal{T}_u|\mathcal{X}_t}(\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_p} + \sigma_n^2\mathbf{I})^{-1}\left(\boldsymbol{\theta}^*_{\mathcal{T}_u} - \bar{\mu}\right) + \bar{\mu}\\ \tilde{\sigma}^2_{t|\mathcal{T}_u} &\triangleq k(t, t|\mathcal{X}_t) - \mathbf{k}_{t\mathcal{T}_u|\mathcal{X}_t}(\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_p} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_{\mathcal{T}_u t|\mathcal{X}_t}\,.\end{aligned} \tag{3}$$

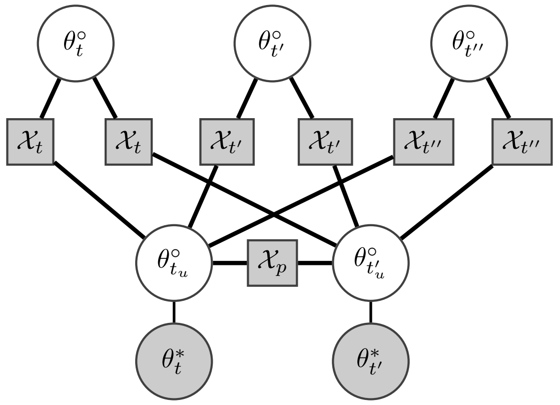
A shrewd reader may notice that the task kernel does not have a consistent definition in (3): $\mathbf{k}_{t\mathcal{T}_u|\mathcal{X}_t}$ is computed based on $\mathcal{X}_t$ while $\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_p}$ is computed based on $\mathcal{X}_p$. Although this approach still gives a competitive empirical performance in our experiments, the GP posterior with a consistent definition of the task kernel should be defined as:

$$\begin{aligned}\mu_{t|\mathcal{T}_u} &\triangleq \mathbf{k}_{t\mathcal{T}_u|\mathcal{X}_t}(\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_t} + \sigma_n^2\mathbf{I})^{-1}\left(\boldsymbol{\theta}^*_{\mathcal{T}_u} - \bar{\mu}\right) + \bar{\mu}\\ \sigma^2_{t|\mathcal{T}_u} &\triangleq k(t, t|\mathcal{X}_t) - \mathbf{k}_{t\mathcal{T}_u|\mathcal{X}_t}(\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_t} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_{\mathcal{T}_u t|\mathcal{X}_t}\end{aligned} \tag{4}$$

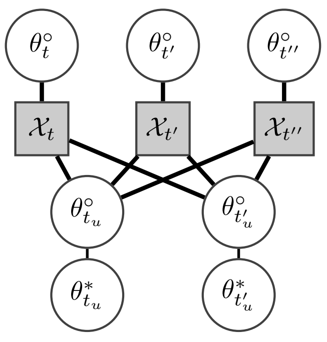
where all kernels are defined based on $\mathcal{X}_t$ consistently.

Point estimate of the initializer. Given the posterior in (3) or (4), we can define h as the posterior mean, i.e., $h(t) = \tilde{\mu}_{t|\mathcal{T}_u}$ or $h(t) = \mu_{t|\mathcal{T}_u}$. This effectively means that the GP posterior variance is set to 0 and the meta model is similar to a sparse GP model, namely the *subset of regressors* [Quiñonero-Candela and Rasmussen, 2005, Silverman, 1985, Smola and Bartlett, 2001]. The difference is that instead of estimating a distribution of the inducing variables, we only learn point estimates of a noisy version of the inducing variables. The meta parameters consist of the optimal parameters of the inducing tasks ($\boldsymbol{\theta}^*_{\mathcal{T}_u}$), the GP prior mean ($\bar{\mu}$), and the GP hyperparameters (σ_s^2, $l_{\mathbf{x}}$, and σ_n^2). Optimizing $\boldsymbol{\theta}^*_{\mathcal{T}_u}$ plays the role of optimizing both inducing variables and inducing inputs in the traditional sparse GP regression.

The computational advantage of $\tilde{\mu}_{t|\mathcal{T}_u}$ (3) is that the inversion of $\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_p} + \sigma_n^2\mathbf{I}$, which incurs $\mathcal{O}(n_u^3)$, can be reused in the evaluation of $\tilde{\mu}_{t|\mathcal{T}_u}$ for different tasks. On the other hand, $\mu_{t|\mathcal{T}_u}$ (4) requires the inversion of different matrices $\mathbf{K}_{\mathcal{T}_u\mathcal{T}_u|\mathcal{X}_t} + \sigma_n^2\mathbf{I}$ for different tasks. Therefore, we call the approach in (3) *Gaussian process meta-learning lite* (GPML-lite) (described in Fig. 1a) and the approach in (4) GPML (described in Fig. 1b). The GPML model consists of different GP models for different tasks all of which share the inducing tasks $\mathcal{T}_u$ and the GP hyperparameters, but have different evaluations of the task kernel.



(a) GPML-lite



(b) GPML

Figure 1: Plots of GPML-lite and GPML models: inducing tasks are $\mathcal{X}_u = \{t_u, t'_u\}$; shaded square boxes show the set of inputs that are used to compute the covariance between initializers in the circles connected to them.

Probabilistic estimate of the initializer. Equations in (4) specify a Gaussian posterior of $h(t) = \theta_t^\circ$. Thus, it can be used to define a probabilistic estimate of the initializer θ_t°. In such case, during meta training, the expectation of the test loss over $\theta_t^\circ \sim p(\theta_t^\circ | \boldsymbol{\theta}^*_{\mathcal{T}_u})$, i.e.,

$$\mathbb{E}_{\theta_t^\circ \sim p(\theta_t^\circ | \boldsymbol{\theta}^*_{\mathcal{T}_u})} \left(|\mathcal{T}|^{-1} \sum_{t \in \mathcal{T}} l(y_t(\mathcal{X}_t^*), g_t^{(m)}(\theta_t^\circ)) \right) .$$

is minimized with a stochastic optimization method. We call this method *GPML-Bayes* to differentiate from GPML and GPML-lite in the experiments.

Remark 2 (Representativeness of inducing tasks)**.** Similar to sparse GP regression with the SE kernel, the initializer of a task is correlated with that of an inducing task if their distance is small. In contrast, if the distance of a task to all inducing tasks is large, the initializer of the task is similar to the GP prior mean $\bar{\mu}$. This implies that the inducing tasks should be representative of the task distribution if the number of inducing tasks is large.

Recall that we have been developing a meta-learning algorithm for a 1-dimensional θ_t. When $\boldsymbol{\theta}_t$ is a vector of many values (e.g., the parameters of a neural network), the mappings from the training set of t to different elements of $\boldsymbol{\theta}_t$ are modelled with different GP models sharing the inducing tasks $\mathcal{T}_u$. It is the shared inducing tasks that forge a relationship between elements of $\boldsymbol{\theta}_t$. As different model parameters are modelled with different GPs, their computation can be executed in parallel. Thus, the computational complexity of GPML mainly depends on the number of inducing tasks, whose scalability is similar to sparse GP models. Furthermore, the number of inducing tasks mostly depends on the complexity of the task distribution (e.g., multimodal). To reduce the number of GPs, we can model a subset of the task parameters with GPs and the other task parameters with MAML or combine GPML with the *latent embedding optimization* (LEO) approach [Rusu et al., 2019]. In this paper, all task parameters are modelled with GPs. We also use a constant length-scale $l_{\mathbf{x}} = l$ in our experiments. A possible improvement is to implement the *automatic relevance determination* in the kernel, i.e., defining an input-dependent length-scale [Rasmussen and Williams, 2006].

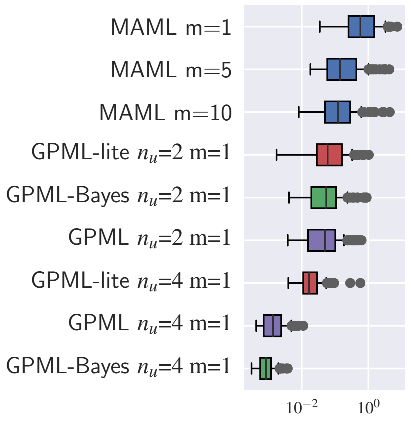
4 EXPERIMENTS

In these experiments, the performance is visualized by plotting the boxplot of the MSE of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ on the test set over 100 tasks in the task distribution. From the top to bottom of a boxplot, algorithms are positioned in the descending order of the average of MSE. Our models f are feed forward neural network with ReLU activation function in these experiments. Due to the lack of space, description of the model f and some comparisons in our experiments are deferred to the appendix.

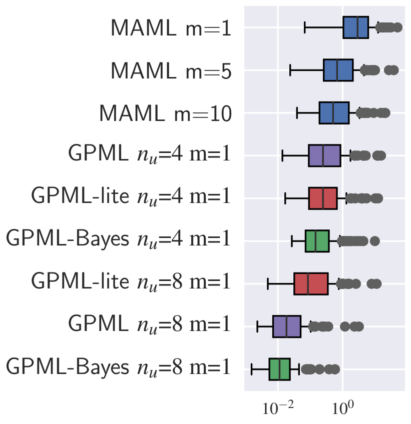
4.1 SYNTHETIC FUNCTIONS

There are two 5-shot regression problems on learning a random cosine function, namely *cosine tasks*, and a mixture of cosine and linear functions, namely *cosine and linear tasks* (further details are in Appendix B). The empirical performance of our approaches including GPML-lite, GPML, and GPML-Bayes are compared against MAML with different number m of GD updates.

Predictive performance. The MSE values of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ trained with MAML, GPML-lite, GPML, and GPML-Bayes are shown in Fig. 2. Even though the performance of MAML improves as m increases from 5 to 10, MAML is still outperformed by our approaches as it only learns 1 initializer for all tasks. Given more computational resources, the performance of our methods is improved by increasing n_u (e.g., from 2 to 4 in Fig. 2a and from 4 to 8 in Fig. 2b) to capture the task diversity. Although GPML-lite outperforms MAML, its performance



(a) Cosine tasks.



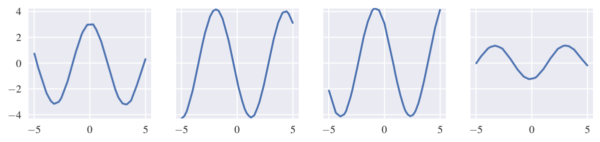
(b) Cosine and linear tasks.

Figure 2: Plots of the MSE of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ for 5-shot regression.

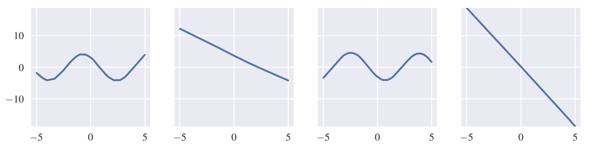
is not as good as GPML and GPML-Bayes. This is noticeable when we compare GPML-lite, GPML, and GPML-Bayes with $n_u = 4$ in Fig. 2a and $n_u = 8$ in Fig. 2b. It is because the task kernels in the posterior of GPML-lite are not defined consistently in the posterior as explained in the paragraph after (3).

Representativeness of inducing tasks. Fig. 3 shows 4 inducing tasks obtained from training GPML-Bayes on cosine tasks and cosine and linear tasks. These inducing tasks modelled with neural networks successfully capture cosine functions of different phases and amplitudes in the former (Fig. 3a) and cosine and linear functions in the latter (Fig. 3b). Thus, we observe that inducing tasks are representative of the task distribution. Furthermore, we can observe that a random task in the task distribution is close to an inducing task because the distribution of MSE between a random task and its closest inducing task peaks at 0 in Fig. 4.

Visualization of predicted tasks. Figs. 5 and 6 show $f(\mathbf{x}, \boldsymbol{\theta}_t^\circ)$ and $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ trained with MAML ($m = 10$) and GPML-Bayes ($m = 1$ and different values of n_u)

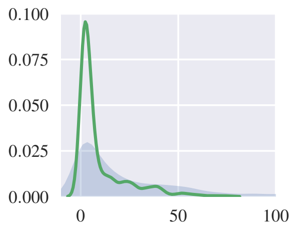


(a) Cosine tasks.

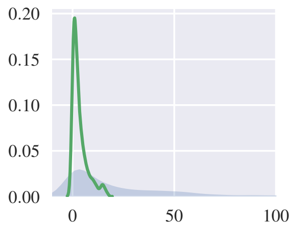


(b) Cosine and linear tasks.

Figure 3: Plots of inducing tasks.



(a) Cosine tasks.



(b) Cosine and linear tasks.

Figure 4: Plots of the distribution of MSE between a random task and its closest inducing task (plotted as green lines). The shaded area shows the distribution of MSE between random tasks in the task distribution.

on 3 regression tasks. When GPML-Bayes does not fit the tasks well ($n_u = 2$ in Fig. 5 and $n_u = 4$ in Fig. 6), the variance of the predicted functions is large. When GPML-Bayes fits all 3 tasks well (thanks to the representative inducing tasks in Fig. 3), the predicted functions have large variance. Regarding MAML, it only learns 1 initializer for all 3 tasks, so its performance is not as good as GPML-Bayes even with $m = 10$.

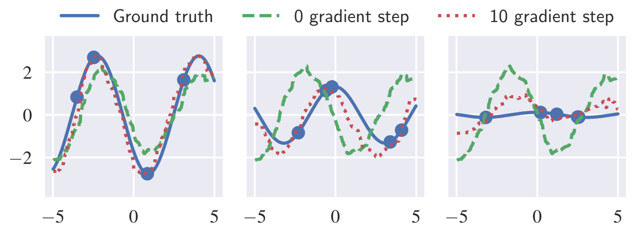
We also perform a 10-shot regression with a noisy sine function adopted from the *Bayesian MAML* (BMAML) work [Yoon et al., 2018] which is discussed in Appendix C.

4.2 REAL-WORLD DATASETS

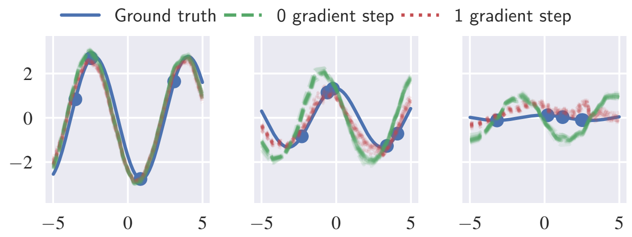
Table 1: Cross entropy and MAE of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ in experiments with the sushi dataset.

	Cross entropy	MAE
MAML $m = 1$	0.440 ± 0.086	1.021 ± 0.543
GPML $n_u = 4$ $m = 1$	0.418 ± 0.111	0.906 ± 0.531
GPML $n_u = 8$ $m = 1$	0.417 ± 0.112	0.901 ± 0.534

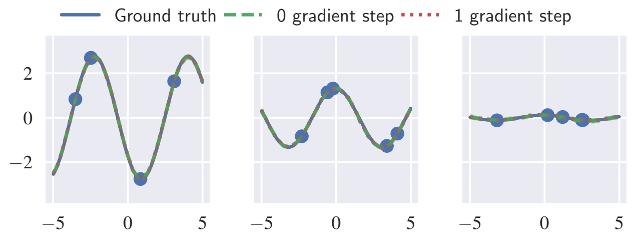
Sushi dataset. We introduce the use of few-shot learning in collaborative filtering with the sushi dataset [Kamishima,



(a) MAML $m = 10$.

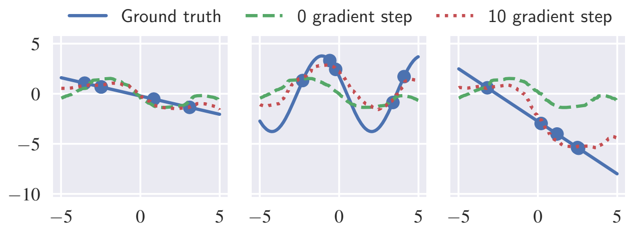


(b) GPML-Bayes $m = 1$ $n_u = 2$.

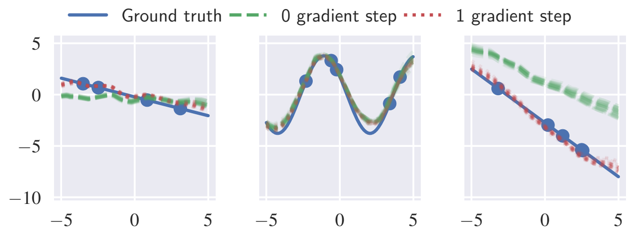


(c) GPML-Bayes $m = 1$ $n_u = 4$.

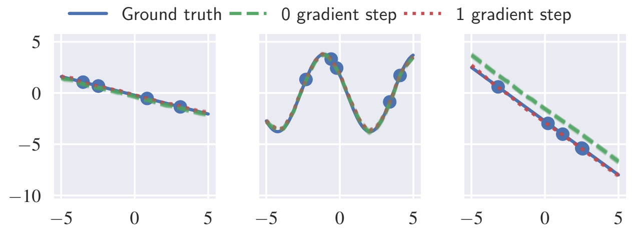
Figure 5: Plots of predicted functions.



(a) MAML $m = 10$.



(b) GPML-Bayes $m = 1$ $n_u = 4$.



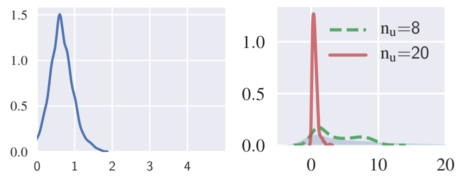
(c) GPML-Bayes $m = 1$ $n_u = 8$.

Figure 6: Plots of the predicted functions.

2003] which consists of 5000 user ratings in a 5-point scale. Each rating is over a subset of 100 types of sushi identified by 7 features. We view the ratings of each user over different types of sushi as 1 dataset/task (i.e., different people have different preferences over sushi). By randomly drawing a user in 5000 users, we generate a task which outputs a rating given a type of sushi represented by a vector of 7 features. The 5-shot learning model is to predict the a user's rating of a type of sushi given the features of the sushi and the ratings of that user on 5 random types of sushi. Table 1 shows that GPML outperforms MAML in both the cross entropy and the *mean absolue error* (MAE) loss. It is also observed that a random task in the task distribution is close to an inducing task because the distribution of MSE between a random task and its closest inducing task peaks around 0 in Fig. 7a. In other words, inducing tasks are representative of the task distribution. This can give us more insight, e.g., an inducing task shows that the highest rated sushi (i.e., 5) is often in the groups of octopus and crab as shown in Fig. 8.

Light sensor dataset. We apply our methods to recover missing data in an environmental sensing dataset which includes sensor readings at a number of locations over a time period. We consider the dataset of a task as the sensor readings at these locations at a particular timestamp. Since a number of random readings may be missing due to faulty sensors, we want to recover the missing readings from the available readings at a timestamp. This is cast as a 10-shot regression problem: given a timestamp, recovering the sensor readings at all locations from the readings at only 10 random locations. We use the log of the light readings at 54 locations in the Intel Berkeley sensor lab data.[3] The dataset contains light readings of 23890 timestamps (23890 tasks), which is divided into 22695 meta training tasks and 1195 meta test tasks. Table 2 shows the MSE values of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ trained with BMAML, MAML (varying $m = 1, 5, 10$), and GPML (varying $n_u = 8, 20$). While increasing m from 5 to 10 does not improve the performance of MAML, the gap between GPML and MAML increases as n_u increases. The distribution of MSE between a random task and its closest inducing task is shown in Fig. 7b which indicates that a large number of tasks in the task distribution are close to an inducing task, especially when $n_u = 20$.

[3]The dataset is available at `http://db.csail.mit.edu/labdata/labdata.html`.



(a) Sushi with $n_u = 8$. (b) Light sensor.

Figure 7: Plots of the distribution of MSE between a random task and its closest inducing task.

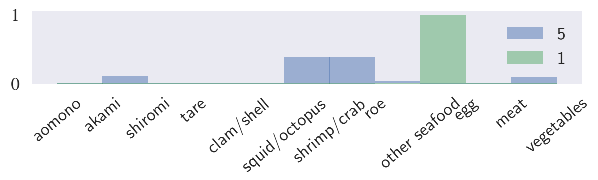


Figure 8: Plot of predicted ratings over different sushi groups from an inducing task.

5 RELATED WORKS

There have been serveral works that tackle the limitation of a common initializer for all tasks in MAML which is exacerbated in problems with multimodal task distributions, i.e., tasks are diverse. It can be addressed in two ways: (a) improving the prediction of the task parameters given the common initializer, and (b) making the initializer task-dependent, i.e., different tasks have different initializers.

Regarding the first direction, an initial work is the *light-weight Laplace approximation for meta-adaptation* (LLAMA) [Grant et al., 2018]. By casting MAML as empirical Bayes in a hierarchical probabilistic model, LLAMA applies local Laplace approximation to replace the point estimate of the task parameters in MAML with a Gaussian distribution at the cost of approximating a high-dimensional covariance matrix. Another work is *Bayesian MAML* (BMAML) that uses *Stein variational GD* (SVGD) to obtain the posterior of the task parameters given the initializer [Yoon et al., 2018]. However, as admitted by its authors, BMAML cannot capture the correct posterior distribution of the task parameters due to the limited number of SVGD steps and initialzer samples (for fast adaptation to a new task). Instead, BMAML aims to optimize for a set of initializer samples that are useful for many tasks. Nonetheless, these initializer samples are assumed to be independent from the task in BMAML, which decreases its performance for problems with a multimodal task distribution. This is a common disadvantage shared by LLAMA as its Laplace approximation is applied to the task parameters obtained with early stopping for fast adaptation. Even if we can afford an expensive adaptation by performing many GD updates, the overfitting issue may arise due to the small training set. Regarding our GPML-Bayes, given a task, the task parameters are transformed from the initializer following the GP posterior distribution through GD updates. Hence, the expressive power of our methods mainly comes from the second direction as below.

Table 2: Mean and standard deviation (SD) of MSE of $f(\mathbf{x}, g_t^{(m)}(\boldsymbol{\theta}_t^\circ))$ in experiments with the light sensor dataset.

	Mean	SD
BMAML with 10 particles	4.746	1.945
MAML $k = 10$ $m = 1$	1.008	1.075
MAML $k = 10$ $m = 5$	0.575	0.581
MAML $k = 10$ $m = 10$	0.580	0.532
GPML $k = 10$ $n_u = 8$ $m = 1$	0.342	0.439
GPML $k = 10$ $n_u = 20$ $m = 1$	0.261	0.435

As the first direction is limited due to the need of fast adaptation, it is promising to consider the second direction. Probabilistic MAML [Finn et al., 2018] injects the dependency between the initializer and the training set (i.e., task) through an inference network. In particular, the inference is approximated by adding a Gaussian noise into the gradient-descent updated parameters. Nonetheless, as the number of the GD updates is small, the relationship between the initializer and the task is limited. *Latent embedding optimization* (LEO) [Rusu et al., 2019] explicitly learns a task-dependent initializer by encoding the training set (or its inputs) into a latent space where the gradient-based meta-learning is performed. The latent space is transformed into the parameter (or initializer) space via a decoder network, which effectively means that the initializer of a task depends on its training set. However, the objective function requires additional weighted regularization terms to ensure that the network is maximally expressive and the latent space is disentangled. On the other hand, GPML learns a task-dependent initializer by defining the covariance between initializers via the distance between their tasks. Unlike LEO, the training of GPML naturally encourages the diversity (disentangled representation) and interpretability of the inducing tasks without additional regularization terms.

Metric-based meta-learning. Although there has not been any work explicitly modelling the distance between tasks, there are several existing meta-learning methods that use the notion of the distance between the embeddings of images in classification tasks. These methods are under the *metric-based meta-learning* approach which aims to enforce similar images (or their embeddings) to belong to the same class. The most rudimentary approach is the siamese network [Bromley et al., 1994] which is used to rank the similarity between image pairs as a pre-training step; then, during testing, the pre-trained siamese network is used to classify images [Koch et al., 2015]. The separation between these two processes implies that the few-shot learning objective is not explicitly enforced. The prototypical network assumes that there exists a latent space (embedding space) such that images of each class form a cluster [Snell et al., 2017]. Hence, the approach assigns a test image to the class whose embedding cluster centroid is closest. The assumption is further relaxed such that a class can form multiple clusters in the embedding space [Allen et al., 2019]. There is an attempt to measure the distance between tasks implicitly by introducing the full context embedding which includes the whole training set (i.e., the support set) as inputs to the embedding function in the matching network approach [Vinyals et al., 2016]. In contrast to these methods, GPML explicitly models the distance between tasks to cast the few-shot regression problem as a GP.

6 CONCLUSION AND FUTURE WORK

This paper presents GPML viewing the few-shot regression as a GP. Unlike metric-based meta-learning that utilizes the distance between data inputs, GPML is capable of modelling the task distance via inducing tasks to construct a novel task kernel. The meta training naturally encourages both diversity and representativeness of these inducing tasks, which leads to both superior predictive performance in handling diverse tasks and useful interpretation of the task distribution. We empirically illustrate these advantages of GPML through both synthetic problems: learning a random cosine function and a mixture of cosine and linear functions; and new real-world applications with the sushi and sensor datasets. For future work, we would like to explore two main directions to extend GPML: (a) Bayesian modelling of the inducing tasks and (b) developing new task kernels for classification and reinforcement learning tasks.

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (Award No: AISG2-RP-2020-018). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

K. Allen, E. Shelhamer, H. Shin, and J. Tenenbaum. Infinite mixture prototypes for few-shot learning. In *Proc. ICML*, pages 232–241, 2019.

M. A. Alvarez and N. D. Lawrence. Computationally efficient convolved multiple output Gaussian processes. *JMLR*, 12:1459–1500, 2011.

E. V. Bonilla, K. Chai, and C. Williams. Multi-task Gaussian process prediction. In *Proc. NeurIPS*, volume 20, 2008.

J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a "Siamese" time delay neural network. In *Proc. NIPS*, pages 737–744, 1994.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135, 2017.

C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. In *Proc. NeurIPS*, pages 9516–9527, 2018.

E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. Recasting gradient-based meta-learning as hierarchical Bayes. In *Proc. ICLR*, 2018.

T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proc. ACM SIGKDD*, pages 583–588, 2003.

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proc. ICML Workshop on Deep Learning*, 2015.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *JMLR*, 6:1939–1959, 2005.

A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In *Proc. NeurIPS*, pages 113–124, 2019.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *Proc. ICLR*, 2019.

B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society: Series B (Methodological)*, 47(1):1–21, 1985.

A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In *Proc. NIPS*, pages 619–625, 2001.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Proc. NeurIPS*, pages 4077–4087, 2017.

S. Thrun and L. Pratt. *Learning to Learn*. Springer US, 1998.

J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Proc. NIPS*, pages 3630–3638, 2016.

J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. In *Proc. NeurIPS*, pages 7332–7342, 2018.