5th Symposium on Advances in Approximate Bayesian Inference, 2023 1–41

A Study of Bayesian Neural Network Surrogates for Bayesian Optimization

Yucen Lily Li, *New York University* YUCENLI@NYU.EDU
Tim G. J. Rudner, *New York University* TIM.RUDNER@NYU.EDU
Andrew Gordon Wilson, *New York University* ANDREWGW@CIMS.NYU.EDU

Abstract

Bayesian optimization is a highly efficient approach to optimizing objective functions which are expensive to query. These objectives are typically represented by Gaussian process (GP) surrogate models, which are easy to optimize and support exact inference. While standard GP surrogates have been well-established in Bayesian optimization, Bayesian neural networks (BNNs) have recently become practical function approximators, with many benefits over standard GPs, such as the ability to naturally handle non-stationarity and learn representations for high-dimensional data. In this paper, we study BNNs as alternatives to standard GP surrogates for optimization. We consider a variety of approximate inference procedures for finite-width BNNs, including high-quality Hamiltonian Monte Carlo, low-cost stochastic MCMC, and heuristics such as deep ensembles. We also consider infinite-width BNNs and partially stochastic models such as deep kernel learning. We evaluate this collection of surrogate models on diverse problems with varying dimensionality, number of objectives, non-stationarity, and discrete and continuous inputs. We find: (i) the ranking of methods is highly problem dependent, suggesting the need for tailored inductive biases; (ii) HMC is the most successful approximate inference procedure for fully stochastic BNNs; (iii) full stochasticity may be unnecessary as deep kernel learning is relatively competitive; (iv) deep ensembles perform relatively poorly; (v) infinite-width BNNs are particularly promising, especially in high dimensions.

For a full-length version of this paper, see `https://arxiv.org/abs/2305.20028`.

1. Introduction

Bayesian optimization (O'Hagan, 1978) is a distinctly compelling success story of Bayesian inference. In Bayesian optimization, we place a prior over the objective we wish to optimize, and use a *surrogate model* to infer a posterior predictive distribution over the values of the objective at all feasible points in space. We then combine this predictive distribution with an *acquisition function* that trades-off exploration (moving to regions of high uncertainty) and exploitation (moving to regions with a high expected value, for maximization). The resulting approach converges quickly to a global optimum, with strong performance in many expensive black-box settings ranging from experimental design, to learning parameters for simulators, to hyperparameter tuning (Frazier, 2018; Garnett, 2023).

While many acquisition functions have been proposed for Bayesian optimization (e.g. Frazier et al., 2008; Wang and Jegelka, 2017), Gaussian processes (GPs) with standard Matérn or RBF kernels are almost exclusively used by default as a surrogate model for the objective, without checking whether other alternatives would be more appropriate, despite the fundamental role that the surrogate model plays in Bayesian optimization.

Thus, despite promising advances in Bayesian optimization research, there is an elephant in the room: *should we be considering other surrogate models?* It has become particularly timely to evaluate Bayesian neural network (BNN) surrogates as alternatives to Gaussian processes with standard kernels: In recent years, there has been extraordinary progress in making BNNs practical (e.g. Daxberger et al., 2021; Khan and Rue, 2021; Rudner et al., 2022; Tran et al., 2022; Wilson and Izmailov, 2020). Moreover, BNNs can flexibly represent the non-stationary behavior typical of optimization objectives, discover similarity measures as part of representation learning which is useful for higher dimensional inputs, and naturally handle multi-output objectives. In parallel, new *Monte-Carlo* acquisition functions (Balandat et al., 2020) have been developed which only require posterior samples, significantly lowering the barrier to using non-GP surrogates that do not provide closed-form predictive distributions.

In this paper, we exhaustively evaluate Bayesian neural networks as surrogate models for Bayesian optimization. We consider conventional fully stochastic multilayer BNNs with a variety of approximate inference procedures, ranging from high-quality full-batch Hamiltonian Monte Carlo (Izmailov et al., 2021; Neal, 1996, 2010), to stochastic gradient Markov Chain Monte Carlo (Chen et al., 2014), to heuristics such as deep ensembles (Lakshminarayanan et al., 2017). We also consider infinite-width BNNs (Lee et al., 2017; Neal and Neal, 1996), corresponding to GPs with fixed non-stationary kernels derived from a neural network architecture, as well as partially Bayesian last-layer deep kernel learning methods (Wilson et al., 2016). This particularly wide range of neural network-based surrogates allows us to evaluate the role of representation learning, non-stationarity, and stochasticity in modeling Bayesian optimization objectives.

We also extensively study a variety of synthetic and real-world objectives—with a wide range of input space dimensionalities, single- and multi-dimensional output spaces, and both discrete and continuous inputs, and non-stationarities.

Our study provides several key findings: (1) while stochasticity is often prized in Bayesian optimization (Garnett, 2023; Snoek et al., 2012), due to the small data sizes in Bayesian optimization, fully stochastic BNNs do not consistently dominate deep kernel learning, which is not stochastic about network parameters; (2) of the fully stochastic BNNs, HMC generally works the best for Bayesian optimization, and deep ensembles work surprisingly poorly, given their success in other settings; (3) on standard benchmarks, standard GPs are relatively competitive, due to their strong priors and simple exact inference procedures; (4) there is no single method that dominates across most problems, demonstrating that there is significant variability across Bayesian optimization objectives, where tailoring the surrogate to the objective has particular value; (5) infinite-width BNNs are surprisingly effective at high-dimensional optimization, even relative to DKL and stochastic multilayer BNNs. These results suggest that the non-Euclidean similarity metrics constructed from neural networks are valuable for high-dimensional Bayesian optimization, but representation learning (provided by DKL and finite-width BNNs) is not as valuable as a strong prior derived from a neural network architecture (provided by the infinite-width BNN).

This study also serves as an evaluation framework for considering alternative surrogate models for Bayesian optimization.

Our code is available at `https://github.com/yucenli/bnn-bo`.

2. Preliminaries and Related Work

We discuss related work in Appendix 1. This paper provides several key contributions in the context of this prior work, as GP surrogates are nearly always used with Bayesian optimization. While BNN surrogates have been attempted, they have been limited to SGHMC inference, and applied in specialized settings, without an effort to understand their properties. Little is known about whether BNNs could generally be used as an alternative to GPs for Bayesian optimization, especially in light of more recent general advances in BNN research. This is the first paper to provide a comprehensive study of BNN surrogates, considering a range of approximate inference procedures and experimental settings. We test the utility of BNNs in a variety of contexts, exploring the behavior as we change the dimensionality of the problem and the number of objectives, investigating their performance on non-stationary functions, and also incorporating problems with a mix of discrete and continuous input parameters. Rather than champion a specific approach, we provide an objective assessment, also highlighting the continued benefits of GP surrogates for general Bayesian optimization.

We consider a wide variety of surrogate models, helping to separately understand the role of stochasticity, representation learning, and strong priors for surrogate models. We provide additional background on these surrogates and Bayesian optimization in Appendix 2, and we also conduct a sensitivity study into the effect of key design choices such as network architecture and approximate inference procedure for BNN surrogate models in Appendix 5.1.

3. Empirical Evaluation

We provide an extensive empirical evaluation of BNN surrogates for Bayesian optimization. We first assess how BNNs compare to GPs in relatively simple and well-understood settings through commonly used synthetic objective functions, and we perform an empirical comparison between GPs and different types of BNNs (HMC, SGHMC, LLA, ENSEMBLE, I-BNN, and DKL). To further ascertain whether BNNs may be a suitable alternative to GPs in real-world Bayesian optimization problems, we additionally study six real-world datasets. We also perform a case study of Bayesian optimization tasks where simple Gaussian process models may fail but BNN models would be expected to prevail. We conclude our evaluation by analyzing the impacts of kernel selection and hyperparameter optimization on GP surrogate models. We use Monte-Carlo based Expected Improvement (Balandat et al., 2020) as our acquisition function for all problems.

3.1. Synthetic Benchmarks

We evaluate BNN and GP surrogates on a variety of synthetic benchmarks described in Appendix 3.1, and the results can be found in Appendix 5.2.

3.2. Real-World Benchmarks

To provide an evaluation of BNN surrogates in more realistic optimization problems, we consider a diverse selection of real-world applications which span a variety of domains, such as solving differential equations and monitoring cellular network coverage (Dreifuerst et al., 2021; Eriksson et al., 2019; Maddox et al., 2021; Oh et al., 2019; Wang et al., 2020). Many of these problems, such as the development of materials to clean oil spills, have consequential

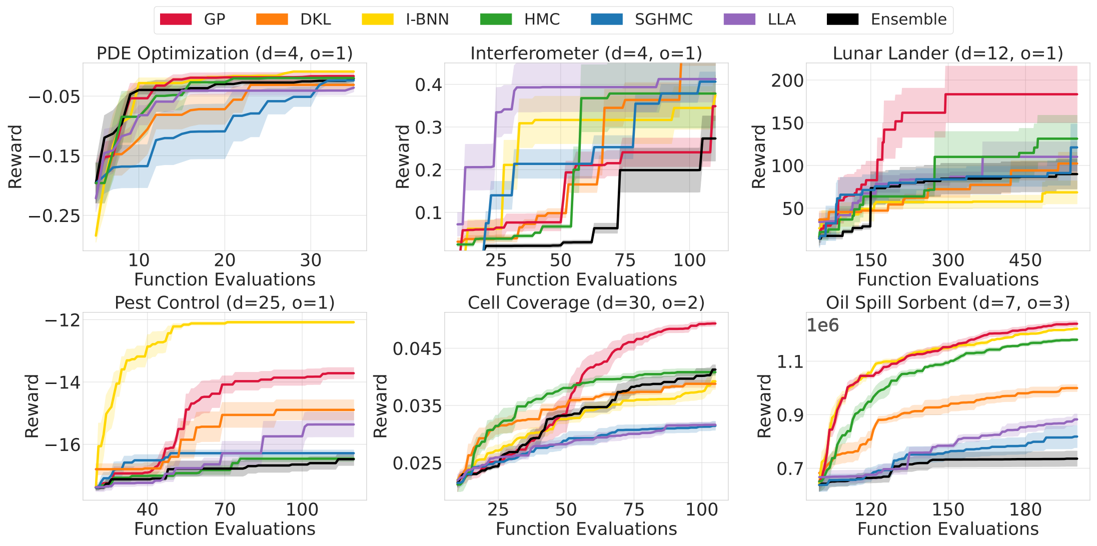


Figure 1: **Real world benchmarks show mixed results**. BNNs outperform GPs on some problems and underperform on others, and there does not seem to be a noticeable preference for any particular surrogate as we increase the number of input dimensions. Additionally, there is not a clear distinction between the top row of experiments, which optimize over continuous parameters, and the bottom row of experiments, which include discrete inputs. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.

applications; however, these objectives are often multi-modal and are difficult to optimize globally. Additionally, unlike the synthetic benchmarks, many real-world applications consist of input data with ordinal or categorical values, which may be difficult for GPs to handle. Several of the problems also require multiple objectives to be optimized. Detailed problem descriptions are provided in Appendix 3.

We share the results of our experiments in Figure 1, and details about the experiment setup can be found in Appendix 4. The results are mixed: BNNs are able to significantly outperform GPs in the Pest Control dataset, while GPs find the maximum reward in the Cell Coverage and Lunar Lander experiments. The Pest Control, Cell Coverage, and Oil Spill Sorbent experiments all include discrete input parameters, and there seems to be a slight trend of GP and I-BNNs performing well, and SGHMC and ENSEMBLEs performing more poorly. Similar to the findings from the synthetic benchmarks, we see that the different approximate inference methods for finite-width BNNs lead to significantly different Bayesian optimization performance, with HMC generally finding higher rewards compared to SGHMC, LLA, and deep ensembles. Additionally, it appears that GPs perform well in the two multi-objective problems, although that may not be generalizable to additional multi-objective problems and may be more related to the curvature of the specific problem space.

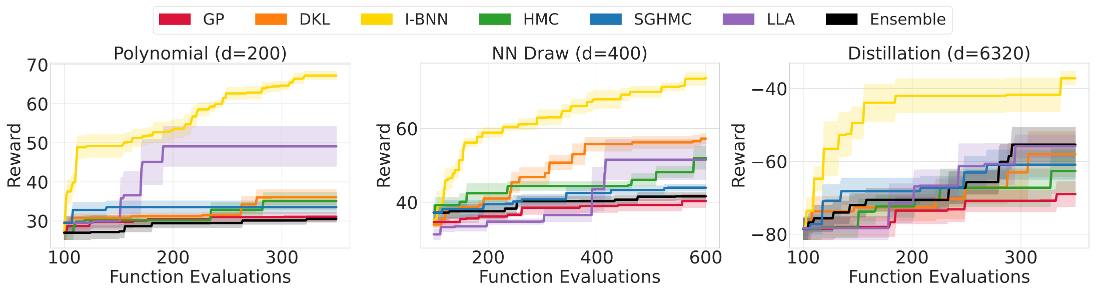


Figure 2: **I-BNNs outperform other surrogates in many high-dimensional settings**. We show the results of three different experiments with high-dimensional and non-stationary objectives, and we find that BNNs consistently find higher rewards than GPs across all problems. We plot the mean and one standard error of the mean over 5 trials, and d corresponds to the number of input dimensions.

3.3. Limitations of GP Surrogate Models

Although popular, GPs suffer from well-known limitations that directly impact their usefulness as surrogate models. To contrast BNN and GP surrogates, we explore failure modes of GPs and demonstrate that the increased flexibility provided by BNN surrogate models can overcome these issues and improve performance in Bayesian optimization.

Non-Stationary Objective Functions. In Appendix 5.8, we discuss the limits of GPs for non-stationary objective functions and compare the performance of Bayesian optimization with BNN and GP surrogate models in this setting.

High-Dimensional Input Spaces. Due to the curse of dimensionality, GPs do not scale well to high-dimensional input spaces without careful human intervention. Common covariance functions may fail to faithfully represent high-dimensional input data, making the design of custom-tailored kernel functions necessary. In contrast, neural networks are well-suited for modeling high-dimensional input data (Krizhevsky et al., 2012).

To measure the effect of dimensionality on the performance of GPs and BNNs, we first benchmark the ability of the surrogate models to maximize synthetic test functions provided by high-dimensional polynomial functions and function draws from neural networks. To better understand the behaviors in real-world settings, we also construct a high-dimensional problem by using Bayesian optimization to set the parameters of a neural network in the context of knowledge distillation. Knowledge distillation refers to the act of "distilling" information from a larger teacher model to a smaller student model by matching model outputs (Hinton et al., 2015), and it is known to be a difficult optimization problem (Stanton et al., 2021). For full descriptions of the high-dimensional problems, see Appendix 3.

We share the results of our findings in Figure 2 and Appendix 5.4. We see I-BNNs clearly stand out for their good performance in these high-dimensional settings. The I-BNN has several appealing features in this setting: (1) it provides a non-Euclidean and non-stationary similarity metric, which can be particularly valuable in high-dimensions; (2) it does not have any hyperparameters for learning, and thus is not "data hungry" — which is especially important in high dimensional problems with small data sizes, since these settings provide relatively little information for representation learning.

Additionally, we find that other BNN surrogate models also outperform GPs across the high-dimensional problems, providing a compelling motivation for BNNs as surrogate models for Bayesian optimization.

3.4. Further Empirical Results

We present further empirical results in the appendix, including the following findings:

- The design choices of BNNs greatly impact their performance, and neural architecture search can improve the behavior of HMC (Appendix 4.4).
- Deep ensembles may underperform on Bayesian optimization problems due to the low diversity between models in the small-data setting (Appendix 5.3).
- I-BNNs, which can handle non-stationary functions and use non-Euclidean distance metrics, consistently outperform GPs in high dimensions (Appendix 5.4).
- The optimal combination of hyperparameter selection method and kernel choice for GP surrogates is problem-dependent (Appendix 5.5).

4. Discussion

While Bayesian optimization research has made significant progress over the last few decades (Garnett, 2023), the surrogate model is a crucial and highly underexplored design choice. Although standard GP models are the default surrogate, it is not because they have been shown to be superior to alternatives—we simply have had almost no evidence about how alternatives would perform. It is particularly timely to consider neural network surrogates, given significant recent advances in Bayesian neural networks and related approaches. The setting of Bayesian optimization is also quite different from where BNNs are typically applied, involving particularly small datasets and online learning.

The fact that DKL is competitive with BNNs calls into question the importance of fully stochastic surrogates—a surprising finding given the small datasets common to Bayesian optimization, where overfitting would be a more significant concern. Moreover, I-BNNs show promising performance in general, but especially for higher dimensional settings. Given that they also do not involve learning many hyperparameters, and do not require approximate inference, it is possible they could become a standard surrogate for Bayesian optimization.

We also show that the Bayesian optimization objectives are sufficiently different such that one method does not generally dominate. This finding supports the use of simple models with strong but generic assumptions, such as standard GP models, which indeed provide relatively competitive performance. On the other hand, perhaps it is self-fulfilling that standard GPs would be competitive on standard benchmarks, since most problems were designed without alternative models in mind. It may be time for the community to consider new benchmarks that evolve with the advances in our choice of surrogate models.

Acknowledgements

We thank Greg Benton for helpful guidance in the beginning stages of this research and Sanyam Kapoor for discussions. This work is supported by NSF CAREER IIS-2145492, NSF I-DISRE 193471, NIH R01DA048764-01A1, NSF IIS-1910266, NSF 1922658 NRT-HDR, Meta Core Data Science, Google AI Research, BigHat Biosciences, Capital One, and an Amazon Research Award.

References

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

Tianqi Chen, Emily B Fox, Carlos Guestrin, and Michael I Jordan. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning*, pages 1683–1691, 2014.

Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 2021.

Ryan M Dreifuerst, Samuel Daulton, Yuchen Qian, Paul Varkey, Maximilian Balandat, Sanjay Kasturia, Anoop Tomar, Ali Yazdan, Vish Ponnampalam, and Robert W Heath. Optimizing coverage and capacity in cellular networks using machine learning. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8138–8142. IEEE, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

Peter I Frazier. A tutorial on bayesian optimization. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Peter I Frazier, Warren B Powell, and Savas Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 2008.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for bayesian optimization. In *Artificial Intelligence and Statistics*, 2017.

Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, May 2013. ISSN 1532-4435.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

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.

Pavel Izmailov, Sharad Vikram, Matthew D Hoffman, and Andrew Gordon Gordon Wilson. What are bayesian neural network posteriors really like? In *International conference on machine learning*, pages 4629–4640. PMLR, 2021.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International conference on machine learning*, 2015.

Mohammad Emtiyaz Khan and Håvard Rue. The bayesian learning rule. *arXiv preprint arXiv:2107.04562*, 2021.

Samuel Kim, Peter Y Lu, Charlotte Loh, Jamie Smith, Jasper Snoek, and Marin Soljacic. Deep learning for bayesian optimization of scientific problems with high-dimensional structure. *Transactions of Machine Learning Research*, 2021.

Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, and Vincent Fortuin. Promises and pitfalls of the linearized laplace in bayesian optimization. *arXiv preprint arXiv:2304.08309*, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25:*, pages 1106–1114, 2012.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

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.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Wesley J Maddox, Maximilian Balandat, Andrew G Wilson, and Eytan Bakshy. Bayesian optimization with high-dimensional outputs. *Advances in Neural Information Processing Systems*, 34:19274–19287, 2021.

Henry Moss, David Leslie, Daniel Beck, Javier Gonzalez, and Paul Rayson. Boss: Bayesian optimization over string spaces. *Advances in Neural Information Processing Systems*, 2020.

Radford M Neal. Bayesian Learning for Neural Networks. 1996.

Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162, 2010.

Radford M Neal and Radford M Neal. Priors for infinite networks. *Bayesian learning for neural networks*, pages 29–53, 1996.

Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. Combinatorial bayesian optimization using the graph cartesian product. *Advances in Neural Information Processing Systems*, 32, 2019.

A O'Hagan. On curve fitting and optimal design for regression. *J. Royal Stat. Soc. B*, 40: 1–32, 1978.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems 32*. 2019.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.

Tim G. J. Rudner, Freddie Bickford Smith, Qixuan Feng, Yee Whye Teh, and Yarin Gal. Continual Learning via Sequential Function-Space Variational Inference. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2022.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

Dmitry Sorokin, Alexander Ulanov, Ekaterina Sazhina, and Alexander Lvovsky. Interferobot: aligning an optical interferometer by a reinforcement learning agent. *Advances in Neural Information Processing Systems*, 33:13238–13248, 2020.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29, 2016.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew G Wilson. Does knowledge distillation really work? *Advances in Neural Information Processing Systems*, 34:6906–6919, 2021.

Samuel Stanton, Wesley Maddox, Nate Gruver, Phillip Maffettone, Emily Delaney, Peyton Greenside, and Andrew Gordon Wilson. Accelerating Bayesian optimization for biological sequence design with denoising autoencoders. In *International Conference on Machine Learning*, 2022.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in neural information processing systems*, 26, 2013.

Dustin Tran, Jeremiah Liu, Michael W. Dusenberry, Du Phan, Mark Collier, Jie Ren, Kehang Han, Zi Wang, Zelda Mariet, Huiyi Hu, Neil Band, Tim G. J. Rudner, Karan Singhal, Zachary Nado, Joost van Amersfoort andAndreas Kirsch, Rodolphe Jenatton, Nithum Thain, Honglin Yuan, Kelly Buchanan, Kevin Murphy, D. Sculley, Yarin Gal, Zoubin Ghahramani, Jasper Snoek, and Balaji Lakshminarayanan. Plex: Towards Reliability Using Pretrained Large Model Extensions. 2022.

Boqian Wang, Jiacheng Cai, Chuangui Liu, Jian Yang, and Xianting Ding. Harnessing a novel machine-learning-assisted evolutionary algorithm to co-optimize three characteristics of an electrospun oil sorbent. *ACS Applied Materials & Interfaces*, 12(38):42842–42849, 2020.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, 2016.

Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. Bayesian optimization with gradients. *Advances in neural information processing systems*, 30, 2017.

Appendix

A Study of Bayesian Neural Network Surrogates for Bayesian Optimization

Code for our evaluation framework is available at

https://github.com/yucenli/bnn-bo

Table of Contents

1. Related Work

There is a large body of literature on improving the performance of Bayesian optimization. However, an overwhelming majority of this research only considers Gaussian process surrogate models, focusing on developing new acquisition functions (e.g. Frazier et al., 2008; Wang and Jegelka, 2017), additive covariance functions (Gardner et al., 2017; Kandasamy et al., 2015), the inclusion of gradient information (Wu et al., 2017), multi-objectives (Swersky et al., 2013), trust region methods that use input partitioning for higher dimensional and non-stationary data (Eriksson et al., 2019), and covariance functions for discrete inputs and strings (Moss et al., 2020). For a comprehensive review, see Garnett (2023).

There has been some prior work focusing on other types of surrogate models for Bayesian optimization, such as random forests (Hutter et al., 2011) and tree-structured Parzen estimators (Bergstra et al., 2013). Moreover, Snoek et al. (2015) apply a Bayesian linear regression model to the last layer of a deterministic neural network for computational scaling, which can be helpful for the added number of objective queries associated with higher dimensional inputs. Deep kernel learning (Wilson et al., 2016), which transforms the inputs of a Gaussian process kernel with a deterministic neural network, is sometimes also used with Bayesian optimization, especially in specialized applications like protein engineering (Stanton et al., 2022). Additionally, the linearized-Laplace approximation to produce a linear model from a neural network has recently been applied to Bayesian optimization in concurrent work (Kristiadi et al., 2023).

Despite the recent practical advances in developing Bayesian neural networks for many tasks (e.g. Wilson and Izmailov, 2020), and recent Monte-Carlo acquisition functions which make it easier to use surrogates like BNNs that do not provide closed-form predictive distributions (Balandat et al., 2020), there is a vanishingly small body of work that considers BNNs as surrogates for Bayesian optimization. This is surprising, since we would indeed expect BNNs to have properties naturally aligned with Bayesian optimization, such as the ability to learn non-stationary functions without explicit modeling interventions and gracefully handle high-dimensional input and output spaces.

The possible first attempt to use a Bayesian neural network surrogate for Bayesian optimization (Springenberg et al., 2016) came before most of these advances in BNN research, and used a form of stochastic gradient Hamiltonian Monte Carlo (SGHMC) (Chen et al., 2014) for inference. Like Snoek et al. (2015), the focus was largely on scalability advantages over Gaussian processes; however, the reported performance gains were marginal, and puzzling in that they were largest for a *small* number of objective function queries (where the neural net would not be able to learn a rich representation, and scalability would not be required). Kim et al. (2021) used the same method for BNNs with Bayesian optimization, also with SGHMC, targeted at scientific problems with known structures and high dimensionality. In these applications, BNNs leverage auxiliary information, domain knowledge, and intermediate data, which would not typically be available in many Bayesian optimization problems.

Our paper provides several key contributions in the context of this prior work, where standard GP surrogates are nearly always used with Bayesian optimization. While finite-width BNN surrogates have been attempted, they have been largely limited to SGHMC inference, and are often applied in specialized settings without an effort to understand their properties. Little is known about whether BNNs could generally be used as an alternative to GPs for Bayesian optimization, especially in light of more recent general advances in BNN research. This is the first paper to provide a comprehensive study of BNN surrogates, considering a range of model types and experimental settings. We test the utility of BNNs in a variety of contexts, exploring their behavior as we change the dimensionality of the problem and the number of objectives, investigating their performance on non-stationary functions, and also incorporating problems with a mix of discrete and continuous input parameters. Moreover, we are the first to study infinite BNN models in Bayesian optimization, and to consider the role of stochasticity and representation learning in neural network based Bayesian optimization surrogates. Finally, rather than champion a specific approach, we provide an objective assessment, also highlighting the benefits of GP surrogates for general Bayesian optimization problems.

2. Background

2.1. Bayesian Optimization

Bayesian optimization is a global optimization method commonly used for black-box functions which are costly to evaluate. Specifically, we formulate the optimization as a maximization problem where we want to solve $\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$, where $\mathbf{x}$ represents all possible inputs to objective function f, using a limited number of function queries. In Bayesian optimization, we iteratively select new function evaluations using the following procedure: (1) use a *surrogate model* to find $p(f|\mathcal{D})$, where $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^t$ and y_i is a noisy observation of $f(\mathbf{x}_i)$; (2) select the next $\mathbf{x}_{t+1}$ to query by maximizing an *acquisition function*, which is an inexpensive computation that typically uses the mean and variance of the posterior $p(f|\mathcal{D})$ to compute how useful a function evaluation would be; and (3) query the function at value $\mathbf{x}_{t+1}$ to observe $y_{t+1} \sim \mathcal{N}(f(\mathbf{x}_{t+1}), \sigma^2_{\text{obs}})$ and add the result to the dataset $\mathcal{D} := \mathcal{D} \cup \{\mathbf{x}_{t+1}, y_{t+1}\}$.

Historically, Bayesian optimization has seen wide success in a range of fields including drug design, engineering challenges, and hyperparameter optimization. A more recent and growing area of interest is in higher dimensional settings, where the objective, $f(x)$ is multi-dimensional consisting of several tasks we wish to optimize jointly. In multi-objective Bayesian optimization, we want to find the input $\mathbf{x}^*$ that maximizes k related objective functions $\mathcal{F} = \{f_1(\mathbf{x}), \cdots, f_k(\mathbf{x})\}$. There is typically no single solution that maximizes all K objectives simultaneously. Therefore, performance is typically compared using *Pareto dominance*, where one solution Pareto-dominates another if it performs equally well or better on all objectives. Pareto dominance can be measured using *hypervolume*, which indicates how much of a bounded region is dominated by a solution.

Core to the Bayesian optimization procedure is the use of an acquisition function to select the next candidate points given a set of observations $x_i, f(x_i)_{i=1}^N$, and a surrogate model trained to fit these observations. Acquisition functions are functions of the predictive distribution of the surrogate model, and seek candidate points according to criteria such as the probability of improvement or the expected improvement over the currently found minimizer of the objective.

The chosen surrogate model will also have a significant impact on the performance of Bayesian optimization. Since the true function f may take a variety of forms, the surrogate model should be flexible enough to accurately represent it. Additionally, the uncertainty estimates that the surrogate model provides should be well-calibrated to ensure that the input that maximizes f can be found in a limited number of iterations.

2.2. Gaussian Processes

Gaussian processes (GPs) (Rasmussen and Williams, 2006) are distributions over functions entirely specified by a mean function $\mu(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$. For regression tasks with GPs, we assume $y(\mathbf{x}) = f(\mathbf{x}) + \epsilon$ with $f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$, where σ^2 is the likelihood variance. The functions $\mu(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ are the mean and kernel functions of the GP that govern the functional properties and generalization performance of the model. In practice, we tune the hyperparameters of $\mu(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ on the training data to optimize the *marginal likelihood* of the GP, which maximizes the probability that the GP model will have generated the data.

By the definition of a GP, for any finite collection of inputs $\mathbf{x} = [\mathbf{x}_{\text{train}}, \mathbf{x}_{\text{test}}]$, $f(\mathbf{x})$, and thus $y(\mathbf{x})$ are jointly normal. Therefore, we can apply the rules of conditioning partitioned multivariate normals to form a posterior distribution $p\left(f(\mathbf{x}_{\text{test}}) \mid y(\mathbf{x}_{\text{train}})\right)$, which will also be Gaussian.

In the context of Bayesian optimization, this simple conditioning procedure means that given some set of points at which we have already queried the objective function, we can quickly generate a posterior over potential candidate points and, with the help of an acquisition function, select the next points to query the objective.

Throughout our experiments, when we refer to Gaussian processes, we always mean *standard* Gaussian processes, with the Matérn-5/2 kernel that is typically used in Bayesian optimization (Snoek et al., 2012). These Gaussian processes have the advantage of simple exact inference procedures, strong priors, and only a few hyperparameters, such as length-scale, which controls rate of variability. On the other hand, these models are *stationary*, meaning the covariance function is translation invariant and models the objective as having similar properties (such as rate of variation) at different points in input space. They also provide a similarity metric for data points based on simple Euclidean distance of inputs, which is often not suitable for higher dimensional input spaces.

2.3. Bayesian Neural Networks

Bayesian neural networks (BNNs) are neural networks with stochastic parameters for which a posterior distribution is inferred using Bayesian inference.

More specifically, for regression tasks, we can specify a Gaussian observation model, $p(y(\mathbf{x}) \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(h(\mathbf{x} \mid \boldsymbol{\theta}), \sigma^2)$, where $y(\mathbf{x})$ represents a noisy observed value and $h(\mathbf{x} \mid \boldsymbol{\theta})$ represents the output of a neural network with parameter realization $\boldsymbol{\theta}$ for input $\mathbf{x}$. For BNNs, a prior distribution is placed over the stochastic parameters $\boldsymbol{\theta}$ of the neural network, and the corresponding posterior distribution is given by $p(\boldsymbol{\theta} \mid \mathcal{D}) = p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. This posterior distribution can then be used in combination with the acquisition function for Bayesian optimization to select the next set of candidate points to query.

There are many choices to consider when using Bayesian neural networks, such the inference method used to compute the posterior distribution, the architecture of the neural network, and the selection of which parameters are stochastic. In this work, we study the performance of five different types of BNNs with varying inference methods and stochasticity.

2.3.1. Fully-Stochastic Finite-Width Neural Networks

For fully-stochastic Bayesian neural networks, every parameter of the neural network is stochastic and has a prior placed over it. Exact inference over these stochastic network parameters for fully-stochastic finite-width BNNs is analytically intractable, since neural networks are non-linear in their parameters. To enable Bayesian inference in this setting, approximate inference methods can be used to find approximations to the exact posterior $p(\boldsymbol{\theta} \,|\, \mathcal{D})$. In this work, we focus on three types of approximate inference: Hamiltonian Monte Carlo (HMC; Neal (2010)), stochastic-gradient HMC (Chen et al., 2014), and deep ensembles of deterministic neural networks (Lakshminarayanan et al., 2017). These approaches are fully stochastic, and can do representation learning, meaning that they can learn appropriate distance metrics for the data as well as particular types of non-stationarity.

Hamiltonian Monte Carlo. Hamiltonian Monte Carlo is a Markov Chain Monte Carlo method that produces asymptotically exact samples from the posterior distribution (Neal, 2010) and is commonly referred to as the "gold standard" for inference in BNNs. At test time, HMC approximates the integral $p(y(\mathbf{x}_{\text{test}}) \,|\, \mathcal{D}) \approx \frac{1}{M} \sum_{i=1}^{M} p(y(\mathbf{x}_{\text{test}}) \,|\, \boldsymbol{\theta}_i)$ using samples $\boldsymbol{\theta}_i$ drawn from the posterior over parameters.

Full-batch HMC provides the most accurate approximation of the posterior distribution but is computationally expensive and in practice limited to models with only a few hundred thousand parameters (Izmailov et al., 2021). Full-batch HMC allows us to study the performance of BNNs in Bayesian optimization without many of the confounding factors of inaccurate approximations of the predictive distribution.

Stochastic Gradient Hamiltonian Monte Carlo. Stochastic gradient methods (Welling and Teh, 2011; Hoffman et al., 2013) are commonly used as an inexpensive approach to sampling from the posterior distribution. Unlike full-batch HMC, which computes the gradients over the full dataset, stochastic gradient HMC instead samples a mini-batch to compute a noisy estimate of the gradient (Chen et al., 2014). While these methods may seem appealing when working with large models or datasets, they can result in inaccurate approximations and posterior predictive distribution with unfaithful uncertainty representations.

Deep Ensembles. Deep ensembles are ensembles of several deterministic neural networks, each trained using maximum a posteriori estimation using a different random seed and initialization (and sometimes using different subsets of the training data). The ensemble components can be viewed as forming an efficient approximation to the posterior predictive distribution, by choosing parameters that represent typical points in the posterior and have high functional variability (Wilson and Izmailov, 2020). Deep ensembles are easy to implement in practice and have been shown to provide highly accurate predictions and a good predictive uncertainty estimate (Lakshminarayanan et al., 2017; Ovadia et al., 2019).

2.3.2. Fully-Stochastic Infinite-Width Neural Networks

Infinitely-wide neural networks (Neal and Neal, 1996) refer to the behavior of neural networks when the number of nodes in each internal layer increases to infinity. Each one of these nodes continues is stochastic and has a specific prior placed over its parameters.

Infinite-width Neural Network. Infinite-width neural networks (I-BNNs) refer to the behavior of neural networks as the number of nodes per hidden layer increases to infinity. Neal (1996) famously showed with a central limit theorem argument that a BNN with a single infinite-width hidden layer converges to a GP with a neural network covariance function, and this result has been extended to deep neural networks by Lee et al. (2017). I-BNNs are fully stochastic and very different from standard GPs, as they can handle non-stationarity and provide a non-Euclidean notion of distance inspired by a neural network. However, it cannot do representation learning and instead has a fixed covariance function that provides a relatively strong prior.

2.3.3. Linearized Finite-Width Neural Networks

Linearized Laplace Approximation. The linearized-Laplace approximation (LLA) is a deterministic approximate inference method that uses the Laplace approximation (MacKay, 1992; Immer et al., 2021) to produce a linear model from a neural network, and has recently been considered for Bayesian optimization in concurrent work (Kristiadi et al., 2023).

2.3.4. Partially-Stochastic Finite-Width Neural Networks

For partially-stochastic neural networks, we learn point estimates for a subset of the parameters, and we learn the full posterior distribution for the remaining parameters in the neural network.

Deep Kernel Learning. Deep Kernel Learning (DKL) is a hybrid method that combines the flexibility of GPs with the expressivity of neural networks (Wilson et al., 2016). In DKL, a neural network $g_{\mathbf{w}}(\cdot)$ parameterized by weights $\mathbf{w}$ is used to transform inputs into intermediate values, where additional GP kernels can then be applied. Specifically, given inputs $\mathbf{x}$ and a base kernel $k_{\text{BASE}}(\mathbf{x}, \mathbf{x}')$, $k_{\text{DKL}}(\mathbf{x}, \mathbf{x}') = k_{\text{BASE}}(g_{\mathbf{w}}(\mathbf{x}), g_{\mathbf{w}}(\mathbf{x}'))$. Unlike GPs with standard kernels which depend only on the distance between inputs and therefore assume that the mean and smoothness of the function are consistent throughout, DKL does not assume stationarity and is able to represent how the properties of the function change due to its neural network feature extractor. This approach can do non-Euclidean representation learning, handle non-stationarity, and also uses exact inference.

3. Problem Descriptions

3.1. Synthetic Datasets

In the single-objective case, **Branin** is a function with two-dimensional inputs with three global minima, **Hartmann** is a six-dimensional function with six local minima, and **Ackley** is a multi-dimensional function with many local minima which is commonly used to test optimization algorithms. We convert all problems to maximization problems, and the goal in single-objective settings is to find the maximum value of the objective function.

For multi-objective Bayesian optimization benchmarks, **BraninCurrin** is a two-dimensional input and two-objective problem composed of the Branin and Currin functions, and **DTLZ1** and **DTLZ5** are multi-dimensional and multi-objective test functions which are used to measure an optimization algorithm's ability to converge to the Pareto-frontier.

Here, the goal is to find an input corresponding to the multi-dimensional objective value with the maximum hypervolume from a problem-specific reference point.

3.2. Real-World Datasets

Interferometer In this problem, the goal is to tune an optical interferometer through the alignment of two mirrors. We use the simulator in Sorokin et al. (2020) to replicate the Bayesian optimization problem in Maddox et al. (2021). Each mirror has continuous x and y coordinates, which should be optimized so the mirrors reflect light with minimal interference.

Lunar Lander Following Eriksson et al. (2019), we aim to learn the parameters of a controller for a lunar lander as implemented in OpenAI gym. The controller has 12 continuous input dimensions, corresponding to events such as "change the angle of the rover if it is tilted more than x degrees." The objective is to maximize the average final reward over 50 randomly generated environments.

Cellular Coverage We want to optimize the cellular network coverage and capacity from 15 cell towers (Dreifuerst et al., 2021). Each tower has a continuous parameter corresponding to transmit power parameter and an ordinal parameter with 6 values corresponding to tilt, for a total of 15 continuous and 15 ordinal input parameters. There are two different objectives: maximize the cellular coverage, and minimize the interference between cell towers.

Oil Spill Sorbent In this problem, we optimize the properties of a material to maximize its performance as a sorbent for marine oil spills (Wang et al., 2020). We tune 5 ordinal parameters and 2 continuous parameters which control the manufacturing process of electrospun materials, and optimize over three objectives: water contact angle, oil absorption capacity, and mechanical strength.

Pest Control Minimizing the spread of pests while minimizing the prevention costs of treatment is an important problem with many parallels (Oh et al., 2019). In this experiment, we define the setting as a categorical optimization problem with 25 categorical variables corresponding to stages of intervention, with 5 different values at each stage. We optimize over two objectives: minimizing the spread of pests and minimizing the cost of prevention.

PDE Optimization Here, following Maddox et al. (2021), we optimize 4 continuous parameters corresponding to the diffusivity and reaction rates of a Brusselator with spatial coupling. The objective of the problem is to minimize the variance of the PDE output.

3.3. High-Dimensional Problems

Polynomial In this optimization problem, the goal is to find the maximum value of a polynomial function. For input $\mathbf{x} \in \mathbb{R}^d$, the objective value is calculated using $\sum_{i=1}^{d/4} \prod_{j=1}^{4} (\mathbf{x}_{i+j} - c_{i+j})$, where $c_i \sim \text{Normal}(0, 1)$. We set the boundaries of the input space to be $[0, 1]^d$, and this problem setup can be used for any number of dimensions d.

Neural Network Draw We want to find the maximum value of a function specified by a draw from a neural network. For the neural network, we use an MLP with d inputs connected to 2 hidden layers of 256 nodes each with tanh activation, connected to a final layer of size 1 (unless otherwise specified). We set each parameter w in the network to a value drawn from $\mathcal{N}(0, 1)$. The final objective function is equivalent to the output of the neural network with the specified weights. With this setup, we can vary d to alter the input dimensionality of the problem.

Knowledge Distillation The goal of knowledge distillation is to use a larger teacher model to train a smaller student model, typically done by matching the teacher and student predictive distributions. In this problem, we use Bayesian optimization to determine the optimal parameters of the student neural network by setting our objective function as the KL divergence between the teacher and student predictive distributions. Knowledge distillation is known to be a difficult optimization problem, and this is problem also has many more dimensions than typical Bayesian optimization benchmarks. For our experiment, we use the MNIST dataset, and we train a LeNet-5 for our teacher model. For our student model, we use a small CNN with the following architecture:

1. convolutional layer with 16 convolution kernels of 3x3 (ReLU activation)
2. max pool layer 2x2
3. convolutional layer with 12 convolution kernels of 3x3 (ReLU activation)
4. max pool layer 2x2
5. fully connected layer with 10 outputs

4. Experiment Details

4.1. General Setup

For all datasets, we normalize input values to be between [0, 1] and standardize output values to have mean 0 and variance 1. We also use Monte-Carlo based Expected Improvement as our acquisition function.

GP: For single-objective problems, we use GPs with a Matérn 5/2 kernel, adaptive scale, a length-scale prior of $\text{Gamma}(3, 6)$, and an output-scale prior of $\text{Gamma}(2.0, 0.15)$, which combine with the marginal likelihood to form a posterior which we optimize for hyperparameter learning. For multi-objective problems, we use the same GP to independently model each objective.

DKL: We set up the base kernel using the same Matérn 5/2 kernel that we use for GPs. For the feature extractor, we use the same model parameters as found in our full HMC search. For multi-objective problems, we independently model each objective.

I-BNN: We use I-BNNs with 3 hidden layers and the ReLU activation function. We set the variance of the weights to 10.0, and the variance of the bias to 1.3. For multi-objective problems, we independently model each objective.

HMC: We use HMC with an adaptive step size, and we do a small search over different architectures and model choices for each problem. We use MLPs for all experiments, and we first do a grid search over MLP architectures, where we run all combinations of width (64, 128, 256), and depth (2, 3, 4, 5) with the prior and likelihood variance set to 1.0. From the best performing run, we then fix the width and depth of the network and do a grid search over the prior variance (0.1, 1.0, and 10.0) and likelihood variance (0.1, 1.0, and 10.0). We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

SGHMC: We use SGHMC with minibatch size of 5 using the same model parameters as found in our full HMC search. We follow the implementation in Springenberg et al. (2016)

and use scale-adaptive SGHMC with a heteroskedastic likelihood variance as determined by the output of the neural network. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

LLA: We use the same model parameters as found in our HMC search. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

ENSEMBLE: We use an ensemble of 10 models, each with the same architecture as found in the HMC search. Each model is trained on a random 80% of the function evaluations. We model multi-objective problems by setting the number of nodes in the final layer equal to the number of objectives.

We run multiple trials for all experiments, where each trial starts with a different set of initial function evaluations drawn using a Sobol sampler.

4.2. Synthetic Benchmarks

Branin We ran 5 trials using batch size 5 with 10 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 10.0.

Hartmann We ran 5 trials using batch size 10 with 10 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 1.0.

Ackley We ran 5 trials using batch size 10 with 10 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 0.1.

BraninCurrin We ran 5 trials using batch size 10 with 10 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 10.0.

DTLZ1 We ran 5 trials using batch size 5 with 10 initial points. Our BNN models used MLPs with 2 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 10.0.

DTLZ5 We ran 5 trials using batch size 1 with 10 initial points. Our BNN models used MLPs with 5 hidden layers, 10 parameters per layer, tanh activation, likelihood variance of 0.001, and prior variance of 10.0.

4.3. Real-World Benchmarks

PDE Optimization We ran 5 trials using batch size 1 with 5 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 1.0, and prior variance of 1.0.

Interferometer We ran 10 trials using batch size 10 with 10 initial points. Our BNN models used MLPs with 5 hidden layers, 100 parameters per layer, tanh activation, likelihood variance of 10.0, and prior variance of 10.0.

Lunar Lander We ran 5 trials using batch size 50 with 50 initial points. Our BNN models used MLPs with 5 hidden layers, 100 parameters per layer, tanh activation, likelihood variance of 10.0, and prior variance of 10.0.

Pest Control We ran 5 trials using batch size 4 with 20 initial points. Our BNN models used MLPs with 5 hidden layers, 100 parameters per layer, tanh activation, likelihood variance of 10.0, and prior variance of 10.0.

Cell Coverage We ran 5 trials using batch size 5 with 10 initial points. Our BNN models used MLPs with 3 hidden layers, 128 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 0.1.

Oil Spill Sorbent We ran 5 trials using batch size 10 with 10 initial points. Our BNN models used MLPs with 5 hidden layers, 100 parameters per layer, tanh activation, likelihood variance of 0.1, and prior variance of 10.0.

4.4. Neural Architecture Search

We used SMAC (Lindauer et al., 2022) to find the optimal hyperparameters of HMC for Cell Coverage, Pest Control, and DTLZ5 benchmark problems using Bayesian optimization. For each benchmark, our new objective function was the average maximum value found for three runs of Bayesian optimization using the same problem setup as detailed above. We used the hyperparameter optimization facade in SMAC, and for each problem, we used Bayesian optimization to select 200 different HMC architectures to find the optimal combination from the following set of possible values:

- Network width: [32, 512] (continuous)
- Network depth: {1, 2, 3, 4, 5, 6} (discrete)
- Network activation: {"ReLU", "tanh"} (discrete)
- $\log_{10}$ of likelihood variance: [-3.0, 2.0] (continuous)
- $\log_{10}$ of prior variance: [-3.0, 2.0] (continuous)

We then use the optimal architecture and run the benchmark for 5 trials with the same setup as described in Appendix 4.2 and Appendix 4.3 to compare the results of HMC with and without neural architecture search.

For cell coverage, the final HMC model was an MLP with 5 hidden layers, 184 parameters per layer, tanh activation, likelihood variance of 26.3, and prior variance of 0.54.

For pest control, the final HMC model was an MLP with 1 hidden layer of size 321, tanh activation, likelihood variance of 0.26, and prior variance of 0.31.

For DTLZ5, the final HMC model was an MLP with 3 hidden layers, 297 parameters per layer, relu activation, likelihood variance of 0.01, and prior variance of 0.30.

We show the effect of neural architecture search on HMC surrogate models in Figure A.1. On the Cell Coverage problem, the architecture search did not drastically change the performance of HMC. In contrast, extensively optimizing the hyperparameters made a significant difference on the Pest Control problem, leading to HMC finding higher rewards than GPs while using fewer function evaluations; however, on this problem, I-BNN, which does not require specifying an architecture, still performs best. Neural architecture search was also able to improve the results on DTLZ5, leading HMC to be competitive with other surrogate models such as I-BNNs and DKL. The difference in the benefits of the search may be attributed to some problems having less inherent structure than others, where extensive

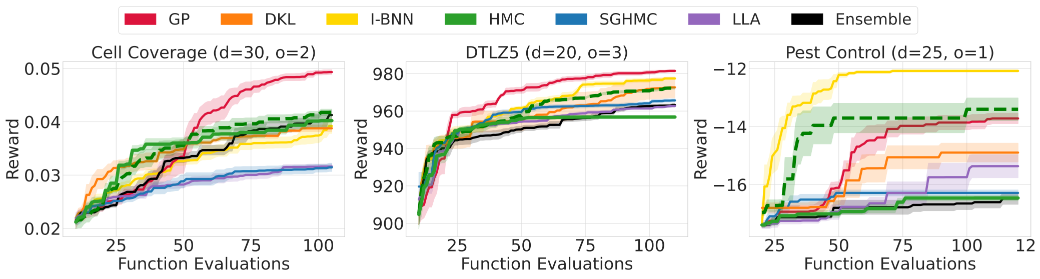


Figure A.1: **The impact of neural architecture search on HMC is problem-dependent.** The dashed green line indicates the performance of HMC after an extensive neural architecture search, compared to the solid green line representing the HMC model selected from a much smaller pool of hyperparameters. We see that it has minimal impact on Cell Coverage (left), moderate impact on DTLZ5 (center), and extensive impact on Pest Control (right), even outperforming GPs. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.

hyperparameter optimization may not be as necessary. Additionally, our original HMC surrogate model choice may already have been a suitable choice for some problems, so an extensive search over architectures may not significantly improve the performance.

4.5. High-Dimensional Experiments

Polynomial We ran 5 trials using batch size 10 with 100 initial points. Our HMC model was an MLP with 2 hidden layers, 256 parameters per layer, tanh activation, likelihood variance of 1.0, and prior variance of 10.0.

Neural Network Draw We ran 5 trials using batch size 10 with 100 initial points. Our HMC model was an MLP with 2 hidden layers, 256 parameters per layer, tanh activation, likelihood variance of 1.0, and prior variance of 10.0.

Knowledge Distillation We ran 5 trials using batch size 10 with 100 initial points. Our HMC model was an MLP with 2 hidden layers, 256 parameters per layer, tanh activation, likelihood variance of 1.0, and prior variance of 10.0.

5. Further Empirical Results

5.1. Role of Architecture

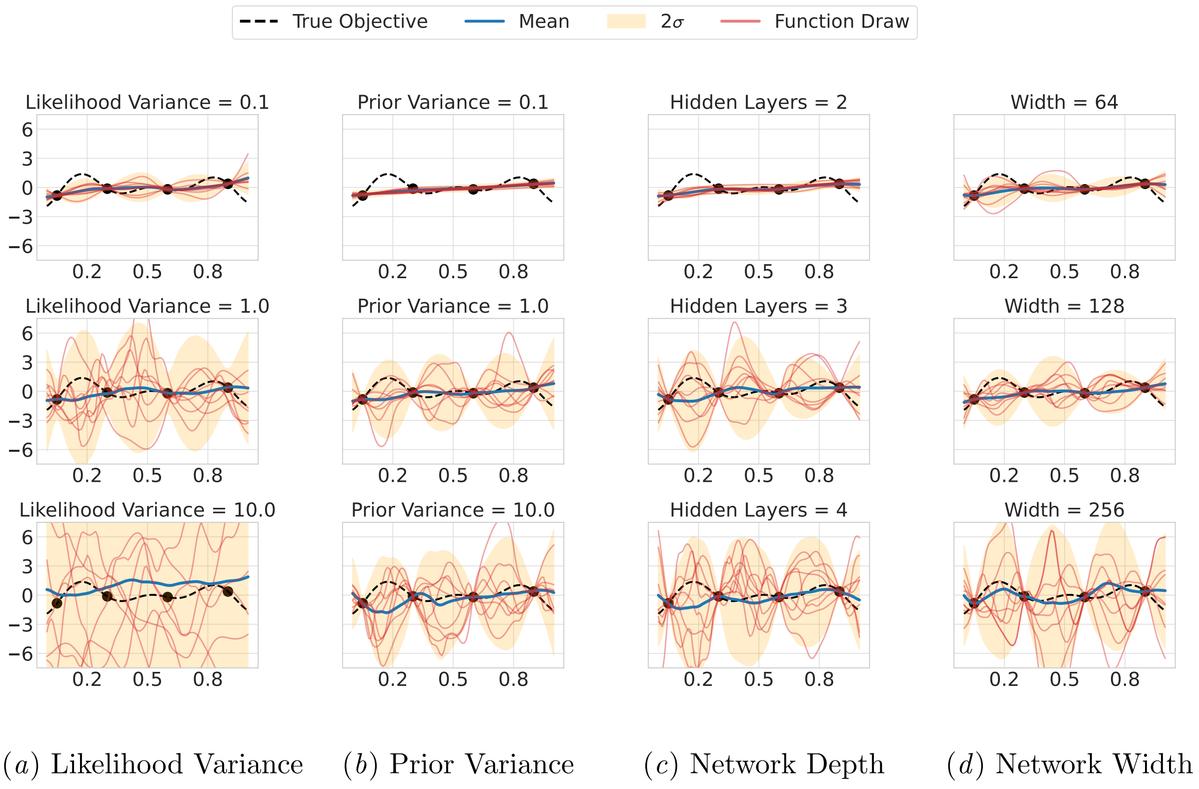
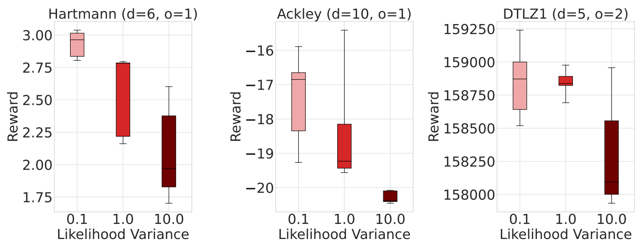


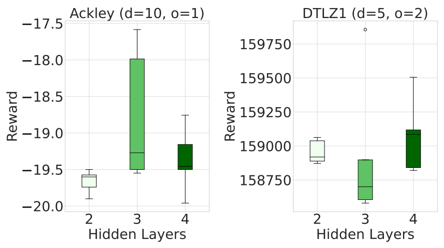
Figure A.2: **The design of the BNN has a significant impact on the uncertainty estimates.** We visualize the uncertainty estimates and function draws produced by full-batch HMC on a simple toy objective function with four function queries (denoted in black). For the visualizations above, we fix all other design choices with the following base parameters: likelihood variance = 1, prior variance = 1, number of hidden layers = 3, and width = 128. We see that varying the different aspects of the model leads to significantly different posterior predictive distributions.

We conduct a sensitivity study into the role of architecture and other key design choices for BNN surrogate models. We highlight results for HMC inference, as it is the gold standard for approximate inference in BNNs (Izmailov et al., 2021).

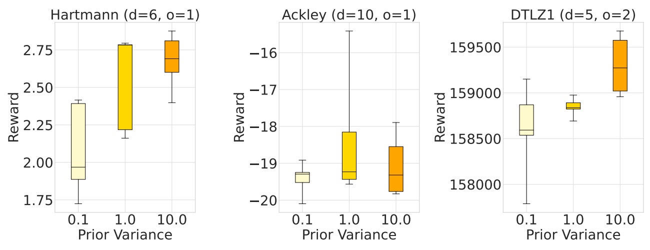
Gaussian processes involve relatively few design choices — essentially only the covariance function, which is often simply set to the RBF or Matérn kernel. Additionally, we are able to have an intuitive understanding of what the induced distributions over functions look like for the different choices of covariance functions.



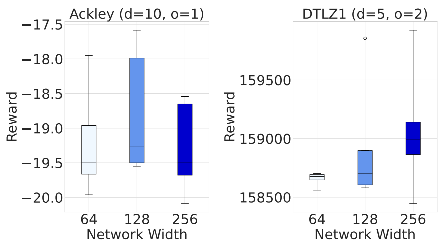
(a) Likelihood Variance



(b) Network Depth



(c) Prior Variance



(d) Network Width

Figure A.3: **There is no single architecture for HMC that performs the best across all problems.** We compare the impact of the design on the Bayesian optimization performance for different benchmark problems. For each set of experiments, we fix all other aspects of the design and plot the values of the maximum reward found using HMC after 100 function evaluations over 10 trials.

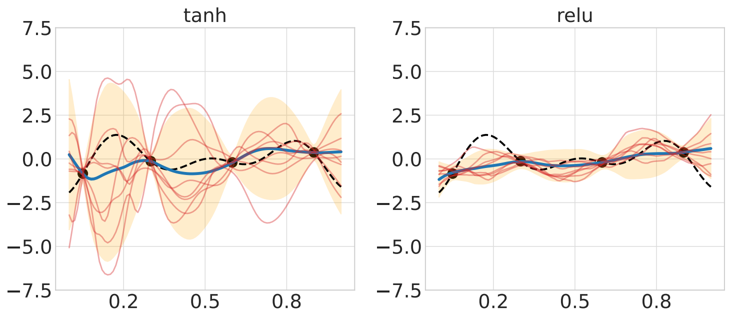
In contrast, with BNNs, we must consider the architecture, the prior over parameters, and the approximate inference procedure. It is also less clear how different modeling choices in BNNs affect the inferred posterior predictive distributions. To illustrate these differences for varying network, prior, and variance parameters, we plot the inferred posterior predictive distributions over functions for different network widths and depths, the activation functions, and likelihood and variance parameters in Figure A.2, and we evaluate the performance under different model choices for three synthetic data problems in Figure A.3. We focus on fully-connected multi-layer perceptrons for this study: while certain architectures have powerful inductive biases for computer vision and language tasks, generic regression tasks such as Bayesian optimization tend to be well-suited for fully-connected multi-layer perceptrons, which have relatively mild inductive biases and make loose assumptions about the structure of the function we are modeling.

Model Hyperparameters. We consider isotropic priors over the neural network parameters with zero mean and variance parameters 0.1, 1, and 10. Similarly, we consider Gaussian likelihood functions with variance parameters 0.1, 1, and 10. The corresponding posterior predictive distributions for full-batch HMC are shown in Figure A.2(a). As would be expected, an increase in the likelihood variance results in a poor fit of the data and virtually no posterior collapse. In contrast, increasing the prior variance results in a higher predictive variance between data points with a good fit to the data points, whereas a prior variance that is too small leads to over-regularization and uncertainty collapse. As shown in Figure A.3(a) and Figure A.3(c), lower likelihood variance parameters and larger prior variance parameters tended to perform best across three synthetic data experiments.

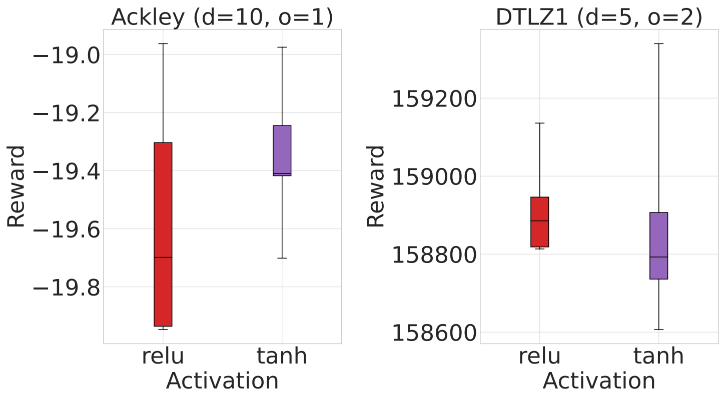
Network Width and Depth. To better understand the effects of the network size on inference, we explore the differences in performance when varying the number of hidden layers and the number of parameters per layer, each corresponding to an increase in model complexity. In Figure A.2(c), we can see that there is a significant increase in uncertainty as we increase the number of hidden layers from two to three, and a BNN with four hidden layers has even larger uncertainty with higher variations in the function draws. Figure A.2(d) also shows an increase in uncertainty as we increase the width of the network, where a smaller width leads to function draws that are much flatter than function draws from a larger width. However, the best size to choose seems to be problem-dependent, as shown in Figure A.3(b) and Figure A.3(d).

Activation Function. The choice of activation function in a neural network determines important characteristics of the function class, such as smoothness or periodicity. The impact of the activation function can be seen in Appendix 5, with function draws from the ReLU BNN appearing more jagged and function draws from the tanh BNN more closely resembling the draws from a GP with a Squared Exponential or Matérn 5/2 covariance function.

We also explore the effect of the activation function on HMC. In Figure A.4, we see that the function draws from BNNs with tanh activations appear quite different from the function draws from BNNs with ReLU activations. The tanh draws are smoother and have more variation, while the ReLU draws seem to be more jagged. We also include results which compare the performance of ReLU and tanh activations in Bayesian optimization problems. We see that there does not appear to be an obvious trend, the optimal choice of activation function is problem-specific.



(a) Posterior predictive distribution



(b) Maximum reward after 100 function evaluations

Figure A.4: The function draws of BNNs with tanh activations appear to be similar to the function draws from a GP. In contrast, the function draws from BNNs with ReLU are often jagged, and the network seems to deviate less from the mean. In practice, ReLU and tanh activation functions can have comparable performance on synthetic functions.

5.2. Synthetic Benchmarks

We evaluate BNN and GP surrogates on a variety of synthetic benchmarks, and we choose problems with a wide span of input dimensions and number of objectives. Detailed problem descriptions can be found in Appendix 3, and we include the experiment setup in Appendix 4.

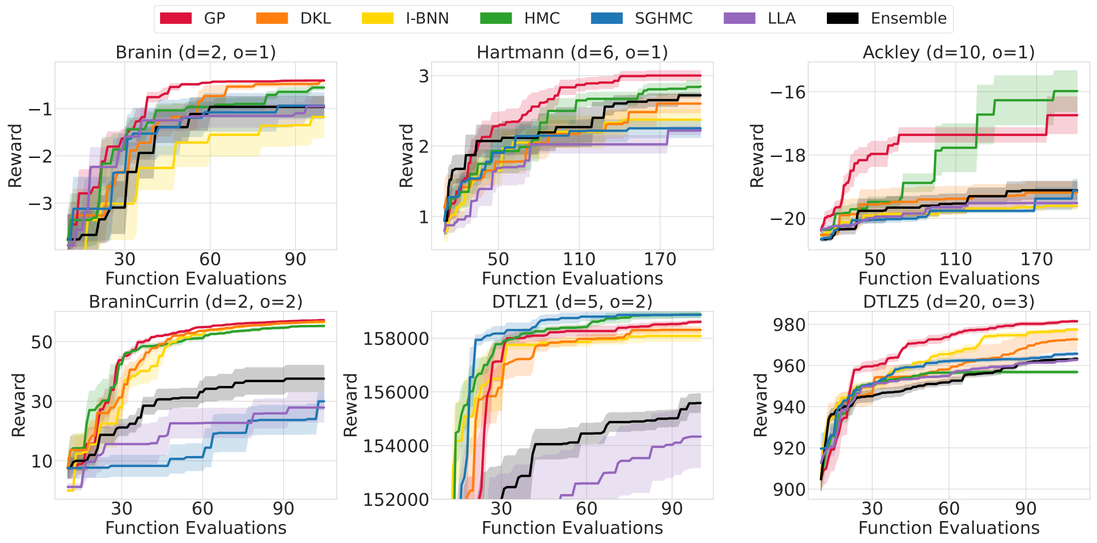


Figure A.5: **BNNs are often comparable to GPs on standard synthetic benchmarks.** However, the type of BNN used has a big impact: HMC is similar to GPs and even outperforms them on Ackley, while SGHMC and deep ensembles have less reliable performance and are often unable to find the maximum. I-BNNs also struggle on these low-dimensional problems. For each benchmark function, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.

In Figure A.5, we find BNN surrogate models to show promising results; however, the specific performance of different BNNs varies considerably per problem. DKL matches GPs in Branin and BraninCurrin, but seems to perform poorly on highly non-stationary problems such as Ackley. I-BNNs also seem to slightly underperform compared to GPs on these synthetic problems, many of which have a small number of input dimensions. In contrast, we find finite-width BNNs using full HMC to be comparable to GPs, performing very similarly in many of the experiments, slightly underperforming in Hartmann and DTLZ5, and outperforming standard GPs in the 10-dimensional Ackley experiment. However, this behavior is not generalizable to all approximate inference methods: the performance of SGHMC and LLA vary significantly per problem, matching the performance of HMC and GPs in some experiments while failing to approach the maximum value in others. Deep ensembles also consistently underperform the other surrogate models, plateauing at noticeably lower objective values on multi-objective problems like BraninCurrin and DTLZ1. This result is surprising, since ensembles are often seen to as an effective way to measure uncertainty (Appendix 5.3).

5.3. Deep Ensembles

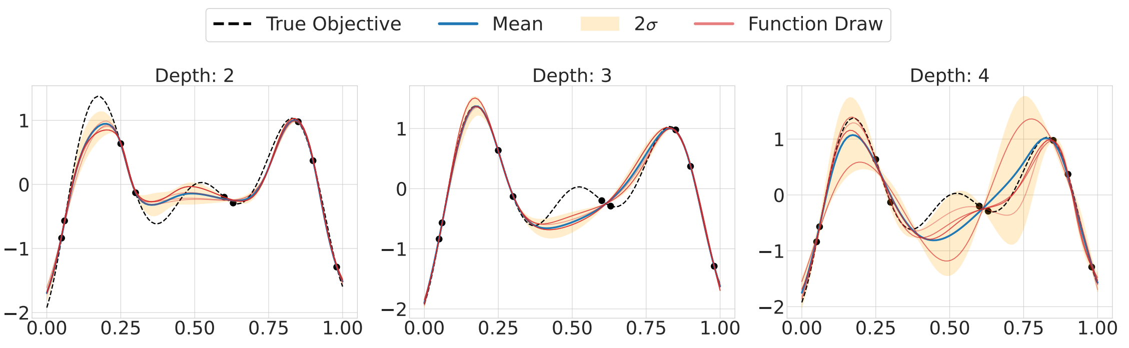
While deep ensembles often provide good accuracy and well-calibrated uncertainty estimates in other settings (Lakshminarayanan et al., 2017), we show they can perform relatively poorly for Bayesian optimization. For instance, when compared to other BNNs on benchmark problems such as BraninCurrin and DTLZ1, the maximum reward found by deep ensembles plateaus at a lower value than other surrogate models.

To further investigate the behavior of deep ensembles, we conduct a sensitivity study, varying the architecture, the amount of training data, and the number of models in the ensemble.

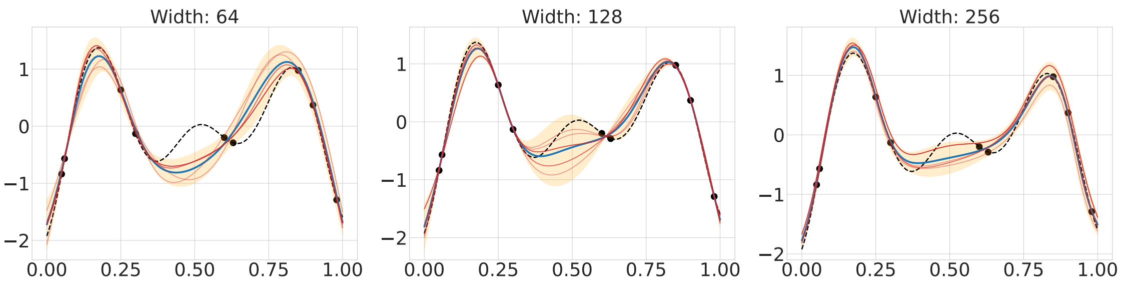
In Figure A.6, we see that smaller training sizes can paradoxically lead to less uncertainty with deep ensembles. A critical component in the success of deep ensembles is the diversity of its models. After training, each model falls within a distinct *basin of attraction*, where solutions across different basins correspond to diverse functions. Intuitively, however, in the low data regime there are fewer settings of parameters that give rise to easily discoverable basins of attraction, making it harder to find diverse solutions simply by re-initializing optimization runs. We consider, for example, the straight path between the weights of Model 1 and Model 2, and we follow how the loss changes as we linearly interpolate between the weights. Specifically, given neural network weights $\mathbf{w}_1$ from Model 1 and $\mathbf{w}_2$ from Model 2, and $\mathcal{L}(\mathbf{w})$ representing the loss of the neural network with weights $\mathbf{w}$ on the training data, we plot the loss $\mathcal{L}(\mathbf{w}_2 + (\mathbf{w}_1 - \mathbf{w}_2) * x)$ for varying values of x.

We share results in Figure A.7 for DTLZ1, a problem where deep ensembles performed poorly. We can see that in low-data regimes, although the loss between the two models does increase, the loss is significantly higher in other regions of the loss landscape. This behavior suggests that the basins are not particularly distinct, as the loss stays relatively flat between them, and thus less likely to provide diverse solutions. However, as we increase the amount of training data, the models are able to find more pronounced basins. We also verify that the flatter regions correspond to a decrease in model diversity by measuring the cosine similarity between the model weights, and we see that in the low-data regime, models have more similar weights and therefore are less diverse.

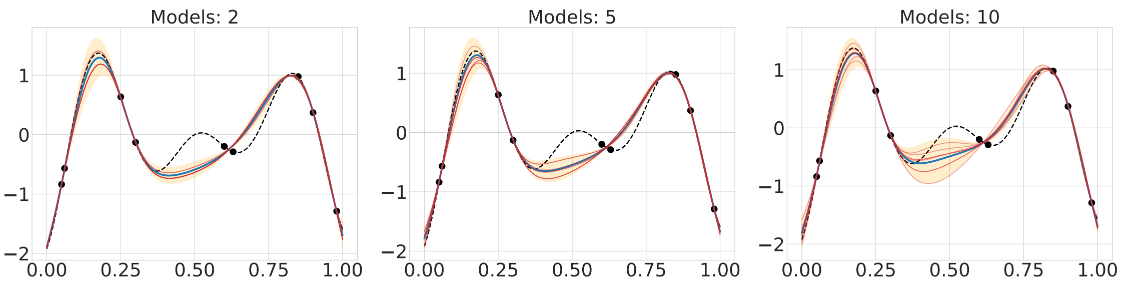
In general, Bayesian optimization problems contain significantly fewer datapoints than where deep ensembles are normally applied. Standard Bayesian optimization benchmarks rarely exceed about 600 data points (objective queries), while in contrast deep ensembles are often trained on problems like CIFAR-10 which have 50,000 training points.



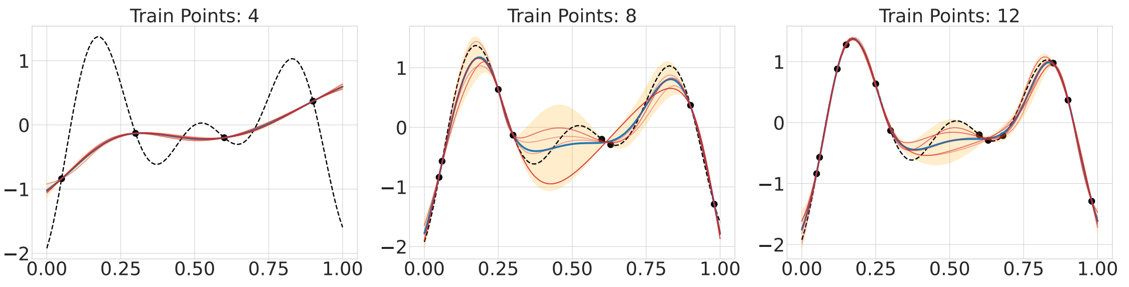
(a) Network Depth



(b) Network Width

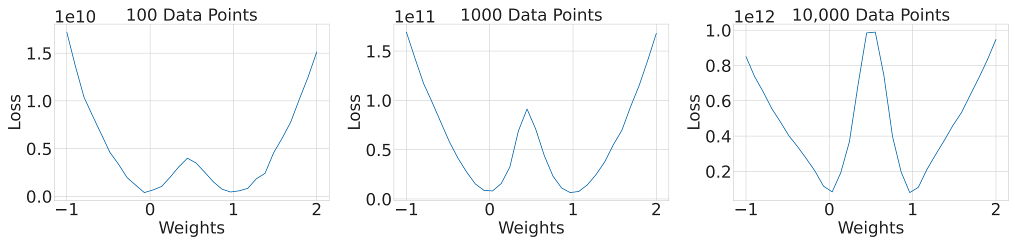


(c) Number of Models

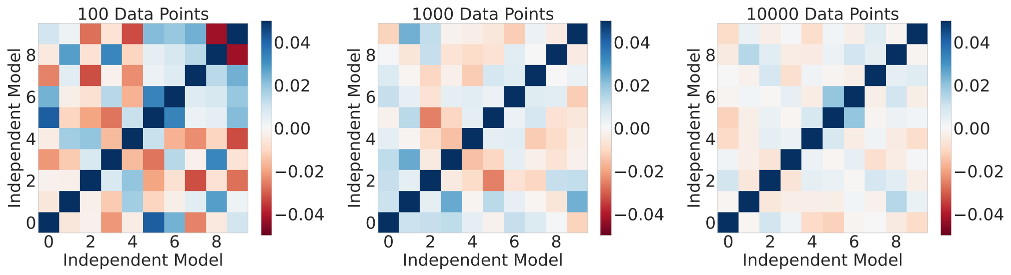


(d) Amount of Training Data

Figure A.6: We visualize the uncertainty of deep ensembles in various scenarios. For each set of experiments, we fix all other design choices with the following base parameters: depth = 3, width = 128, models = 5, train points = 9.



(*a*) Loss landscape



(*b*) Cosine similarity between models

Figure A.7: **With minimal training data, the loss landscape is relatively smooth, and separately-trained models are less diverse.** In the *top* figure, we train two models, corresponding to $x = 0$ and $x = 1$. We then linearly interpolate the weights between the two models to measure how the loss changes. As we increase the number of datapoints, the loss landscape becomes less smooth and models are able to find diverse basins of attraction. For the *bottom* figure, we train 10 models and plot the cosine similarity between weights. With less training data, the weights of the models are more related and the models are less diverse. We plot the results for DTLZ1 with 5 input dimensions and 2 output dimensions.

5.4. Infinite-Width BNNs

I-BNNs outperformed GPs in Bayesian optimization on a variety of high-dimensional problems, and we show results in Figure A.8. The first row of results corresponds to finding the maximum value of a random polynomial function, and the second and third rows show the results of maximizing a function draw from a neural network. For the neural network function draw benchmark, we experimented with many different architectures for the neural network to ensure that we had a diverse set of objective functions to maximize, as denoted in the title of each plot. In all cases, I-BNNs were able to find significantly larger values than GPs.

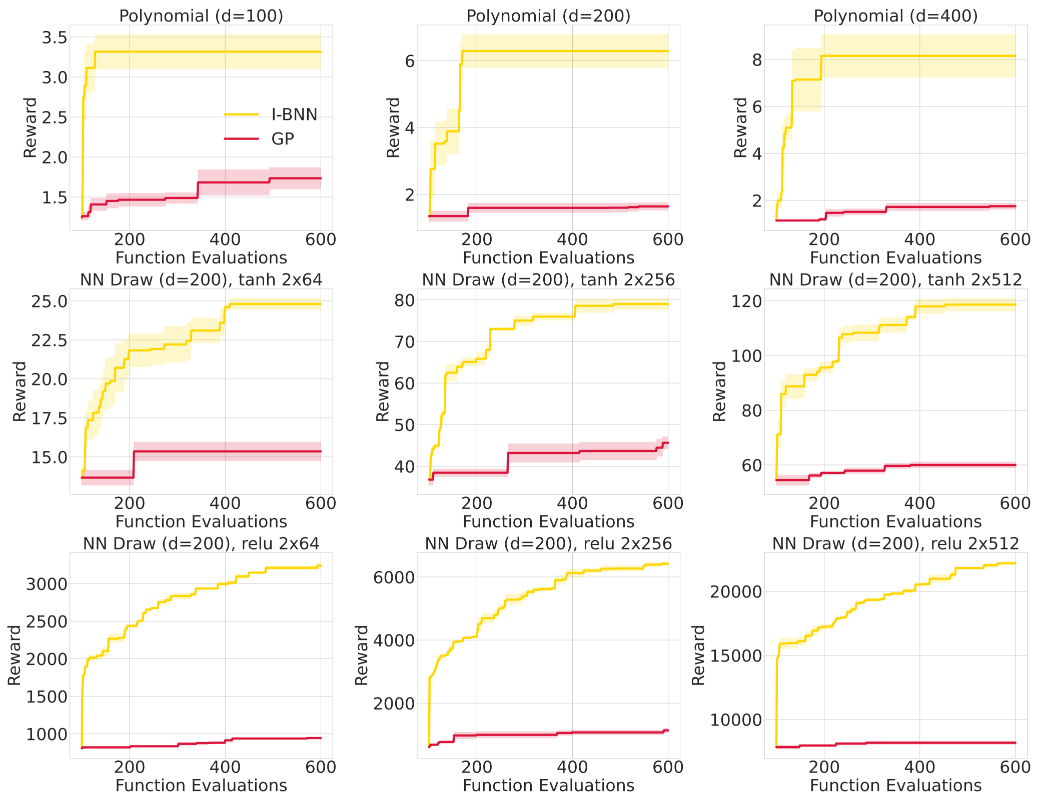


Figure A.8: **I-BNNs outperform GPs in high dimensions.** We plot the mean and one standard error of the mean over 3 trials, and d corresponds to the number of input dimensions. For the neural network draw benchmark, the architecture of the neural network which the function is drawn from is also included, where 2×64 means the network has 2 hidden layers of 64 nodes each.

In Figure A.9, we conduct a sensitivity analysis on the design of I-BNNs, showing how the posterior changes as we vary certain hyperparameters. Unlike standard GPs with RBF or Matérn kernels which are based in Euclidean similarity, I-BNNs instead have a non-stationary and non-Euclidean similarity metric which is more suitable for high-dimensional problems. Additionally, I-BNNs consist of a relatively strong prior, which is particularly useful in the data-scarce settings common to high-dimensional-problems.

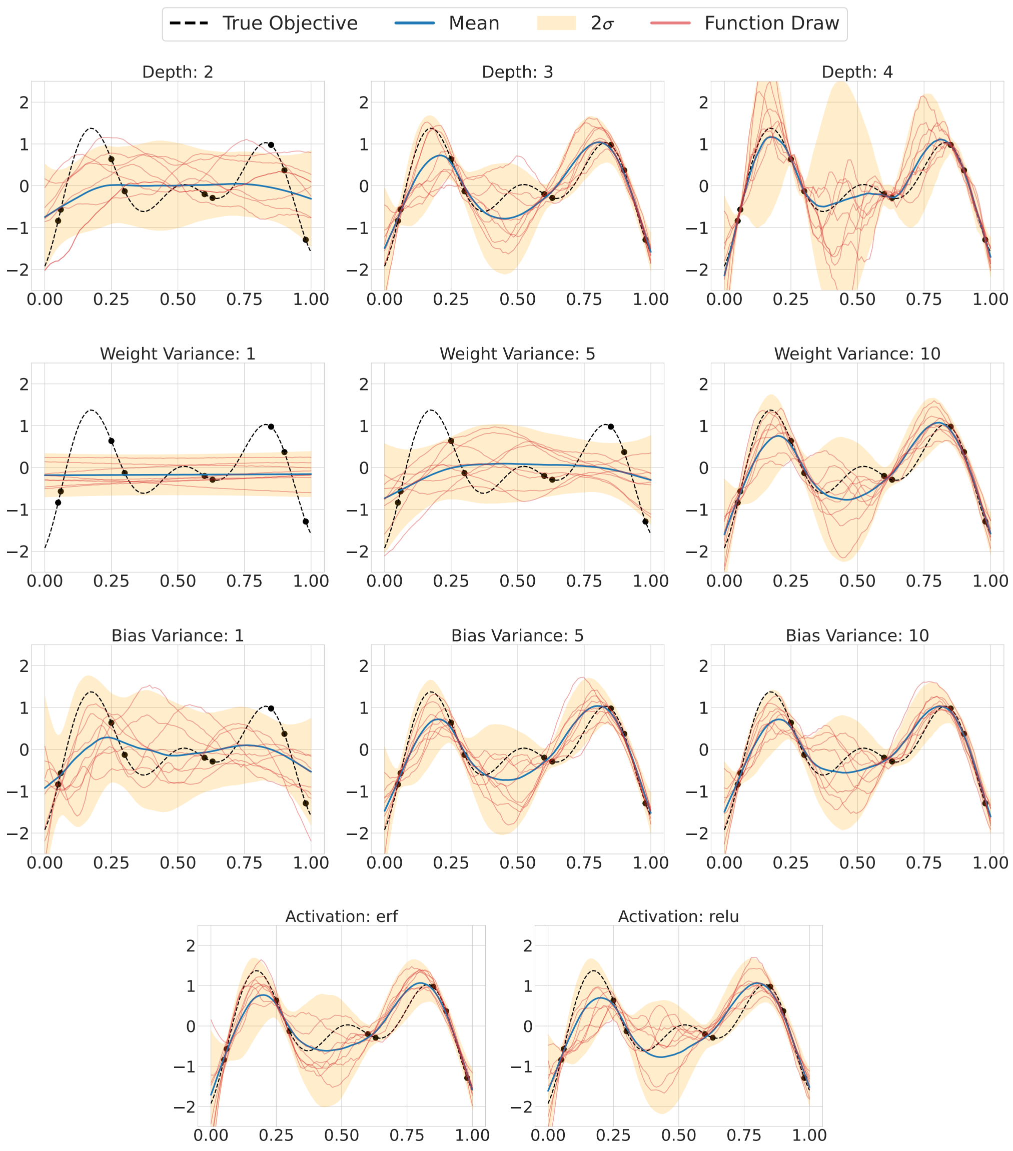


Figure A.9: We visualize how posterior predictive distribution of I-BNNs changes with different design choices. For each set of experiments, we fix all other design choices with the following base parameters: depth = 3, weight variance = 10, bias variance = 5, activation = relu.

Problem	Dim	GP MLL	I-BNN MLL
Branin	2	-3775.3	-1354.1
Hartmann	6	-1320.4	-1630.1
Ackley	10	-1463.6	-1824.3
Ackley	20	-1657.3	-2070.2
Ackley	50	-2788.9	-2316.4
Ackley	100	-4131.6	-2456.9
NN Draw	10	-3226.9	-1829.6
NN Draw	50	-15544.3	-2344.7
NN Draw	100	-39197.5	-2453.8
NN Draw	200	-35775.2	-2584.0
NN Draw	400	-48710.9	-2737.2
NN Draw	800	-63123.4	-2868.9
Distillation	6230	-2126658.7	-3059.8

Table A.1: **I-BNNs have larger marginal likelihoods than GPs on high-dimensional problems.** Here, we show the marginal log likelihood (MLL) of GPs and I-BNNs on various benchmark problems, and we estimate the MLL by sampling 1000 points randomly from the input domain.

To explore the suitability of the neural network kernel, we compare the marginal likelihood of GPs and I-BNNs on problems with various dimensions, and we report results in Table A.1. We see that I-BNNs and GPs have similar marginal likelihoods on problems with fewer dimensions, such as Branin and Hartmann, but as we increase the number of dimensions, the marginal likelihood of GPs becomes lower than that of I-BNNs. Interestingly, we see that the marginal likelihood of GPs does not decrease as quickly for Ackley as it does for the neural network draw test problem. This may be due to the neural network draw objective function having higher non-stationarity, so the standard GP kernel is less suitable for this problem compared to Ackley. For the knowledge distillation experiment, we see that the marginal likelihood of GPs plummets, and it was also not able to find high rewards for the problem as shown in Figure 2. Overall, I-BNNs have a higher marginal likelihood than GPs on high-dimensional problems, indicating that the I-BNN prior may be more reasonable in these settings.

5.5. Revisiting Standard Gaussian Process Assumptions

While Gaussian processes are typically used as the default surrogate model for Bayesian optimization, there are many design choices which play a crucial role in the performance of the GP. The choice of kernel significantly impacts the uncertainty estimates from a GP, and the Matérn kernel is often used as the default choice over other kernels such as the Radial Basis Function (RBF) kernel (e.g. Eriksson et al., 2019; Snoek et al., 2012), although there has been limited prior work comparing the performance of GP kernels on various Bayesian optimization benchmarks. The method for selecting GP hyperparameters is also important. While many works choose to compute a point estimate for the hyperparameters by optimizing the marginal likelihood, others use a fully Bayesian approach, where the posterior distribution is computed by marginalizing over all possible values of the hyperparameters.

In Figure A.10, we study the impact of the hyperparameter choice and the performance of Matérn and RBF kernel on our benchmark tasks. We find the method of hyperparameter selection to play a significant role in the performance of the GP; there is a significant gap between the maximum rewards found using the point estimate vs marginalization for many problems such as BraninCurrin and Oil Spill Sorbent. For discrete problems such as Pest Control and Cell Coverage, we find marginalization to work better than the point estimate. The optimal kernel choice is also problem-dependent: while the Matérn and RBF kernel kernels perform similarly on a majority of the datasets, the Matérn kernel corresponds to less smooth functions compared to the RBF kernel and performs well on many real-world high-dimensional problems.

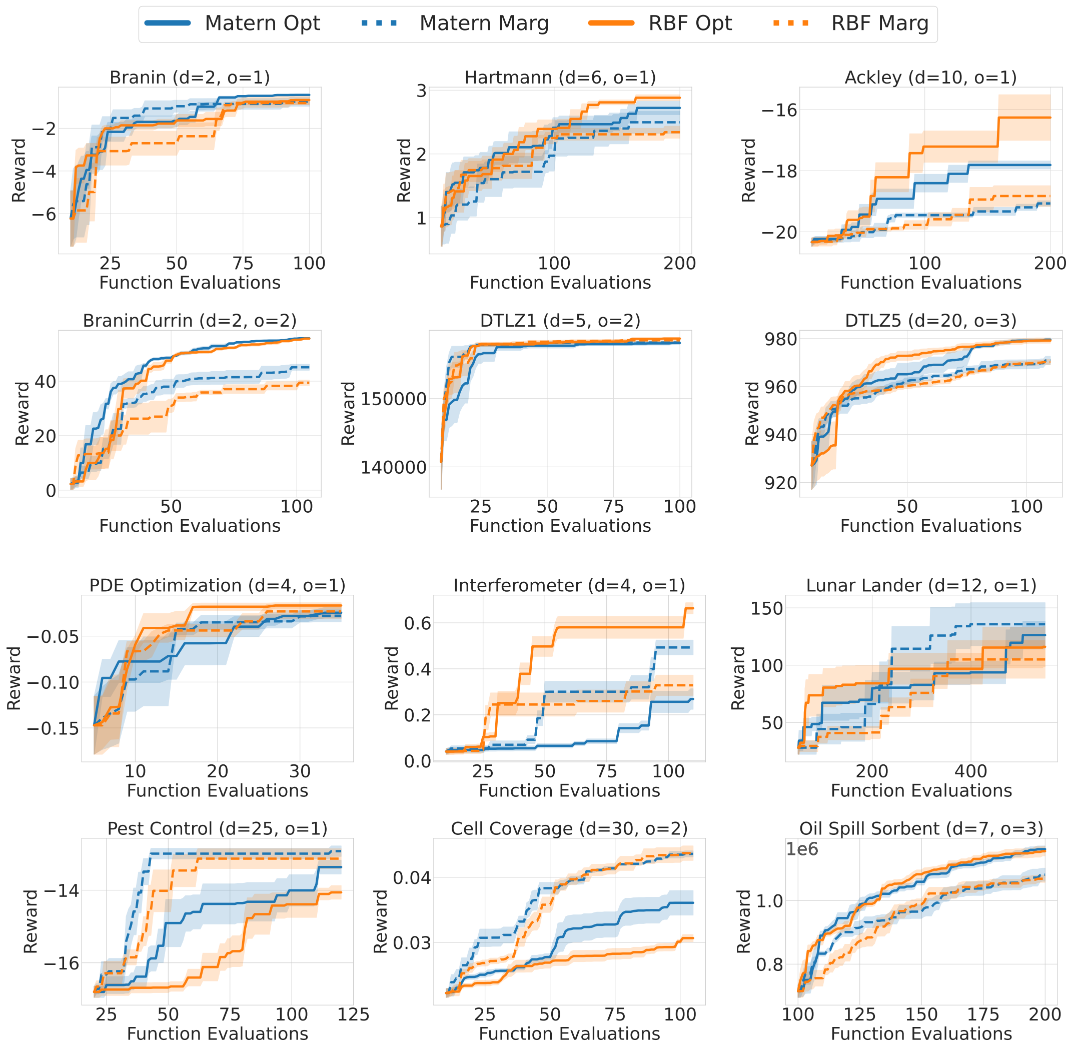
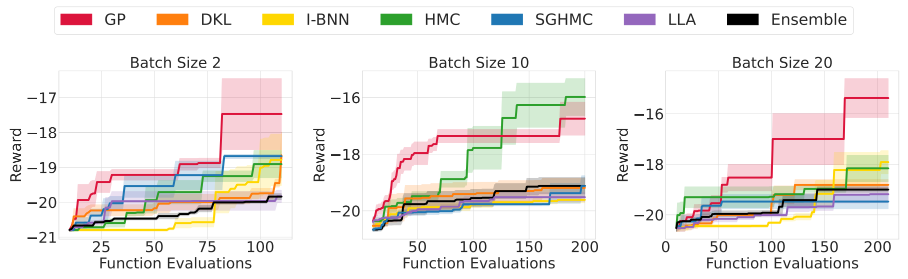
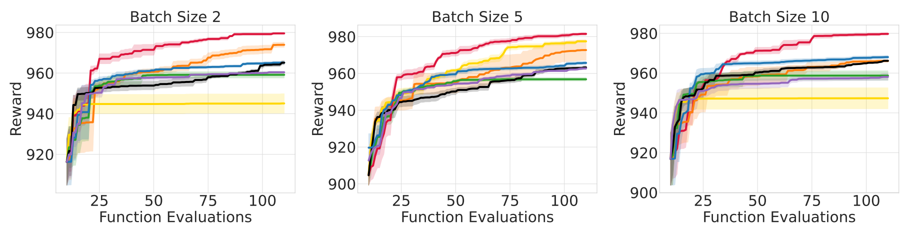


Figure A.10: **The method of hyperparameter selection and the kernel used makes a large impact on the performance of the GP.** The solid lines refer to hyperparameter optimization using a point estimate, while dashed lines correspond to fully Bayesian marginalization. We see that optimal choice of GP kernel is problem-dependent: while Matérn and RBF perform similarly in many instances, we see that Matérn outperforms RBF on Lunar Lander and Pest Control, while RBF performs well on Ackley and Interferometer. The method of hyperparameter optimization is also important, and we find marginalization to be helpful on high-dimensional real-world benchmarks such as Pest Control and Cell Coverage. We plot the mean and one standard error of the mean over 5 trials, d refers to the number of input dimensions, and o refers to the number of output dimensions.

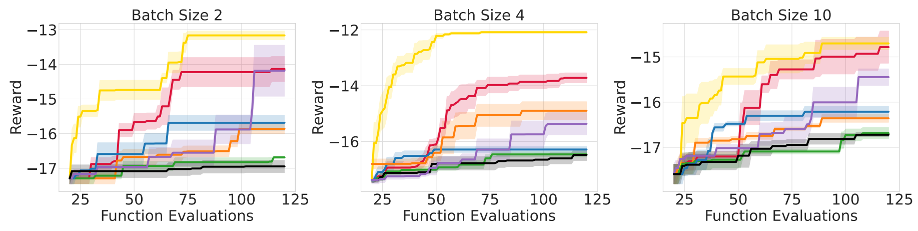
5.6. Acquisition Batch Size



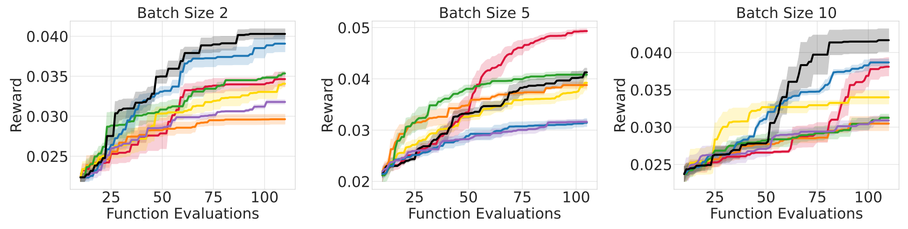
(a) Ackley (d=10, o=1)



(b) DTLZ5 (d=20, o=3)



(c) Pest Control (d=25, o=1)



(d) Cell Coverage (d=30, o=2)

Figure A.11: The batch size of the acquisition function impacts the performance of the models, but the general trends remain similar. There are a few notable exceptions: the I-BNN seems to plateau on DTLZ5, and is unable to find high rewards. GPs also seem to perform significantly better on Cell Coverage with a batch size of 5 compared to other sizes. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.

5.7. Smaller Architecture

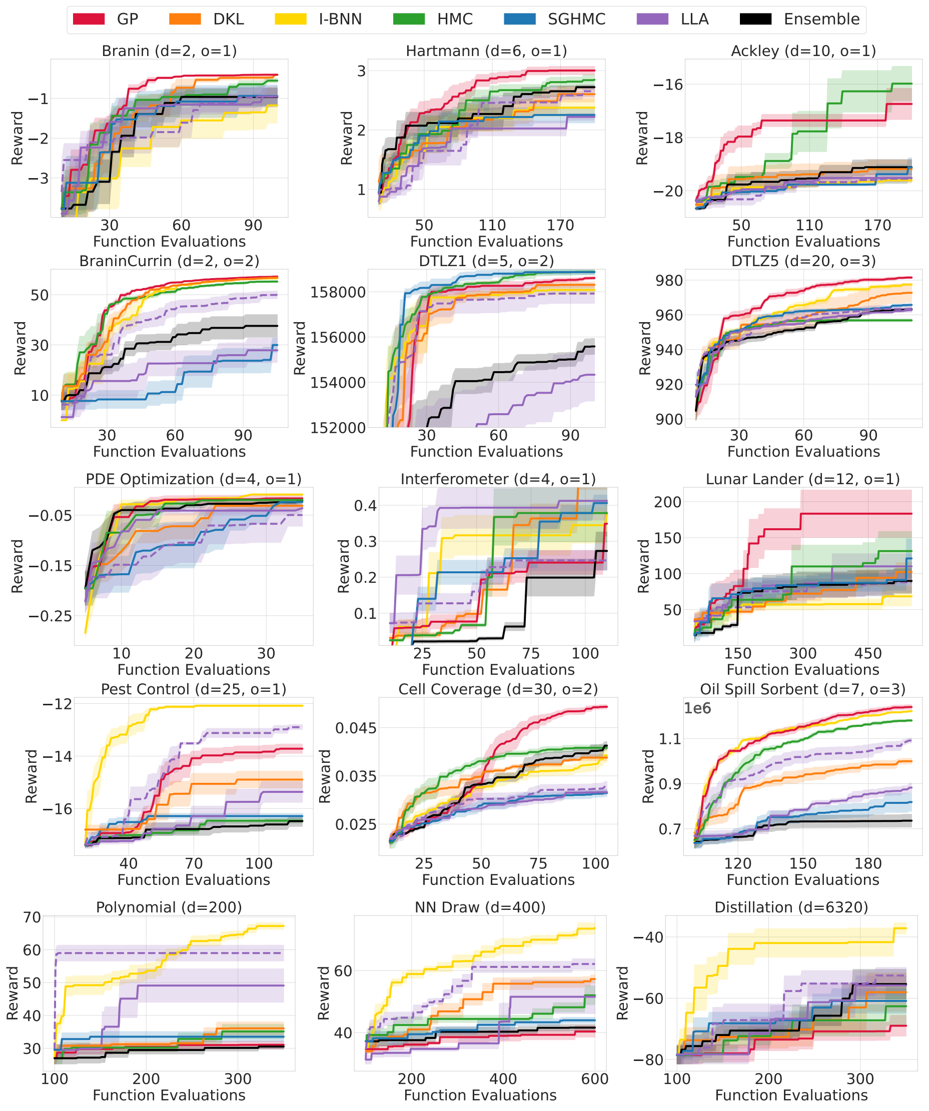
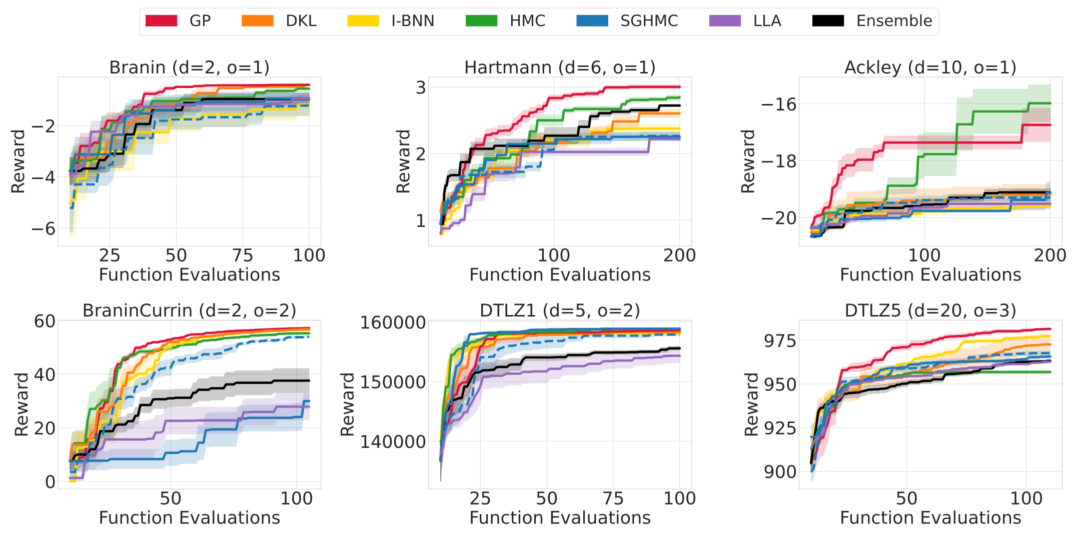
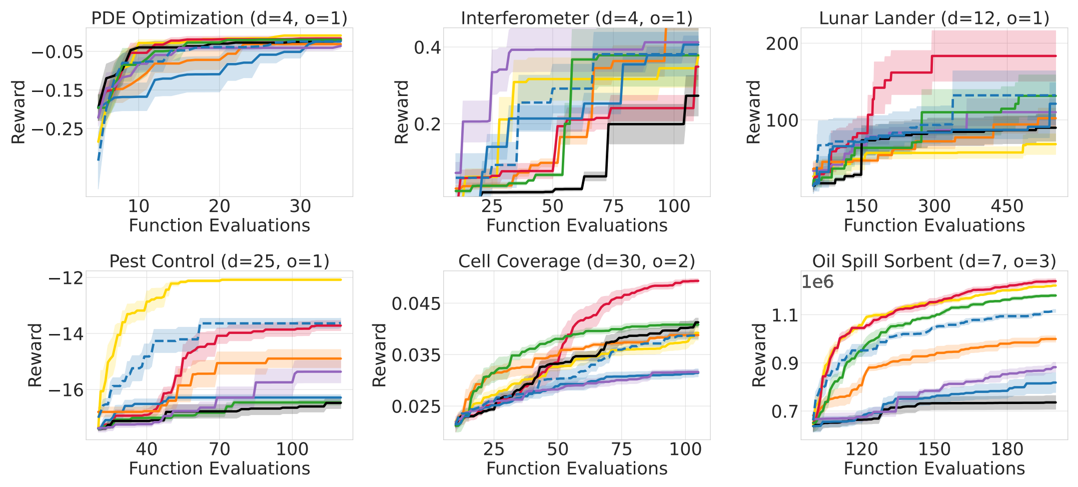


Figure A.12: **The linearized Laplace approximation with a different architecture**. The solid line purple line represents the performance of LLA with the same architecture as the other BNN methods as specified in Appendix 4. The dashed purple line represents the performance of LLA with the relatively small architecture specified by Kristiadi et al. (2023). For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials. Neither architecture is best for all problems, consistent with our findings in Appendix 5.1. While the smaller architecture tends to improve performance for LLA, the I-BNN, HMC, and the standard GP generally provide stronger Bayesian optimization performance than LLA.

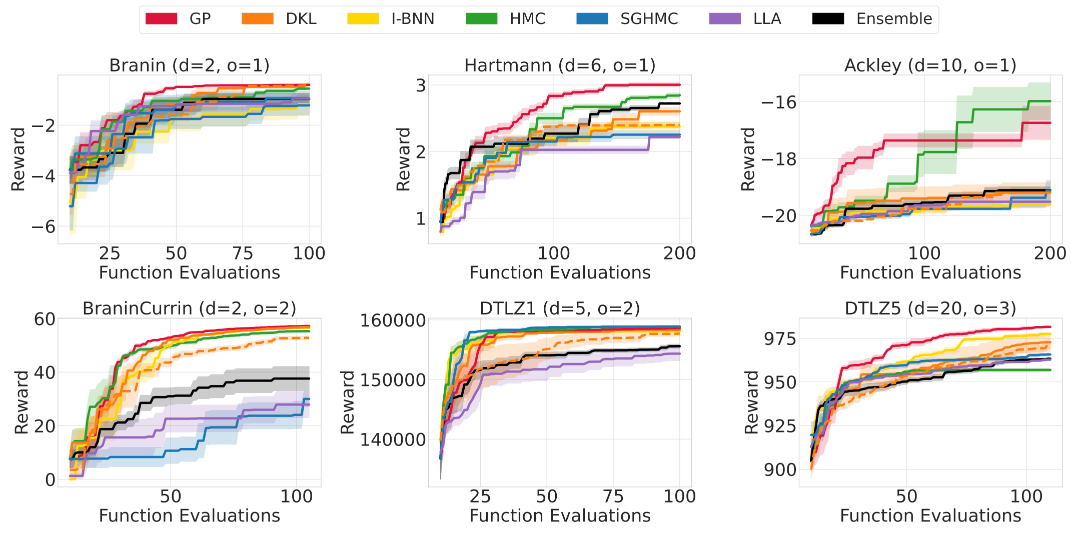


(*a*) Synthetic Benchmarks

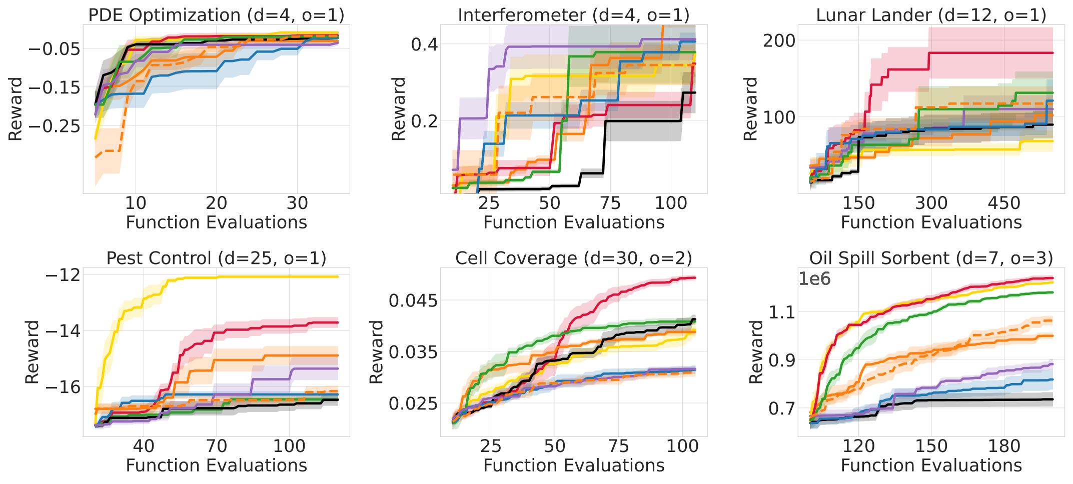


(*b*) Real-World Benchmarks

Figure A.13: **SGHMC with a different architecture**. The solid line blue line represents the performance of SGHMC with the same architecture as the other BNN methods as specified in Appendix 4, and the dashed blue line represents the performance of SGHMC with the relatively small architecture specified by Kristiadi et al. (2023): an MLP with 2 hidden layers of size 50 with ReLU activation. We find that for SGHMC, the smaller architecture often outperforms the larger one and finds higher rewards for many of the problems. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.



(a) Synthetic Benchmarks



(b) Real-World Benchmarks

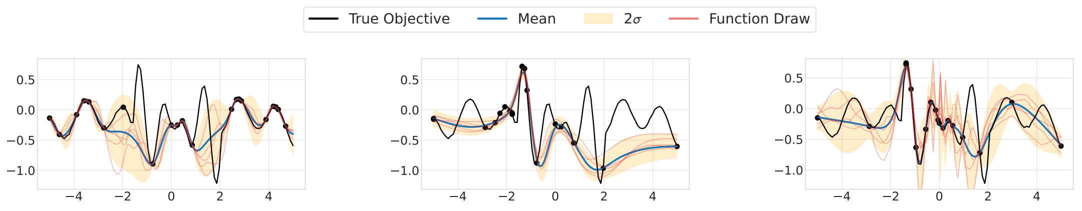
Figure A.14: **Deep kernel learning with a different architecture**. The solid line orange line represents the performance of DKL with the same architecture as the other BNN methods as specified in Appendix 4, and the dashed orange line represents the performance of DKL with the relatively small architecture specified by Kristiadi et al. (2023): an MLP with 2 hidden layers of size 50 with ReLU activation. We find that the smaller architecture finds lower rewards for many of the problems, especially in the real-world datasets. For each benchmark, we include d for the number of input dimensions, and o for the number of objectives. We plot the mean and one standard error of the mean over 5 trials.

5.8. Limitations of Gaussian Process Surrogate Models

To use GP surrogates, we must specify a kernel function class that governs the covariance structure over data points and learn the kernel hyperparameters. We typically constrain model selection to models with kernel of the form $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$. This constraint makes it easier to describe the functional form and learn the hyperparameters of the kernel. However, because the covariance between two values only depends on their distance and not on the values themselves, this setup assumes the function is stationary and has similar mean and smoothness throughout the input space. Unfortunately, this assumption of stationarity does not hold true in many real-world settings. For example, in the common Bayesian optimization application of choosing hyperparameters of a neural network, the true loss function landscape may have vastly different behavior in one part of the input space compared to another. BNN surrogates, in contrast to GP surrogates, are able to model non-stationary functions without similar constraints.

Because the GP assumes that the behavior of the function is the same throughout the input domain, it cannot accurately model the input-dependent variation of the function and leads to underfitting around the true optimum. In contrast, BNN surrogates can learn the non-stationarity of the function.

Due to their assumptions of stationarity, GPs struggle in non-stationary settings. In Figure A.15, we compare the posterior distribution from a GP with the posterior from different BNN surrogate models. We show results after 20 iterations of Bayesian optimization, starting with an initial point of $x = 0$. The function we wish to maximize is non-stationary: the function has greater variance between -2 and 2, and there is also a slight downward trend. We see that due to their stronger assumptions, GPs are not able to find the true global maximum of 0.8, instead getting suck in local optima. In contrast, HMC and I-BNNs are able to find the global maximum within 20 iterations.

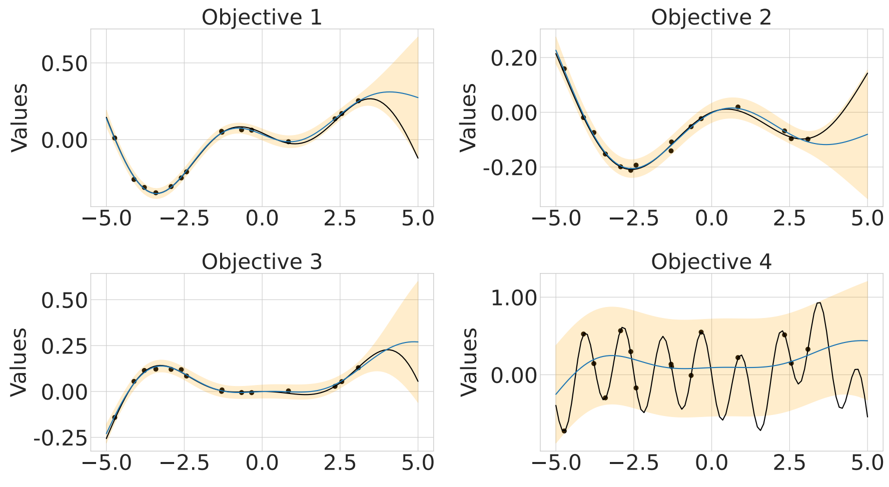


(a) GP (b) HMC (c) I-BNN

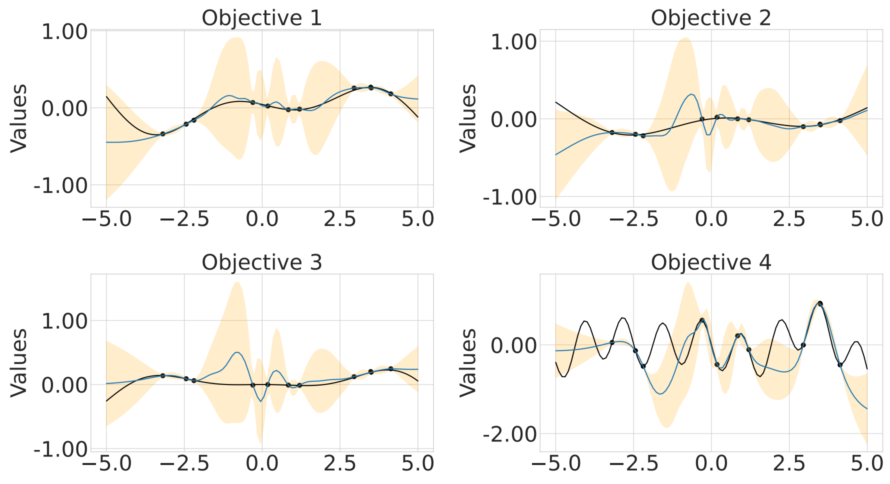
Figure A.15: **GPs struggle to find the global maximum for non-stationary functions**. After 20 function evaluations, the GP does not accurately model the true function around $x = -1$ because it is unable to account for the sudden increase in scale due to its assumption of stationarity. BNNs do not suffer from the same pathologies and are able to find the true maximum.

An additional limitation of GP surrogate models, which we demonstrate in Figure A.16, is its performance on multi-objective problems in Bayesian optimization. Although GPs have been successfully extended to a wide range of multi-objective problems, in the interest of making the approaches scalable, there are many assumptions placed onto the kernel. In the most naive setting, we can model each objective independently. While this approach is convenient, it completely ignores the relationship between objective values and has no notion of shared structure, so it is unable to take advantage of all the information in the problem. Multi-objective covariance functions can also be decomposed as Kronecker product kernels. While this approach can have significant computational advantages compared to modeling the full covariance function, it requires each objective to itself be modeled with the same underlying kernel. Thus, this method of modeling multiple objectives will fail to capture the nuances of each particular objective when the functions have differing properties.

In Figure A.16, we show the result of twenty iterations of Bayesian optimization over a synthetic multitask example, where we care about optimizing over the fourth function but provide additional information through the other three datasets. We use the GP with Kronecker product kernels to model the multiple objectives. In our experiment, although the GP is able to learn the proper length scale and variance over the three additional functions with similar length scales, it struggles to accurately fit the fourth objective. Because the GP is unable to account for the differences between the four functions, it does not find the global optimum. Unlike GPs, BNNs are not restricted to strict covariance structures and are able to produce well-calibrated uncertainty estimates in multi-objective settings. The BNNs are able to accurately fit all four functions, including the fourth objective function which has a much smaller length scale compared to the others.



(*a*) GP



(*b*) HMC

Figure A.16: **GPs have a hard time finding the global maximum in multi-objective settings**. Multi-task GPs learn one length scale across all objectives, which may not be suitable for many datasets. In this example, it does not find the global minimum in the 4th objective because it treats the shorter length scale as noise. In contrast, BNNs are able to appropriately model the uncertainty for the 4th objective and find its true minimum.