Deep Mean Functions for Meta-Learning in Gaussian Processes

Vincent Fortuin
Department of Computer Science
ETH Zürich
Zürich, Switzerland
fortuin@inf.ethz.ch

Gunnar Rätsch
Department of Computer Science
ETH Zürich
Zürich, Switzerland
raetsch@inf.ethz.ch

1 Introduction

Bayesian methods are well suited for learning tasks in the low-data limit, because they offer a principled way to include prior knowledge about the problem [22]. If the prior knowledge is suitable for the task, only very few observations may be needed to fit the posterior well. However, the acquisition and effective encoding of this prior knowledge has been a longstanding challenge in the field [7]. A powerful framework to acquire prior knowledge about a task is meta-learning [34]. Meta-learning refers to a method in which knowledge is gained by solving a set of specific tasks (the meta-tasks) and subsequently used to improve the model's performance on a different task (the target task) [34]. A more thorough treatment of meta-learning and related approaches is deferred to the appendix (Sec. A).

The largest benefits can be gained from meta-learning in a setting in which there is a large amount of data in the meta-tasks, but only little data in the target task. This leads to the hypothesis that the meta-model should be expressive enough to model large data sets, while the target model should handle small amounts of data well. In this work, we thus propose to use deep neural networks as meta-models and Gaussian processes (GPs) as target models. Moreover, we present the first evidence that learning a GP's mean function can outperform kernel learning on meta-learning tasks.

We make the following contributions:

- Formalize meta-learning in Gaussian processes via mean function learning.
- Present analytical and empirical evidence that mean function learning can be superior to kernel learning in this setting.
- Discuss the risk of overfitting and connections to model-agnostic meta-learning.

2 Meta-learning in Gaussian Processes

We are following the common setting from Baxter [3], where we have a so-called *environment* τ and our tasks are sampled *i.i.d.* from this environment. For each task, we sample a data distribution and a sample size from the environment $(\mathcal{P}_i, n_i) \sim \tau$ and then sample a data set $\mathcal{D}_i \sim \mathcal{P}_i^{n_i}$, where $\mathcal{P}_i^{n_i}$ is the distribution of data sets of size n_i under $\mathcal{P}_i$.

The set of meta-tasks $\mathcal{M}$ consists of m data sets $\mathcal{M} := \{\mathcal{D}_i\}_{i=1}^m$, with one data set for each meta-task. Each of these data sets contains observations $\mathcal{D}_i := \{\mathbf{x}_i, \mathbf{y}_i\}$, where $\mathbf{x}_i \in \mathbb{R}^{n_i \times d}$ and $\mathbf{y}_i \in \mathbb{R}^{n_i \times p}$ for tasks in which the respective functions to be learned are defined as $f_i : \mathbb{R}^d \to \mathbb{R}^p$. In this setting, all meta-tasks share the same input and output dimensionalities d and p, but they can have different numbers of observations n_i. Additionally to the meta-tasks, we have a target task with a data set $\tilde{\mathcal{D}} := \{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*\}$, where $\tilde{\mathbf{x}} \in \mathbb{R}^{\tilde{n} \times d}$ and $\tilde{\mathbf{y}} \in \mathbb{R}^{\tilde{n} \times p}$ are the training points and their respective values and $\tilde{\mathbf{x}}^* \in \mathbb{R}^{\tilde{n}^* \times d}$ and $\tilde{\mathbf{y}}^* \in \mathbb{R}^{\tilde{n}^* \times p}$ are the test points and their respective values. We assume

4th workshop on Bayesian Deep Learning (NeurIPS 2019), Vancouver, Canada.

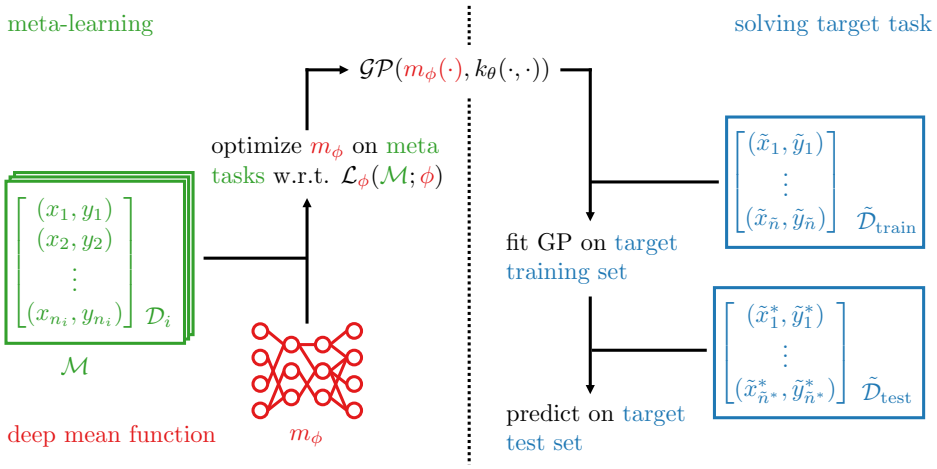


Figure 1: Overview of the proposed deep mean function meta-learning framework. Note that a similar procedure can be used to meta-learn the kernel function parameters θ, or both, mean and kernel function parameters.

this target task to also be sampled from the environment τ. As mentioned above, we assume there to be much more data in the meta-tasks than in the target task, that is, $n_{\text{train}} = \sum_{i=1}^{m} n_i \gg \tilde{n}$.

In order to predict on $\{\tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*\} =: \tilde{\mathcal{D}}_{\text{test}}$, we want to fit a GP to $\{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}\} =: \tilde{\mathcal{D}}_{\text{train}}$ with a prior $f(\cdot) \sim \mathcal{GP}(m_\phi(\cdot), k_\theta(\cdot,\cdot))$, where the mean and kernel functions are parameterized by sets of parameters ϕ and θ respectively. These parameters can now be optimized on the meta-task set, that is, $\phi^* = \arg\min_\phi \mathcal{L}_\phi(\mathcal{M};\phi)$ and $\theta^* = \arg\min_\theta \mathcal{L}_\theta(\mathcal{M};\theta)$ with a suitable loss function $\mathcal{L}_\psi$ for parameters $\psi \subset \{\phi, \theta\}$. This approach can be seen as gaining knowledge from solving the meta-tasks $\mathcal{M}$ and using the parameters ϕ and θ to encode the thus acquired meta-knowledge into the GP prior for the task on $\tilde{\mathcal{D}}$.

3 Learning deep mean functions for meta-learning GPs

The following section outlines our proposed model. Technical details are deferred to the appendix (Sec. B).

3.1 Mean functions can be superior to kernel functions

While learning kernel functions for GPs is a popular area of research, it entails a number of challenges. The most fundamental challenge is that the kernel function has to be positive definite in order to define an inner product in a suitable Hilbert space [23]. GP mean functions, in contrast, do not suffer from any such constraints and can therefore be learned more freely without taking any special precautions.

The question remains whether we can incorporate more prior knowledge into ϕ or into θ, that is, whether we should rather learn the mean or the kernel function. Given that many approaches in the literature resort to using a zero mean function and only adjust the kernel, they seem to be implicitly making the following assumption.

Assumption 1. *Given a Gaussian Process prior $\mathcal{GP}(m_\phi(\cdot), k_\theta(\cdot,\cdot))$, all the available prior knowledge can be effectively incorporated into the kernel parameters θ when a naïve mean function (i.e., $m_\phi(\cdot) = 0$) is used, such that complete prior knowledge yields an optimal posterior.*

It is easy to see that this assumption does not always hold. One can for instance construct a counter-example by having a true mean function that takes 0 values in certain parts of the domain and nonzero values in other parts. If this mean function is known *a priori*, it can be used as a mean function for the GP, leading to an optimal posterior fit. In contrast to that, there is no kernel that can guarantee an optimal fit if a zero mean function is used. The analytical details for this counter-example are deferred to the appendix (Sec. B).

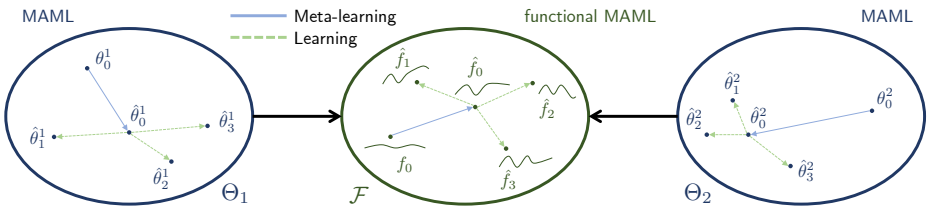


Figure 2: Sketch of the mechanism of MAML [9] in two different parameterizations of the learning model and of our mean function learning approach cast as *functional MAML*, acting directly in the function space $\mathcal{F}$, instead of the parameter spaces Θ_i.

Given these insights and the fact that learning mean functions poses a less constrained problem than kernel learning, we propose that mean functions can be a better and more natural choice than kernels for meta-learning in GPs under certain conditions. We will empirically validate this hypothesis in the experimental section (Sec. 4).

3.2 On the risk of overfitting in GP prior learning

When optimizing the hyperparameters of a GP model, the objective is often chosen to be the *log marginal likelihood* (LML), which can be seen as an approximation to Bayesian model selection (see also Sec. B). For the GP model, the LML can be computed in closed form [29] as

$$\begin{aligned}\log p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}},\psi) = & -\frac{1}{2}(\tilde{\mathbf{y}}-m_\phi(\tilde{\mathbf{x}}))^\top \left(K_\theta^{xx}+\sigma^2 I\right)^{-1}(\tilde{\mathbf{y}}-m_\phi(\tilde{\mathbf{x}})) \\ & -\frac{1}{2}\log|K_\theta^{xx}+\sigma^2 I| - \frac{n}{2}\log 2\pi \,,\end{aligned} \tag{1}$$

with hyperparameters $\psi := (\phi, \theta)$ and where $|M|$ denotes the determinant of matrix M.

It is evident that this LML naturally decomposes into three terms. The first one measures the goodness-of-fit, the second one the complexity and the third one is a normalization constant. Notice that the objective function contains an automatic tradeoff between data-fit and model complexity when it comes to the kernel parameters (second term), but it does not contain any complexity penalty on the mean function parameters. Optimizing the mean function parameters w.r.t. the LML directly on the training data can therefore lead to overfitting (as we will also see empirically in Sec. 4). This might explain why mean function learning has been largely unpopular in the GP community so far. To avoid this risk, we refrain entirely from optimizing any parameters using the LML on the target training data $\tilde{\mathcal{D}}_{\text{train}}$, and only optimize the parameters on the meta-tasks $\mathcal{M}$. Thus, there is no way for the training data of the target task to inform the GP prior and therefore no possibility of overfitting to the training data.

3.3 Connections to model-agnostic meta-learning

One popular framework for meta-learning, especially with neural networks, is model-agnostic meta-learning [9]. The general idea of this method is to find the best initial parameters $\hat{\theta}_0 \in \Theta$ for the target model, such that the required learning updates to get from $\hat{\theta}_0$ to the optimal parameters $\hat{\theta}_i$ for each meta-task $\mathcal{D}_i$ are minimized.

An issue with this framework is the fact that the parameterization of the models is arbitrary. One can for instance choose two different parameterizations of the model with parameter spaces Θ_1 and Θ_2, leading to different optimal parameters $\{\hat{\theta}_i^1 \in \Theta_1\}$ and $\{\hat{\theta}_i^2 \in \Theta_2\}$. A priori, it is not clear which one of these parameterizations (or of all the other possible parameterizations) will yield a better meta-learning performance.

In contrast to this, our proposed mean function learning framework acts directly in the function space $\mathcal{F}$ and tries to find the best mean function $\hat{f}_0$ that requires the least number of observations to fit the respective posterior means $\hat{f}_i$ of the meta-tasks well. It is thus independent of the parameterization of the functions and optimizes the prior directly with respect to the geometry of the associated function

Table 1: Performance comparison of the different meta-learning GP models and a neural network on MNIST image completion for different numbers of training points. To illustrate overfitting in the non-meta-learning case, we also learned a mean function on the target training set directly (*mean fit on target*).

	$\tilde{n} = 3$		$\tilde{n} = 50$		$\tilde{n} = 300$	
Method	likelihood	MSE	likelihood	MSE	likelihood	MSE
vanilla	-0.52 $\pm$ 0.00	0.112 $\pm$ 0.000	-0.50 $\pm$ 0.00	0.082 $\pm$ 0.000	-0.36 $\pm$ 0.00	0.022 $\pm$ 0.000
learned mean	-0.52 $\pm$ 0.00	**0.090 $\pm$ 0.000**	-0.49 $\pm$ 0.00	**0.068 $\pm$ 0.000**	-0.36 $\pm$ 0.00	**0.021 $\pm$ 0.000**
learned kernel	-0.09 $\pm$ 0.00	0.166 $\pm$ 0.001	-0.08 $\pm$ 0.00	0.075 $\pm$ 0.000	-0.07 $\pm$ 0.00	0.067 $\pm$ 0.000
learned both	**-0.01 $\pm$ 0.00**	0.151 $\pm$ 0.001	**0.00 $\pm$ 0.00**	0.073 $\pm$ 0.000	**0.02 $\pm$ 0.00**	0.059 $\pm$ 0.000
neural network	-	0.097 $\pm$ 0.000	-	0.097 $\pm$ 0.000	-	0.097 $\pm$ 0.000
mean fit on target	-0.75 $\pm$ 0.00	0.200 $\pm$ 0.006	-0.72 $\pm$ 0.00	0.082 $\pm$ 0.001	-0.67 $\pm$ 0.00	0.041 $\pm$ 0.000

Table 2: Performance comparison of the different meta-learning GP models on predicting patient trajectories on the intensive care unit for different clinical variables.

	Likelihood			MSE		
Data	learned mean	learned kernel	learned both	learned mean	learned kernel	learned both
GCS	-2.95 $\pm$ 0.02	-5.46 $\pm$ 0.03	**-2.44 $\pm$ 0.01**	13.98 $\pm$ 0.24	82.93 $\pm$ 0.74	**9.15 $\pm$ 0.17**
Urine	**-25.14 $\pm$ 1.54**	-40.20 $\pm$ 1.82	**-26.09 $\pm$ 1.61**	**20674.24 $\pm$ 1505.23**	35827.67 $\pm$ 1793.72	**21793.27 $\pm$ 1583.14**
HCT	**-2.93 $\pm$ 0.02**	-18.17 $\pm$ 0.09	**-2.92 $\pm$ 0.01**	**19.63 $\pm$ 0.50**	837.31 $\pm$ 4.62	**19.76 $\pm$ 0.48**
Creatinine	-1.75 $\pm$ 0.05	-1.61 $\pm$ 0.02	**-1.52 $\pm$ 0.01**	1.92 $\pm$ 0.16	1.58 $\pm$ 0.10	**0.91 $\pm$ 0.08**
BUN	-13.35 $\pm$ 0.58	-20.01 $\pm$ 0.62	**-8.88 $\pm$ 0.32**	**441.54 $\pm$ 23.51**	1058.40 $\pm$ 38.07	**412.11 $\pm$ 20.99**

space, which can be argued to be more efficient than any arbitrary parameter space [2]. Given this perspective, our approach can therefore be seen as a *functional MAML* (Fig. 2).

4 Experiments

In order to test the performance of mean function learning in general and compare it to kernel learning, we performed experiments on synthetic data, on MNIST handwritten digits [20], and on a challenging real-world medical data set from the 2012 Physionet Challenge [31]. We found strong qualitative and quantitative evidence that mean function learning (on its own and in combination with kernel learning) can outperform kernel learning alone in a wide range of meta-learning scenarios. The experimental details as well as the synthetic data experiments are deferred to the appendix (Sec. C).

MNIST image completion We view the task of image completion, that is, inferring the pixel values in test locations given their values in some random context locations, as a regression task (similar to Garnelo et al. [13]). We try to learn a function that maps from pixel coordinates to the pixel value, that is, $f : \{0, \dots, 27\}^2 \to [0, 1]$. It can be seen that the GP with the learned mean function significantly outperforms the other methods (including the neural network itself, without the GP) in terms of MSE. Moreover, learning both mean function and kernel yields the best performance in terms of likelihood. The risk of overfitting (Sec. 3.2) is illustrated by a GP model where we learned the mean function directly on the training set of the target task. As expected, we observe that this model underperforms compared to all the other models, highlighting the importance of the meta-learning setting for successful mean function learning.

Medical time series prediction These data correspond to the common meta-learning task of predicting the health trajectory of a patient who has recently been admitted to the hospital and for whom only few measurements are available yet. The meta-tasks in this scenario are all the other patients that have been treated previously. The results for some of the most frequently measured vital signs are reported in Table 2. Note that the different variables are measured in different units and have different characteristic scales. The models should therefore not be evaluated in terms of absolute performances, but rather be compared relative to each other. We observe that the GPs that include a learned mean function (*learned mean* and *learned both*) consistently outperform the GP for which only the kernel has been learned. This suggests that mean function learning can also lead to considerable benefits in challenging real-world tasks.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

[3] Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

[4] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36. jmlr.org, 2012.

[5] Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task gaussian process prediction. In J C Platt, D Koller, Y Singer, and S T Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 153–160. Curran Associates, Inc., 2008.

[6] Ronald Newbold Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[7] Bradley P Carlin and Thomas A Louis. *Bayesian methods for data analysis*. CRC Press, 2008.

[8] Alexander G de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A gaussian process library using TensorFlow. *J. Mach. Learn. Res.*, 18(40):1–6, 2017. ISSN 1532-4435.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for fast adaptation of deep networks. March 2017.

[10] Vincent Fortuin, Gideon Dresdner, Heiko Strathmann, and Gunnar Rätsch. Scalable gaussian processes on discrete domains. *arXiv preprint arXiv:1810.10368*, 2018.

[11] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, 2017.

[12] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox Matrix-Matrix gaussian process inference with GPU acceleration. September 2018.

[13] Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and S M Ali Eslami. Conditional neural processes. July 2018.

[14] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, S M Ali Eslami, and Yee Whye Teh. Neural processes. July 2018.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016. ISBN 9780521835688. doi: 10.1038/nmeth.3707.

[16] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting Gradient-Based Meta-Learning as hierarchical bayes. January 2018.

[17] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-Learning priors for efficient online bayesian regression. July 2018.

[18] Tomoharu Iwata and Zoubin Ghahramani. Improving output uncertainty estimation and generalization in deep learning via neural network gaussian processes. July 2017.

[19] Loic Le Gratiet. Bayesian analysis of hierarchical multifidelity codes. *SIAM/ASA Journal on Uncertainty Quantification*, 1(1):244–269, 2013.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[21] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Lecun Y., Bengio Y., and Hinton G. Deep learning. *Nature*, 521(7553):436–444, 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

[22] Daniel McNeish. On using bayesian methods to address small sample problems. *Structural Equation Modeling: A Multidisciplinary Journal*, 23(5):750–773, 2016.

[23] James Mercer. Functions of positive and negative type, and their connection the theory of integral equations. *Philos. Trans. R. Soc. Lond. A*, 209(441-458):415–446, January 1909. ISSN 0080-4614. doi: 10.1098/rsta.1909.0016.

[24] Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *J. Mach. Learn. Res.*, 7(Dec):2651–2667, 2006. ISSN 1532-4435, 1533-7928.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[26] Paris Perdikaris, Daniele Venturi, and George Em Karniadakis. Multifidelity information fusion algorithms for high-dimensional systems and massive data sets. *SIAM Journal on Scientific Computing*, 38(4):B521–B538, 2016.

[27] John C Platt, Christopher J C Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a gaussian process prior for automatically generating music playlists. In T G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1425–1432. MIT Press, 2002.

[28] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[29] Carl Edward Rasmussen and Christopher K I Williams. *Gaussian Processes for Machine Learning*. MIT Press, new edition, January 2006. ISBN 9780262182539.

[30] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Netw.*, 61:85–117, 2015. ISSN 0893-6080, 1879-2782. doi: 10.1016/j.neunet.2014.09.003.

[31] Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pages 245–248. IEEE, 2012.

[32] Grigorios Skolidis. *Transfer learning with Gaussian processes*. PhD thesis, June 2012.

[33] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

[34] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of Meta-Learning. *Artificial Intelligence Review*, 18(2):77–95, June 2002. ISSN 1573-7462. doi: 10.1023/A:1019956318069.

[35] Christian Widmer, Nora C Toussaint, Yasemin Altun, and Gunnar Rätsch. Inferring latent task structure for multitask learning by multiple kernel learning. *BMC bioinformatics*, 11(8):S5, 2010.

[36] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. November 2015.

[37] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Stochastic variational deep kernel learning. November 2016.

[38] K Yu, V Tresp, and A Schwaighofer. Learning gaussian processes from multiple tasks. *International Conference on Machine Learning*, 2005.

Appendix

A Related work

GP prior learning. Gaussian processes have gained a lot of attention in the field of machine learning in recent years [29], especially combined with approaches that improve their scalability [10, 28, 33]. Moreover, kernel learning for GPs has a long history [27]. It can be seen as an extension to Maximum-Likelihood-II (ML-II) parameter optimization, where the kernel parameters are optimized with respect to the *log marginal likelihood* of the GP [29]. The flexibility of these learned kernels grows with the number of parameters, culminating in deep kernel learning, where the kernel is parameterized by a deep neural network [36, 37]. While kernels have thus enjoyed a lot of attention, the research into learning mean functions is still in its infancy. Mean functions have been optimized over different levels of recursive autoregressive GPs in multifidelity modeling [19, 26], but these approaches have used very limited functional forms for the mean functions and have not considered true meta-learning, but only transfer learning across different recursion levels on the same task. The only documented deep mean function learning for GPs [18] deals with standard optimization of the mean function parameters on the training set and not with a meta-learning setting.

Meta-learning. Meta-learning has recently been explored in great detail [34], especially in the case of deep neural networks [4, 9]. Especially the work on MAML [9] bears certain similarities to our proposed approach (see Sec. 3.3). It has been shown that these approaches can be seen as inference in a hierarchical Bayesian model [16]. These explorations extend to GP-like neural network models [13, 14], but the setting has been underexplored when it comes to classical GPs. While there has been some work on meta-learning kernel functions [5, 32, 35], parameters for Bayesian linear regression [17], and parameters in hierarchical Bayesian models [38], the aforementioned GP mean function learning approach has not been studied in this setting yet.

B Model details

B.1 SGD for meta-learning the GP prior

In GP regression, the loss function is often chosen to be the *negative log marginal likelihood* (LML) [29]. The LML on a meta-task can be computed as

$$\begin{aligned}\mathcal{L}_\psi(\mathcal{D}_i;\psi) &= -\log p(\mathbf{y}_i|\mathbf{x}_i;\psi)\\ &= -\log \int p(\mathbf{y}_i|\mathbf{f}_i,\mathbf{x}_i)\, p(\mathbf{f}_i|\mathbf{x}_i;\psi)\, d\mathbf{f}_i \ ,\end{aligned} \tag{2}$$

where all learnable parameters of the GP prior (i.e., the mean parameters, kernel parameters, or both) are denoted as ψ and $\mathbf{f}_i$ is a vector of the function values of the latent function $f_i(\cdot)$ evaluated at the points $\mathbf{x}_i$.

Given this loss function over a single meta-task, we define the loss over all meta-tasks as a sum over their individual losses, that is,

$$\mathcal{L}_\psi\left(\mathcal{M};\psi\right) := \sum_{i=1}^{m} \mathcal{L}_\psi\left(\mathcal{D}_i;\psi\right) \ . \tag{3}$$

This loss can then be optimized using any general-purpose optimization method. In this work, we use Stochastic Gradient Descent (SGD) (see also Sec. B).

Once the parameters of the GP prior are optimized, we can use the prior to fit a GP to $\tilde{\mathcal{D}}_{\text{train}}$ and predict on $\tilde{\mathcal{D}}_{\text{test}}$. If we evaluate the predictive posterior of the GP on the test points, it yields [29]

$$p\left(\tilde{\mathbf{f}}^*|\tilde{\mathbf{y}},\tilde{\mathbf{x}},\tilde{\mathbf{x}}^*\right) = \mathcal{N}\left(\mathbf{m}^*,\mathbf{K}^*\right) \tag{4}$$

with

$$\mathbf{m}^* = m_\phi(\tilde{\mathbf{x}}^*) + K_\theta^{*x}\left(K_\theta^{xx}+\sigma^2 I\right)^{-1}\left(\tilde{\mathbf{y}} - m_\phi(\tilde{\mathbf{x}})\right)$$

$$\mathbf{K}^* = K_\theta^{**} - K_\theta^{*x}\left(K_\theta^{xx}+\sigma^2 I\right)^{-1} K_\theta^{x*}$$

where $K_\theta^{x*} = K_\theta^{*x\top}$ denotes the kernel matrix (also known as the Gram matrix) with $(K_\theta^{x*})_{ij} = k_\theta(\tilde{x}_i, \tilde{x}_j^*)$ and similarly for K_θ^{xx} and K_θ^{**}.

Algorithm 1 outlines the procedure to optimize the GP prior parameters ψ (which can stand for the mean function parameters ϕ, the kernel function parameters θ, or both; see Eqs. (**??**), (2)).

Algorithm 1 Algorithm to optimize the GP prior parameters ψ on the set of meta-tasks $\mathcal{M}$ using SGD.

Require: set of meta-tasks $\mathcal{M} = \{\mathcal{D}_i\}_{i=1}^m$, learning rate η
  while not converged **do**
    for all $\mathcal{D}_i \in \mathcal{M}$ **do**
      Compute $\mathcal{L}_\psi(\mathcal{D}_i; \psi) = -\log p(\mathbf{y}_i|\mathbf{x}_i; \psi)$
      Update $\psi \leftarrow \psi - \eta \nabla_\psi \mathcal{L}_\psi(\mathcal{D}_i; \psi)$
    end for
  end while

B.2 Counter-example against Assumption 1

Let us assume for simplicity that we generate data from a known noiseless process with nonzero mean, that is, $\exists x : m_{\text{true}}(x) \neq 0$, $\sigma^2 = 0$ and $\forall x : y(x) = m_{\text{true}}(x)$.

If we want to fit a GP to these data and want it to yield correct predictions, we need the posterior to satisfy

$$m_{true}(\tilde{\mathbf{x}}^*) = m_\phi(\tilde{\mathbf{x}}^*) + K_\theta^{*x}\, K_\theta^{xx^{-1}} \left(m_{true}(\tilde{\mathbf{x}}) - m_\phi(\tilde{\mathbf{x}})\right) , \tag{5}$$

where $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}^*$ are defined as above (Sec. 2).

Let us now assume that we only happen to observe training points where the process is zero, but that it is nonzero for some test points, that is, $\forall x \in \tilde{\mathbf{x}} : m_{true}(x) = 0$ and $\exists x \in \tilde{\mathbf{x}}^* : m_{true}(x) \neq 0$ (this assumption is more likely to hold for small $\tilde{n}$). If we try to encode all our prior knowledge into θ and choose an uninformative ϕ, that is, $m_\phi(\cdot) = 0$, the RHS of Equation 5 will always be zero, regardless of the choice of θ, while the LHS will sometimes be nonzero. However, if we encode our prior knowledge into ϕ, that is, choose $m_\phi(\cdot) = m_{true}(\cdot)$, Equation 5 will always hold, regardless of the choice of θ.

As mentioned above, the assumptions of this counter-example are more likely to hold for small values of $\tilde{n}$. In fact, for $\tilde{n} \to \infty$, one can show that under mild assumptions we can always find a kernel that will make the posterior approach $m_{true}(\cdot)$ arbitrarily closely [24]. However, in the case of small $\tilde{n}$, which we assume to be more common in the meta-learning setting (see Sec. 2), we believe Proposition 1 to be more relevant than the asymptotic kernel universality results.

B.3 Bayesian model selection

One of the most principled ways of choosing the hyperparameters ψ of the GP prior is Bayesian model selection [29]. Bayesian model selection chooses the hyperparameters according to their posterior probabilities given the data. These probabilities can be computed as

$$p(\psi|\mathbf{y}, \mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x}, \psi)\, p(\psi)}{p(\mathbf{y}|\mathbf{x})} , \tag{6}$$

where the denominator is given by

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x}, \psi)\, p(\psi)\, d\psi \, . \tag{7}$$

Instead of this maximum *a posteriori* (MAP) inference over ψ, practitioners often resort to a maximum likelihood estimate (MLE) for reasons of tractability, by optimizing the likelihood term in the numerator of (6) with respect to ψ. Note that the negative logarithm of this term is exactly the LML from (2). Since this approach uses an MLE method to optimize the *hyperparameters* instead of the *parameters* of the model, it is sometimes called *type II* maximum likelihood (ML-II) approximation [29].

B.4 Deep mean functions for GPs

Given the insight that meta-learning the GP's mean function might help us in solving our target task, we are still faced with the problem of choosing a suitable parameterization, that is, a suitable function class for this mean function. Since we expect to have a reasonably large amount of data from the meta-tasks ($n_{\text{train}} \gg \tilde{n}$, see Sec. 2), we want to choose a function class that can scale easily to such amounts of data and that is flexible enough to incorporate the meta-knowledge. A class of parametric functions that exhibit these two properties are deep neural networks [15, 21, 30].

As an illustrative example, let us assume that we want to parameterize the mean function as a feed-forward neural network with two hidden layers. The mean value at a point $\hat{x}$ will then be given by

$$m_\phi(\hat{x}) = W_3\sigma(W_2\sigma(W_1\hat{x} + b_1) + b_2) + b_3 \ , \tag{8}$$

where the W_i's are the weight matrices of the layers, the b_i's are their biases, and $\sigma(\cdot)$ is a nonlinear activation function (e.g., sigmoid or ReLU). For our notation, we would then consume all the trainable parameters into the vector ϕ, that is, $\phi = (W_i, b_i)_{i=1}^3$.

Given this functional form, we can train the mean function according to Section 2. We compute the gradients of the LML with respect to ϕ using backpropagation, as implemented in various deep learning frameworks (in our experiments, we use `TensorFlow` [1] and `PyTorch` [25]).

It has been shown that using deep neural networks as kernel functions for GPs yields neural networks with nonparametric (or equivalently, "infinitely wide") last layers [36]. Similarly, using deep mean functions amounts to fitting a neural network to the data and then modeling the residuals with a GP [18]. It thus offers a natural way to combine the predictive power of neural networks with the calibrated uncertainties of GPs and can therefore be seen as an avenue of *Bayesian Deep Learning* [11].

C Experimental details

This section contains additional details regarding the experiments (Sec. 4). We implemented the algorithms in Python, using the `GPflow` package [8] and the `GPyTorch` [12] package.

C.1 Performance measures

In our experiments, we assess the performance of the models on the target task with two different measures: the test mean squared error (MSE) and the test data log likelihood (often just denoted as *likelihood*). Note that the likelihood depends on the whole predictive posterior, while the MSE only depends on its mean. Since the mean function of the GP only affects the posterior by shifting its mean, it can be hypothesized that learning a good mean function for the GP will affect the MSE more strongly than the likelihood. Similarly, since the kernel function parameterizes the covariance of the process, it could be expected to affect the likelihood more strongly than the MSE. When interpreting the results of our experiments, one should therefore keep in mind that the MSE slightly favors GPs with a good mean function, while the likelihood slightly favors good kernel functions.

However, the decision which one of the two measures is more important depends on the intended use of the GP in the target task. If the GP is used to predict values at test points from its predictive posterior mean, the MSE is the more relevant measure. If it is instead used to draw multiple samples from the whole posterior or to estimate the probability of different outcomes, the likelihood is more relevant. We do not make any limiting assumptions on the use cases in this work and the ultimate decision for a measure (and therefore potentially a preferred model) is hence left to the practitioner.

C.2 Sinusoid function regression

As a proof of concept, we aim to assess the general performance of mean function meta-learning. To this end, we simulated functions from a known generating Gaussian process. The process had a sinusoid mean and a radial basis function (RBF) covariance. For each function, we sampled the value at 50 equally spaced points in the $[-5, 5]$ interval. Samples from the process are depicted in Figure S1.

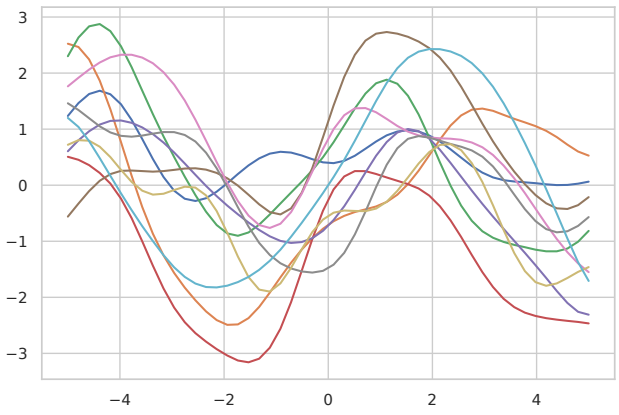


Figure S1: Samples from the generative sinusoid process for the synthetic data experiment.

Table S1: Performance comparison of GP regression with a zero mean function, a learned mean function and the true mean function of the generating process on synthetic sinusoid function data for different numbers of training points. The performance is measured in terms of likelihood and mean squared error. The reported values are means and their respective standard errors of 200 runs. The learned mean function performs comparably with the true mean function.

	$\tilde{n} = 1$		$\tilde{n} = 5$		$\tilde{n} = 20$	
Method	likelihood	MSE	likelihood	MSE	likelihood	MSE
zero mean	-1.31 $\pm$ 0.07	1.21 $\pm$ 0.04	-5.68 $\pm$ 0.13	1.02 $\pm$ 0.15	19.99 $\pm$ 0.22	**0.00** $\pm$ **0.00**
true mean	**-0.74** $\pm$ **0.08**	**0.79** $\pm$ **0.03**	**-4.32** $\pm$ **0.14**	**0.53** $\pm$ **0.04**	**21.55** $\pm$ **0.27**	**0.00** $\pm$ **0.00**
learned mean	**-0.75** $\pm$ **0.08**	**0.80** $\pm$ **0.03**	**-4.43** $\pm$ **0.13**	**0.52** $\pm$ **0.03**	20.76 $\pm$ 0.22	**0.00** $\pm$ **0.00**

We trained a deep feedforward neural network on 1000 sampled functions and used it as a mean function for GP regression on 200 unseen functions that were sampled from the same process. We used a neural network with two hidden layers of size 64 each with sigmoid activation functions. It was trained for 100 epochs using stochastic gradient descent (SGD). We compared it against a GP with zero mean function and one with the true sinusoid mean function for different numbers of training points (Tab. S1).

It can be seen that the learned mean function performs comparably with the true mean function and significantly better than the zero mean function in terms of likelihood and MSE. It also becomes evident that this effect diminishes when the number of training points increases, following the intuition that the prior becomes less important once there are enough observations (see Sec. 3.1). These findings support the efficacy of the mean function learning approach in the low-data limit.

C.3 Step function regression

To assess the performance of the mean function learning approach on a traditionally more challenging problem for GPs and compare it to kernel learning, we chose the task of step function regression. Step functions are hard to fit for GPs due to their discontinuity [29]. We hypothesize that this discontinuity can be modeled more easily by a mean function than by a kernel function, since the kernel function always interpolates between neighboring points to some extent and therefore implicitly assumes continuity.

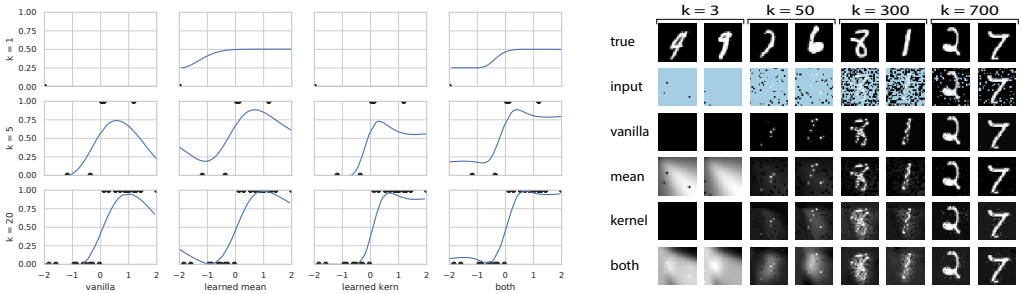
A step function is simply defined as

$$f_{\text{step}}(x) = \begin{cases} y_1 & \text{if} \quad x < x_{\text{step}} \\ y_2 & \text{otherwise} \end{cases} \tag{9}$$

with the step location x_{step} and the two function values (y_1, y_2) before and after the step, respectively. For our set of tasks, we choose a dataset of different step functions, namely the Heaviside step function

Table S2: Performance comparison of GPs with a zero mean function and RBF kernel (*vanilla*), a learned mean function and RBF kernel (*learned mean*), a zero mean and learned kernel function (*learned kernel*) and a learned mean and learned kernel function (*learned both*) on step function regression for different numbers of training points. The performance is measured in terms of likelihood and mean squared error. The values are means and their standard errors of 1000 randomly sampled step functions and sets of training points. Learning both the mean function and the kernel function consistently outperforms the other methods.

	$\tilde{n} = 1$		$\tilde{n} = 5$		$\tilde{n} = 20$	
Method	likelihood	MSE	likelihood	MSE	likelihood	MSE
vanilla	0.28 ± 0.00	0.335 ± 0.006	0.35 ± 0.00	0.113 ± 0.003	0.47 ± 0.01	0.026 ± 0.001
learned mean	0.29 ± 0.00	0.196 ± 0.002	0.36 ± 0.00	0.095 ± 0.002	0.46 ± 0.01	0.027 ± 0.001
learned kernel	0.54 ± 0.00	0.323 ± 0.007	$\mathbf{0.61 \pm 0.00}$	0.093 ± 0.004	0.68 ± 0.01	$\mathbf{0.024 \pm 0.001}$
learned both	$\mathbf{0.55 \pm 0.00}$	$\mathbf{0.186 \pm 0.002}$	$\mathbf{0.61 \pm 0.00}$	$\mathbf{0.078 \pm 0.002}$	$\mathbf{0.69 \pm 0.01}$	$\mathbf{0.022 \pm 0.001}$



(a) Step function reconstructions.

(b) MNIST reconstructions

Figure S2: Reconstructions of Heaviside step functions (a) and of MNIST test digits (b) using the different methods and different numbers of training points k.

[6] (i.e., $(y_1, y_2) = (0, 1)$) and its mirrored version along the x_{step}-axis (i.e., $(y_1, y_2) = (1, 0)$). We sample the step location x_{step} uniformly at random from the $[-1, 1]$ interval. The whole function consists of 50 evenly spaced points in the $[-2, 2]$ domain.

We compare a vanilla GP (zero mean and RBF kernel) with a learned deep mean function, a learned deep kernel function (following Wilson et al. [36]) and a GP with deep mean and deep kernel function learned concurrently. The learned mean and kernel functions are parameterized by the same deep feed-forward neural network architecture (except for the dimension of the last layer). We use neural networks with two hidden layers of size 128 and 64 with sigmoid activation functions. We train each model on 10,000 randomly sampled step functions from our function space using stochastic gradient descent.

The respective performances of the methods in terms of likelihood and MSE are reported in Table S2 and some example regression outputs on the standard Heaviside step function for the different models and numbers of training points are depicted in Figure S2a. It can be seen that in the low-data regime, mean function learning outperforms kernel learning in terms of MSE, while kernel learning yields a better likelihood. However, the gap between mean function learning and kernel learning in terms of MSE narrows with an increasing number of training points, following our intuition that the prior becomes less important with increasing amounts of data.

It can also be seen that learning both the mean and the kernel function consistently performs best in all data regimes. Especially in the low-data regime, we can see that this approach combines the high likelihood of the kernel learning with the low MSE of the mean function learning. It stands to question, however, whether this effect is due to the relative simplicity of the task or whether it also holds for more complex data.

C.4 MNIST image completion

The learned mean and kernel functions are parameterized by the same deep feed-forward neural network architecture (except for the dimension of the last layer). We use neural networks with

two hidden layers of size 128 and 64 with sigmoid activation functions. We train each model on the MNIST training set of images using stochastic gradient descent. We again compare the two approaches against a "vanilla" GP (zero mean and RBF kernel) and a GP where both functions are learned concurrently. We report performances on the MNIST test set with different numbers of training points (Tab. 1).

C.5 Medical time series prediction

In this data set, there are 4,000 patients in the training set and 4,000 in the test set. The patients are observed for 48 hours on the intensive care unit (ICU), where different vital signs are measured at different frequencies. During meta-training, we learn the mean function and kernel function of the GP on the whole time series of 48 hours. We use the same neural network architectures as in the MNIST experiment. We learn a different GP prior for each of the different vital signs. During testing, we condition a GP with the meta-learned prior on the first 24 hours and try to predict the remaining 24 hours.