Bayesian Optimization with a Neural Network Meta-learned on Synthetic Data Only

Samuel Müller [⋆ 1] **Sebastian Pineda** [⋆ 1] **Matthias Feurer** [1]
Noah Hollmann [2] **Josif Grabocka** [1] **Frank Hutter** [1,3]
[1] University of Freiburg, [2] Charité Berlin, [3] Bosch Center for Artificial Intelligence
Correspondence to Samuel Müller: muellesa@cs.uni-freiburg.de
[⋆] Equal Contribution

Abstract

Bayesian Optimization (BO) is an effective approach to optimize black-box functions, relying on a probabilistic surrogate to model the response surface. In this work, we propose to use a Prior-data Fitted Network (PFN) as a cheap and flexible surrogate. PFNs are neural networks that approximate the Posterior Predictive Distribution (PPD) in a single forward-pass. Most importantly, they can approximate the PPD for any prior distribution that we can sample from efficiently. Additionally, we show what is required for PFNs to be used in a standard BO setting with common acquisition functions. We evaluated the performance of a PFN surrogate for Hyperparameter optimization (HPO), a major application of BO. While the method can still fail for some search spaces, we fare comparable or better than the state-of-the-art on the HPO-B and PD1 benchmark.

1 Introduction

Gaussian processes (GPs) are today's de facto standard surrogate model in Bayesian Optimization (BO) [7]. This dominance can be attributed to both their strong performance and their mathematical convenience. The set of priors that GPs can model is restricted, though. A GP can only model priors that can be encoded as a valid kernel function and, as the name says, only normally-distributed priors. Moreover, kernel hyperparameters are typically not treated in a Bayesian manner, likely due to the high cost of running Markov Chain Monte Carlo for this full Bayesian view, which was already shown to yield strong results a decade ago [1, 16]. Instead, GPs are usually fitted with maximum likelihood in most BO packages; this procedure, often referred to as "Empirical Bayes" makes Bayesian Optimization less principled from a Bayesian point of view than it could be.

The recently proposed Prior-data Fitted Networks (PFNs, 14) show that fast approximate Bayesian inference is possible by training a neural network to mimic the posterior predictive distribution (PPD). This is a powerful approach, as it makes approximate Bayesian inference readily usable in novel applications and allows using any prior that we can sample from. E.g., it has been applied to perform Bayesian inference on tabular data [10] with a prior over different neural architectures and their weights, generalizing the usual notion of Bayesian deep learning to not only quantify uncertainty over the weights of a fixed architecture but over the joint space of architectures and weights.

In this work, we show how to use PFNs as a more flexible, Bayesian, drop-in replacement for Gaussian Processes for real-world Bayesian optimization problems. We show that PFNs can be applied to Bayesian optimization and find important pre-processing steps to make them a strong BO surrogate. In our experiments, we focus on BO for hyperparameter optimization (HPO 6). This is an important task to bring machine learning algorithms to top performance and a major BO application [3, 16, 9].

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

2 Bayesian Optimization with PFNs

Prior-data Fitted Networks (PFNs, 14) are neural networks (meta-)trained to perform approximate Bayesian prediction. That is, PFNs are trained to predict some output $y \in \mathbb{R}$ conditioned on an input $\boldsymbol{x} \in \mathbb{R}^k$ and a training set of given input-output examples $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. The PFN is trained on this task with samples obtained from a prior distribution p, i.e., $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N), (\boldsymbol{x}, y)\} \sim p(\mathcal{D})$. The algorithm thus is solely trained on artificial data. The loss function for training a PFN q_θ with parameters θ is the cross entropy for predicting the hold-out example's label

$$\ell_\theta = \mathbb{E}_{(\boldsymbol{x},y)\cup D \sim p(\mathcal{D})}[-\log q_\theta(y|\boldsymbol{x}, D)]. \tag{1}$$

After training on artificial datasets drawn from the prior $p(\mathcal{D})$, we use the trained PFN to approximate the PPD of this prior, based on which we compute the acquisition values. Last, we maximize the acquisition values for new inputs to find the configuration to be queried next.

We contrast Bayesian optimization with PFNs and GPs (with Empirical Bayes) in Algorithm 1 in the Appendix. The major difference is that PFNs have an up-front cost of fitting the PFN once, while Empirical Bayes incurs the online cost of fitting the hyperparameters in each iteration. We emphasize that PFNs incur the training cost exactly once per prior, and that a single trained PFN can be used for BO on different tasks with different dimensions, just like the code base for GP regression is a one-time investment that is then generally applicable.

2.1 A HEBO-inspired Prior for BO

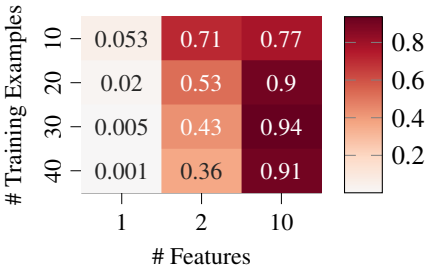
HEBO [4] is a state-of-the-art BO method that won the NeurIPS blackbox competition [20] and demonstrated excellent performance in a recent comparison [5]. Thus, we build a prior based on the one used by HEBO. Our HEBO-inspired prior considers parameters $\phi = \{\text{outputscale}, \text{lengthscale}, \text{noise}, \text{Kumaraswamy parameters}\}$ modelled like random variables subject to some distribution $p(\phi; \psi)$ that depend on hyperparameters ψ. To sample functions, we perform the following four steps per dataset: *i)* Sample $\boldsymbol{x}$ from a uniform distribution, *ii)* sample the parameters $\phi \sim p(\phi; \psi)$, *iii)* transform $\boldsymbol{x}$ using the Kumaraswamy CDF, *iv)* draw the outputs for our dataset $y \sim \mathcal{N}(0, K(\boldsymbol{x}; \phi))$, where K is the covariance matrix defined by the 3/2 Matérn kernel, which uses the sampled outputscale, lengthscale and noise. For additional details on ψ see Appendix B.

2.2 PFN Adaptions for BO

Acquisition Functions When we condition a PFN on new observations, we get an approximation of the PPD in a forward-pass $p(y|\boldsymbol{x}, D) \approx q_\theta(y|\boldsymbol{x}, D)$ in the form of a Riemann distribution, i.e., a distribution over bins spanning a reasonable output range. The Riemann distribution [14] allows us to calculate the utility exactly for different acquisition functions, e.g., EI, PI or UCB. To exemplify the general approach, we outline how to compute the $PI(f^*) = \int_{-\infty}^{\infty}[y > f^*]p(y)dy$ for the unbounded Riemann distribution in Appendix D. We experimented with EI, PI, UCB and the HEBO acquisition function ensemble [4] and found that simple EI works well enough and is stable.

Inference Strategies We found that two more tricks improve the performance of our PFNs considerably. *i)* We use a power transform to transform the observed outputs to a distribution more similar to a standard-normal, like proposed by HEBO [4]. *ii)* We found that simple min-max input normalization between 0 and 1 works well. In many search spaces, dimensions that naturally lie on a logarithmic scale are not correctly specified, though; this e.g., inspired input warping [18]. We found that we can improve performance in some scenarios by computing the PPD for an ensemble of $d + 1$ members, where one is normalized like above and for all others we apply a logarithm to a single dimension first. We also found that ensembling over different permutations of the features can improve performance, thus we do it when we do not otherwise ensemble.

PFN Training Procedure We only train a single PFN for one particular prior, even though the search spaces can have different numbers of features. To do this, we sample datasets with different numbers of dimensions during PFN training. Following [10], we zero-pad and linearly scale the features when the number of features k is smaller than the maximum number of features K by $\frac{K}{k}$ to make sure the magnitude of the inputs is similar across different numbers of features. We used the original PFNs settings for training [14]: an embedding size of 512 and six layers, using Adam [12] and



	# Features Approx.	1	2	10
Mean Regret	Emp. Bay.	0.057	0.059	0.110
	PFN	0.054	0.052	0.124
# Wins	Emp. Bay.	206	144	169
	PFN	239	154	171
Mean Rank	Emp. Bay.	1.517	1.506	1.501
	PFN	1.483	1.494	1.499

Figure 1: Left: Fraction of cases in which the PFN gives a higher likelihood to unseen examples across $1\,000$ datasets drawn from the prior. The optimization of the GP failed in 4.5% of cases. We ignored these cases to be as fair as possible. Right: BO performance after 50 evaluations on the prior with EI over $1\,000$ sampled datasets. The majority of runs yielded ties.

cosine-annealing [13] without any special tricks. The learning rate was chosen based on a simple grid search for minimal training loss.

3 Results

We present results on three problems: optimizing functions sampled from the HEBO prior, HPO on the HPO-B [15] benchmark and HPO on the PD1 [22] benchmark. We only consider the discrete setting, but plan to scale the results to continuous spaces in the future. We consider a budget of 50 evaluations and repeat experiments for five seeds per task for the real-world experiments.

3.1 Optimizing Prior Functions

We first study whether PFNs can do BO out of the box with as few confounders as possible, by evaluating them on functions sampled from the prior that they were trained on.

We first verified that the PFN predictions match the true GP posterior without Empirical Bayes, where it is computable; as shown in Figure 5 in the appendix. This is not the case when training the PFN on too few (100k) prior samples but the approximation becomes very tight with enough samples (we evaluated training on up to 80M samples, but performance was already stable with 20M).

Next, we consider the HEBO prior, following their hyper-prior setup as close as possible. We do not tune the hyper-prior hyperparameters, e.g. the distribution over the lengthscale, in this experiment to have a fair comparison. We evaluate the HEBO posterior based on our reimplementation of the HEBO model in GPyTorch with Empirical Bayes [8]. We stayed as close as possible to HEBO, but had to make some adaptions as one cannot sample from the HEBO model as it is. Specifically, we had to introduce simple priors for the hyperparameters of both the kernels used in HEBO (Matérn and linear kernel), as they did not have a prior attached. We introduce uniform priors ($\mathcal{U}(0, 1)$) for the $\mathrm{lengthscale}$ of both the Matérn and the Linear kernel and the $\mathrm{variance}$ of the linear kernel. Next, we trained a PFN using the HEBO prior with distributions for the hyperparameters (see Section 2.1) in order to enable PFNs to perform approximate MCMC over the hyperpriors. We compare the resulting PFN's fit vs. the Empirical Bayes approximation, as used in HEBO. Figure 1 shows that the likelihood assigned to held-out outputs is higher for the PFN in many cases; PFNs also become better with more features. We hypothesize that this is because HEBO's empirical Bayes approximation becomes too greedy in high dimensions.

Next, we perform BO on datasets sampled from the prior. Here, we have to introduce the acquisition function as a confounding factor, which might favor one or the other method. We chose expected improvement, since it is a default choice in BO, but are aware that this might have an impact on our results. For every feature size we sample $1\,000$ different functions, and for each function 1000 uniform random points in $[0.0, 1.0]$. Across dimensions, the GP with empirical Bayes and our PFN approximation worked very similar for Bayesian optimization. To assess our method qualitatively, we provide example query points of optimizations in Figures 4 and 5 in the appendix.

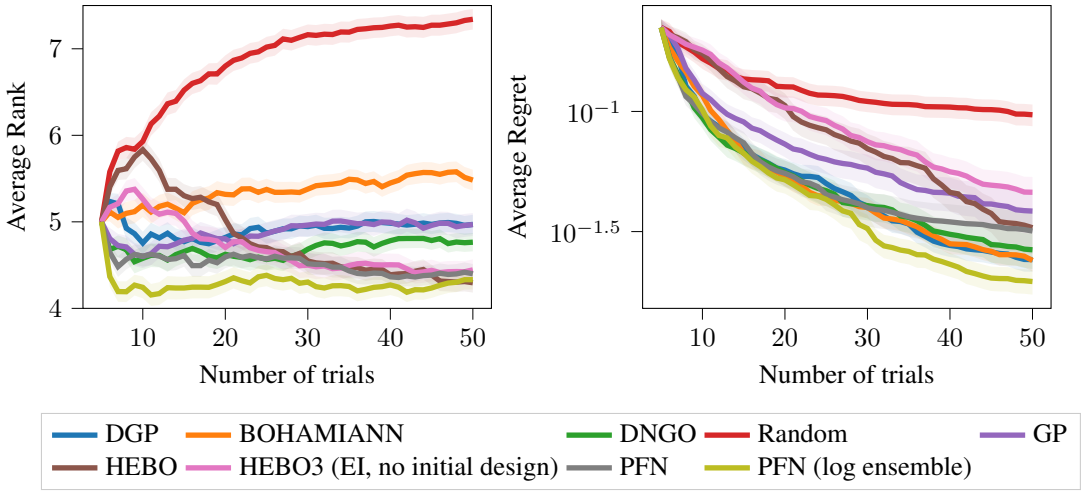


Figure 2: Performance over number of trials for the test search spaces of HPO-B for the test search spaces of HPO-B. Figure 6 shows per-search-space results.

3.2 Hyperparameter Optimization

.

Finally, we conduct a large-scale evaluation of PFNs as a surrogate for HPO. We perform experiments on the large-scale benchmark HPO-B [15], which contains 16 search spaces on 85 datasets. On HPO-B, we compare our method to the following baselines: 1) *Random Search (RS)* [2], 2) *Gaussian Processes (GP)* [11, 16] 3) *DNGO* [17] 4) *BOHAMIANN* [19], 5) *Deep Kernel Gaussian Processes* [23], 6) *HEBO* [4] and 7) a variant of HEBO that only uses EI and does not have initial design sampling, like our method.

For this experiment, we used a model with the prior described in Section 2.1. To decide on the prior hyperparameters ψ, we ran a random search over 50 configurations, picked the model that yielded the lowest regret in a subset of validation search spaces and report results on a separate set of test search spaces. We specify details on this split in Appendix A. Figure 2 reports results for the average rank and average regret across all the datasets from the test search spaces, showing the PFN to attain competitive results compared to the baselines. Moreover, an ensemble of PFNs with logged dimensions outperforms HEBO, the *state-the-of-the-art* in HPO. However, we would like to mention that the performance rankings are very inconsistent across search spaces, and that all methods, including the PFN, had some validation search spaces for which they performed very poorly; see Figure 6 in the appendix for examples.

We also evaluated our PFN for the very different image classification tuning benchmarks in PD1 [22]; the results in Appendix E show that in this case it performs very similarly to HEBO.

4 Conclusion & Limitations

We showed that PFNs can be adapted to perform as efficient BO surrogates. Still, we believe that there is a lot of room to improve upon our work. So far, this work only considers two real-world benchmarks and no continuous benchmark, as we did not manage to obtain strong results in a continuous optimization setting yet. This will therefore be the subject of our focus next.

5 Acknowledgments

Robert Bosch GmbH is acknowledged for financial support. We acknowledge funding by European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

Funded by the European Union

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG.

References

[1] R. Benassi, J. Bect, and E. Vazquez. Robust gaussian process-based global optimization using a fully bayesian expected improvement criterion. In C. Coello, editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2011.

[2] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[3] E. Brochu, V. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599v1 [cs.LG]*, 2010.

[4] A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. Griffiths, A. Maraval, H. Jianye, J. Wang, J. Peters, and H. Bou-Ammar. Hebo: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022.

[5] K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In Vanschoren et al. [21].

[6] M. Feurer and F. Hutter. Hyperparameter Optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book.

[7] P. Frazier. A tutorial on Bayesian optimization. *arXiv:1807.02811 [stat.ML]*, 2018.

[8] J. Gardner, G. Pleiss, D. Bindel, K. Weinberger, and A. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Curran Associates, 2018.

[9] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2022. in preparation.

[10] N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter. Meta-learning a real-time tabular automl method for small data. *arXiv:2207.01848*, 2022.

[11] D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015. Published online: `iclr.cc`.

[13] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

[14] S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, 2022. Published online: `iclr.cc`.

[15] S. Pineda Arango, H. Jomaa, M. Wistuba, and J. Grabocka. HPO-B: A large-scale reproducible benchmark for black-box hpo based on OpenML. In Vanschoren et al. [21].

[16] J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pages 2960–2968. Curran Associates, 2012.

[17] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pages 2171–2180. Omnipress, 2015.

[18] J. Snoek, K. Swersky, R. Zemel, and R. Adams. Input warping for Bayesian optimization of non-stationary functions. In E. Xing and T. Jebara, editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 1674–1682. Omnipress, 2014.

[19] J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*. Curran Associates, 2016.

[20] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In H. Escalante and K. Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 3–26, 2021.

[21] J. Vanschoren, S. Yeung, and M. Xenochristou, editors. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[22] Z. Wang, G. Dahl, K. Swersky, C. Lee, Z. Mariet, Z. Nado, J. Gilmer, J. Snoek, and Z. Ghahramani. Pre-training helps bayesian optimization too. *arXiv:2207.03084*, 2022.

[23] A. Wilson, Z. Hu, R. Salakhutdinov, and E. Xing. Deep kernel learning. In A. Gretton and C. Robert, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS'16)*, volume 51, pages 370–378. Proceedings of Machine Learning Research, 2016.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See our experiments in Section 3.
   (b) Did you describe the limitations of your work? [Yes] We do have limitations. We only evaluate on discrete benchmarks and we see that our method can utterly fail. These are described in the experiments in Section 3 and the conclusion in Section4
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not see any likely bad impact on society
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We are releasing the code once we are advancing this to a full conference submission. So far, it is still heavily under development.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For the main results we provide 95% confidence intervals.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The total amount of compute required to train each of the two PFNs used in this paper was 8 GPU-days on RTX 2080 Ti GPUs.
3. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Both benchmarks we use are cited, as well as the GP framework, we use.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

Algorithm 1: Bayesian Optimization with PFNs and Bayesian optimization with GPs

Input : A prior distribution over datasets $p(\mathcal{D})$, hyperparameter prior settings, initial observations $D = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_K, y_K)\}$, search space $\mathcal{X}$, number of BO iterations K, black-box function to optimize f, acquisition function α

Output : Best observed configuration $\boldsymbol{x}_*$ and response value $y*$

Train neural network q_θ by minimizing objective function in Equation 1 (once, offline)

for $i \leftarrow 1$ **to** N **do**

  Fit GP model $\hat{f}$ to data D

  Suggest next candidate $\boldsymbol{x} \in \arg\max_{\hat{\boldsymbol{x}} \in \mathcal{X}} \alpha(\hat{\boldsymbol{x}}, D, q_\theta$ or $\hat{f})$

  Update history with response $D \leftarrow D \cup \{(\boldsymbol{x}, f(\boldsymbol{x}))\}$

end

Return top performing configuration: $\arg\max_{(\boldsymbol{x}_i, y_i) \in D} y_i$

A Search spaces split

We split search spaces so that we have similar algorithms within the same group, therefore procuring minimal overlap between search spaces. For choosing the prior with the validation dataset, we use 5527, 5891, 5906, 5971, 6767, 6766, 5860. The test search spaces are 5860, 5906, 5965, 5970, 5971, 6766, 6767, 6794.

B Details on the HEBO-inspired Prior

For our final model, not the one used for the direct comparison with HEBO on artificial data, we made some adaptions to the HEBO prior. We do not use a linear kernel and use the following priors, found by through BO on the validation search spaces of HPO-B, as described in Section 3.2. We use $Gamma(concentration, rate)$ distributions. For the outputscale we use $Gamma(0.8452, 0.3993)$ and for the lengthscale we use $Gamma(1.2107, 1.5212)$. For the noise we follow the original kernel exactly with $\log(\epsilon) \sim \mathcal{N}(-4.63, 0.5)$. The warping was performed with $\log$ normal distributions for its concentrations, s.t. $\log(\textit{concentration0}) \sim \mathcal{N}(0, 0.5939)$ and $\log(\textit{concentration1}) \sim \mathcal{N}(0, 0.9722)$.

C Training Details

We only train a single PFN for one particular prior, even though the search spaces can have different numbers of features. To do this, we sample datasets with different numbers of dimensions during PFN training. Following [10], we zero-pad and linearly scale the features when the number of features k is smaller than the maximum number of features K by $\frac{K}{k}$ to make sure the magnitude of the inputs is similar across different numbers of features.

Our PFNs were trained in the standard PFN settings. We use an embedding size of 512 and six layers. Our models were trained with Adam [12] and cosine-annealing [13] without any special tricks. The lr was chosen based on simple grid searches for minimal training loss.

D Riemman Distribution and Acqusition Function

We outline how to compute the $PI(f^*) = \int_{-\infty}^{\infty} [y > f^*] p(y) dy$ for the unbounded Riemann distribution.

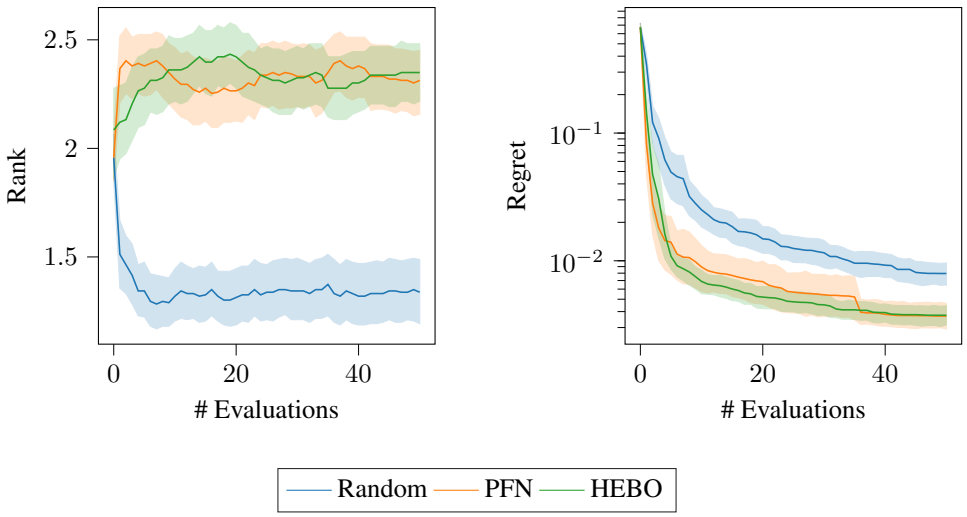


Figure 3: Average rank and regret on the image classification tasks of PD1.

$$\int_{-\infty}^{\infty} [y > f^*] p(y) dy \tag{2}$$

$$= \int_{-\infty}^{y_1} [y > f^*] p(y) dy + \sum_{i=1}^{M} \int_{y_i}^{y_{i+1}} [y > f^*] p(y) dy + \int_{y_{M+1}}^{\infty} [y > f^*] p(y) dy \tag{3}$$

$$= (1 - F_l(y_1 - f^*)) + \sum_{i=1}^{M} \left\{ \begin{array}{ll} (y_{i+1} - f^*) \frac{p(b_i)}{y_{i+1} - y_i} & \text{, if } y_i < f^* < y_{i+1} \\ [y_i \leq f^*] p(b_i) & \text{, else} \end{array} \right\} + F_r(f^* - y_{M+1}) \tag{4}$$

$$= (1 - F_l(y_1 - f^*)) + \sum_{i=1}^{M} (y_{i+1} - min(y_{i+1}, max(f^*, y_i))) \frac{p(b_i)}{y_{i+1} - y_i} + F_r(f^* - y_{M+1}), \tag{5}$$

where F_l (F_r) is the CDF of the half-normal distribution used for the left (right) side. The acquisition function is divided into three terms: (1) a term that governs the probability mass for values lower than interval b_1 and that goes up to values of y_1, (2) a term that summarizes over all intervals $b_1, \ldots, b_M$, and (3) a term that governs the probability mass for values larger than the interval b_M and that start from values of $y_M + 1$.

E Evaluation on PD1

We evaluated on all vision tasks of PD1, which amounts to a total of 17 tasks in the same search space. We make this choice, as these share their evaluation metric: accuracy. For each task we ran 5 seeds with a different (shared) single initial given evaluation. In Figure 3, we can see that even on a completely different benchmark the PFN is very competetive with HEBO.

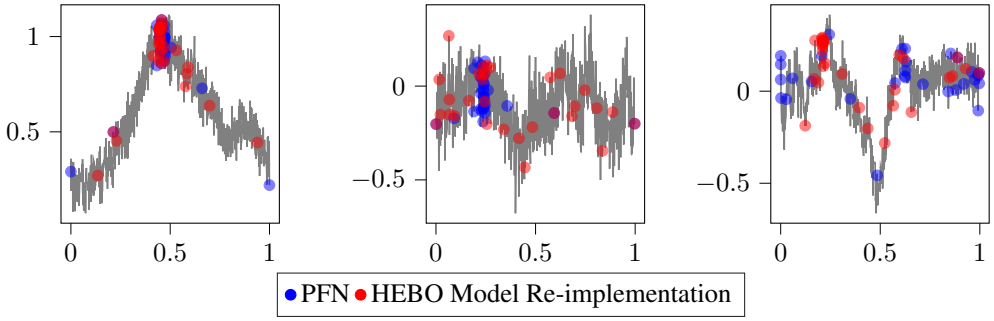


Figure 4: Examples of optimization trajectories on functions sampled from the our HEBO prior. We compare to our re-implementation of HEBO

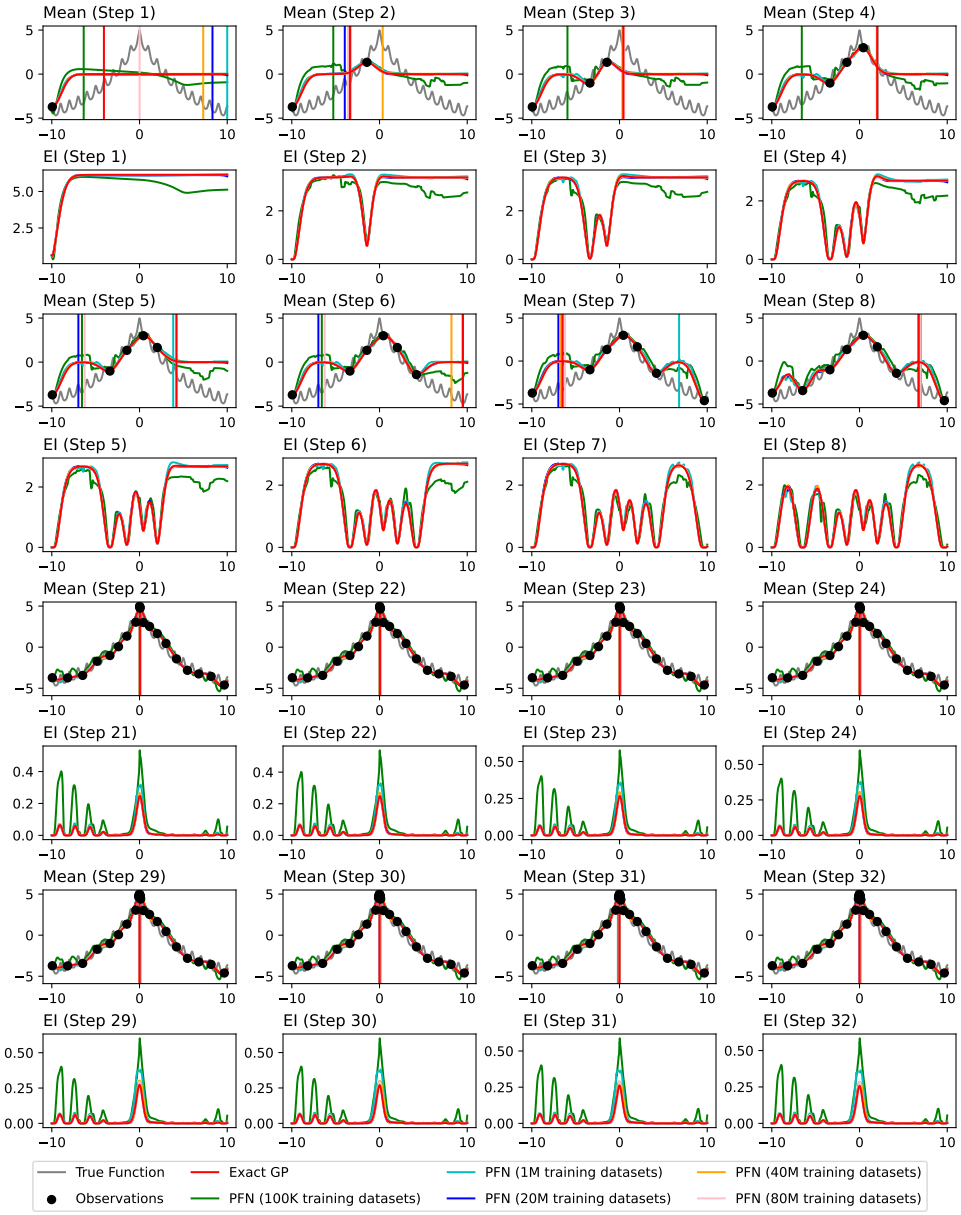


Figure 5: In this figure, we compare models trained on a simple GP prior (with fixed hyper-parameters), thus we can compare to the exact posterior of the GP. We show how PFNs behave differently depending on how much they were trained. Vertical lines mark the maximum acquisition function.

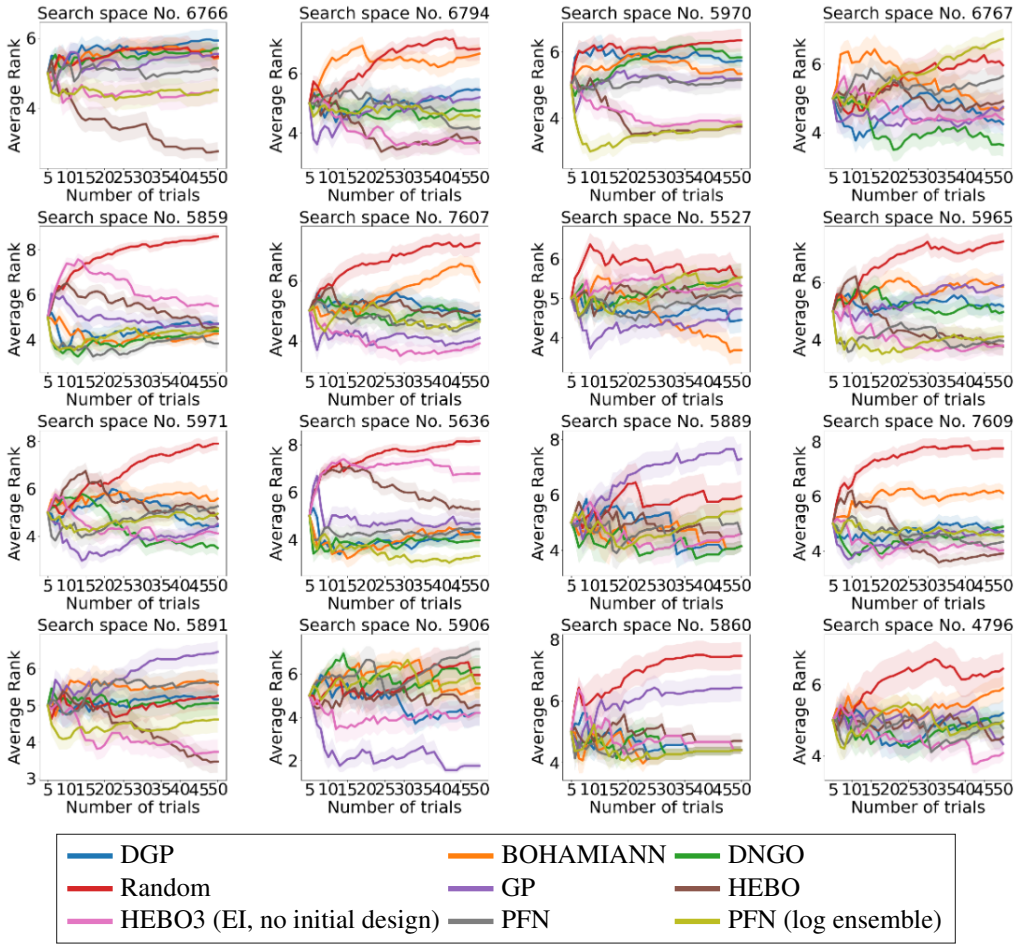


Figure 6: Average Ranks across search spaces. We can see that the PFN generally works well, but also has clear failure modes for some search spaces, which need further investigation.