Uncertainty-Aware Meta-Learning for Multimodal Task Distributions

Cesar Almecija[1], **Apoorva Sharma**[2*], **Young-Jin Park**[1] **& Navid Azizan**[1]
[1] Massachusetts Institute of Technology, [2] NVIDIA Research
{almecija,azizan,youngp}@mit.edu, apoorvas@nvidia.com

Abstract

Meta-learning is a popular approach for learning new tasks with limited data (i.e., *few-shot learning*) by leveraging the commonalities among different tasks. However, meta-learned models can perform poorly when context data is limited, or when data is drawn from an out-of-distribution (OoD) task. Especially in safety-critical settings, this necessitates an uncertainty-aware approach to meta-learning. In this work, we present UNLIMITD (*uncertainty-aware meta-learning for multimodal task distributions*), a novel method for meta-learning that (1) makes probabilistic predictions on in-distribution tasks efficiently, (2) is capable of detecting OoD context data at test time, and (3) performs on heterogeneous, multimodal task distributions. To achieve this goal, we take a probabilistic perspective and train a parametric, tuneable distribution over tasks on the meta-dataset. We construct this distribution by performing Bayesian inference on a linearized neural network, leveraging Gaussian process theory. We demonstrate that UNLIMITD's predictions compare favorably to, and outperform in most cases, the standard baselines, especially in the low-data regime. Furthermore, we show that UNLIMITD is effective in detecting data from OoD tasks. Finally, we confirm that both of these findings continue to hold in the multimodal task-distribution setting. [2]

1 Introduction

Meta-learning has emerged as a popular approach to enable models to perform well on new tasks using limited data. It involves first a *meta-training* process, when the model learns valuable features from a set of tasks. Then, at test time, using only few *context data* from a new, unseen task, the model (1) *adapts* to this new task, and then (2) *infers* by making predictions on new, unseen *query inputs* from the same task. A popular baseline for meta-learning, which has attracted a large amount of attention, is Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017], in which the adaptation process consists of fine-tuning the parameters of the model via gradient descent.

However, meta-learning methods can often struggle in several ways. First, accurate prediction becomes challenging when context data is limited. As these predictions can be untrustworthy, this necessitates the development of meta-learning methods that can express uncertainty during adaptation [Yoon et al., 2018, Harrison et al., 2018]. In addition, meta-learning models may not successfully adapt to "unusual" tasks, i.e., when test-time context data is drawn from an *out-of-distribution* (OoD) task not well represented in the training dataset [Jeong and Kim, 2020, Iwata and Kumagai, 2022]. Finally, special care has to be taken when learning tasks that have a large degree of heterogeneity. An important example is the case of tasks with a *multimodal* distribution, i.e., when there are no common features shared across all the tasks, but the tasks can be broken down into subsets (modes) in a way that the ones from the same subset share common features [Vuorio et al., 2019].

[*] This work was completed prior to Apoorva starting at NVIDIA Research.
[2] An implementation is available at https://github.com/azizanlab/unlimtd

6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans.

Our contributions. This paper presents UNLIMITD, a novel meta-learning method that leverages probabilistic tools to address the aforementioned issues. Specifically, UNLIMITD models the true distribution of tasks with a learnable distribution constructed over a linearized neural network and uses analytic Bayesian inference to perform uncertainty-aware adaption. We further extend the framework to the multimodal task-distribution setting.

2 Problem statement

A task $\mathcal{T}^i$ consists of a function f_i from which data is drawn. At test time, the prediction steps are broken down into (1) *adaptation*, that is identifying f_i using K context datapoints $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ from the task, and (2) *inference*, that is making predictions for f_i on the *query inputs* $\boldsymbol{X}^i_*$. We take a probabilistic, functional perspective and represent a cluster by $p(f)$, a theoretical distribution over the function space that describes the probability of a task belonging to the cluster.

In practice, however, we are not given $p(f)$, but only a meta-training dataset $\mathcal{D}$ that we assume is sampled from $p(f)$: $\mathcal{D} = \{(\widetilde{\boldsymbol{X}}^i, \widetilde{\boldsymbol{Y}}^i)\}_{i=1}^N$, where N is the number of training tasks, and $(\widetilde{\boldsymbol{X}}^i, \widetilde{\boldsymbol{Y}}^i) \sim \mathcal{T}^i$ is the entire pool of data from which we can draw subsets of context data. Consequently, in the meta-training phase, we aim to optimize $\tilde{p}_\xi(f)$ to capture properties of $p(f)$, using $\mathcal{D}$.

3 Our Approach: UNLIMITD

In our approach, we choose $\tilde{p}_\xi(f)$ to be the GP distribution over functions that arises from a Gaussian prior on the weights of the linearization of a neural network. Recent works [Jacot et al., 2018, Neal, 1996, Azizan et al., 2021, Lee et al., 2017] have demonstrated that the linearized deep neural network, wherein the Jacobian of the network operates as the feature map, can well approximate the training behavior of wide nonlinear deep neural networks. Appendix A.1 contains detail about the GP and linearized neural networks, on which the proposed framework is based.

Consider a particular task $\mathcal{T}^i$ and a batch of K context data $(\boldsymbol{X}^i \in \mathbb{R}^{N_x \times K}, \boldsymbol{Y}^i \in \mathbb{R}^{N_y \times K})$. The resulting GP prior predictive distribution after evaluating on the context inputs, is $\boldsymbol{Y}|\boldsymbol{X}^i \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i}, \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i})$, where

$$\boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\mu}, \quad \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\Sigma}\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)^\top + \boldsymbol{\Sigma}_\varepsilon \tag{1}$$

where $\boldsymbol{J}(\cdot,\cdot) : \mathbb{R}^P \times \mathbb{R}^{N_x \times K} \to \mathbb{R}^{N_y K \times P}$ is the vectorized Jacobian of the network with respect to the parameters on the given K inputs.

In this setup, the parameters ξ of $\tilde{p}_\xi(f)$ that we wish to optimize are the linearization point $\boldsymbol{\theta}_0 \in \mathbb{R}^P$, and the parameters of the prior over the weights $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Given this Gaussian prior, it is straightforward to compute the joint NLL of the context labels $\boldsymbol{Y}^i$,

$$\mathrm{NLL}(\boldsymbol{X}^i, \boldsymbol{Y}^i) = \frac{1}{2}\left(\left\| \boldsymbol{Y}^i - \boldsymbol{\mu}_{\boldsymbol{Y}|\boldsymbol{X}^i} \right\|^2_{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{Y}|\boldsymbol{X}^i}} + \log\det \boldsymbol{\Sigma}_{\boldsymbol{Y}|\boldsymbol{X}^i} + N_y K \log 2\pi \right). \tag{2}$$

The NLL (a) serves as a loss function quantifying the quality of ξ during training and (b) serves as an uncertainty signal at test time to evaluate whether context data $(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ is OoD. Given this model, *adaptation* is tractable as we can condition this GP on the context data analytically.

3.1 Tractably parameterizing the prior covariance over the weights

When working with deep neural networks, the number of weights P can surpass 10^6. The memory complexity of UNLIMITD quickly increases with P due to the dense prior covariance matrix over the weights $\boldsymbol{\Sigma} \in \mathbb{R}^{P \times P}$.

Learning a low-dimensional representation of the covariance. To relieve the burden, this paper proposes a low-rank representation of $\boldsymbol{\Sigma}$, allowing for a learnable weight-space prior covariance that can encode correlations. Specifically, we consider a covariance of the form $\boldsymbol{\Sigma} = \boldsymbol{Q}^\top \mathrm{diag}(\boldsymbol{s}^2)\boldsymbol{Q}$, where $\boldsymbol{Q}$ is a fixed projection matrix on an s-dimensional subspace of $\mathbb{R}^P$. In this case, the parameters that are learned are $\xi = (\boldsymbol{\theta}_0, \boldsymbol{\mu}, \boldsymbol{s})$. We define $\boldsymbol{S} := \mathrm{diag}(\boldsymbol{s}^2)$. The computation of the covariance of the prior predictive (equation 1) could then be broken down into two steps:

$$\begin{cases} A := \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{Q}^\top \\ \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)\boldsymbol{\Sigma}\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)^\top = A\boldsymbol{S}A^\top \end{cases}$$

Algorithm 1 UNLIMITD-F: meta-training with a learnt low-dimensional covariance

Find intermediate $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with UNLIMITD-I ▷ see Alg. 3 in Appendix
Find $\boldsymbol{Q}$ via FIMPROJ(s); initialize $\boldsymbol{s}$. ▷ see Alg. 2 in Appendix
for all epoch **do**:
  Sample n tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
  for all $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**:
    $NLL_i \leftarrow \text{GAUSSNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu},\ \boldsymbol{J}\boldsymbol{Q}^\top \text{diag}(\boldsymbol{s}^2)\boldsymbol{Q}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon)$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
  Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu} \cup \boldsymbol{s}} \sum_i NLL_i$

which requires a memory footprint of $O(P(s + N_y K))$, if we include the storage of the Jacobian. Because $N_y K \ll P$ in typical deep learning contexts, it suffices that $s \ll P$ so that it becomes tractable to deal with this new representation of the covariance.

One way to choose the informative projection matrix $\boldsymbol{Q}$ is constructing the projection map by choosing the top s eigenvectors of the Fisher information matrix (FIM) evaluated on the training dataset $\mathcal{D}$. Recent work has shown that the FIM for deep neural networks tends to have rapid spectral decay [Sharma et al., 2021], which suggests that keeping only a few of the top eigenvectors of the FIM is enough to encode an expressive task-tailored prior. We named this approach UNLIMITD-F. The detail about UNLIMITD-F and other parameterizing methods (UNLIMITD-I and UNLIMITD-R) are described in Appendix A.2 and Appendix A.3, respectively.

3.2 Generalizing the structure to a mixture of Gaussians

When learning on multiple clusters of tasks, $p(f)$ can become non-unimodal. Instead of using single GP, we can capture this multimodality by structuring $\tilde{p}_\xi(f)$ as a *mixture* of GPs.

Building a more general structure. We assume that at train time, a task $\mathcal{T}^i$ comes from any cluster with equal probability. Thus, we construct $\tilde{p}_\xi(f)$ as an equal-weighted mixture of α GPs.

For each element of the mixture, the structure is similar to the single cluster case, where the parameters of the cluster's weight-space prior are given by $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. We choose to have both the projection matrix $\boldsymbol{Q}$ and the linearization point $\boldsymbol{\theta}_0$ (and hence, the feature map $\phi(\cdot) = \boldsymbol{J}(\boldsymbol{\theta}_0, \cdot)$) shared across the clusters. This yields improved computational efficiency, as we can compute the projected features once, simultaneously, for all clusters. This yields the parameters $\xi_\alpha = (\boldsymbol{\theta}_0, \boldsymbol{Q}, (\boldsymbol{\mu}_1, \boldsymbol{s}_1), \ldots, (\boldsymbol{\mu}_\alpha, \boldsymbol{s}_\alpha))$.

This can be viewed as a mixture of linear regression models, with a common feature map but separate, independent prior distributions over the weights for each cluster. These separate distributions are encoded using the low-dimensional representations $\boldsymbol{S}_j$ for each $\boldsymbol{\Sigma}_j$. Notice how this is a generalization of the single cluster case, for when $\alpha = 1$, $\tilde{p}_\xi(f)$ becomes a Gaussian and $\xi_\alpha = \xi$.

Prediction and likelihood computation. The NLL of a batch of inputs under this mixture model can be computed as:

$$\text{NLL}_{\text{mixt}}(\boldsymbol{X}^i, \boldsymbol{Y}^i) = \log \alpha - \log \text{sum} \exp(-\text{NLL}_1(\boldsymbol{X}^i, \boldsymbol{Y}^i), \ldots, -\text{NLL}_\alpha(\boldsymbol{X}^i, \boldsymbol{Y}^i)), \quad (3)$$

where $\text{NLL}_j(\boldsymbol{X}^i, \boldsymbol{Y}^i)$ is the NLL with respect to each individual Gaussian, as computed in equation 2.

To make exact predictions, we would require conditioning this mixture model. As this is not directly tractable, we propose to first *infer the cluster* from which a task comes from, by identifying the Gaussian $\mathcal{G}_{j_0}$ that yields the highest likelihood for the context data $\left(\boldsymbol{X}^i, \boldsymbol{Y}^i\right)$. Then, we can *adapt* by conditioning $\mathcal{G}_{j_0}$ with the context data and finally *infer* by evaluating the posterior on the query $\boldsymbol{X}^i_*$.

3.3 Meta-training the Parametric Task Distribution

The key to our meta-learning approach is to estimate the quality of $\tilde{p}_\xi(f)$ via the NLL of context data from training tasks, and use its gradients to update the parameters of the distribution ξ. Optimizing this loss over tasks in the dataset draws $\tilde{p}_\xi(f)$ closer to the empirical distribution present in the dataset, and hence towards the true distribution $p(f)$. The training details can be found in Appendix A.4

Computing the likelihood. In the algorithms, the function GAUSSNLL($\boldsymbol{Y}^i$; m, K) stands for NLL of $\boldsymbol{Y}^i$ under the Gaussian $\mathcal{N}(m, K)$ (see equation 2). In the mixture case, we instead use MIXTNLL,

which wraps equation 3 and calls GAUSSNLL for the individual NLL computations (see discussion in Section 3.2). In this case, $\boldsymbol{\mu}$ becomes $\{\boldsymbol{\mu}_j\}_{j=1}^{j=\alpha}$ and $\boldsymbol{s}$ becomes $\{\boldsymbol{s}_j\}_{j=1}^{j=\alpha}$ when applicable.

Finding the FIM-based projections. The FIM-based projection matrix aims to identify the elements of $\phi = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})$ that are most relevant for the problem (see Section 3.1 and Appendix A.3). However, this feature map evolves during training, because it is $\boldsymbol{\theta}_0$-dependent. How do we ensure that the directions we choose for $\boldsymbol{Q}$ remain relevant during training? We leverage results from Fort et al. [2020], stating that the NTK (the kernel associated with the Jacobian feature map, see Section A.1.2) changes significantly at the beginning of training and that its evolution slows down as training goes on. This suggests that as a heuristic, we can compute the FIM-based directions after partial training, as they are unlikely to deviate much after the initial training. For this reason, UNLIMITD-F (Algorithm 1) first calls UNLIMITD-I (Algorithm 3) before computing the FIM-based $\boldsymbol{Q}$. Then the usual training takes place with the learning of $\boldsymbol{s}$ in addition to $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$.

4 Related work

Bayesian inference with linearized DNNs. Bayesian inference with neural networks is often intractable because the posterior predictive has rarely a closed-form expression. Whereas UNLIMITD linearizes the network to allow for practical Bayesian inference, existing work has used other approximations to tractably express the posterior. For example, it has been shown that in the infinite-width approximation, the posterior predictive of a Bayesian neural network behaves like a GP [Neal, 1996, Lee et al., 2017]. This analysis can in some cases yield a good approximation to the Bayesian posterior of a DNN [Garriga-Alonso et al., 2018]. It is also common to use Laplace's method to approximate the posterior predictive by a Gaussian distribution and allow practical use of the Bayesian framework for neural networks. This approximation relies in particular on the computation of the Hessian of the network: this is in general intractable, and most approaches use the so-called Gauss-Newton approximation of the Hessian instead [Ritter et al., 2018]. Recently, it has been shown that the Laplace method using the Gauss-Newton approximation is equivalent to working with a certain linearized version of the network and its resulting posterior GP [Immer et al., 2021].

Meta-learning. MAML is a meta-learning algorithm that uses as adaptation a few steps of gradient descent [Finn et al., 2017]. It has the benefit of being model-agnostic (it can be used on any model for which we can compute gradients w.r.t. the weights), whereas UNLIMITD requires the model to be a differentiable regressor. MAML has been further generalized to probabilistic meta-learning models such as PLATIPUS or BaMAML [Yoon et al., 2018, Finn et al., 2018], where the simple gradient descent step is augmented to perform approximate Bayesian inference. These approaches, like ours, learn (during meta-training) and make use of (at test-time) a prior distribution on the weights. In contrast, however, UNLIMITD uses exact Bayesian inference at test-time. MAML has also been improved for multimodal meta-learning via MMAML [Vuorio et al., 2019, Abdollahzadeh et al., 2021]. Similarly to our method, they add a step to identify the cluster from which the task comes from [Vuorio et al., 2019]. OoD detection in meta-learning has been studied by Jeong and Kim [2020], who build upon MAML to perform OoD detection in the classification setting, to identify unseen classes during training. Iwata and Kumagai [2022] also implemented OoD detection for classification, by learning a Gaussian mixture model on a latent space. UNLIMITD extends these ideas to the regression task, aiming to identify when test data is drawn from an unfamiliar function.

ALPaCA is a Bayesian meta-learning algorithm for neural networks, where only the last layer is Bayesian [Harrison et al., 2018]. Such framework yields an exact linear regression that uses as feature map the activations right before the last layer. Our work is a generalization of ALPaCA, in the sense that UNLIMITD restricted to the last layer matches ALPaCA's approach. More on this link between the methods is discussed in Appendix A.5.

5 Results and Discussion

We wish to evaluate following key aspects of UNLIMITD. (1) At test time, how do the probabilistic predictions compare to baselines? (2) How well does the detection of context data from OoD tasks perform? We construct a multimodal task distribution, regression problems inspired from Vuorio et al. [2019], with training data consisting of sinusoids as well as lines with varying slopes. OoD tasks are quadratic functions. We evaluate the performance of the proposed approach and compared

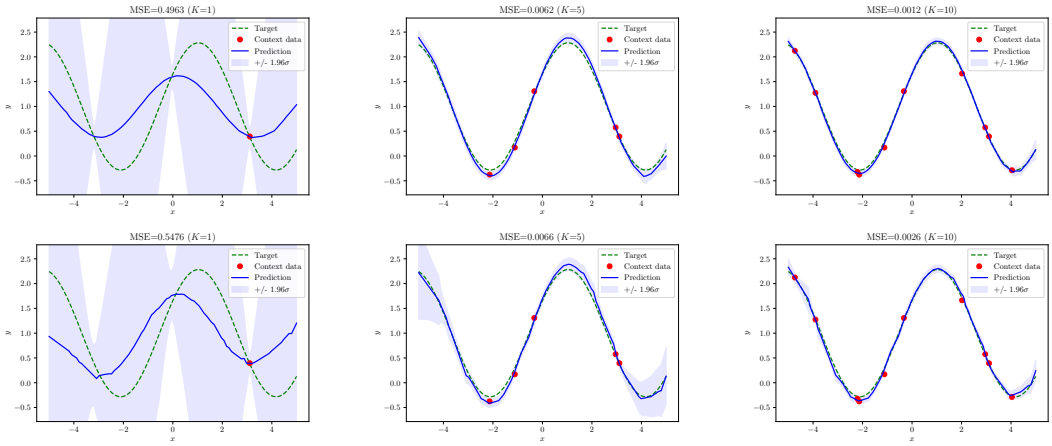
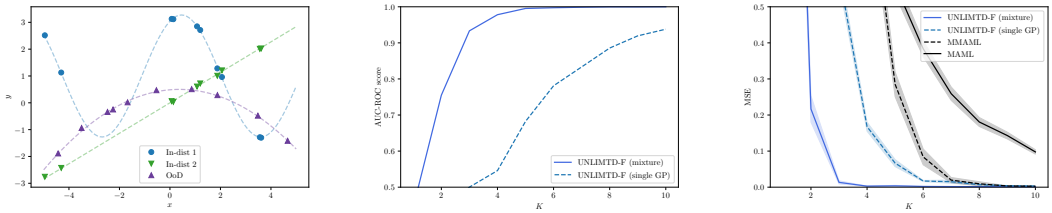


Figure 1: Example of predictions for a varying number of context inputs K, after meta-training with UNLIMITD-F. Top: UNLIMITD-F, infinite task dataset. Bottom: UNLIMITD-F, finite task dataset. The standard deviation is from the posterior predictive distribution. Note how the uncertainty levels are coherent with the actual prediction error. Also, note how uncertainty decreases when there is more context data. Notice how UNLIMITD-F recovers the shape of the sine even with a low number of context inputs. Finally, note how UNLIMITD-F is able to reconstruct the sine even when trained on fewer tasks (bottom). More comprehensive plots available in Figure 5.



(a) Examples of context data (b) AUC for OoD detection (c) MSE on predictions

Figure 2: Performance of UNLIMITD for OoD detection and inference, as a function of the number of context datapoints K. The training dataset includes both sines and lines, while OoD tasks are quadratic functions. We compare the different variants (UNLIMITD-F with a single GP or a mixture model), and against MAML/MMAML for predictions. Note how both versions of UNLIMITD-F yield better predictions than the baselines. In particular, even with a single GP, UNLIMITD-F outperforms the baselines.

it to the UNLIMITD with single GP structure as well as baselines including MAML and MMAML. Details on the problems is in Appendix A.7, while the experiment details are in Appendix A.8 and A.9. Further discussion about our models on the unimodal setting is described in Appendix A.6.1.

Both the OoD detection and the prediction performances are better with the mixture structure than with the single GP structure (Figure 2), indicating that the mixture model is a useful structure for $\tilde{p}_\xi(f)$. This is reflected in the quality of the learned priors (see Appendix A.6.2 for more results). Further note how our UNLIMITD have efficient OoD detection and outperforms MAML; it achieves much better generalization when decreasing the number of context samples K (Figure 2). This demonstrates the strength of our probabilistic approach for multimodal meta-learning: even if the probabilistic assumptions are not optimal, the predictions are still accurate and can beat baselines. The performance and trade-off among different variants (i.e., UNLIMITD-I, UNLIMITD-R, and UNLIMITD-F) of our proposed method is further discussed in Appendix A.6.1

Acknowledgements

The authors acknowledge the MIT SuperCloud [Reuther et al., 2018] and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. The authors would like to thank MISTI MIT-France for supporting this research. C.A. further acknowledges support from Mines Paris Foundation. N.A. acknowledges support from the Edgerton Career Development Professorship.

References

Milad Abdollahzadeh, Touba Malekzadeh, and Ngai-Man Man Cheung. Revisit multimodal meta-learning through the lens of multi-task learning. *Advances in Neural Information Processing Systems*, 34:14632–14644, 2021.

Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic mirror descent on overparameterized nonlinear models. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *Advances in neural information processing systems*, 31, 2018.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020.

Ning Gao, Hanna Ziesche, Ngo Anh Vien, Michael Volpp, and Gerhard Neumann. What matters for meta-learning vision regression tasks? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14776–14786, 2022.

Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*, 2018.

James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 318–337. Springer, 2018.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of bayesian neural nets via local linearization. In *International Conference on Artificial Intelligence and Statistics*, pages 703–711. PMLR, 2021.

Tomoharu Iwata and Atsutoshi Kumagai. Meta-learning for out-of-distribution detection via density estimation in latent space. *arXiv preprint arXiv:2206.09543*, 2022.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Taewon Jeong and Heeyoung Kim. Ood-maml: Meta-learning for few-shot out-of-distribution detection and classification. *Advances in Neural Information Processing Systems*, 33:3907–3916, 2020.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

Wesley Maddox, Shuai Tang, Pablo Moreno, Andrew Gordon Wilson, and Andreas Damianou. Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 2737–2745. PMLR, 2021.

Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Apoorva Sharma, Navid Azizan, and Marco Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. In *Uncertainty in Artificial Intelligence*, pages 1958–1967. PMLR, 2021.

Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4): 1454–1485, 2017.

Risto Vuorio, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in Neural Information Processing Systems*, 32, 2019.

Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *Advances in neural information processing systems*, 31, 2018.

A Appendix

A.1 Background

A.1.1 Bayesian linear regression and Gaussian Processes

Efficient Bayesian meta-learning requires a tractable inference process at test time. In general, this is only possible analytically in a few cases. One of them is the Bayesian linear regression with Gaussian noise and a Gaussian prior on the weights. Viewing it from a nonparametric, functional approach, this model is equivalent to a Gaussian process (GP) [Rasmussen and Williams, 2005].

Let $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_K) \in \mathbb{R}^{N_x \times K}$ be a batch of K N_x-dimensional inputs, and let $\boldsymbol{y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_K) \in \mathbb{R}^{N_y K}$ be a vectorized batch of N_y-dimensional outputs. In the Bayesian linear regression model, these quantities are related according to $\boldsymbol{y} = \phi(\boldsymbol{X})^\top \hat{\boldsymbol{\theta}} + \varepsilon \in \mathbb{R}^{N_y K}$ where $\hat{\boldsymbol{\theta}} \in \mathbb{R}^P$ are the weights of the model, and the inputs are mapped via $\phi : \mathbb{R}^{N_x \times K} \to \mathbb{R}^{P \times N_y K}$. Notice how this is a generalization of the usual one-dimensional linear regression ($N_y = 1$).

If we assume a Gaussian prior on the weights $\hat{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and a Gaussian noise $\varepsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_\varepsilon)$ with $\boldsymbol{\Sigma}_\varepsilon = \sigma_\varepsilon^2 \boldsymbol{I}$, then the model describes a multivariate Gaussian distribution on $\boldsymbol{y}$ for any $\boldsymbol{X}$. Equivalently, this means that this model describes a GP distribution over functions, with mean and covariance function (or kernel)

$$\begin{aligned} \boldsymbol{\mu}_{\text{prior}}(\boldsymbol{x}_t) &= \phi(\boldsymbol{x}_t)^\top \boldsymbol{\mu}, \\ \text{cov}_{\text{prior}}(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) &= \phi(\boldsymbol{x}_{t_1})^\top \boldsymbol{\Sigma} \phi(\boldsymbol{x}_{t_2}) + \boldsymbol{\Sigma}_\varepsilon =: k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}) + \boldsymbol{\Sigma}_\varepsilon. \end{aligned} \tag{4}$$

This GP enables tractable computation of the likelihood of any batch of data $(\boldsymbol{X}, \boldsymbol{Y})$ given this distribution over functions. The structure of this distribution is governed by the feature map ϕ and the prior over the weights, specified by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

This distribution can also easily be conditioned to perform inference. Given a batch of data $(\boldsymbol{X}, \boldsymbol{Y})$, the posterior predictive distribution is also a GP, with an updated mean and covariance function

$$\begin{aligned} \boldsymbol{\mu}_{\text{post}}(\boldsymbol{x}_{t_*}) &= k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_*}, \boldsymbol{X}) \left(k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{\Sigma}_\varepsilon\right)^{-1} \boldsymbol{Y}, \\ \text{cov}_{\text{post}}(\boldsymbol{x}_{t_{1_*}}, \boldsymbol{x}_{t_{2_*}}) &= k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_{1_*}}, \boldsymbol{x}_{t_{2_*}}) - k_{\boldsymbol{\Sigma}}(\boldsymbol{x}_{t_{1_*}}, \boldsymbol{X}) \left(k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{\Sigma}_\varepsilon\right)^{-1} k_{\boldsymbol{\Sigma}}(\boldsymbol{X}, \boldsymbol{x}_{t_{2_*}}). \end{aligned} \tag{5}$$

Here, $\boldsymbol{\mu}_{\text{post}}(\boldsymbol{X}_*)$ represents our model's adapted predictions for the test data, which we can compare to $\boldsymbol{Y}_*$ to evaluate the quality of our predictions, for example, via mean squared error (assuming that test data is clean, following Rasmussen and Williams [2005]). The diagonal of $\text{cov}_{\text{post}}(\boldsymbol{X}_*, \boldsymbol{X}_*)$ can be interpreted as a per-input level of confidence that captures the ambiguity in making predictions with only a limited amount of context data.

A.1.2 The linearization of a neural network yields an expressive linear regression model

As discussed, the choice of feature map ϕ plays an important role in specifying a linear regression model. In the deep learning context, recent work has demonstrated that the linear model obtained when linearizing a deep neural network with respect to its weights at initialization, wherein the Jacobian of the network operates as the feature map, can well approximate the training behavior of wide nonlinear deep neural networks [Jacot et al., 2018, Neal, 1996, Azizan et al., 2021, Lee et al., 2017].

Let f be a neural network $f : (\boldsymbol{\theta}, \boldsymbol{x}_t) \mapsto \boldsymbol{y}_t$, where $\boldsymbol{\theta} \in \mathbb{R}^P$ are the parameters of the model, $\boldsymbol{x} \in \mathbb{R}^{N_x}$ is an input and $\boldsymbol{y} \in \mathbb{R}^{N_y}$ an output. The linearized network (w.r.t. the parameters) around $\boldsymbol{\theta}_0$ is

$$f(\boldsymbol{\theta}, \boldsymbol{x}_t) - f(\boldsymbol{\theta}_0, \boldsymbol{x}_t) \approx \boldsymbol{J}_{\boldsymbol{\theta}}(f)(\boldsymbol{\theta}_0, \boldsymbol{x}_t)(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

where $\boldsymbol{J}_{\boldsymbol{\theta}}(f)(\cdot, \cdot) : \mathbb{R}^P \times \mathbb{R}^{N_x} \to \mathbb{R}^{N_y \times P}$ is the Jacobian of the network (w.r.t. the parameters).

In the case where the model accepts a batch of K inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K)$ and returns $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K)$, we generalize f into $g : \mathbb{R}^P \times \mathbb{R}^{N_x \times K} \to \mathbb{R}^{N_y \times K}$, with $\boldsymbol{Y} = g(\boldsymbol{\theta}, \boldsymbol{X})$. Consequently, we generalize the linearization:

$$g(\boldsymbol{\theta}, \boldsymbol{X}) - g(\boldsymbol{\theta}_0, \boldsymbol{X}) \approx \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})(\boldsymbol{\theta} - \boldsymbol{\theta}_0),$$

where $\boldsymbol{J}(\cdot, \cdot) : \mathbb{R}^P \times \mathbb{R}^{N_x \times K} \to \mathbb{R}^{N_y K \times P}$ is a shorthand for $\boldsymbol{J}_{\boldsymbol{\theta}}(g)(\cdot, \cdot)$. Note that we have implicitly vectorized the outputs, and throughout the work, we will interchange the matrices $\mathbb{R}^{N_y \times K}$ and the vectorized matrices $\mathbb{R}^{N_y K}$.

This linearization can be viewed as the $N_y K$-dimensional linear regression

$$\boldsymbol{z} = \phi_{\boldsymbol{\theta}_0}(\boldsymbol{X})^\top \hat{\boldsymbol{\theta}} \in \mathbb{R}^{N_y K}, \tag{6}$$

where the feature map $\phi_{\boldsymbol{\theta}_0}(\cdot) : \mathbb{R}^{N_x \times K} \to \mathbb{R}^{P \times N_y K}$ is the transposed Jacobian $\boldsymbol{J}(\boldsymbol{\theta}_0, \cdot)^\top$. The parameters of this linear regression $\hat{\boldsymbol{\theta}} = (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$ are the *correction* to the parameters chosen as the linearization point. Equivalently, this can be seen as a kernel regression with the kernel $k_{\boldsymbol{\theta}_0}(\boldsymbol{X}_1, \boldsymbol{X}_2) = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}_1)\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}_2)^\top$, which is commonly referred to as the Neural Tangent Kernel (NTK) of the network. Note that the NTK depends on the linearization point $\boldsymbol{\theta}_0$. Building on these ideas, Maddox et al. [2021] show that the NTK obtained via linearizing a DNN *after* it has been trained on a task yields a GP that is well-suited for adaptation and fine-tuning to new, similar tasks. Furthermore, they show that networks trained on similar tasks tend to have similar Jacobians, suggesting that neural network linearization can yield an effective model for multi-task contexts such as meta-learning. In this work, we leverage these insights to construct our parametric functional distribution $\tilde{p}_\xi(f)$ via linearizing a neural network model.

A.2 The variants of UNLIMITD: Parameterizing the prior covariance over the weights

Imposing a unit covariance. One simple way to tackle this issue would be to remove $\boldsymbol{\Sigma}$ from the learnable parameters ξ, i.e., fixing it to be the identity $\boldsymbol{\Sigma} = \boldsymbol{I}_P$. In this case, $\xi = (\boldsymbol{\theta}_0, \boldsymbol{\mu})$. This computational benefit comes at the cost of model expressivity, as we lose a degree of freedom in how

we can optimize our learned prior distribution $\tilde{p}_\xi(f)$. In particular, we are unable to choose a prior over the weights of our model that captures correlations between elements of the feature map.

A trade-off between feature-map expressiveness and learning a rich prior over the weights. Note that even if a low-dimensional representation of $\boldsymbol{\Sigma}$ enriches the prior distribution over the weights, it also restrains the expressiveness of the feature map in the kernel by projecting the P-dimensional features $\boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})$ on a subspace of size $s \ll P$ via $\boldsymbol{Q}$. This presents a trade-off: we can use the full feature map, but limit the weight-space prior covariance to be the identity matrix by keeping $\boldsymbol{\Sigma} = \boldsymbol{I}$ (case UNLIMITD-I). Alternatively, we could learn a low-rank representation of $\boldsymbol{\Sigma}$ by randomly choosing s orthogonal directions in $\mathbb{R}^P$, with the risk that they could limit the expressiveness of the feature map if the directions are not relevant to the problem that is considered (case UNLIMITD-R). As a compromise between these two cases, we can choose the projection matrix more intelligently and project to the most impactful subspace of the full feature map — in this way, we can reap the benefits of a tuneable prior covariance while minimizing the useful features that the projection drops. To select this subspace, we construct this projection map by choosing the top s eigenvectors of the Fisher information matrix (FIM) evaluated on the training dataset $\mathcal{D}$ (case UNLIMITD-F). Recent work has shown that the FIM for deep neural networks tends to have rapid spectral decay [Sharma et al., 2021], which suggests that keeping only a few of the top eigenvectors of the FIM is enough to encode an expressive task-tailored prior.

A.3 A tractable way of finding the perturbation directions in weight space that impact the most the predictions of an entire dataset

Deep neural networks have a large number of parameters, making the feature map $\phi_{\boldsymbol{\theta}_0}$ high-dimensional. However, recent work has shown that only a small subspace of the weight space is impactful. For example, to perform continual learning, Farajtabar et al. [2020] leverage the fact that it is sufficient to update the parameters orthogonally to a few directions only to avoid catastrophic forgetting. Sagun et al. [2017] have shown that the Hessian of a deep neural network can be summarized in a few number of directions, due to rapid spectral decay. This encourages finding a method to extract these meaningful directions of the weight space.

A.3.1 Link with the Fisher Information Matrix

We define these main directions as the ones that have the most impact on the predictions of a whole dataset. To find them, we first find a way to quantify the influence of an infinitesimal weight perturbation. Using the second-order approximation of that quantity, we then describe in the deep-learning context a tractable way to find the directions.

Setting We take the same setting as Section A.1.2, and we describe a method to quantify the influence of a parameter perturbation $\tilde{\boldsymbol{\theta}}$ on the predictions of a dataset of tasks $\mathcal{D}$. To do so, we leverage a probabilistic interpretation of the model: we assume a Gaussian pdf over the observations for given inputs and parameters $p_{\boldsymbol{\theta}}(\boldsymbol{Y}|\boldsymbol{X}) \sim \mathcal{N}(g(\boldsymbol{\theta}, \boldsymbol{X}),\ \boldsymbol{\Sigma}_\varepsilon)$, where the covariance of the noise $\boldsymbol{\Sigma}_\varepsilon$ is diagonal $\boldsymbol{\Sigma}_\varepsilon = \sigma_\varepsilon^2 \boldsymbol{I}$ (just as in Section A.1.1).

Perturbation of the prediction of a batch of inputs. Before quantifying the influence of the perturbation on the predictions of the whole task dataset, we do it for the prediction of a batch of inputs $g(\boldsymbol{\theta}_0, \boldsymbol{X})$.

We borrow the method from Sharma et al. [2021]: we quantify the influence of a parameter perturbation by computing the Kullback-Leibler divergence between $p_{\boldsymbol{\theta}_0}(\boldsymbol{Y}|\boldsymbol{X})$ and $p_{\boldsymbol{\theta}_0+\tilde{\boldsymbol{\theta}}}(\boldsymbol{Y}|\boldsymbol{X})$. The expansion is:

$$\delta(\boldsymbol{\theta}_0, \boldsymbol{X})(\tilde{\boldsymbol{\theta}}) := D_{\mathrm{KL}}(p_{\boldsymbol{\theta}_0}(\boldsymbol{Y}|\boldsymbol{X}) \| p_{\boldsymbol{\theta}_0+\tilde{\boldsymbol{\theta}}}(\boldsymbol{Y}|\boldsymbol{X})) \approx \tilde{\boldsymbol{\theta}}^\top \boldsymbol{F}(\boldsymbol{\theta}_0, \boldsymbol{X}) \tilde{\boldsymbol{\theta}} + o(\|\tilde{\boldsymbol{\theta}}\|^2) \tag{7}$$

where $\boldsymbol{F}(\boldsymbol{\theta}_0, \boldsymbol{X}) := \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})^\top \boldsymbol{\Sigma}_\varepsilon^{-1} \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}) = \sigma_\varepsilon^{-2} \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}) \in \mathbb{R}^{P \times P}$ is the empirical Fisher Information Matrix (FIM) of the batch of inputs $\boldsymbol{X}$, computed on the parameters $\boldsymbol{\theta}_0$.

Generalization: perturbation of the prediction of a dataset. Now we can define the influence of a parameter perturbation on the whole training dataset $\mathcal{D}$, by generalizing the previous definition:

$$\delta(\boldsymbol{\theta}_0, \mathcal{D})(\tilde{\boldsymbol{\theta}}) := \frac{1}{N} \sum_{i=1}^{N} \delta(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)(\tilde{\boldsymbol{\theta}})$$

Using equation 7, this quantity verifies:

$$\delta(\boldsymbol{\theta}_0, \mathcal{D})(\tilde{\boldsymbol{\theta}}) \approx \tilde{\boldsymbol{\theta}}^\top \left(\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{F}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) \right) \tilde{\boldsymbol{\theta}} + o(\|\tilde{\boldsymbol{\theta}}\|^2) \tag{8}$$

which gives a natural definition for the FIM of the whole dataset by analogy with equation 7:

$$\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D}) := \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{F}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) = \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) \in \mathbb{R}^{P \times P}$$

The expansion in equation 8 shows that the FIM of the dataset is a second-order approximation describing the influence of a parameter perturbation over the entire dataset. In particular, the eigenvectors of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$ with the highest eigenvalues are the directions that impact the most the predictions.

A.3.2 Computing the top eigenvectors of the Fisher Information Matrix of the dataset in a deep learning context

Naively computing the top eigenspace of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$ requires processing a $P \times P$ matrix, which is intractable in the deep-learning context (where P can surpass 10^6). Instead, we decide to use the method used by Sharma et al. [2021], which leverages a low-rank-approximation-based technique (namely matrix sketching) developed by Tropp et al. [2017].

Sketching the FIM of the dataset The key idea behind this technique is to build two small random sketches of the FIM, $(\boldsymbol{Y}, \boldsymbol{W}) := \mathcal{S}\left(\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})\right)$, that with high probability contain enough information to reconstruct the top s eigenvectors of $\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})$. The linearity of $\mathcal{S}$ simplifies the sketching process by breaking down the computation into individual sketches:

$$(\boldsymbol{Y}, \boldsymbol{W}) := \mathcal{S}(\boldsymbol{F}(\boldsymbol{\theta}_0, \mathcal{D})) = \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} \mathcal{S}\left(\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i) \right) =: \frac{1}{N\sigma_\varepsilon^2} \sum_{i=1}^{N} (\boldsymbol{Y}^i, \boldsymbol{W}^i)$$

In particular, the sketch $(\boldsymbol{Y}, \boldsymbol{W})$ can be updated in-place and does not require to store all the individual sketches. Given a sketch budget $k + l$ (Sharma et al. [2021] recommends choosing $k := 2s + 1$ and $l := 4s + 3$) and two random normal matrices $\boldsymbol{\Omega} \in \mathbb{R}^{k \times P}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{l \times P}$, the random individual sketches $(\boldsymbol{W}^i, \boldsymbol{Y}^i)$ are defined as:

$$\begin{cases} \boldsymbol{Y}^i & := & ((\boldsymbol{\Omega} \boldsymbol{J}_i^\top) \boldsymbol{J}_i)^\top & \in \mathbb{R}^{P \times k} \\ \boldsymbol{W}^i & := & (\boldsymbol{\Psi} \boldsymbol{J}_i^\top) \boldsymbol{J}_i & \in \mathbb{R}^{l \times P} \end{cases}$$

where $\boldsymbol{J}_i := \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)$.

Sketch-based computation of the top eigenspace Once the sketches are computed, the function FIXEDRANKSYMAPPROX by Tropp et al. [2017] computes the first eigenvectors and eigenvalues of the FIM of the dataset. Overall, the memory footprint to find the sketches and the top eigenspace is $O(P(s + N_y M))$, where s is the number of queried eigenvectors and M is the size of $\widetilde{\boldsymbol{X}}^i$. As long as $s \ll P$, this computation is tractable, given that $N_y M \ll P$ is usual deep learning contexts. Algorithm 2 summarizes the process that yields the FIM-based projections via sketching. We drop the scaling coefficient σ_ε^{-2} as it doesn't affect the computation, given that we only want orthogonal eigenvectors and eigenvalues.

A.4 Meta-training the Parametric Task Distribution

The key to our meta-learning approach is to estimate the quality of $\tilde{p}_\xi(f)$ via the NLL of context data from training tasks, and use its gradients to update the parameters of the distribution ξ. Optimizing

Algorithm 2 Computing the FIM-based projections

Require: s (desired size of the subspace)

$k \leftarrow 2s + 1$
$l \leftarrow 4s + 3$
Draw $\boldsymbol{\Omega} \in \mathbb{R}^{k \times P}$, $\boldsymbol{\Psi} \in \mathbb{R}^{l \times P}$, two random normal matrices
Initialize $\boldsymbol{Y} = 0 \in \mathbb{R}^{P \times k}$, $\boldsymbol{W} = 0 \in \mathbb{R}^{l \times P}$
for all training task $\mathcal{T}^i$ **do**:
    $\boldsymbol{J}_i \leftarrow \boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}^i})$
    $\boldsymbol{Y} \leftarrow \boldsymbol{Y} + 1/N((\boldsymbol{\Omega}\boldsymbol{J}_i^\top)\boldsymbol{J}_i)^\top$
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + 1/N(\boldsymbol{\Psi}\boldsymbol{J}_i^\top)\boldsymbol{J}_i$

Algorithm 3 UNLIMITD-I: meta-training with identity prior covariance

Initialize $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$.
for all epoch **do**:
    Sample n tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
    for all $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**:
        $NLL_i \leftarrow \text{GAUSSNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu},\ \boldsymbol{J}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon)$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
    Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu}} \sum_i NLL_i$

this loss over tasks in the dataset draws $\tilde{p}_\xi(f)$ closer to the empirical distribution present in the dataset, and hence towards the true distribution $p(f)$.

We present three versions of UNLIMITD, depending on the choice of structure of the prior covariance over the weights (see Section 3.1 and Appendix A.2 for more details). UNLIMITD-I (Algorithm 3) is the meta-training with the fixed identity prior covariance. UNLIMITD-R and UNLIMITD-F (Algorithm 4) learn a low-dimensional representation of that prior covariance, either with random projections or with FIM-based projections.

Computing the likelihood. In the algorithms, the function GAUSSNLL($\boldsymbol{Y}^i$; m, K) stands for NLL of $\boldsymbol{Y}^i$ under the Gaussian $\mathcal{N}(m, K)$ (see equation 2). In the mixture case, we instead use MIXTNLL, which wraps equation 3 and calls GAUSSNLL for the individual NLL computations (see discussion in Section 3.2). In this case, $\boldsymbol{\mu}$ becomes $\{\boldsymbol{\mu}_j\}_{j=1}^{j=\alpha}$ and $\boldsymbol{s}$ becomes $\{\boldsymbol{s}_j\}_{j=1}^{j=\alpha}$ when applicable.

Finding the FIM-based projections. The FIM-based projection matrix aims to identify the elements of $\phi = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X})$ that are most relevant for the problem (see Section 3.1 and Appendix A.3). However, this feature map evolves during training, because it is $\boldsymbol{\theta}_0$-dependent. How do we ensure that the directions we choose for $\boldsymbol{Q}$ remain relevant during training? We leverage results from Fort et al. [2020], stating that the NTK (the kernel associated with the Jacobian feature map, see Section A.1.2) changes significantly at the beginning of training and that its evolution slows down as training goes on. This suggests that as a heuristic, we can compute the FIM-based directions after partial training, as they are unlikely to deviate much after the initial training. For this reason, UNLIMITD-F (Algorithm 4) first calls UNLIMITD-I (Algorithm 3) before computing the FIM-based $\boldsymbol{Q}$ that yields intermediate parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$. Then the usual training takes place with the learning of $\boldsymbol{s}$ in addition to $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$.

A.5 UNLIMITD as a generalization of ALPaCA

Restraining the linearization to the last layer Remember the linear regression of equation 6, that we obtained by linearizing the network with all its layers. Let's separate the parameters of the network $\boldsymbol{\theta}$ into two: the parameters of all the layers but the last one $\boldsymbol{\lambda}$, and the parameters of the last layer $\boldsymbol{\rho}$: $\boldsymbol{\theta} = \boldsymbol{\lambda} \cup \boldsymbol{\rho}$.

We assume that the last layer is dense with biases: we will note $\boldsymbol{\rho}^w$ the weight matrix and $\boldsymbol{\rho}^b$ the biases of this last layer. N_ψ will stand as the dimension of the activations right before the last layer: in particular, $\boldsymbol{\rho}^w \in \mathbb{R}^{N_y \times N_\psi}$ and $\boldsymbol{\rho}^b \in \mathbb{R}^{N_y}$. P' will stand as the size of $\boldsymbol{\rho}$: in our case,

Algorithm 4 UNLIMITD-R and UNLIMITD-F: meta-training with a learnt covariance

if using random projections **then**
  Find random projection $\boldsymbol{Q}$
  Initialize $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$
else if using FIM-based projections **then**
  Find intermediate $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$ with UNLIMITD-I ▷ see Alg. 3
  Find $\boldsymbol{Q}$ via FIMPROJ(s); initialize $\boldsymbol{s}$. ▷ see Alg. 2
for all epoch **do**:
  Sample n tasks $\{\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)\}_{i=1}^{i=n}$
  for all $\mathcal{T}^i, (\boldsymbol{X}^i, \boldsymbol{Y}^i)$ **do**:
    $NLL_i \leftarrow \text{GAUSSNLL}(\boldsymbol{Y}^i; \boldsymbol{J}\boldsymbol{\mu},\ \boldsymbol{J}\boldsymbol{Q}^\top \text{diag}(\boldsymbol{s}^2)\boldsymbol{Q}\boldsymbol{J}^\top + \boldsymbol{\Sigma}_\varepsilon)$ ▷ $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{\theta}_0, \boldsymbol{X}^i)$
  Update $\boldsymbol{\theta}_0$, $\boldsymbol{\mu}$, $\boldsymbol{s}$ with $\nabla_{\boldsymbol{\theta}_0 \cup \boldsymbol{\mu} \cup \boldsymbol{s}} \sum_i NLL_i$

$P' = N_\psi \times N_y + N_y$. We implicitly vectorize $\boldsymbol{\rho}$, such that:

$$\boldsymbol{\rho} = \begin{pmatrix} \text{vec}\, \boldsymbol{\rho}^w \\ \boldsymbol{\rho}^b \end{pmatrix} \in \mathbb{R}^{N_\psi N_y + N_y} = \mathbb{R}^{P'}$$

We now restrain the linear regression to the last layer as follows:

$$\boldsymbol{y} = \boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}))(\boldsymbol{\rho} - \boldsymbol{\rho}_0) + \varepsilon \tag{9}$$

where:

- $\psi_{\boldsymbol{\lambda}}(\cdot) : \mathbb{R}^{N_x \times K} \to \mathbb{R}^{N_\psi \times K}$ stands for the function that maps the inputs and the activations right before the last layer;
- $\boldsymbol{J}'(\cdot, \cdot) : \mathbb{R}^{P'} \times \mathbb{R}^{N_\psi \times K} \to \mathbb{R}^{N_y K \times P'}$ stands for the jacobian of the last layer *with respect to the parameters* $\boldsymbol{\rho}$. We can write the jacobian in closed-form due to the linearity of the last layer:

  $$\boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(\boldsymbol{X})) = \begin{pmatrix} \psi_{\boldsymbol{\lambda}}(\boldsymbol{X})^\top \otimes \boldsymbol{I}_{N_y} & \boldsymbol{I}_{N_y} \end{pmatrix} \in \mathbb{R}^{N_y K \times P'}$$

  Note that $\boldsymbol{J}'(\boldsymbol{\rho}_0, \psi_{\boldsymbol{\lambda}}(x))$ does not depend on the linearization point $\boldsymbol{\rho}_0$ (could be expected, given the linearity of the last layer).
- $\boldsymbol{\rho} - \boldsymbol{\rho}_0 \in \mathbb{R}^{P'}$ is the *correction* to the last parameters. We will note the correction to the weights as $\hat{\boldsymbol{\rho}}^w := \boldsymbol{\rho}^w - \boldsymbol{\rho}_0^w$ and the correction to the bias as $\hat{\boldsymbol{\rho}}^b := \boldsymbol{\rho}^b - \boldsymbol{\rho}_0^b$.

Using Kronecker's product identities, the linear regression in equation 9 can be rewritten:

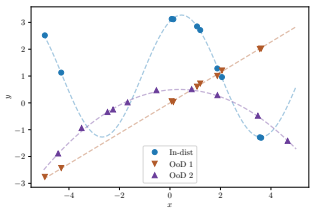
$$\boldsymbol{y} = \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}) + \hat{\boldsymbol{\rho}}^b + \varepsilon \tag{10}$$

Also, as a side note, another way of getting this linear regression (equation 10) is to rewrite the initial linearization of Section A.1.2, but with respect to $\boldsymbol{\rho}$ only:

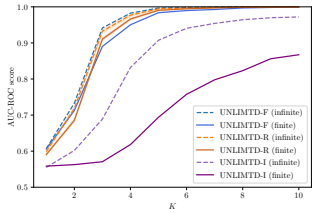
$$\begin{aligned} f(\boldsymbol{\lambda} \cup \boldsymbol{\rho}, \boldsymbol{x}_t) - f(\boldsymbol{\lambda} \cup \boldsymbol{\rho}_0, \boldsymbol{x}_t) &= \boldsymbol{\rho}^w \psi_{\boldsymbol{\lambda}}(x) + \boldsymbol{\rho}^b - (\boldsymbol{\rho}_0^w \psi_{\boldsymbol{\lambda}}(x) + \boldsymbol{\rho}_0^b) \\ &= (\boldsymbol{\rho}^w - \boldsymbol{\rho}_0^w)\psi_{\boldsymbol{\lambda}}(x) + (\boldsymbol{\rho}^b - \boldsymbol{\rho}_0^b) \\ &= \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(x) + \hat{\boldsymbol{\rho}}^b \end{aligned}$$

Doing it this way has the benefit to show that the linearization is exact when restricted to the last layer. For non-linear neural networks, the linearization is always an approximation.

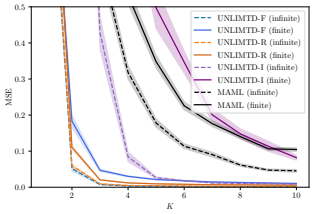
Adapting UNLIMITD to the last layer Just like what we did in the general case, we use the GP theory to make Bayesian inference on this linear regression. The new parameters of the GP $\tilde{p}_\xi(f)$ are now $\xi = (\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the parameters of the Gaussian prior over $\hat{\boldsymbol{\rho}}$. Note how $\boldsymbol{\rho}_0$ have disappeared from ξ: contrary to the general case, the linearization point is not optimized, as it does not impact the computation. Also note that $\boldsymbol{\lambda}$ has replaced $\boldsymbol{\rho}_0$, as it parameterizes the feature map.



(a) Examples of context data from in-dist. and OoD tasks



(b) AUC for OoD detection



(c) MSE on predictions

Figure 3: Performance of UNLIMITD for OoD detection and inference, as a function of the number of context datapoints K on the unimodal task. The training dataset consists of sinusoids, while OoD tasks are lines and quadratic tasks.

Comparison with ALPaCA In the ALPaCA setting, the linear regression is:

$$\boldsymbol{y}_a = \hat{\boldsymbol{\rho}}^w \psi_{\boldsymbol{\lambda}}(\boldsymbol{X}) + \varepsilon \tag{11}$$

Note that $\hat{\boldsymbol{\rho}}^w$ still plays the same role in the linear regression, but is not any *correction* anymore. Substracting 11 - 10 yields:

$$\boldsymbol{y} - \boldsymbol{y}_a = \hat{\boldsymbol{\rho}}^b$$

UNLIMITD restricted to the last layer is closely related to ALPaCA, in that they both perform a linear regression with the same kernel: the only difference lies in the additional bias term that is not considered in ALPaCA. Thus, we can think of UNLIMITD as a generalization of ALPaCA to all the layers of the network.

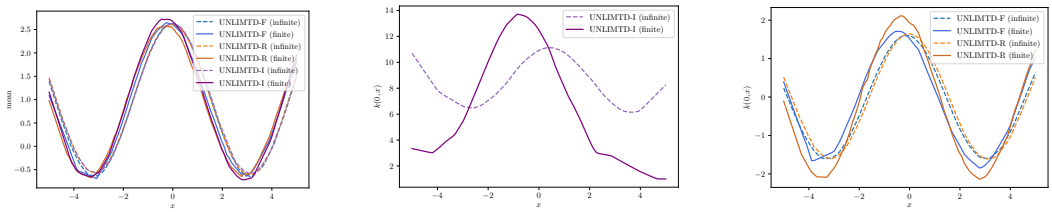
A.6 Additional results

A.6.1 Additional results (single-cluster case)

Unimodal meta-learning: The meta-learned prior accurately fits the tasks. First, we investigate the performance of UNLIMITD on a unimodal task distribution consisting of sinusoids of varying amplitude and phase, using the single GP structure for $\tilde{p}_\xi(f)$. We evaluate the performance of the proposed approaches and MAML. We also compare the results between training with an infinite amount of available sine tasks (infinite task dataset), and with a finite amount of available tasks (finite task dataset). Examples of predictions at the test time are available in Figure 5, along with confidence levels.

In both OoD detection and quality of predictions, UNLIMITD-R, and UNLIMITD-F perform better than UNLIMITD-I (Figure 3), and this is reflected in the quality of the learned prior $\tilde{p}_\xi(f)$ in each case (see Appendix A.6.1). With respect to the trade-off mentioned in Section 3.1, we find that for small networks, a rich prior over the weights matters more than the full expressiveness of the feature map, making both UNLIMITD-R, and UNLIMITD-F appealing. However, after running further experiments on a deep-learning image-domain problem, this conclusion does not hold for deep networks (see Appendix A.6.3), where keeping an expressive feature map is important (UNLIMITD-I and UNLIMITD-F are appealing in that case). Thus, UNLIMITD-F is the variant that we retain, for it allows similar or better performances than the other variants in all situations.

Comparison between variants In both OoD detection and quality of predictions, UNLIMITD-R, and UNLIMITD-F perform better than UNLIMITD-I (Figure 3), and this is reflected in the quality of the learned prior $\tilde{p}_\xi(f)$ in each case (see Appendix A.6.1). With respect to the trade-off mentioned in Section 3.1, we find that for small networks, a rich prior over the weights matters more than the full expressiveness of the feature map, making both UNLIMITD-R, and UNLIMITD-F appealing. However, after running further experiments on a deep-learning image-domain problem, this conclusion does not hold for deep networks (see Appendix A.6.3), where keeping an expressive feature map is important (UNLIMITD-I and UNLIMITD-F are appealing in that case). Thus, UNLIMITD-F is the variant that we retain, for it allows similar or better performances than the other variants in all situations.



(a) Mean $\mathbb{E}(f)$ (UNLIMITD-I, UNLIMITD-R, UNLIMITD-F)

(b) Covariance $\text{cov}(f(0), f(x))$ (UNLIMITD-I)

(c) Covariance $\text{cov}(f(0), f(x))$ (UNLIMITD-R, UNLIMITD-F)

Figure 4: Mean (4a) and covariance (4b, 4c) functions of $\tilde{p}_\xi(f)$ after different meta-trainings on the sine cluster (UNLIMITD-I, UNLIMITD-R and UNLIMITD-F, with a finite or infinite dataset). Note the scale, and how UNLIMITD-I has a less accurate covariance function that UNLIMITD-R and UNLIMITD-F.

Quality of the priors To qualitatively analyze the prior after meta-training on the sines, we plot the mean functions and the covariance functions of the resulting GP, that is $\tilde{p}_\xi(f)$ (Figure 4). All the trainings yield a similar mean for $\tilde{p}_\xi(f)$ (that is a cosine with amplitude 1.5 and offset 1) (Figure 4a), which is close to the theoretical mean of $p(f)$ (a cosine with amplitude 2.5 and offset 1). The covariance function of UNLIMITD-R and UNLIMITD-F (Figure 4c) resembles what we would expect for $p(f)$ (e.g. periodicity, negative correlation between 0 and π, etc.), but it is not the case for UNLIMITD-I (Figure 4b). This empirical analysis confirms the quantitative comparison in terms of OoD detection and prediction performance carried in Section 5.

Examples of predictions Figure 5 summarizes the predictions of the model meta-trained on sines, for a varying number of context inputs K, breaking down all the different cases of training.

A.6.2 Additional results (multi-cluster case)

Quality of the priors Figure 6 and Figure 7 show the mean and covariance functions of the GP (when training with a single GP), and the mean and covariance functions of the GPs composing the mixture(when training with a mixture of GPs). We note that in the case of the mixture, both of the Gaussians composing the mixture have correctly captured the common features shared by each of the clusters (respectively the linear and the sine cluster). For instance, the mean of the line cluster is the zero function, which matches Figure 6b, and the correlation between $x = 1$ and the other inputs is correctly rendered for linear tasks (Figure 7b).

When learning with a single GP, the learnt mean and covariance do not match any of the two clusters. For example, the mean has an intermediate offset between the offset of the sine cluster and the line cluster. This empirical analysis comforts the conclusions of Section 5: the mixture model yields better results than the single GP case.

A.6.3 UNLIMITD yields effective predictions on large-scale vision problems

We consider a regression-vision meta-learning problem from Gao et al. [2022], Shapenet1D, aiming to predict object orientations in space. In this problem, each task consists of a different object of which we want to predict the orientation. For each task, the context data consists of some images of the same object, but with different orientations; the query inputs are other images of the same object, with unknown orientations. Details on the problems and the datasets can be found in appendix A.7.

We train a deep learning model on Shapenet1D, and we compare the performances on the test set between UNLIMITD-I, UNLIMITD-R and UNLIMITD-F (Figure 8). More training and test details can be found in the Appendix A.10.

Both UNLIMITD-I and UNLIMITD-F yield better performances than MAML, achieving low angle errors. UNLIMITD-R however gives poor results, worse than that of MAML.

In terms of the trade-off of Section 3.1, our conclusion from small networks does not scale up to deep models. Here, the loss of valuable features is perceptible (that is what happens with UNLIMITD-R, when the randomness of the directions may drop such features), and not learning a rich prior over the

weights is not burdensome (UNLIMITD-I). However, UNLIMITD-F plays a role of compromise, by learning the prior covariance while keeping the few important features of the jacobian, as it gives comparable results to UNLIMITD-I.

A.7 Details on the regression problems

A.7.1 Simple regression problems

Our simple regression problems are inspired by Vuorio et al. [2019]'s work. They consists of three clusters of different types of tasks, and varying offset. The first cluster consists of sines with constant frequency and offset, but with a varying amplitude and phase:

$$\{x \mapsto A\sin(x+\varphi)+1 | A \in [0.1, 5], \varphi \in [0, \pi]\}$$

The second cluster consists of lines with a varying slope and no offset:

$$\{x \mapsto ax | a \in [-1, 1]\}$$

The last cluster consists of quadratic functions, with a varying quadratic coefficient and phase, and with a constant offset:

$$\left\{x \mapsto a(x-\varphi)^2 + 0.5 | a \in [-0.2, 0.2], \varphi \in [-2, 2]\right\}$$

In all these clusters, we add an artificial Gaussian noise on the context observations $\mathcal{N}(0,\ 0.05)$. The query datapoints remain noiseless, to remain coherent with the assumption from Section A.1.1 borrowed from Rasmussen and Williams [2005].

A.7.2 Vision problem

We consider the meta-learning vision, regression problem recently created by Gao et al. [2022] (namely Shapenet1D), that consists in estimating objects orientation from images. The objects have a vertical angular degree of freedom, and Figure 9 shows an example of such objects in different positions.

We use the same training and test datasets as Gao et al. [2022], with no kind of augmentation (in particular, no artificial noise on the ground-truth angles). In particular, we use the same intra-category (IC) evaluation dataset (that is, objects from the same categories as the objects used for training) and cross-category (CC) evaluation dataset (that is, objects from different categories as the ones used for training).

A.8 Training and test details for the single cluster case

A.8.1 Training details of UNLIMITD (single cluster case)

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our single-cluster experiment, the cluster is the sine cluster (see Appendix A.7).

In the case where there are an infinite number of available sine tasks during training (*ie* $N = +\infty$), the training is performed with $n = 24$ tasks per epoch, and at each epoch the context inputs are randomly drawn from $[-5, 5]$ (which means that $M = +\infty$). In the case where we restrict the available tasks to a finite number, we randomly choose $N = 10$ tasks and $M = 50$ context datapoints per task before training (they are shared among all the "finite" trainings) and perform the trainings with $n = 6$ tasks per epoch.

For all the trainings, the number of context inputs seen during training is $K = 10$.

For all trainings, we train UNLIMITD on $60,000$ epochs. When dealing with the UNLIMITD-F case, we allow half of the epochs $(30,000)$ to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$ (see Algorithm 4), and the other half *after*.

In the UNLIMITD-F case with infinite available tasks, a finite number of tasks is needed to compute the FIM: thus we build an artificial finite dataset of $N = 100$ and $M = P$ (arbitrary, but chosen so that $\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top$ gets a chance to be full-rank *ie* contain as much information as possible), that is used only for that computation.

In the UNLIMITD-F and the UNLIMITD-R cases, the subspace size is $s = 10$.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise $\sigma_\varepsilon = 0.05$, equal to the noise added to the context data. We compute the NLL using Rasmussen and Williams [2005]'s implementation.

A.8.2 Training details of MAML (single cluster case)

We train MAML baselines to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Finn et al. [2017]'s work.

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our single-cluster experiment, the cluster is the sine cluster (see Appendix A.7).

In the case where there are an infinite number of available sine tasks during training (*ie* $N = +\infty$), the training is performed with $n = 24$ tasks per epoch, and at each epoch the context and query inputs are randomly drawn from $[-5, 5]$. In the case where we restrict the available tasks to a finite number, we randomly choose $N = 10$ tasks and $M = 50$ datapoints per task (used for both context and query batches) before training (they are shared among all the "finite" trainings) and perform the trainings with $n = 6$ tasks per epoch.

For all trainings, the number of context datapoints is $K = 10$, and the number of query datapoints is $L = 10$. We meta-train for $70,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The inner learning-rate is kept constant, at 0.001. The number of inner updates is 5 during training, and 10 at test time.

A.8.3 Test details (single-cluster case)

For the OoD detection evaluation, we plot the AUC as a function of the number of the context inputs K. The AUC is computed using the NLL of the context inputs wrt to $\tilde{p}_\xi(f)$ (our uncertainty metric). The true-positives are the OoD tasks (lines and quadratic tasks) flagged as such; the false-positives are the in-distribution tasks (sines) flagged as OoD.

For the predictions, we plot the average and ci95 of the MSE on 1,000 tasks (100 queried inputs each). In the UNLIMITD-R case, we also compute the average and ci95 on 5 different random projections trainings.

A.9 Training and test details for the multi cluster case

A.9.1 Training details of UNLIMITD (multi-cluster case)

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our multi-cluster experiment, $\alpha = 2$: the clusters consist of the sine cluster and the linear cluster (see Appendix A.7).

For all the trainings, the training is performed with $n = 24$ tasks per epoch (with an infinite number of available sine tasks and linear tasks during training *ie* $N = +\infty$), and at each epoch the context inputs are randomly drawn from $[-5, 5]$ (which means that $M = +\infty$). In accordance with our equal probability assumption from Section 3.2, at each epoch $n/2 = 12$ tasks come from the sine cluster and $n/2 = 12$ tasks come from the linear cluster.

For all trainings, the number of context inputs seen during training is $K = 10$.

For all trainings, we train UNLIMITD algorithm on $60,000$ epochs: because we deal with the UNLIMITD-F case, we allow half of the epochs $(30,000)$ to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\{\boldsymbol{\mu}_j\}_{j=1}^{j=\alpha}$ (see Algorithm 4), and the other half *after*.

In the UNLIMITD-F case, a finite number of tasks is needed to compute the FIM: thus we build an artificial finite dataset of $N = 100$ and $M = P$ (arbitrary, but chosen so that $\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)\boldsymbol{J}(\boldsymbol{\theta}_0, \widetilde{\boldsymbol{X}}^i)^\top$ gets a chance to be full-rank *ie* contain as much information as possible), that is used only for that computation.

For all trainings, the subspace size is $s = 10$.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise $\sigma_\varepsilon = 0.05$, equal to the noise added to the context data. We compute the NLL using Rasmussen and Williams [2005]'s implementation.

When training with MIXT, we make sure to initialize $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ randomly with $\mathcal{N}(\mathbf{0},\ 0.5\boldsymbol{I})$, so that the meta-learning can effectively differentiate the two clusters. Also, in the MIXT case, the mean $\boldsymbol{\mu}$ is unique *before* computing the FIM. Once we have found the projection directions, we initialize $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$ with the intermediate $\boldsymbol{\mu}$, thus yielding two Gaussians.

A.9.2 Training details of MAML (multi-cluster case)

We reimplement and train a MAML baseline to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Finn et al. [2017]'s work.

The model is a neural network with 2 hidden layers, 40 neurons each, with a ReLU non-linearity. In our multi-cluster experiment, $\alpha = 2$: the clusters consist of the sine cluster and the linear cluster (see Appendix A.7).

The training is performed with $n = 24$ tasks per epoch (with an infinite number of available sine and linear tasks during training *ie* $N = +\infty$) and at each epoch the context and query inputs are randomly drawn from $[-5, 5]$. In accordance with our equal probability assumption from Section 3.2, at each epoch $n/2 = 12$ tasks come from the sine cluster and $n/2 = 12$ tasks come from the linear cluster.

For all trainings, the number of context datapoints is $K = 10$, and the number of query datapoints is $L = 10$. We meta-train for $70,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The inner learning-rate is kept constant, at 0.001. The number of inner updates is 5 during training, and 10 at test time.

A.9.3 Training details of MMAML (multi-cluster case)

We train a MMAML by using Vuorio et al. [2019]'s code, using the best setting mentionned in their paper (FiLM). We adapt their code to train it on our sine and linear clusters: we add an offset to their sine cluster (via their parameter `bias`), change the phase from $\sin(x - \varphi)$ to $\sin(x + \varphi)$ and specify via the arguments the slope range ($[-1, 1]$) and the y-intercept range ($[0, 0]$, because it does not vary in our case) of the line tasks. We also change the number of context inputs that we set to $K = 10$, to remain coherent with the rest of the trainings. The rest of the hyperparameters are kept identical to the command specified in the repository of MMAML. Finally, at test time we make the query ground-truths noiseless, to remain coherent with the rest of the test conditions.

A.9.4 Test details (multi-cluster case)

For the OoD detection evaluation, we plot the AUC as a function of the number of the context inputs K. The AUC is computed using the NLL of the context inputs wrt to $\tilde{p}_\xi(f)$ (our uncertainty metric). The true-positives are the OoD tasks (quadratic tasks) flagged as such; the false-positives are the in-distribution tasks (lines and sines) flagged as OoD.

For the predictions, we plot the average and ci95 of the MSE on 1,000 tasks (100 queried inputs each): half of them are sines and half of them are lines.

A.10 Training and test details for the vision problem

The model is a deep neural network, identical to the one used by Gao et al. [2022] except for the last layer: instead of doing a one-dimensional regression (where the output stands for an angle prediction), we perform a two-dimensional regression (where the output stands for a cosine and sine prediction). This choice is motivated by the fact that the Gaussian noise assumed in Section A.1.1 cannot capture the complexity of an angle error (e.g., predicting 361° should yield a low error when compared to the ground-truth angle 0°), but better renders the MSE that can be applied on cosine and sine.

A.10.1 Training details of UNLIMITD (vision case)

For all the trainings, there are $n = 10$ tasks per epoch and $K = 15$ context inputs per task. When dealing with the UNLIMITD-F case, we allow half of the epochs $(5,000)$ to the training *before* finding the intermediate $\boldsymbol{\theta}_0$ and $\boldsymbol{\mu}$ (see Algorithm 4), and the other half after.

In the UNLIMITD-F and UNLIMITD-R cases, the subspace size is $s = 100$.

For all the trainings, the meta-optimizer is Adam with an initial learning rate of 0.001. The noise is $\sigma_\varepsilon = 0.01$. We compute the NLL using Rasmussen and Williams [2005]'s implementation.

A.10.2 Training details of MAML (vision case)

We train a MAML baseline to compare our results with that of MAML. A large part of these hyperparameters is directly inspired from Gao et al. [2022].

The training is performed with $n = 10$ tasks per epoch. The number of context datapoints is $K = 15$, and the number of query datapoints is $L = 10$. We meta-train for $50,000$ epochs.

For all trainings, the meta-optimizer is Adam with an initial learning rate of 0.0005. The inner learning-rate is kept constant, at 0.002. The number of inner updates is 5 during training, and 20 a test time.

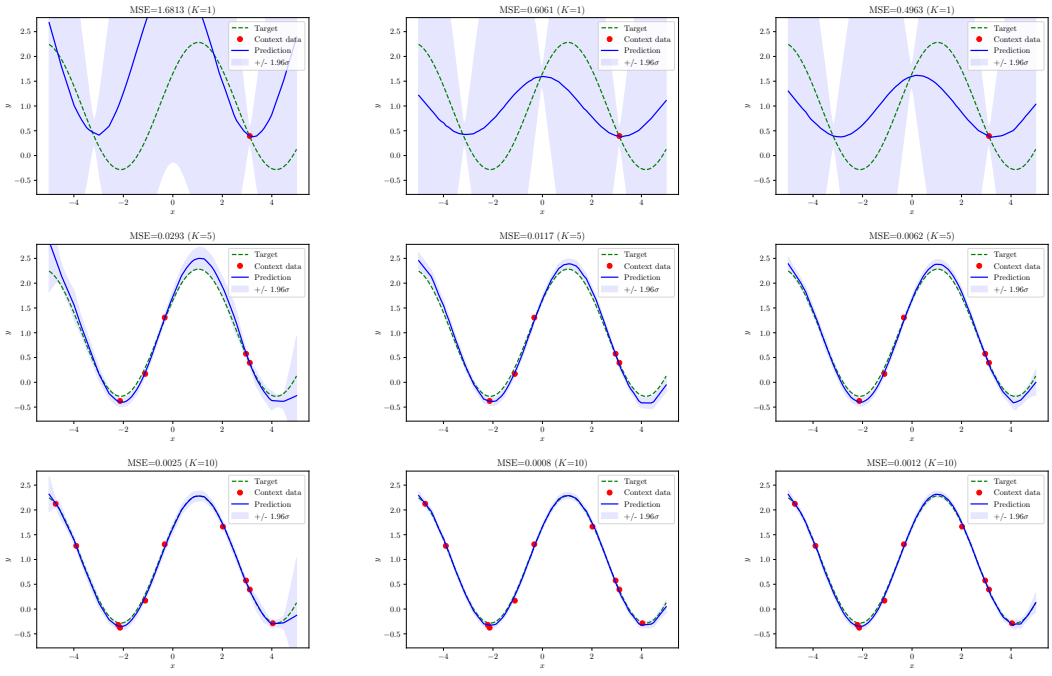
A.10.3 Test details (vision problem)

At test time, we wrap our model with the $\arctan$ function to convert the predictions angle predictions. Then, we compare the angle predictions with the ground-truth angles. To do so, we use the following error from Gao et al. [2022] to compare two angles:

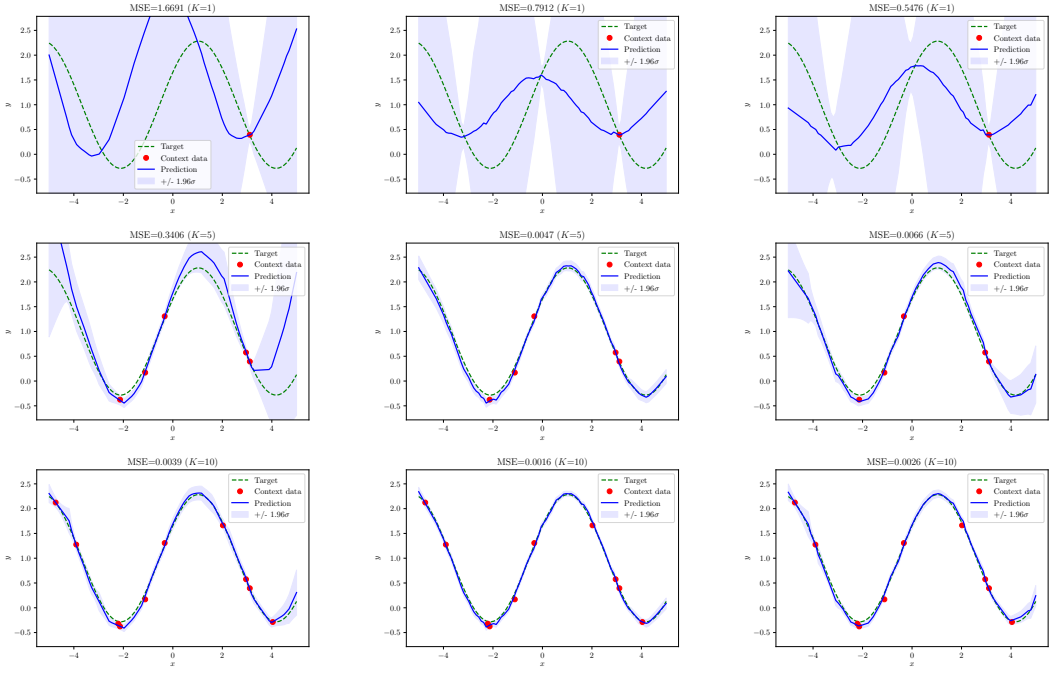
$$\mathcal{E}(\beta, \beta^*) = \min\{\mathcal{E}_{\beta^+,\beta^*}, \mathcal{E}_{\beta,\beta^*}, \mathcal{E}_{\beta^-,\beta^*}\}$$

where $\mathcal{E}_{\beta^\pm,\beta^*} = |y \pm 360 - y^*|$ and $\mathcal{E}_{\beta,\beta^*} = |y - y^*|$.

When plotting the performance (Figure 8), we plot the average and ci95 on 100 tasks (15 queried inputs each). For UNLIMITD-R, the average and ci95 are computed on 5 different random projection trainings.

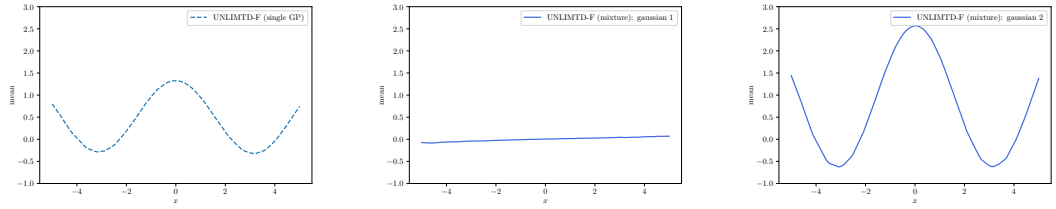


(a) UNLIMITD-I (inf) (b) UNLIMITD-R (inf) (c) UNLIMITD-F (inf)



(d) UNLIMITD-I (fin) (e) UNLIMITD-R (fin) (f) UNLIMITD-F (fin)

Figure 5: Example of predictions for a varying number of context inputs K, after different meta-trainings on the sine cluster (UNLIMITD-I, UNLIMITD-R and UNLIMITD-F, in the infinite and finite case). Standard deviation is from the posterior predictive distribution. Note how UNLIMITD-R and UNLIMITD-F perform better than UNLIMITD-I when it comes to reconstructing the sine with a smaller amount of context inputs.

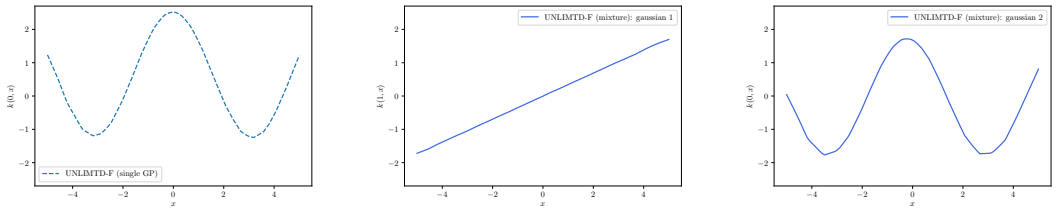


(a) Mean function $\mathbb{E}(f)$ of the GP (UNLIMITD-F with a single GP).

(b) Mean function $\mathbb{E}(f)$ of the first GP of the mixture (UNLIMITD-F with a mixture of GPs).

(c) Mean function $\mathbb{E}(f)$ of the second GP of the mixture (UNLIMITD-F with a mixture of GPs).

Figure 6: Mean functions of the GP (when learning with a GP) / the GPs composing the mixture (when learning a mixture of GPs), after training on both the sine and line cluster with UNLIMITD-F. Note how the mean of the single GP is intermediate between the ones of the mixture.



(a) Covariance function $\text{cov}(f(0), f(x))$ of the GP (UNLIMITD-F with a single GP).

(b) Covariance function $\text{cov}(f(1), f(x))$ of the first GP of the mixture (UNLIMITD-F with a mixture of GPs)

(c) Covariance function $\text{cov}(f(0), f(x))$ of the second GP of the mixture (UNLIMITD-F with a mixture of GPs)

Figure 7: Covariance functions of the GP (when learning with a GP) / the GPs composing the mixture (when learning a mixture of GPs), after training on both the sine and line cluster with UNLIMITD-F. Note how the covariance of the single GP is not accurate for any of the two clusters.

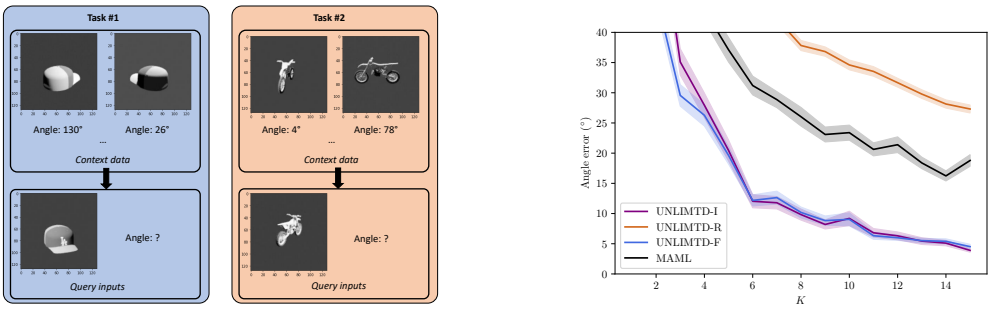
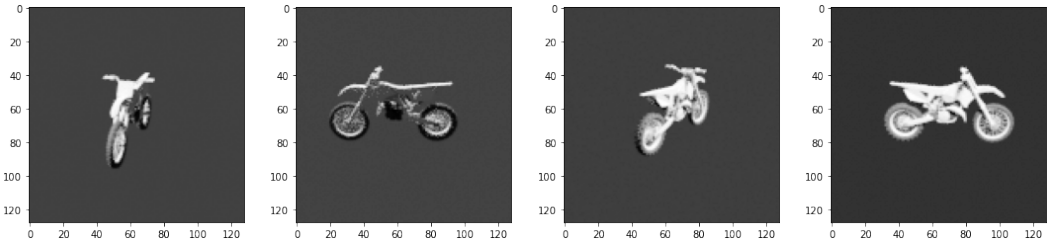


Figure 8: Left: Examples of vision tasks. Right: angle error as a function of the number of context datapoints K, after different meta-trainings on Shapenet1D (UNLIMITD-I, UNLIMITD-R and UNLIMITD-F vs MAML). Note how UNLIMITD-I and UNLIMITD-F make better predictions than MAML. Also note that contrary to the small network case, UNLIMITD-R performs worse than MAML and than the two other variants.



(a) Angle: 4° (c) Angle: 78° (e) Angle: 202° (g) Angle: 261°

(b) Angle: 26° (d) Angle: 130° (f) Angle: 243° (h) Angle: 285°

Figure 9: Examples of images and ground-truth angles from the Shapenet1D dataset [Gao et al., 2022]