Leveraging Deep Learning for Enhanced Bayesian Optimization in Scientific Domains with Complex Structures

Abstract

Bayesian optimization (BO) is a widely used technique for the global optimization of costly black-box functions. However, many real-world scenarios involve functions that are not entirely black-box. These functions may possess known structures, such as symmetries, or the data generation process might be a composite one that provides valuable intermediate information beyond the optimization objective's value. Traditional surrogate models used in BO, like Gaussian Processes (GPs), do not scale well with large datasets and struggle to incorporate known structures. This paper introduces the use of Bayesian neural networks (BNNs), which are scalable and adaptable surrogate models with inductive biases, to enhance BO for intricate, structured problems in high-dimensional spaces. We showcase the application of BO on various practical challenges in physics and chemistry. This includes optimizing the topology of photonic crystal materials using convolutional neural networks and refining chemical properties of molecules with graph neural networks. Our findings indicate that neural networks frequently surpass GPs as surrogate models for BO in these complex tasks, achieving greater sampling efficiency and reduced computational expenses.

1 Introduction

Bayesian optimization (BO) is a powerful technique for global optimization, particularly suited for expensive, derivative-free functions. It has found applications across various scientific and engineering domains, including hyperparameter tuning in machine learning. BO operates by iteratively selecting the next data point to evaluate, aiming to maximize sampling efficiency and minimize the number of evaluations needed to find the optimum. This is crucial when experiments or simulations are time-consuming or resource-intensive.

In numerous fields, the system under investigation is not a complete black box. For instance, high-dimensional input spaces like images or molecules often exhibit known structures, symmetries, and invariances. Moreover, the function might be decomposable into other functions, where the data collection process yields intermediate or auxiliary information that can be used to compute the objective function more efficiently. Examples include scientific experiments or simulations that produce high-dimensional observations or multiple measurements simultaneously, such as the optical scattering spectrum of a nanoparticle across various wavelengths or multiple quantum chemistry properties of a molecule from a single density functional theory (DFT) calculation. These physically-informed insights into the system are valuable for designing surrogate models with appropriate inductive biases, but they are often underutilized in current methods.

BO relies on a surrogate model to represent a distribution over potential functions, incorporating uncertainty in its predictions. Gaussian Processes (GPs) are commonly used as surrogate models due to their analytical tractability. However, GPs face challenges: (1) their inference time scales cubically with the number of observations and output dimensionality, making them less suitable for large datasets or problems with high output dimensionality without kernel approximations, and (2) they are most naturally applied to continuous, low-dimensional input spaces, requiring careful manual formulation of kernels for high-dimensional data with complex structures. Consequently, encoding inductive biases can be difficult.

Neural networks (NNs) and Bayesian neural networks (BNNs) have emerged as alternatives to GPs due to their scalability and flexibility. Another approach involves using neural networks to generate continuous latent spaces, making it easier to apply BO with standard GPs. The ability of BNN architectures to incorporate various constraints, symmetries, and inductive biases opens up possibilities for applying BO to more complex tasks involving structured data.

This work demonstrates the application of deep learning to facilitate BO for complex, real-world scientific datasets, without relying on pre-trained models. Specifically:

- We utilize auxiliary or intermediate information to enhance BO for tasks with high-dimensional observations.
- We apply BO to complex input spaces, including images and molecules, using convolutional and graph neural networks, respectively.

- We implement BO on several realistic scientific datasets, such as the optical scattering of a nanoparticle, topology optimization of a photonic crystal material, and chemical property optimization of molecules from the QM9 dataset.

Our results demonstrate that neural networks can significantly outperform GPs as surrogate models on these problems. We believe these strong results will generalize to other contexts, enabling the application of BO to a wider range of problems. While our methods build upon existing techniques, we employ a novel combination of these methods to adapt existing BO frameworks to real-world, complex applications.

2 Related Work

Several methods have been developed to improve the scalability of GPs for larger problems. For example, one framework for multi-output GPs scales linearly with the dimensionality of a low-dimensional subspace of the data. Multi-task GPs have also been used for BO over problems with large output dimensionalities. Furthermore, GPs have been demonstrated on very large datasets using GPUs and intelligent preconditioners, or through various approximations.

Another strategy for scaling BO to larger problems involves combining it with other methods, reducing the need for the surrogate model to train on the entire dataset. For instance, one method uses a collection of independent probabilistic models in different trust regions, iteratively deciding where to perform BO, effectively reducing the problem to a set of local optimizations. Other methods build upon this approach and dynamically learn the partition function separating different regions.

GPs have been adapted to complex problem settings to broaden the applicability of BO. For example, some approaches decompose synthetic problems as a composition of other functions, leveraging the additional structure to improve BO. However, the multi-output GP used in these approaches scales poorly with output dimensionality, limiting their use to simpler problems. GP kernels have also been developed for complex input spaces, including convolutional and graph kernels. Graph kernels have been used to apply BO to neural architecture search (NAS), where the architecture and connectivity of a neural network itself can be optimized.

Deep learning has been employed as a scalable and flexible surrogate model for BO. For instance, neural networks have been used as adaptive basis functions for Bayesian linear regression, enabling BO to scale to large datasets. This approach also allows for transfer learning of the adaptive basis across multiple tasks and modeling of auxiliary signals to improve performance. Additionally, Bayesian neural networks (BNNs) that use Hamiltonian Monte Carlo to sample the posterior have been used for single-task and multi-task BO for hyperparameter optimization.

A popular approach for BO in high-dimensional spaces is latent-space optimization. Here, an autoencoder, such as a VAE, is trained on a dataset to create a continuous latent space representing the data. Then, conventional optimization algorithms, like BO with GPs, can be used to optimize over this continuous latent space. This approach has been applied to tasks such as arithmetic expression optimization and chemical design. Note that these approaches focus on both data generation and optimization, whereas our work focuses solely on the optimization process.

Random forests have also been used for iterative optimization, such as sequential model-based algorithm configuration (SMAC), as they do not face scaling challenges. Tree-structured Parzen Estimators (TPE) are another popular choice for hyperparameter tuning. However, these approaches still encounter difficulties in encoding complex, structured inputs like images and graphs.

Deep learning has also been applied to improve tasks other than BO. For example, active learning, similar to BO, aims to optimize a model's predictive ability with as few data points as possible. The inductive biases of neural networks have enabled active learning on various high-dimensional data, including images, language, and partial differential equations. BNNs have also been applied to the contextual bandits problem, where the model chooses between discrete actions to maximize expected reward.

3 Methodology

3.1 Bayesian Optimization Prerequisites

We will now briefly introduce the BO methodology. We formulate our optimization task as a maximization problem, where we aim to find the input x2̌217 2̌208 X that maximizes a function f, such that x2̌217 = arg maxx f(x). The input x can be a real-valued continuous vector, but it can also be generalized to categorical variables, images, or discrete objects like molecules. The function f returns the objective value y = f(x), which we also refer to as the "label" of x, and can represent a performance metric we want to maximize. In general, f can be a noisy function.

A crucial component of BO is the surrogate model, which provides a distribution of predictions instead of a single point estimate. Ideally, these surrogate models are Bayesian, but in practice, various approximate Bayesian models or even frequentist distributions have been used. In iteration N, a Bayesian surrogate model M is trained on a labeled dataset Dtrain = (xn, yn)N n=1. An acquisition function 0̌3b1 then uses M to suggest the next data point xN+1 2̌208 X to label, where:

$$x_{N+1} = \arg\max_{x \in X} \alpha(x; M, D_{train}) \tag{1}$$

The new data is evaluated to obtain yN+1 = f(xN+1), and (xN+1, yN+1) is added to Dtrain.

3.2 Acquisition Function

A key consideration in BO is selecting the next data point xN+1 2̌208 X given the model M and labeled dataset Dtrain. This is parameterized through the acquisition function 0̌3b1, which is maximized to determine the next data point to label, as shown in Equation 1.

We utilize the expected improvement (EI) acquisition function 0̌3b1EI. When the posterior predictive distribution of the surrogate model M is a normal distribution N(0̌0b5(x), 0̌3c32(x)), EI can be expressed analytically as:

$$\alpha_{EI}(x) = \sigma(x)[\gamma(x)\Phi(\gamma(x)) + \phi(\gamma(x))] \tag{2}$$

where 0̌3b3(x) = (0̌0b5(x) 2̌212 ybest)/0̌3c3(x), ybest = max(ynN n=1) is the best observed objective function value so far, and 0̌3c6 and 0̌3a6 are the PDF and CDF of the standard normal distribution N(0, 1), respectively. For surrogate models without an analytical form for the posterior predictive distribution, we sample from the posterior NMC times and use a Monte Carlo (MC) approximation of EI:

$$\alpha_{EI}^{MC}(x) \approx \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \max(\mu^{(i)}(x) - y_{best}, 0) \tag{3}$$

where 0̌0b5(i) is a prediction sampled from the posterior of M. While some works fit the surrogate model's output to a Gaussian to use Equation 2 for acquisition, this is not valid when the model prediction for y is not Gaussian, which is generally the case for composite functions (see Section 2.4).

EI has advantages over other acquisition functions because the MC approximation (1) remains differentiable, facilitating optimization of the acquisition function in the inner loop (unlike the MC approximation of upper confidence bound (UCB), which is not differentiable and can result in ties), and (2) is inexpensive (unlike naive Thompson sampling for ensembles, which would require retraining a model from scratch in each iteration).

3.3 Continued Training with Learning Rate Annealing

A challenge in BO is the computational cost of training a surrogate model on Dtrain from scratch in every optimization loop, especially since neural networks ideally require extensive training until convergence. To reduce the training time of BNNs in each optimization loop, we use the model trained in the Nth optimization loop iteration as the initialization (a "warm start") for the (N+1)th iteration, rather than starting from a random initialization. Specifically, we employ the cosine annealing learning rate, which starts with a high learning rate and gradually reduces it to 0. For more details, refer to Section A.3 in the Appendix.

3.4 Auxiliary Information

Typically, we assume f is a black-box function, so we train M : X 2̌192 Y to model f. Here, we consider scenarios where the experiment or observation may provide intermediate or auxiliary information z 2̌208 Z, such that f can be decomposed as:

$$f(x) = h(g(x)) \tag{4}$$

where g : X 2̌192 Z is the expensive labeling process, and h : Z 2̌192 Y is a known objective function that can be computed cheaply. This is also known as "composite functions". In this case, we train M : X 2̌192 Z to model g, and the approximate EI acquisition function becomes:

$$\alpha_{EI}^{MC-aux}(x) \approx \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \max(h(\mu^{(i)}(x)) - y_{best}, 0) \tag{5}$$

which can be seen as a Monte Carlo version of the acquisition function presented in prior work. We denote models trained using auxiliary information with the suffix "-aux." Because h is not necessarily linear, h(0̌0b5(i)(x)) is not generally Gaussian even if 0̌0b5(i) itself may be, making the MC approximation convenient or even necessary.

4 Surrogate Models

Bayesian models capture uncertainty associated with both data and model parameters in the form of probability distributions. This is achieved by placing a prior probability distribution P(0̆3b8) on the model parameters and calculating the posterior belief of the parameters using Bayes' theorem after observing new data. Fully Bayesian neural networks have been studied in small architectures but are impractical for realistically sized neural networks, as nonlinearities between layers make the posterior intractable, requiring MCMC methods to sample the posterior. However, in the last decade, numerous proposals for approximate Bayesian neural networks have emerged, capable of capturing some Bayesian properties and producing a predictive probability distribution. In this work, we compare several different options for the BNN surrogate model, along with other non-BNN baselines. We list some notable models here, with model details and results in Section A.4.1 of the Appendix.

Ensembles combine multiple models to improve predictive performance by averaging their results. Ensembles of neural networks have been reported to be more robust than other BNNs, and we use "Ensemble" to denote an ensemble of neural networks with identical architectures but different random initializations, providing enough variation for individual models to give different predictions. Using individual models can be interpreted as sampling from a posterior distribution, so we use Equation 5 for acquisition. Our ensemble size is NMC = 10.

Other BNNs: We also compare to variational BNNs, including Bayes by Backprop (BBB) and Multiplicative Normalizing Flows (MNF); BOHAMIANN; and NeuralLinear. For BBB, we also experiment with KL annealing, denoted by "-Anneal."

GP Baselines: GPs are largely defined by their kernel (also called "covariance functions"), which determines the prior and posterior distributions, how different data points relate to each other, and the type of data the GP can operate on. In this work, "GP" refers to a standard specification using a Mat0̆0e9rn 5/2 kernel, a popular kernel for real-valued continuous spaces. For images, we use a convolutional kernel, labeled as "ConvGP", implemented using the infinite-width limit of a convolutional neural network. For graphs, we use the Weisfeiler-Lehman (WL) kernel, labeled as "GraphGP", which can operate on undirected graphs with node and edge features, making it suitable for chemical molecule graphs. We also compare against "GP-aux," which uses multi-output GPs for problems with auxiliary information (composite functions). In the Appendix, we also examine GPs using infinite-width and infinite-ensemble neural network limits as kernels, as well as TuRBO, which combines GP-based BO with trust regions.

VAE-GP uses a VAE trained beforehand on an unlabeled dataset representative of X. This allows us to encode complex input spaces, such as chemical molecules, into a continuous latent space where conventional GP-based BO methods can be applied, even enabling the generation and discovery of novel molecules not in the original dataset. Here, we modified an existing implementation that uses a junction tree VAE (JTVAE) to encode chemical molecules. More details can be found in the Appendix.

Other Baselines: We compare against two variations of Bayesian optimization, TuRBO and TPE. We also compare against several global optimization algorithms that do not use surrogate models and are computationally inexpensive, including LIPO, DIRECT-L, and CMA-ES.

We emphasize that ensembles and variational methods can easily scale to high-dimensional outputs with minimal increase in computational cost by simply changing the output layer size. Neural Linear and GPs scale cubically with output dimensionality (without covariance approximations), making them difficult to train on high-dimensional auxiliary or intermediate information.

5 Results

We now examine three real-world scientific optimization tasks, all of which provide intermediate or auxiliary information that can be leveraged. In the latter two tasks, the structure of the data also becomes important, and hence BNNs with various inductive biases significantly outperform GPs and other baselines. For simplicity, we highlight results from select architectures (see Appendix for full results, dataset, and hyperparameter details). All BO results are averaged over multiple trials, and the shaded area in the plots represents 0̆0b1 one standard error over the trials.

5.1 Multilayer Nanoparticle

We first consider the problem of light scattering from a multilayer nanoparticle, which has various applications requiring a tailored optical response, including biological imaging, improved solar cell efficiency, and catalytic materials. The nanoparticle we consider consists of a lossless silica core and 5 spherical shells of alternating TiO2 and silica. The nanoparticle is parameterized by the core radius and layer thicknesses, which we restrict to the range of 30 nm to 70 nm. Due to the nanoparticle's size being on the order of the wavelength of light, its optical properties can be tuned by adjusting the number and thicknesses of the layers. The scattering spectrum can be calculated semi-analytically, as detailed in Section A.1.1 of the Appendix.

Our goal is to optimize the scattering cross-section spectrum over a range of visible wavelengths. We compare two different objective functions: the narrowband objective, which aims to maximize scattering in the small wavelength range of 600 nm to 640 nm and minimize it elsewhere, and the highpass objective, which aims to maximize scattering above 600 nm and minimize it elsewhere. While conventional GPs are trained using the objective function as the label directly, BNNs with auxiliary information can be trained to predict the full scattering spectrum (the auxiliary information z 2̆208 R201), which is then used to calculate the objective function.

The BO results are presented in Figure 2. The addition of auxiliary information significantly improves BO performance for BNNs. They are also competitive with GPs, making BNNs a viable approach for scaling BO to large datasets. In Appendix A.5, we observe similar trends for other types of BNNs. Due to the poor scaling of multi-output GPs with respect to output dimensionality, we can only run GP-aux for a limited number of iterations within a reasonable time frame. Within these few iterations, GP-aux performs poorly, only slightly better than random sampling. We also find in the Appendix that BO with either GPs or BNNs is comparable with or outperforms other global optimization algorithms, including DIRECT-L and CMA-ES.

5.2 Photonic Crystal Topology

Next, we examine a more complex, high-dimensional domain with symmetries that are not easily exploited by GPs. Photonic crystals (PCs) are nanostructured materials engineered to exhibit unique optical properties not found in bulk materials, such as photonic band gaps, negative refractive index, and angular selective transparency. With advancements in fabrication techniques enabling smaller feature sizes, there is growing interest in inverse design and topology optimization to design more sophisticated PCs for applications in photonic integrated circuits, flat lenses, and sensors.

Here, we consider 2D PCs consisting of periodic unit cells represented by a 32 0̆0d7 32 pixel image, with white and black regions representing vacuum (or air) and silicon, respectively. Optimizing over raw pixel values may lead to pixel-sized features or intermediate pixel values that are not physically realizable. Therefore, we parameterize the PCs with a level-set function 0̆3c6 : X 2̆192 V that converts a 51-dimensional feature vector x = [c1, c2, ..., c50, 2̆206] 2̆208 R51, representing the level-set parameters, into an image v 2̆208 R320̆0d732 representing the PC. More details can be found in Section A.1.2 of the Appendix.

We test BO on two different data distributions, PC-A and PC-B. In the PC-A distribution, x spans ci 2̆208 [2̆2121, 1], 2̆206 2̆208 [2̆2123, 3]. In the PC-B distribution, we arbitrarily restrict the domain to ci 2̆208 [0, 1]. The PC-A data distribution is translation invariant, meaning that any PC with a translational shift will also be in the data distribution. However, the PC-B data distribution is not translation invariant.

The optical properties of PCs can be characterized by their photonic density of states (DOS). We choose an objective function that aims to minimize the DOS in a certain frequency range while maximizing it elsewhere, corresponding to opening up a photonic band gap in that frequency range. We train GPs directly on the level-set parameters X, whereas we train the Bayesian convolutional NNs (BCNNs) on the more natural unit cell image space V. BCNNs can also be trained to predict the full DOS as auxiliary information z 2̆208 R500.

The BO results, shown in Figure 4(a), demonstrate that BCNNs outperform GPs by a significant margin on both datasets. This is due to both the auxiliary information and the inductive bias of the convolutional layers, as shown in Figure 4(b). Because the behavior of PCs is determined by their topology rather than individual pixel values or level-set parameters, BCNNs are much better suited to analyze this dataset compared to GPs. Additionally, BCNNs can be made much more data-efficient since they directly encode translation invariance and thus learn the behavior of a whole class of translated images from a single image. Because GP-aux is extremely expensive compared to GP (5000̆0d7 longer on this dataset), we are only able to run GP-aux for a small number of iterations, where it performs comparably to random sampling. We also compare to GPs using a convolutional kernel (2̆01cConvGP-NNGP2̆01d) in Figure 4(a). ConvGP-NNGP only performs slightly better than random sampling, likely due to a lack of auxiliary information and inflexibility to learn the most suitable representation for this dataset.

For our main experiments with BCNNs, we use an architecture that respects translation invariance. To demonstrate the effect of another commonly used deep learning training technique, we also experiment with incorporating translation invariance into a translation-dependent architecture using a data augmentation scheme in which each image is randomly translated, flipped, and rotated during training. We expect data augmentation to improve performance when the data distribution exhibits the corresponding symmetries. As shown in Figure 4(c), we indeed find that data augmentation improves the BO performance of the translation-dependent architecture when trained on the translation-invariant PC-A dataset, even matching the performance of a translation-invariant architecture on PC-A. However, on the translation-dependent PC-B dataset, data augmentation initially hurts the BO performance of the translation-dependent architecture because the model is unable to quickly specialize to the more compact distribution of PC-B, putting its BO performance more on par with models trained on PC-A. These results show that techniques used to improve generalization performance (such as data augmentation or invariant architectures) for training deep learning architectures can also be applied to BO surrogate models and, when used appropriately, directly translate into improved BO performance. Note that data augmentation would not be feasible for GPs without a hand-crafted kernel, as the increased size of the dataset would cause inference to become computationally intractable.

5.3 Organic Molecule Quantum Chemistry

Finally, we optimize the chemical properties of molecules. Chemical optimization is of significant interest in both academia and industry, with applications in drug design and materials optimization. This is a difficult problem where computational approaches such as density functional theory (DFT) can take days for simple molecules and are intractable for larger molecules; synthesis is expensive and time-consuming, and the space of synthesizable molecules is large and complex. There have been many approaches to molecular optimization that largely revolve around finding a continuous latent space of molecules or hand-crafting kernels to operate on molecules.

Here, we focus on the QM9 dataset, which consists of 133,885 small organic molecules along with their geometric, electronic, and thermodynamic quantities calculated with DFT. Instead of optimizing over a continuous space, we draw from the fixed pool of available molecules and iteratively select the next molecule to add to Dtrain. This is a problem setting especially common to materials design, where databases are incomplete and the space of experimentally feasible materials is small.

We use a Bayesian graph neural network (BGNN) for our surrogate model, as GNNs have become popular for chemistry applications due to the natural encoding of a molecule as a graph with atoms and bonds as nodes and edges, respectively. For baselines that operate over continuous spaces (i.e., GPs and simple neural networks), we use the Smooth Overlap of Atomic Positions (SOAP) descriptor to produce a fixed-length feature vector for each molecule.

We compare two different optimization objectives derived from the QM9 dataset: the isotropic polarizability 0̆3b1 and (0̆3b5LUMO 2̆212 2̆0acg,,) where 2̆0acg,;, is the HOMO-LUMO energy gap. Other objectives are included in the Appendix. Because many of the chemical properties in the QM9 dataset can be collectively computed by a single DFT or molecular dynamics calculation, we can treat a group of labels from QM9 as auxiliary information z and train our BGNN to predict this entire group simultaneously. The objective function h then simply picks out the property of interest.

As shown in Figure 5(c), GraphGP and the BGNN variants significantly outperform GPs, showing that the inductive bias in the graph structure leads to a much more natural representation of the molecule and its properties. In the case of maximizing the polarizability 0̆3b1, including the auxiliary information improves BO performance, showing signs of positive transfer. However, it does not have a significant impact on the other objectives, which may be due to the small size of the available auxiliary information (only a handful of chemical properties from the QM dataset) compared with the nanoparticle and photonic crystal tasks. In a more realistic online setting, we would have significantly more physically informative information available from a DFT calculation, e.g., we could easily compute the electronic density of states (the electronic analogue of the auxiliary information used in the photonics task).

As seen in Figure 5(d), we also note that the GraphGP is relatively computationally expensive (150̆0d7 longer than GPs for small N and 8000̆0d7 longer than BGNNs for N = 100) and so we are only able to run it for a limited N in a reasonable time frame. We see that BGNNs perform comparably or better than GraphGPs despite incurring a fraction of the computational cost.

VAE-GP uses a modified version of the latent-space optimization method implementation provided by Tripp et al. (2020). Rather than optimizing over a continuous latent space of the VAE, we feed the data pool through the VAE encoder to find their latent space representation and then apply the acquisition function to the latent points to pick out the best unlabeled point to sample. We keep as many hyperparameters the same as the original implementation as possible, except for the weighted retraining, which we forgo since we have a fixed data pool that was used to train the VAE. This setup is similar to GraphNeuralLinear in that a deep learning architecture is used to encode the molecule as a continuous vector, although GraphNeuralLinear is only trained on the labeled data. The results for this experiment show that VAE-GP performs worse than BNNs on two of the three objective functions we tested and slightly better on one objective. We also note that the performance of VAE-GP depends very heavily on the pre-training of the VAE, as choosing different hyperparameters or even a different random seed can significantly deteriorate performance (see Figure 15 in the Appendix).

6 Discussion

Introducing physics-informed priors (in the form of inductive biases) into the model is critical for performance. Well-known inductive biases in deep learning include convolutional and graph neural networks for images and graph structures, respectively, which significantly improve BO performance. Another inductive bias we introduce is the addition of auxiliary information present in composite functions, which significantly improves the performance of BO for the nanoparticle and photonic crystal tasks. We conjecture that the additional information forces the BNN to learn a more consistent physical model of the system since it must learn features shared across the multi-dimensional auxiliary information, thus enabling the BNN to generalize better. For example, the scattering spectrum of the multilayer particle consists of multiple resonances (sharp peaks), the width and location of which are determined by the material properties and layer thicknesses. The BNN could potentially learn these more abstract features, and thus the deeper physics, to help it interpolate more efficiently, akin to data augmentation. Auxiliary information can also be interpreted as a form of data augmentation. Indeed, tracking the prediction error on a validation set shows that models with auxiliary information tend to have a lower loss than those without (see Appendix A.5). It is also possible that the loss landscape for the auxiliary information is smoother than that of the objective function and that the auxiliary information acts as implicit regularization that improves generalization performance.

Interestingly, GP-aux performs extremely poorly on the nanoparticle and photonic crystal tasks. One possible reason is that we are only able to run GP-aux for a few iterations, and it is not uncommon for GP-based BO to require some critical number of iterations to reach convergence, especially in high-dimensional systems where the size of the covariance matrix scales with the square of the dimensionality. It may also be possible that GP-aux only works on certain types of function decompositions and cannot be broadly applied to all composite functions, as the inductive biases in GPs are often hard-coded.

There is an interesting connection between how well BNNs are able to capture and explore a multi-modal posterior distribution and their performance in BO. For example, we have noticed that larger batch sizes tend to significantly hurt BO performance. On the one hand, larger batch sizes may result in poorer generalization as the model finds sharper local minima in the loss landscape. Another explanation is that the stochasticity inherent in smaller batch sizes allows the BNN to more easily explore the posterior distribution, which is known to be highly multi-modal. Indeed, BO often underperforms for very small dataset sizes N but quickly catches up as N increases, indicating that batch size is an important hyperparameter that must be balanced with computational cost.

All our results use continued training (or warm restart) to minimize training costs. We note that re-initializing M and training from scratch in every iteration performs better than continued training on some tasks (results in the Appendix), which points to how BNNs may not sufficiently represent a multi-modal posterior distribution or that continued training may skew the training distribution that the BNN sees. Future work will consider using stochastic training approaches such as SG-MCMC methods for exploring posterior distributions, as well as other continual learning techniques to further minimize training costs, especially for larger datasets.

When comparing BNN architectures, we find that ensembles tend to consistently perform among the best, which is supported by previous literature showing that ensembles capture uncertainty much better than variational methods, especially in multi-modal loss landscapes. Ensembles are also attractive because they require no additional hyperparameters and are simple to implement. Although training costs increase linearly with the size of the ensemble, this can be easily parallelized on modern computing infrastructures. Furthermore, recent work that aims to model efficient ensembles that minimize computational cost could be an interesting future direction. NeuralLinear variants are also quite powerful and cheap, making them very promising for tasks without high-dimensional auxiliary information. Integrating Neural Linear with multi-output GPs is an interesting direction for future work. The other BNNs either require extensive hyperparameter tuning or perform poorly, making them difficult to use in practice. Additional discussion can be found in Appendix A.5.5.

As seen in Appendix A.5.4, VAE-GP performs worse than our method on two of the chemistry objectives and better on one objective. While latent-space optimization methods are often applied to domains where one wants to simultaneously generate data and optimize over the data distribution, these methods can also be applied to the cases in this work, where a data pool (e.g., QM9 dataset for the chemistry task) or separate data generation process (e.g., level-set process for the photonic crystal task) is already available. In these cases, the VAE is not used as a generative model but rather as a way to learn appropriate representations. While latent-space approaches can take advantage of well-developed and widely available optimization algorithms, they also require unsupervised pre-training on a sizable dataset and a suitable autoencoder model with the necessary inductive biases. Such models are available in chemistry, where there has been significant development, but are more limited in other domains such as photonics. On the other hand, our method can incorporate the data structure or domain knowledge in an end-to-end manner during training, although future work is needed to evaluate more carefully how much of an advantage this is and whether it depends on specific dataset or domain characteristics. For settings where we do not need a generative model, it would also be interesting to replace the autoencoder with a self-supervised model or semi-supervised model to create a suitable latent space.

7 Conclusion

We have demonstrated global optimization on multiple tasks using a combination of deep learning and BO. In particular, we have shown how BNNs can be used as surrogate models in BO, enabling the scaling of BO to large datasets and providing the flexibility to incorporate a wide variety of constraints, data augmentation techniques, and inductive biases. We have demonstrated that integrating domain knowledge on the structure and symmetries of the data into the surrogate model, as well as exploiting intermediate or auxiliary information, significantly improves BO performance, all of which can be interpreted as physics-informed priors. Intuitively, providing the BNN surrogate model with all available information allows the BNN to learn a more faithful physical model of the system of interest, thus enhancing the performance of BO. Finally, we have applied BO to real-world, high-dimensional scientific datasets, and our results show that BNNs can outperform our best-effort GPs, even with strong domain-dependent structure encoded in the covariance functions. We note that our method is not necessarily tied to any particular application domain and can lower the barrier of entry for design and optimization.

Future work will investigate more complex BNN architectures with stronger inductive biases. For example, output constraints can be placed through unsupervised learning or by variationally fitting a BNN prior. Custom architectures have also been proposed for partial differential equations, many-body systems, and generalized symmetries, which will enable effective BO on a wider range of tasks. The methods and experiments presented here enable BO to be effectively applied in a wider variety of settings. There are also variants of BO, including TuRBO, which perform extremely well on our tasks, and so future work will also include incorporating BNNs into these variants.

8 Appendix

8.1 Datasets

The dimensionalities of the datasets are summarized in Table 1. The continuous input dimension for chemical molecules refers to the SOAP descriptor. While the space of chemical molecule graphs in general does not have a well-defined dimensionality as chemical molecules can be arbitrarily large and complex, we limit the size of molecules by only sampling from the QM9 dataset, and can define the dimensionality as the sum of the adjacency, node, and edge matrix dimensionalities.

The high dimensionalities of all of these problems make Bayesian neural networks well-suited as surrogate models to enable scaling. Note that the nanoparticle scattering problem can be adjusted to be less or more difficult by either changing the input dimensionality (i.e. the number of nanoparticle layers) or the auxiliary dimension (i.e. the resolution or range of wavelengths that are sampled).

Table 1: Summary of dataset dimensionalities. Note that alternate inputs for photonic crystal and organic molecule datasets are binary images and molecule graphs, respectively.

	CONTINUOUS INPUT DIMENSION	ALTERNATE INPUT DIMENSION	AUXILIARY DIMENSION
NANOPARTICLE SCATTERING	6	N/A	201
PHOTONIC CRYSTAL DOS	51	32 x 32 = 1024	500
MOLECULE QUANTUM CHEMISTRY	480	9 + 9 0̆0d7 9 + 9 0̆0d7 9 = 171	9

8.2 Nanoparticle Scattering

The multilayer nanoparticle consists of a lossless silica core surrounded by alternating spherical layers of lossless TiO2 and lossless silica. The relative permittivity of silica is 0̆3b5silica = 2.04. The relative permittivity of TiO2 is dispersive and depends on the wavelength of light:

$$\varepsilon_{TiO2} = 5.913 + \frac{0.2441}{\lambda^2 - 0.0803} \tag{6}$$

where 0̆3bb is the wavelength given in units of nm. The entire particle is surrounded by water, which has a relative permittivity of 0̆3b5water = 1.77.

For a given set of thicknesses, we analytically solve for the scattering spectrum, i.e. the scattering cross-section 0̆3c3(0̆3bb) as a function of wavelength 0̆3bb, using Mie scattering. The code for computing 0̆3c3 was adapted from existing work.

The objective functions for the narrowband and highpass objectives are:

$$h_{nb}(z) = \frac{\int_{\lambda \in nb} \sigma(\lambda) d\lambda}{\int_{\lambda \notin nb} \sigma(\lambda) d\lambda} \approx \frac{\sum_{i=126}^{145} z_i}{\sum_{i=1}^{125} z_i + \sum_{i=146}^{201} z_i} \tag{7}$$

$$h_{hp}(z) = \frac{\int_{\lambda \in hp} \sigma(\lambda) d\lambda}{\int_{\lambda \notin hp} \sigma(\lambda) d\lambda} \approx \frac{\sum_{i=126}^{201} z_i}{\sum_{i=1}^{125} z_i} \tag{8}$$

where z 2̆208 R201 is the discretized scattering cross-section 0̆3c3(0̆3bb) from 0̆3bb = 350 nm to 750 nm.

8.3 Photonic Crystal

The photonic crystal (PC) consists of periodic unit cells with periodicity a = 1 au, where each unit cell is depicted as a 2̆01ctwo-tone2̆01d image, with the white regions representing silicon with permittivity 0̆3b51 = 11.4 and black regions representing vacuum (or air) with permittivity 0̆3b50 = 1.

The photonic crystal (PC) structure is defined by a spatially varying permittivity 0̆3b5(x, y) 2̆208 0̆3b50, 0̆3b51 over a 2D periodic unit cell with spatial coordinates x, y 2̆208 [0, a]. To parameterize 0̆3b5, we choose a level set of a Fourier sum function 0̆3c6, defined as a linear combination of plane waves with frequencies evenly spaced in the reciprocal lattice space up to a maximum cutoff. Intuitively, the upper limit on the frequencies roughly corresponds to a lower limit on the feature size such that the photonic crystal remains within reasonable fabrication constraints. Here we set the cutoff such that there are 25 complex frequencies corresponding to 50 real coefficients c = (c1, c2, ..., c50).

Explicitly, we have

$$\phi[c](x,y) = \Re\left(\sum_{k=1}^{25} (c_k + i c_{k+25}) e^{2\pi i (n_x x + n_y y)/a}\right) \tag{9}$$

where each exponential term is composed from the 25 different pairs nx, ny with nx, ny 2̆208 2̆2122, 2̆2121, 0, 1, 2. We then choose a level-set offset 2̆206 to determine the PC structure, where regions with 0̆3c6 > 2̆206 are assigned to be silicon and regions where 0̆3c6 2̆264 2̆206 are vacuum. Thus, the photonic crystal unit cell topology is parameterized by a 51-dimensional vector, [c1, c2, ..., c50, 2̆206] 2̆208 R51. More specifically,

$$\varepsilon(x,y) = \varepsilon[c,\Delta](x,y) = \{ \varepsilon_1 \, \phi[c](x,y) > \Delta \varepsilon_0 \phi[c](x,y) \leq \Delta \tag{10}$$

which is discretized to result in a 32 0̆0d7 32 pixel image v 2̆208 0̆3b50, 0̆3b5132̆00d732. This formulation also has the advantage of enforcing periodic boundary conditions.

For each unit cell, we use the MIT Photonics Bands (MPB) software to compute the band structure of the photonic crystal, 0̆3c9(k), up to the lowest 10 bands, using a 32 0̆0d7 32 spatial resolution (or equivalently, 32 0̆0d7 32 k-points over the Brillouin zone 2̆212 0̆3c0 a < k < 0̆3c0 a). We also extract the group velocities at each k-point and compute the density-of-states (DOS) via an extrapolative technique. The DOS is computed at a resolution of 20,000 points, and a Gaussian filter of kernel size 100 is used to smooth the DOS spectrum. To normalize the frequency scale across the different unit cells, the frequency is rescaled via 0̆3c9 2̆192 0̆3c9norm, where 0̆3b5avg is the average permittivity over all pixels. Finally, the DOS spectrum is truncated at 0̆3c9norm = 1.2 and interpolated using 500 points to give z 2̆208 R500.

The objective function aims to minimize the DOS in a small frequency range and maximize it elsewhere. We use the following:

$$h_{DOS}(z) = \sum_{i=1}^{300} z_i + \frac{1}{\sum_{i=351}^{500} z_i + 1} \tag{11}$$

where the 1 is added in the denominator to avoid singular values.

8.4 Organic Molecule Quantum Chemistry

The Smooth Overlap of Atomic Positions (SOAP) descriptor uses smoothed atomic densities to describe local environments for each atom in the molecule through a fixed-length feature vector, which can then be averaged over all the atoms in the molecule to produce a fixed-length feature vector for the molecule. This descriptor is invariant to translations, rotations, and permutations. We use the SOAP descriptor implemented by DScribe using the parameters: local cutoff rcut = 5, number of radial basis functions nmax = 3, and maximum degree of spherical harmonics lmax = 3. We use outer averaging, which averages over the power spectrum of different sites.

The graph representation of each molecule is processed by the Spektral package. Each graph is represented by a node feature matrix X 2̆208 Rs0̆0d7dn, an adjacency matrix A 2̆208 Rs0̆0d7s, and an edge matrix E 2̆208 Re0̆0d7de, where s is the number of atoms in the molecule, e is the number of bonds, and dn, de are the number of features for nodes and edges, respectively.

The properties that we use from the QM9 dataset are listed in Table 2. We separate these properties into two categories: (1) the ground state quantities which are calculated from a single DFT calculation of the molecule and include geometric, energetic, and electronic quantities, and (2) the thermodynamic quantities which are typically calculated from a molecular dynamics simulation.

Table 2: List of properties from the QM9 dataset used as labels

Property	Unit	Description
Ground State Quantities		
A	GHz	Rotational constant
B	GHz	Rotational constant
C	GHz	Rotational constant
μ	D	Dipole moment
α	a_0^3	Isotropic polarizability
ϵ_{HOMO}	Ha	Energy of HOMO
ϵ_{LUMO}	Ha	Energy of LUMO
$\Delta\epsilon$	Ha	Gap ($\epsilon_{LUMO} - \epsilon_{HOMO}$)
$\langle R^2 \rangle$	a_0^2	Electronic spatial extent
Thermodynamic Quantities at 298.15 K		
U_0	Ha	Internal energy at 0 K
U	Ha	Internal energy at 298.15 K
H	Ha	Enthalpy at 298.15 K
G	Ha	Free energy at 298.15 K
c_v	$\frac{cal}{molK}$	Heat capacity at 298.15 K

The auxiliary information for this task consists of the properties listed in Table 2 that are in the same category as the objective property, as these properties would be calculated together. The objective function then simply picks out the corresponding feature from the auxiliary information. More precisely, for the ground state objectives, the auxiliary information is:

$$z = [A, B, C, \mu, \alpha, \epsilon_{HOMO}, \epsilon_{LUMO}, \epsilon_{gap}, < R^2 >] \in \mathbb{R}^9 \tag{12}$$

and the objective functions are:

$$h_\alpha(z) = \frac{z_5}{25} - 6 \tag{13}$$

$$h_{\epsilon_{gap}}(z) = \frac{z_8}{0.6} - 0.02 \tag{14}$$

where the quantities for the latter objective are normalized so that they have the same magnitude.

8.5 Bayesian Optimization and Acquisition Function

Our algorithm for Bayesian optimization using auxiliary information z is shown in Algorithm 1. This algorithm reduces to the basic BO algorithm in the case where h is the identity function and Z = Y such that we can ignore mention of z in Algorithm 1.

Algorithm 1 Bayesian optimization with auxiliary information

```
1:  Input: Labelled dataset $D_{train} = \{(x_n, z_n, y_n)\}_{n=1}^{N_{start}=5}$
2:  for $N = 5$ to 1000 do
3:      Train $M : X \to Z$ on $D_{train}$
4:      Form an unlabelled dataset, $X_{pool}$
5:      Find $x_{N+1} = \arg\max_{x \in X_{pool}} \alpha(x; M, D_{train})$
6:      Label the data $z_{N+1} = g(x_{N+1}), y_{N+1} = h(z_{N+1})$
7:      $D_{train} = D_{train} \cup (x_{N+1}, z_{N+1}, y_{N+1})$
    end for
```

As mentioned in the main text, the inner optimization loop in line 5 of Algorithm 1 is performed by finding the maximum value of 03b1 over a pool of |Xpool| randomly sampled points. We can see in Figure 6 that increasing |Xpool| in the acquisition step tends to improve BO performance. Thus, there is likely further room for improvement of the inner optimization loop using more sophisticated algorithms, possibly using the gradient information provided by BNNs. Unless otherwise stated, we optimize the inner loop of Bayesian optimization to choose the next data point to label by maximizing EI on a pool of |Xpool| = 10^5 randomly sampled points.

[width=0.5]figures/figure6.png

Figure 1: Effect of m = |Xpool| used in the inner optimization loop to maximize the acquisition function on overall BO performance. ybest is taken from the narrowband objective function using the ensemble architecture. The 201caux201d in the legend denotes using auxiliary information and the numbers represent the architecture (i.e. 8 layers of 256 units or 16 layers of 512 units).

8.6 Continued Training

As mentioned in Section 2.3 of the main text, the BNN is ideally trained from scratch until convergence in each iteration loop, although this comes at a great computational cost. An alternative is the warm restart method of continuing the training from the previous iteration which enables the model2019s training loss to converge in only a few epochs. However, as shown in Figure 7, we have found that naive continued training can result in poor BO performance. This is likely because (a) training does not converge for the new data point Dnew = (xN +1, yN +1) relative to the rest of the data under a limited computational budget, resulting in the acquisition function possibly labeling similar points in consecutive iterations, and (b) the BNN gets trapped in a local minima in the loss landscape that is not ideal for learning future data points. To mitigate this, we use the cosine annealing learning rate. The large learning rate at the start of training allows the model to more easily escape local minima and explore a multimodal posterior, while the small learning rate towards the end of the annealing cycle allows the model to converge more easily. Note that the idea of warm restart is similar to 201ccontinual learning,201d which is an open and active sub-problem in machine learning research. In particular, we re-train the BNN using 10 epochs.

[width=0.5]figures/figure7.png

Figure 2: Effect of restarting the BNN training from scratch in each BO iteration.

8.7 Models and Hyperparameters

8.7.1 Additional Surrogate Models

Variational BNNs model a prior and posterior distribution over the neural network weights but use some approximation on the distributions to make the BNN tractable. In particular, we use Bayes by Backprop (BBB) (also referred to as the 201cmean field201d approximation), which approximates the posterior over the neural network weights with independent normal distributions. We also compare Multiplicative Normalizing Flows (MNF), which uses normalizing flows on top of each layer output for more expressive posterior distributions.

BOHAMIANN proposed to use BNNs in BO by using stochastic gradient Hamiltonian Monte Carlo (SGHMC) to approximately sample the BNN posterior, combined with scale adaptation to adapt it for an iterative setting.

NeuralLinear trains a conventional neural network on the data but then replaces the last layer with Bayesian linear regression such that the neural network serves as an adaptive basis for the linear regression.

TuRBO (trust region Bayesian Optimization) is a method that maintains M trust regions and performs Bayesian optimization within each trust region, maintaining M local surrogate models, to scale BO to high-dimensional problems that require thousands of observations. We use M = 1 and M = 5, labeled as ̆201cTuRBO-1̆201d and ̆201cTuRBO-5̆201d, respectively.

TPE (Tree Parzen Estimator) is a method that instead of modeling p(y|x), models p(x|y) and p(y) for the surrogate model and fits into the BO framework. The tree-structure of the surrogate model allows it to define leaf variables only when node variables take particular values, which makes it well-suited for hyper-parameter search (e.g. the learning rate momentum is only defined for momentum-based gradient descent methods).

LIPO is a parameter-free algorithm that assumes the underlying function is a Lipschitz function and estimates the bounds of the function. We use the implementation provided by the dlib library.

DIRECT-L (DIviding RECTangles-Local) systematically divides the search domain into smaller and smaller hyperrectangles to efficiently search the space. We use the implementation provided by the NLopt library.

CMA-ES (covariance matrix adaptation evolution strategy) is an evolutionary algorithm that samples new data based on a multivariate normal distribution and refines the parameters of this distribution until reaching convergence. We use the implementation provided by the pycma library.

8.7.2 Implementation Details

Unless otherwise stated, we set NMC = 30. All BNNs other than the infinitely-wide networks are implemented in TensorFlow v1. Models are trained using the Adam optimizer using the cosine annealing learning rate with a base learning rate of 10̆22123. All hidden layers use ReLU as the activation function, and no activation function is applied to the output layer.

Infinite-width neural networks are implemented using the Neural Tangents library. We use two different types of infinite networks: (1) ̆201cGP-̆201d refers to a closed-form expression for Gaussian process inference using the infinite-width neural network as a kernel, and (2) ̆201cInf-̆201d refers to an infinite ensemble of infinite-width networks that have been ̆201ctrained̆201d with continuous gradient descent for an infinite time. We compare NNGP and NTK kernels as well as the parameterization of the layers. By default, we use the NTK parameterization, but we also use the standard parameterization, denoted by ̆201c-std̆201d.

We implement BO using GPs with a Mat̆00e9rn kernel using the GPyOpt library. The library optimizes over the acquisition function in the inner loop using the L-BFGS algorithm.

8.8 Additional Results

8.8.1 Test Functions

We test BO on several common synthetic functions used for optimization, namely the Branin and 6-dimensional Hartmann functions. We use BNNs with 4 hidden layers and 256 units in each hidden layer, where each hidden layer is followed by a ReLU activation function. Plots of the best value ybest at each BO iteration are shown in Figure 8. As expected, GPs perform the best. Ensembles and BBB also perform competitively and much better than random sampling, showing that deep BO is viable even for low-dimensional black-box functions.

[width=0.45]figures/branin.png [width=0.45]figures/hartmann.png

Figure 3: BO results for the Branin and Hartmann-6 functions.

8.8.2 Nanoparticle Scattering

Detailed BO results for the nanoparticle scattering problem are shown in Table 3.

All the BNNs used for the nanoparticle scattering problem use an architecture consisting of 8 hidden layers with 256 units each, with the exception of BOHAMIANN where we used the original architecture consisting of 2 hidden layers with 50 units each. The infinite-width neural networks for the nanoparticle task consist of 8 hidden layers of infinite width, each of which are followed by ReLU activation functions.

[width=0.45]figures/narrowband$_bnn.png[width = 0.45]figures/highpass_bnn.png[width = 0.45]figures/narrowband_other.png[width = 0.45]figures/highpass_other.png$

Figure 4: Additional optimization result curves for the nanoparticle scattering task. (Top) Various BNNs. Note that results using auxiliary information are denoted by a solid line, while those that do not are denoted by a dashed line. Also note that the y-axis is zoomed in to differentiate the curves. (Bottom) Various non-BO algorithms. Ensemble-aux is replicated here for ease of comparison.

We also experiment with KL annealing in BBB, a proposed method to improve the performance of variational methods for BNNs in which the weight of the KL term in the loss function is slowly increased throughout training. For these experiments, we exponentially anneal the KL term with weight 0̆3c3KL(i) = 10i/5002̆2125 as a function of epoch i when training from scratch; during the continued training, the weight is held constant at 0̆3c3KL = 102̆2123.

KL annealing in the BBB architecture significantly improves performance for the narrowband objective, although results are mixed for the highpass objective. Additionally, KL annealing has the downside of introducing more parameters that must be carefully tuned for optimal performance. MNF performs poorly, especially on the highpass objective where it is comparable to random sampling, and we have found that MNF is quite sensitive to the choice of hyperparameters for uncertainty estimates even on simple regression problems.

The different variants infinite-width neural networks do not perform as well as the BNNs on both objective functions, despite the hyperparameter search.

LIPO seems to perform as well as GPs on both objective functions, which is impressive given the computational speed of the LIPO algorithm. Interestingly DIRECT-L does not perform as well as LIPO or GPs on the narrowband objective, and actually performs comparably to random sampling on the highpass objective. Additionally, CMA performs poorly on both objectives, likely due to the highly multimodal nature of the objective function landscape.

We also look at the effect of model size in terms of number of layers and units in Figure 10 for ensembles. While including auxiliary information clearly improves performance across all architectures, there is not a clear trend of performance with respect to the model size. Thus, the performance of BO seems to be somewhat robust to the exact architecture as long as the model is large enough to accurately and efficiently train on the data.

[width=0.5]figures/model$_size.png$

Figure 5: Comparison of ybest at N = 1000 for the nanoparticle narrowband objective function for a variety of neural network sizes. All results are ensembles, and 2̆01caux2̆01d denotes using auxiliary information.

Examples of the optimized structures by the 2̆01cEnsemble-aux2̆01d architecture are shown in Figure 11. We can see that the scattering spectra peak in the shaded region of interest, as desired by the respective objective functions.

[width=0.45]figures/narrowband$_optimized.png[width = 0.45]figures/highpass_optimized.png$

Figure 6: Examples of optimized nanoparticles and their scattering spectrum using the 2̆01cEnsemble-aux2̆01d architecture for the (a) narrowband and (c) highpass objectives. Orange shaded regions mark the range over which we wish to maximize the scattering.

8.8.3 Photonic Crystal

The BNN and BCNN architectures that we use for the PC task are listed in Table 4. The size of the 2̆01cFC2̆01d architectures are chosen to have a similar number of parameters as their convolutional counterparts. Unless otherwise stated, all results in the main text and here use the 2̆01cConv-TI2̆01d and 2̆01cFC2̆01d architectures for BCNNs and BNNs, respectively.

The infinite-width convolutional neural networks (which act as convolutional kernels for GPs) in the PC task consist of 5 convolutional layers followed by 4 fully-connected layers of infinite width. Because the pooling layers in the Neural Tangents library are currently too slow for use in application, we increased the size of the filters to 5 0̆0d7 5 to increase the receptive field of each filter.

Detailed BO results for the PC problem are shown in Table 5. For algorithms that optimize over the level set parameterization R51, we see that GPs perform consistently well, although BNNs using auxiliary information (e.g. Ensemble-Aux) can outperform GPs. DIRECT-L and CMA perform extremely well on the PC-A distribution but performs worse than GP on the PC-B distribution.

Adding convolutional layers and auxiliary information improves performance such that BCNNs significantly outperform GPs. Interestingly, the infinite-width networks perform extremely poorly, although this may be due to a lack of pooling layers in their architecture which limits the receptive field of the convolutions.

Examples of the optimized structures by the 2̆01cEnsemble-aux2̆01d architecture are shown in Figure 12. The photonic crystal unit cells generally converged to the same shape: a square lattice of silicon posts with periodicity.

Validation Metrics

Table 3: Various architectures for BNNs and BCNNs used in the PC problem. Numbers represent the number of channels and units for the convolutional and fully-connected layers, respectively. All convolutional layers use 3 ŏ0d7 3-sized filters with stride (1, 1) and periodic boundaries. ž01cMPž01d denotes max-pooling layers of size 2 ŏ0d7 2 with stride (2, 2), and ž01cAPž01d denotes average-pooling layers of size 2 ŏ0d7 2 with stride (1, 1). ž01cConvž01d denotes BCNNs whereas ž01cFCž01d denotes BNNs (containing only fully-connected layers) that act on the level-set parameterization x rather than on the image v. ž01cTIž01d denotes translation invariant architectures, whereas ž01cTDž01d denotes translation dependent architectures (i.e. not translation invariant).

Architecture	Convolutional Layers	Fully-Connected Layers
Conv-TI	16-MP-32-MP-64-MP-128-MP-256	256-256-256-256
Conv-TD	8-AP-8-MP-16-AP-32-MP-32-AP	256-256-256-256
FC	n/a	256-256-256-256-256

[width=0.45]figures/pc$_{ao}ptimized.png[width=0.45]figures/pc_{bo}ptimized.png$

Figure 7: Examples of optimized photonic crystal unit cells over multiple trials for (a) PC-A distribution and (c) PC-B distribution. (b,d) Examples of the optimized DOS. Note that the DOS has been minimized to nearly zero in a thin frequency range. Orange shaded regions mark the frequency range in which we wish to minimize the DOS. All results were optimized by the ž01cEnsemble-auxž01d architecture.

To explore more deeply why certain surrogate models perform well while others do not, we track various metrics of the model during BO on a validation dataset with 1000 randomly sampled data points. In particular, we look at the mean squared error (MSE), the mean absolute error (MAE), the negative log-likelihood (NLL), and the calibration error on the PC-A data distribution. Results are shown in Figure 13(a).

The calibration error is a quantitative measure of the uncertainty of the model, which is important for the performance of BO as the acquisition function uses the uncertainty to balance exploration and exploitation. Intuitively, we expect that a 50% confidence interval contains the correct answer 50

$$cal(F_1, y_1, ..., F_T, y_T) = \frac{1}{m}\sum_{j=1}^{m}(p_j - \hat{p}_j)^2 \tag{15}$$

where Fj is the CDF of the predictive distribution, pj is the confidence level, and ŏ2c6pj is the empirical frequency. We choose to measure the error along the confidence levels pj = (j ž212 1)/10 for j = 1, 2, ..., 11. The CDF Fj(yj) an be analytically calculated for models that have an analytical predictive distribution. For models that do not have an analytical predictive distribution, we use the empirical CDF:

$$F(y) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\mu^{(i)}\leq y} \tag{16}$$

where 1 is the indicator function. We also plot the calibration, (pj, ŏ2c6pj)M j=1, in Figure 13(b). Perfectly calibrated predictions correspond to a straight line.

[width=]figures/figure13.png

Figure 8: (a) Various metrics tracked during BO of the PC-A dataset distribution on a validation dataset of 1000 datapoints. (b) Uncertainty calibration curves measured at various points during BO. Note that the calibration curve for GP-aux is only shown for N = 50, as it becomes computationally intractable for larger N.

Figure 13 shows that the infinite neural network kernel (NTK) has the highest prediction error, which is likely a contributing factor to its poor BO performance. Interestingly, vanilla GPs have the lowest MSE, so the prediction error is not the only indicator for BO performance. Looking at the calibration, the infinite neural network kernel has the highest calibration error, and we see from Figure 13(b) that it tends to be overconfident in its predictions. GPs have a higher calibration error than the ensemble neural network methods and tend to be significantly underconfident in their predictions. GP-aux has higher validation loss, calibration error, and NLL than most, if not all, of the other methods, which explain its poor performance.

The ensemble NN methods tend to be reasonably well-calibrated. Within the ensemble NNs, the "-aux" methods have lower MSE and calibration error than their respective counterparts, and ConvEnsemble-aux has the lowest NLL calibration error out of all the methods, although interestingly Ensemble-aux seems to have the lowest MSE and MAE out of the ensemble NNs.

These results together show that calibration of Bayesian models is extremely important for use as surrogate models in BO.

8.8.4 Organic Molecule Quantum Chemistry

The Bayesian graph neural networks (BGNNs) used for the chemical property optimization task consist of 4 edge-conditioned graph convolutional layers with 32 channels each, followed by a global average pooling operation, followed by 4 fully-connected hidden layers of 64 units each. The edge-conditioned graph convolutional layers are implemented by Spektral.

More detailed results for the quantum chemistry dataset are shown in Table 6 and Figure 14. The architecture with the Bayes by Backprop variational approximation applied to every layer, including the graph convolutional layers (ˇ201cBBBˇ201d), performs extremely poorly, even worse than random sampling in some cases. However, only making the fully-connected layers Bayesian (ˇ201cBBB-FCˇ201d) performs surprisingly well.

Table 4: BO results for the four different quantum chemistry objective functions. ˇ2217 denotes that ybest is measured at N = 100 due to computational constraints.

	α		ϵ_{gap}		μ		$(\epsilon_{LUMO} - \epsilon_{HOMO})/2$	
Model	Mean	SD	Mean	SD	Mean	SD	Mean	SD
GP	0.41	0.04	-0.10	0.02	101.08	1.05	0.29	0.07
GraphGP	*0.62	0.00	*ˇ22120.10	0.02	*131.99	14.59	*0.24	0.03
Ensemble	0.62	0.00	-0.08	0.00	86.56	0.31	0.28	0.00
Ensemble-aux	0.62	0.00	-0.10	0.02	83.86	4.45	0.13	0.05
GraphEnsemble	0.62	0.00	-0.10	0.00	143.53	0.00	0.49	0.00
GraphEnsemble-aux	0.62	0.00	-0.10	0.00	143.53	0.00	0.49	0.00
GraphBBB	0.38	0.01	-0.11	0.01	94.46	1.16	0.25	0.01
GraphBBB-FC	0.62	0.00	-0.10	0.00	135.64	13.67	0.39	0.14
GraphNeuralLinear	0.62	0.00	-0.10	0.00	143.53	0.00	0.46	0.09
VAE-GP	0.62	0.06	-0.10	0.02	123.3 VAE-GP-2	-	-	-
-	110.84	16.68	0.56	0.35				
VAE-GP-latent128	-	-	-	-	154.66	35.96	0.40	0.10
VAE-GP-LATENT128-BETA0.001	-	-	-	-	133.66	13.25	0.42	0.13
VAE-GP-LATENT32	-	-	-	-	114.83	14.64	0.53	0.38
Random	0.38	0.02	-0.10	0.02	105.19	7.87	0.29	0.07

[width=0.45]figures/alpha.png [width=0.45]figures/gap.png

Figure 9: Additional BO results for several different objective functions on the chemistry dataset. GP and GraphEnsemble-aux curves are replicated from the main text for convenience.

Ensembles trained with auxiliary information (ˇ201cEnsemble-auxˇ201d) and neural linear (ˇ201cNeuralLinearˇ201d) perform the best on all objective functions. Adding auxiliary information to ensembles helps for the ˇ03b1 objective function, and neither helps nor hurts for the other objective functions. Additionally, BNNs perform at least as well or significantly better than GPs in all cases. GPs perform comparably or worse than random sampling in several cases.

As noted in the main text, the performance of VAE-GP depends on the quality of the pre-trained VAE, as shown in Figure 15. The VAE-GP benchmark uses the same pre-trained VAE, and ˇ201cVAE-GP-2ˇ201d refers to the same method using a different random seed for the VAE. Even with the exact same method, VAE-GP-2 performs significantly worse on both objective functions. We also increase the latent space dimensionality from 52 to 128 in the ˇ201cVAE-GP-LATENT128ˇ201d benchmark, which performs even worse on the ˇ03b1 ˇ2212 ˇ20acgap benchmark although it performs significantly better on the ˇ03c9 benchmark. We also adjust the learning rate momentum to ˇ03b7 = 0.001 in ˇ201cVAE-GP-LATENT128-BETA0.001ˇ201d, and the latent space dimensionality to 32 in ˇ201cVAE-GP-LATENT32ˇ201d. There is no clear trend with the different hyperparameters, which may point to the random seed of the VAE pre-training being a greater factor in BO performance than the hyperparameters.

[width=0.45]figures/vae$_alpha.png[width = 0.45]figures/vae_gap.png$

Figure 10: Additional BO results for VAE-GP using different pre-trained VAEs.

Validation Metrics

As in Appendix A.5.3, we track the MSE, NLL, and calibration error during optimization on the chemistry task. Results are shown in Figure 16. The various metrics correlate with the respective methodsˇ2019 performances during BO. For example, VAE-GP has an extremely high MSE and calibration error on the ˇ03b1 objective, where it performs poorly, but has an MSE and calibration error more comparable with that of other methods as well as an extremely low NLL on the ˇ03c9 ˇ2212 ˇ20acgap objective, where it performs extremely well. Likewise, the metrics for GRAPHGP are very high on the ˇ03b1 ˇ2212 ˇ20acgap objective, where it performs poorly. GraphEnsemble tends to be among the better methods in terms of these metrics, which translates into good BO performance.

[width=]figures/figure16.png

Figure 11: (a) Various metrics tracked during BO of the chemistry dataset on a validation dataset of 1000 datapoints. (b) Uncertainty calibration curves measured at various points during BO.

8.9 Additional Discussion

BBB performs reasonably well and is competitive with or even better than ensembles on some tasks, but it requires significant hyperparameter tuning. The tendency of variational methods such as BBB to underestimate uncertainty is likely detrimental to their performance in BO. Additionally, prior work shows that BBB has trouble scaling to larger network sizes, which may make them unsuitable for more complex tasks such as those in our work. BOHAMIANN performs very well on the nanoparticle narrowband objective and comparable to other BNNs without auxiliary information on the nanoparticle highpass objective. This is likely due to its effectiveness in exploring a multi-modal posterior. However, the need for SGHMC to sample the posterior makes this method computationally expensive, and so we were only able to run it for a limited number of iterations using a small neural network architecture.

Infinitely wide neural networks are another interesting research direction, as the ability to derive infinitely wide versions of various neural network architectures such as convolutions, and more recently graph convolutional layers, could potentially bring the power of GPs and BO to complex problems in low-data regimes. However, we find they perform relatively poorly in BO, are quite sensitive to hyperparameters (e.g. kernel and parameterization), and current implementations of certain operations such as pooling are too slow for practical use in an iterative setting. In particular, BO using an infinite ensemble of infinite-width networks performs poorly compared to normal ensembles, suggesting that the infinite-width formulations do not fully capture the dynamics of their finite-width counterparts.

Non-Bayesian global optimization methods such as LIPO and DIRECT-L are quite powerful in spite of their small computational overhead and can even outperform BO on some simpler tasks. However, they are not as consistent as BO, performing more comparably to random sampling on other tasks. CMA-ES performs poorly on all the tasks here. Also, like GPs, these non-Bayesian algorithms assume a continuous input space and cannot be effectively applied to structured, high-dimensional problems.

8.10 Compute

All experiments were carried out on systems with NVIDIA Volta V100 GPUs and Intel Xeon Gold 6248 CPUs. All training and inference using neural network-based models, graph kernels, and infinite-width neural network approximations are carried out on the GPUs. All other models are carried out on the CPUs.