Batch Multi-Fidelity Bayesian Optimization with Deep Auto-Regressive Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Bayesian optimization (BO) is a powerful approach for optimizing black-box, expensive-to-evaluate functions. To enable a flexible trade-off between the cost and accuracy, many applications allow the function to be evaluated at different fidelities. In order to reduce the optimization cost while maximizing the benefit-cost ratio, in this paper we propose Batch Multi-fidelity Bayesian Optimization with Deep Auto-Regressive Networks (BMBO-DARN). We use a set of Bayesian neural networks to construct a fully auto-regressive model, which is expressive enough to capture strong yet complex relationships across *all* the fidelities, so as to improve the surrogate learning and optimization performance. Furthermore, to enhance the quality and diversity of queries, we develop a simple yet efficient batch querying method, without any combinatorial search over the fidelities. We propose a batch acquisition function based on Max-value Entropy Search (MES) principle, which penalizes highly correlated queries and encourages diversity. We use posterior samples and moment matching to fulfill efficient computation of the acquisition function, and conduct alternating optimization over every fidelity-input pair, which guarantees an improvement at each step. We demonstrate the advantage of our approach on four real-world hyperparameter optimization applications.

1 Introduction

Many applications demand we optimize a complex function of an unknown form that is expensive to evaluate. Bayesian optimization (Mockus, 2012; Snoek et al., 2012) is a powerful approach to optimize such functions. The key idea is to use a probabilistic surrogate model, typically Gaussian processes (Rasmussen and Williams, 2006), to iteratively approximate the target function, integrate the posterior information to compute and maximize an acquisition function so as to generate new inputs at which to query, update the model with new examples, and meanwhile approach the optimum.

In practice, to enable a flexible trade-off between the computational cost and accuracy, many applications allow us to evaluate the target function at different fidelities. For example, to evaluate the performance of the hyperparameters for a machine learning model, we can train the model thoroughly, *i.e.*, with sufficient iterations/epochs, to obtain the accurate evaluation (high-fidelity yet often costly) or just run a few iterations/epochs to obtain a rough estimate (low-fidelity but much cheaper).

Many multi-fidelity BO algorithms (Lam et al., 2015; Kandasamy et al., 2016; Zhang et al., 2017; Song et al., 2019; Takeno et al., 2019) have therefore been proposed to identify both the fidelities and inputs at which to query, so as to reduce the cost and achieve a good benefit-cost balance. Notwithstanding their success, these methods often overlook the strong yet complex relationships between different fidelities or adopt an over-simplified assumption, (partly) for the sake of convenience in calculating/maximizing the acquisition function considering fidelities. This, however, can restrict the performance of the surrogate model, impair the optimization efficiency and increase the cost. For

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

example, Lam et al. (2015); Kandasamy et al. (2016) learned an independent GP for each fidelity, Zhang et al. (2017) used multitask GPs with a convolved kernel for multi-fidelity modeling and have to use a simple smoothing kernel(*e.g.*, Gaussian) for tractable convolutions. The recent work (Takeno et al., 2019) imposes a linear correlation across different fidelities. In addition, the standard one-by-one querying strategy can underrate the correlation between successive queries, is at higher risk to bring in highly correlated or redundant examples. As a result, it can further lower the learning efficiency and lead to suboptimal total costs.

To overcome these limitations, we propose BMBO-DARN, a novel batch multi-fidelity Bayesian optimization method. First, we develop a deep auto-regressive model to integrate training examples at various fidelities. Each fidelity is modeled by a Bayesian neural network (NN), where the output predicts the objective function value at that fidelity and the input consists of the original inputs and the outputs of *all* the previous fidelities. In this way, our model is adequate to capture the complex, strong correlations (*e.g.*, nonstationary, highly nonlinear) across all the fidelities to enhance the surrogate learning. We use Hamiltonian Monte-Carlo (HMC) sampling for posterior inference. Next, to improve the quality of the queries, we develop a simple yet efficient method to jointly fetch a batch of inputs and fidelities. Specifically, we propose a batch acquisition function based on the state-of-the-art Max-value Entropy Search (MES) principle (Wang and Jegelka, 2017). The batch acquisition function explicitly penalizes highly correlated queries and encourages diversity. To efficiently compute the acquisition function, we use the posterior samples of the NN weights and moment matching to construct a multi-variate Gaussian posterior for all the fidelity outputs and the function optimum. To prevent a combinatorial search over multiple fidelities in maximizing the acquisition function, we develop an alternating optimization algorithm to cyclically update each pair of input and fidelity, which is much more efficient and guarantees an improvement at each step.

For evaluation, we examined BMBO-DARN in both synthetic benchmarks and real-world applications. The synthetic benchmark tasks show that given a small number of training examples, our deep auto-regressive model can learn a more accurate surrogate of the target function than other state-of-the-art multi-fidelity BO models. We then evaluated BMBO-DARN on four popular machine learning models (CNN, online LDA, XGBoost and Physics informed NNs) for hyperparameter optimization. BMBO-DARN can find more effective hyperparameters leading to superior predictive performance, and meanwhile spends smaller evaluation costs, as compared with state-of-the-art multi-fidelity BO algorithms and other popular hyperparameter tuning methods.

2 Background

Bayesian Optimization (BO) (Mockus et al., 1978; Snoek et al., 2012) is a popular approach for optimizing black-box functions that are often costly to evaluate and cannot provide exact gradient information. BO learns a probabilistic surrogate model to predict the function value across the input space and quantifies the predictive uncertainty. At each step, we use this information to compute an acquisition function to measure the utility of querying at different inputs. By maximizing the acquisition function, we find the next input at which to query, which is supposed to be closer to the optimum. Then we add the new example into the training set to improve the accuracy of the surrogate model. The procedure is repeated until we find the optimal input or the maximum number of queries have been finished. There are a variety of acquisition functions, such as Expected Improvement (EI) (Mockus et al., 1978) and Upper Confidence Bound (UCB) (Srinivas et al., 2010). The recent state-of-the-art addition is Maximum-value Entropy Search (MES) (Wang and Jegelka, 2017),

$$a(\mathbf{x}) = \mathbb{I}\big(f(\mathbf{x}), f^*|\mathcal{D}\big), \tag{1}$$

where $\mathbb{I}(\cdot,\cdot)$ is the mutual information, $f(\mathbf{x})$ is the objective function value at $\mathbf{x}$, f^* the minimum, and $\mathcal{D}$ the training data collected so far for the surrogate model. Note that both $f(\mathbf{x})$ and f^* are considered as generated by the posterior of the surrogate model given $\mathcal{D}$; they are random variables.

The most commonly used surrogate is Gaussian process (GP) (Rasmussen and Williams, 2006). Given the training dataset $\mathbf{X} = [\mathbf{x}_1^\top, \ldots, \mathbf{x}_N^\top]^\top$ and $\mathbf{y} = [y_1, \ldots, y_N]^\top$, a GP assumes the outputs $\mathbf{y}$ follow a multivariate Gaussian distribution, $p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\mathbf{m}, \mathbf{K} + v\mathbf{I})$, where $\mathbf{m}$ is the mean function values at the inputs $\mathbf{X}$, often set to $\mathbf{0}$, v is the noise variance, and $\mathbf{K}$ is a kernel matrix on $\mathbf{X}$. Each $[\mathbf{K}]_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, where $\kappa(\cdot,\cdot)$ is a kernel function. For example, a popular one is the RBF kernel, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\beta^{-1}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$. An important advantage of GPs is their convenience in uncertainty quantification. Since GPs assume any finite set of function values follow a multi-variate Gaussian distribution, given a test input $\hat{\mathbf{x}}$, we can compute the predictive (or posterior) distribution $p\left(f(\hat{\mathbf{x}})|\hat{\mathbf{x}}, \mathbf{X}, \mathbf{y}\right)$ via a conditional Gaussian distribution, which is simple and analytical.

Multi-Fidelity BO. Since the evaluation of the exact object function value is often expensive, many practical applications provide multi-fidelity evaluations $\{f_1(\mathbf{x}), \ldots, f_M(\mathbf{x})\}$ to allow us to choose a trade-off between the accuracy and cost. Accordingly, many multi-fidelity BO algorithms have been developed to select both the inputs and fidelities to reduce the cost and to achieve a good balance between the optimization progress and cost, *i.e.*, the benefit-cost ratio. For instance, MF-GP-UCB (Kandasamy et al., 2016) sequentially queries at each fidelity (from the lowest one, *i.e.*, $m = 1$) until the confidence band is over a given threshold. In spite of its great success and guarantees in theory, MF-GP-UCB uses a set of independent GPs to estimate the objective at each fidelity, and hence ignores the valuable correlations between different fidelities. MF-PES (Zhang et al., 2017) uses a multi-task GP surrogate where each task corresponds to one fidelity, and convolves a smoothing kernel with the kernel of a shared latent function to obtain the cross-covariance. The recent MF-MES (Takeno et al., 2019) also builds a multi-task GP surrogate, where the covariance function is

$$\kappa\left(f_m(\mathbf{x}), f_{m'}(\mathbf{x}')\right) = \sum\nolimits_{j=1}^{d} \left(u_{mj} u_{m'j} + \mathbb{1}(m = m') \cdot \alpha_{mj}\right) \rho_j(\mathbf{x}_1, \mathbf{x}_2), \tag{2}$$

where $\alpha_{mj} > 0$, $\mathbb{1}(\cdot)$ is the indicator function, $\{u_{mj}\}_{j=1}^{d}$ is d latent features for each fidelity m, and $\{\rho_j(\cdot, \cdot)\}$ are d bases kernels, usually chosen as a commonly used stationary kernel, *e.g.*, RBF.

3 Deep Auto-Regressive Model for Multi-Fidelity Surrogate Learning

Notwithstanding the elegance and success of the existing multi-fidelity BO methods, they often ignore or oversimplify the complex, strong correlations between different fidelities, and hence can be inefficient for surrogate learning, which might further lower the optimization efficiency and incur more expenses. For example, the state-of-the-art methods MF-GP-UCB (Kandasamy et al., 2016) estimate a GP surrogate for each fidelity independently; MF-PES (Zhang et al., 2017) has to adopt a simple form for both the smoothing and latent function kernel (*e.g.*, Gaussian and delta) to achieve an analytically tractable convolution, which might limit the expressive in estimating the cross-fidelity covariance; MF-MES (Takeno et al., 2019) essentially imposes a linear correlation structure between different fidelities — for any input $\mathbf{x}$, $\kappa\left(f_m(\mathbf{x}), f_{m'}(\mathbf{x})\right) = \mathbf{u}_{m_1}^{\top} \mathbf{u}_{m_2} + \alpha_m$ where $\mathbf{u}_m = [u_{m1}, \ldots, u_{md}]$ and $\widetilde{\alpha}_m = \sum_{j=1}^{d} \alpha_{mj}$ if we use a RBF basis kernel (see (2)).

To overcome this limitation, we develop a deep auto-regressive model for multi-fidelity surrogate learning. Our model is expressive enough to capture the strong, possibly very complex (*e.g.*, highly nonlinear, nonstationary) relationships between all the fidelities to improve the prediction (at the highest fidelity). As such, our model can more effectively integrate multi-fidelity training information to better estimate the objective function.

Specifically, given M fidelities, we introduce a chain of M neural networks, each of which models one fidelity and predicts the target function at that fidelity. Denote by $\mathbf{x}_m$, $\mathcal{W}_m$, and $\psi_{\mathcal{W}_m}(\cdot)$ the NN input, parameters and output mapping at each fidelity m. Our model is defined as follows,

$$\begin{aligned} \mathbf{x}_m &= [\mathbf{x}; f_1(\mathbf{x}); \ldots; f_{m-1}(\mathbf{x})], \\ f_m(\mathbf{x}) &= \psi_{\mathcal{W}_m}(\mathbf{x}_m), \quad y_m(\mathbf{x}) = f_m(\mathbf{x}) + \epsilon_m, \end{aligned} \tag{3}$$

where $\mathbf{x}_1 = \mathbf{x}$, $f_m(\mathbf{x})$ is the prediction (*i.e.*, NN output) at the m-th fidelity, $y_m(\mathbf{x})$ is the observed function value, and ϵ_m is a random noise, $\epsilon_m \sim \mathcal{N}(\epsilon_m | 0, \tau_m^{-1})$. We can see that each input $\mathbf{x}_m$ consists of not only the original input $\mathbf{x}$ of the objective function, but also the outputs from *all* the previous fidelities. Via a series of linear transformation and nonlinear activation inside the NN, we obtain the output at fidelity m. In this way, our model fully exploits the information from the lower fidelities and can flexibly capture arbitrarily complex relationships between the current and *all* the previous fidelities by learning a NN mapping, $f_m(\mathbf{x}) = \psi_{\mathcal{W}_m}\big(\mathbf{x}_m, f_1(\mathbf{x}), \ldots, f_{m-1}(\mathbf{x})\big)$.

We assign a standard Gaussian prior distribution over each element of the NN parameters $\mathcal{W} = \{\mathcal{W}_1, \ldots, \mathcal{W}_M\}$, and a Gamma prior over each noise precision, $p(\tau_m) = \text{Gam}(\tau_m | a_0, b_0)$. Given the dataset $\mathcal{D} = \{\{(\mathbf{x}_{nm}, y_{nm})\}_{n=1}^{N_m}\}_{m=1}^{M}$, the joint probability of our model is given by

$$p(\mathcal{W}, \tau, \mathcal{Y}, \mathcal{S} | \mathcal{X}) = \mathcal{N}(\text{vec}(\mathcal{W}) | \mathbf{0}, \mathbf{I}) \prod_{m=1}^{M} \text{Gam}(\tau_m | a_0, b_0) \prod_{m=1}^{M} \prod_{n=1}^{N_m} \mathcal{N}\left(y_{nm} | f_m(\mathbf{x}_{nm}), \tau_m^{-1}\right), \tag{4}$$

where $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_M]$, $\mathcal{X} = \{\mathbf{x}_{nm}\}$, $\mathcal{Y} = \{y_{nm}\}$, and $\mathrm{vec}(\cdot)$ is vectorization. The graphical representation of our model is given in Fig. 1 of the Appendix. We use Hamiltonian Monte Carlo (HMC) (Neal et al., 2011) sampling to perform posterior inference due to its unbiased, high-quality uncertainty quantification, which is critical to calculate the acquisition function. However, our method allows us to readily switch to other approximate inference approaches as needed (see Sec. 4), *e.g.*, stochastic gradient HMC (Chen et al., 2014) used in the excellent work (Springenberg et al., 2016).

4 Batch Acquisition for Multi-Fidelity Optimization

Given the posterior of our model, we aim to compute and optimize an acquisition function to identify the input and fidelity at which to query next. Popular BO methods query at one input each time and then update the surrogate model. While being successful, this one-by-one strategy can underrate the correlations between consecutive queries. As a result, highly correlated examples are at a much higher risk to be queried and incorporated into the training set. The redundant information within these examples can restrict the learning efficiency, and further lower the optimization efficiency and increase the expense. To improve the query quality and diversity, we develop a batch acquiring approach to jointly identify a set of inputs and fidelities at a time, presented as follows.

4.1 Batch Acquisition Function

We first propose a batch acquisition function based on the MES principle (Zhang et al., 2017) (see (1)). Denote by B the batch size and by $\{\lambda_1, \ldots, \lambda_M\}$ the cost of querying at M fidelities. We want to jointly identify B pairs of inputs and fidelities $(\mathbf{x}_1, \mathbf{m}_1), \ldots, (\mathbf{x}_B, \mathbf{m}_B)$ at which to query. The batch acquisition function is given by

$$a_{\text{batch}}(\mathbf{X}, \mathbf{m}) = \frac{\mathbb{I}(\{f_{m_1}(\mathbf{x}_1), \ldots, f_{m_B}(\mathbf{x}_B)\}, f^* | \mathcal{D})}{\sum_{k=1}^{B} \lambda_{m_k}}, \tag{5}$$

where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_B\}$ and $\mathbf{m} = [m_1, \ldots, m_B]$. As we can see, our batch acquisition function explicitly penalizes highly correlated queries and encourages diversity — if between the outputs $\{f_{m_k}(\mathbf{x}_k)\}_{k=1}^{B}$ are high correlations, the mutual information in the numerator will decrease. Furthermore, by dividing the total querying cost in (5), the batch acquisition function expresses a balance between the benefit of these queries (in probing the optimum) and the price, *i.e.*, benefit-cost ratio. When we set $B = 1$, our batch acquisition function is reduced to the single one used in (Takeno et al., 2019).

4.2 Efficient Computation

Given $\mathbf{X}$ and $\mathbf{m}$, the computation of (5) is challenging, because it involves the mutual information between a set of NN outputs and the function optimum. To address this challenge, we use posterior samples and moment matching to approximate $p(\mathbf{f}, f^* | \mathcal{D})$ as a multi-variate Gaussian distribution, where $\mathbf{f} = [f_{m_1}(\mathbf{x}_1), \ldots, f_{m_B}(\mathbf{x}_B)]$. Specifically, we first draw a posterior sample of the NN weights $\mathcal{W}$ from our model. We then calculate the output at each input and fidelity to obtain a sample of $\mathbf{f}$, and maximize (or minimize) $f_M(\cdot)$ to obtain a sample of f^*. We use L-BFGS (Liu and Nocedal, 1989) for optimization. After we collect L independent samples $\{(\widehat{\mathbf{f}}_1, \widehat{f}_1^*), \ldots, (\widehat{\mathbf{f}}_L, \widehat{f}_L^*)\}$, we can estimate the first and second moments of $\mathbf{h} = [\mathbf{f}; f^*]$, namely, mean and covariance matrix,

$$\boldsymbol{\mu} = \frac{1}{L} \sum_{j=1}^{L} \widehat{\mathbf{h}}_j, \quad \boldsymbol{\Sigma} = \frac{1}{L-1} \sum_{j=1}^{L} (\widehat{\mathbf{h}}_j - \boldsymbol{\mu})(\widehat{\mathbf{h}}_j - \boldsymbol{\mu})^\top,$$

where each $\widehat{\mathbf{h}}_j = [\widehat{\mathbf{f}}_j; \widehat{f}_j^*]$. We then use these moments to match a multivariate Gaussian posterior, $p(\mathbf{h}|\mathcal{D}) \approx \mathcal{N}(\mathbf{h}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Then the mutual information can be computed with a closed form,

$$\mathbb{I}(\mathbf{f}, f^* | \mathcal{D}) = \mathbb{H}(\mathbf{f}|\mathcal{D}) + \mathbb{H}(f^*|\mathcal{D}) - \mathbb{H}(\mathbf{f}, f^*|\mathcal{D}) \approx \frac{1}{2} \log |\boldsymbol{\Sigma}_{\mathbf{ff}}| + \frac{1}{2} \log \sigma_{**} - \frac{1}{2} \log |\boldsymbol{\Sigma}|, \tag{6}$$

where $\boldsymbol{\Sigma}_{\mathbf{ff}} = \boldsymbol{\Sigma}[1:B, 1:B]$, *i.e.*, the first $B \times B$ sub-matrix along the diagonal, which is the posterior covariance of $\mathbf{f}$, and $\sigma_{**} = \boldsymbol{\Sigma}[B+1, B+1]$, *i.e.*, the posterior variance of f^*. The batch acquisition function is therefore calculated from

$$a_{\text{batch}}(\mathbf{X}, \mathbf{m}) \approx \frac{1}{2 \sum_{k=1}^{B} \lambda_{m_k}} \left(\log |\boldsymbol{\Sigma}_{\mathbf{ff}}| + \log \sigma_{**} - \log |\boldsymbol{\Sigma}| \right). \tag{7}$$

Note that $\mathbf{\Sigma}$ is a function of the inputs $\mathbf{X}$ and fidelities $\mathbf{m}$ and hence so are its submatrix and elements, $\mathbf{\Sigma}_{\mathbf{ff}}$ and σ_{**}. To obtain a reliable estimate of the moments, we set $L = 100$ in our experiments. Note that our method can be applied along with any posterior inference algorithm, such as variational inference and SGHMC (Chen et al., 2014), as long as we can generate posterior samples of the NN weights, not restricted to the HMC adopted in our paper.

4.3 Optimizing a Batch of Fidelities and Inputs

Now, we consider maximizing (7) to identify B inputs $\mathbf{X}$ and their fidelities $\mathbf{m}$ at which to query. However, since the optimization involves a mix of continuous inputs and discrete fidelities, it is quite challenging. A straightforward approach would be to enumerate all possible configurations of $\mathbf{m}$, for each particular configuration, run a gradient based optimization algorithm to find the optimal inputs, and then pick the configuration and its optimal inputs that give the largest value of the acquisition function. However, doing so is essentially conducting a combinatorial search over B fidelities, and the search space grows exponentially with B, *i.e.*, $\mathcal{O}(M^B) = \mathcal{O}(e^{B \log M})$. Hence, it will be very costly, even infeasible for a moderate choice of B.

To address this issue, we develop an alternating optimization algorithm. Specifically, we first initialize all the B queries, $\mathcal{Q} = \{(\mathbf{x}_1, m_1), \ldots, (\mathbf{x}_B, m_B)\}$, say, randomly. Then each time, we only optimize one pair of the input and fidelity $(\mathbf{x}_k, m_k)(1 \leq k \leq B)$, while fixing the others. We cyclically update each pair, where each update is much cheaper but guarantees to increase a_{batch}. Specifically, each time, we maximize

$$a_{\text{batch},k}(\mathbf{x}, m) = \frac{\mathbb{I}(\mathcal{F}_{\neg k} \cup \{f_m(\mathbf{x})\}, f^* | \mathcal{D})}{\lambda_m + \sum_{j \neq k} \lambda_{m_j}}, \tag{8}$$

where $\mathcal{F}_{\neg k} = \{f_{m_j}(\mathbf{x}_j) | j \neq k\}$. Note that the computation of (8) still follows (7). We set $(\mathbf{x}_k, m_k)$ to the optimum $(\mathbf{x}^*, m^*)$, and then proceed to optimize the next input location and fidelity $(\mathbf{x}_{k+1}, m_{k+1})$ in $\mathcal{Q}$ with the others fixed. We continues this until we finish updating all the queries in $\mathcal{Q}$, which corresponds to one iteration. We can keep running iterations until the increase of the batch acquisition function is less than a tolerance level or a maximum number of iterations has been reached. Suppose we ran G iterations, the time complexity is $\mathcal{O}(GMB)$, which is linear in the number of fidelities and batch size, and hence is much more efficient than the naive combinatorial search. Our multi-fidelity BO approach is summarized in Algorithm 1.

Algorithm 1 BMBO-DARN ($\mathcal{D}$, B, M, T, $\{\lambda_m\}_{m=1}^M$)

Learn the deep auto-regressive model (4) on $\mathcal{D}$ with HMC.
for $t = 1, \ldots, T$ **do**
  Collect a batch of B queries, $\mathcal{Q} = \{(\mathbf{x}_k, m_k)\}_{k=1}^B$, with Algorithm 2.
  Query the objective function value at each input $\mathbf{x}_k$ and fidelity m_k in $\mathcal{Q}$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_k, y_k, m_k) | 1 \leq k \leq B\}$.
  Re-train the deep auto-regressive model on $\mathcal{D}$ with HMC.
end for

Algorithm 2 BatchAcquisition($\{\lambda_m\}$, B, L, G, ξ)

Initialize $\mathcal{Q} = \{(\mathbf{x}_1, m_1), \ldots, (\mathbf{x}_B, m_B)\}$ randomly.
Collect L independent posterior samples of the NN weights.
repeat
  for $k = 1, \ldots, B$ **do**
    Use the posterior samples to calculate and optimize (8),

$$(\mathbf{x}^*, m^*) = \operatorname*{argmax}_{\mathbf{x} \in \Omega, 1 \leq m \leq M} a_{\text{batch},k}(\mathbf{x}, m),$$

    where Ω is the input domain.
    $(\mathbf{x}_k, m_k) \leftarrow (\mathbf{x}^*, m^*)$.
  end for
until G iterations are done or the increase of a_{batch} in (7) is less than ξ
Return $\mathcal{Q}$.

5 Related Work

Most of Bayesian optimization (BO) (Mockus, 2012; Snoek et al., 2012) methods are based on Gaussian processes (GPs) and a variety of acquisition functions, such as (Mockus et al., 1978; Auer, 2002; Srinivas et al., 2010; Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Wang and Jegelka, 2017; Kandasamy et al., 2017b; Garrido-Merchán and Hernández-Lobato, 2020). Snoek et al. (2015) showed Bayesian neural networks (NNs) can also be used as a general surrogate model, and has excellent performance. Moreover, the training of NNs is also scalable, not suffering from $\mathcal{O}(N^3)$ time complexity (N is the number of examples). Springenberg et al. (2016) further use scale adaption to develop a robust stochastic gradient HMC for the posterior inference in the NN based BO. Recent works that deal with discrete inputs (Baptista and Poloczek, 2018) or mixed discrete and continuous inputs (Daxberger et al., 2019) use an explicit nonlinear feature mapping and Bayesian linear regression, which can be viewed as one-layer Bayesian NNs.

There have been many studies in multi-fidelity BO, *e.g.*, (Huang et al., 2006; Lam et al., 2015; Picheny et al., 2013; Kandasamy et al., 2016, 2017a; Poloczek et al., 2017; McLeod et al., 2017; Wu and Frazier, 2017). Despite being successful, these methods either ignore or oversimplify the strong, complex correlations between different fidelities, and hence might be inefficient in surrogate learning. For example, Picheny et al. (2013); Lam et al. (2015); Kandasamy et al. (2016); Poloczek et al. (2017) learned an independent GP for each fidelity; Song et al. (2019) used all the examples without discrimination to train one single GP; Huang et al. (2006); Takeno et al. (2019) imposed a linear correlation across fidelities, while Zhang et al. (2017) constructed a convolutional kernel as the cross-fidelity covariance and so the involved kernels in the convolution must be simple and smooth enough (yet less expressive) to obtain a closed form. Recently, Perrone et al. (2018) developed an NN-based multi-task BO method for hyper-parameter transfer learning. Their model constructs an NN feature mapping shared by all the tasks, and uses an independent linear combination of the mapped features to predict each task output. While we can consider each task as evaluating the objective at a particular fidelity, the model does not explicitly capture and exploit the correlations across different tasks — given the shared (latent) features, the predictions of these tasks (fidelities) are independent. The most recent work (Li et al., 2020) also developed an NN-based multi-fidelity BO method, which differs from our work in that (1) their model only estimates the relationship between successive fidelities, and hence has less capacity, (2) their work uses a recursive one-dimensional quadrature to calculate the acquisition function, and is difficult to extend to batch acquisitions.

While in our experiments, we mainly use hyperparameter optimization to evaluate our multi-fidelity BO approach, there are many other excellent works specifically designed for hyperparameter tuning/selection, *e.g.*, the non-Bayesian, random search based method Hyberband (Li et al., 2017) which also reflects the multi-fidelity idea: it starts using few training iterations/epochs (low fidelity) to evaluate many candidates, rank them, iteratively selects the top-ranked ones, and further evaluate them with more iterations/epochs (high fidelity). BOHB (Falkner et al., 2018) is a hybrid of KDE based BO (Bergstra et al., 2011) and Hyperband. Li et al. (2018) further developed an asynchronous successive halving algorithm for parallel random search over hyperparameters. Domhan et al. (2015); Klein et al. (2017b) propose to estimate the learning curves, and early halt the evaluation of ominous hyperparameters according to the learning curve predictions. Swersky et al. (2014) introduce a kernel about the training steps, and develops Freeze-thaw BO (Swersky et al., 2014) that can temporarily pause the model training and explore several promising hyperparameter settings for a while and then continue on to the most promising one. The work in (Klein et al., 2017a) jointly estimates the cost as a function of the data size and training steps, which can be viewed as continuous fidelities, like in (Kandasamy et al., 2017a; Wu and Frazier, 2017).

6 Experiment

6.1 Surrogate Learning Performance

We first examined if BMBO-DARN can learn a more accurate surrogate of the objective. We used two popular benchmark functions: (1) *Levy* (Laguna and Martí, 2005) with two-fidelity evaluations, and (2) *Branin* (Forrester et al., 2008; Perdikaris et al., 2017) with three-fidelity evaluations. Throughout different fidelities are nonlinear/nonstationary transforms. We provide the details in the Appendix.

Methods. We compared with the following multi-fidelity learning models used in the state-of-the-art BO methods: (1) MF-GP-UCB (Kandasamy et al., 2016) that learns an independent GP

for each fidelity. (2) MF-MES (Takeno et al., 2019) that uses a multi-output GP with a linear correlation structure across different outputs (fidelities), (3) Scalable Hyperparameter Transfer Learning (SHTL) (Perrone et al., 2018) that uses NN to generate latent bases shared by all the tasks (fidelities) and predicts the output of each task with a linear combination of the bases. (4) Deep Neural Network Multi-Fidelity BO (DNN-MFBO) (Li et al., 2020) that also uses a chain of NNs to model each fidelity, but only estimates the relationship between successive fidelities.

Settings. We implemented our model with PyTorch (Paszke et al., 2019) and HMC sampling based on the Hamiltorch library (`https://github.com/AdamCobb/hamiltorch`). For each fidelity, we used two hidden layers with 40 neurons and `tanh` activation. We ran HMC for 5K steps to reach burn in (by looking at the trace plots) and then produced 200 posterior samples with every 10 steps. To generate each sample proposal, we ran 10 leapfrog steps, and the step size was chosen as 0.012. We implemented DNN-MFBO and SHTL with PyTorch as well. For DNN-MFBO, we used the same NN architecture as in BMBO-DARN for each fidelity, and ran HMC with the same setting for model estimation. For SHTL, we used two hidden layers with 40 neurons and an output layer with 32 neurons to generate the shared bases. We used ADAM (Kingma and Ba, 2014) to estimate the model parameters, and the learning rate was chosen from $\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$. We ran 1K epochs, which are enough for convergence. We used a Python implementation of MF-MES and MF-GP-UCB, both of which use the RBF kernel (consistent with the original papers).

Results. We randomly generated $\{130, 65\}$ examples for *Levy* function at the two increasing fidelities, and $\{320, 130, 65\}$ examples for *Branin* function at its three increasing fidelities. After training, we examined the prediction accuracy of all the models with 100 test samples uniformly sampled from the input space. We calculated the normalized root-mean-square-error (nRMSE) and mean-negative-log-likelihood (MNLL). We repeated the experiment for 5 times, and report their average and standard deviations in Table. 1. As we can see, for both benchmark functions, BMBO-DARN outperforms all the competing models, confirming the advantage of our deep auto-regressive model in surrogate learning. Note that despite using a similar chain structure, DNN-MFBO is still inferior to BMBO-DARN, implying that our fully auto-regressive modeling (see (3)) can better estimate the relationships between the fidelities to facilitate surrogate estimation.

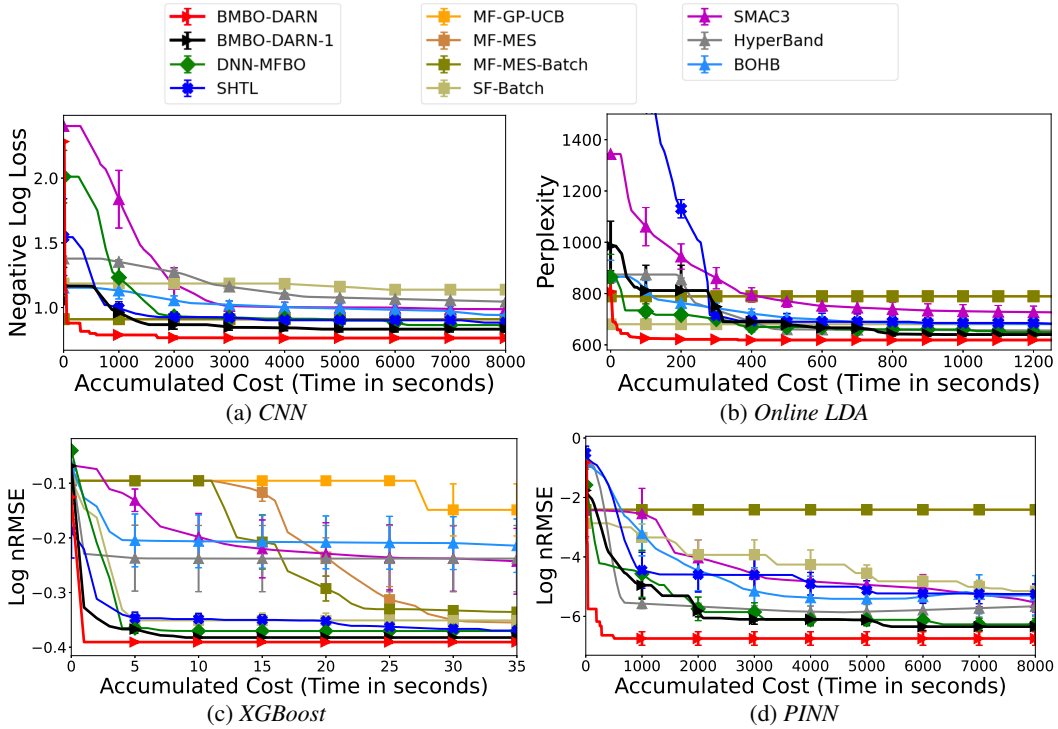
Levy	nRMSE	MNLL
MF-GP-UCB	0.831 ± 0.195	1.824 ± 0.276
MF-MES	0.581 ± 0.032	1.401 ± 0.031
SHTL	0.443 ± 0.009	1.208 ± 0.026
DNN-MFBO	0.365 ± 0.035	1.081 ± 0.011
BMBO-DARN	$\mathbf{0.348 \pm 0.021}$	$\mathbf{1.072 \pm 0.016}$
Branin		
MF-GP-UCB	0.846 ± 0.147	1.976 ± 0.208
MF-MES	0.719 ± 0.099	1.796 ± 0.128
SHTL	0.835 ± 0.218	1.958 ± 0.646
DNN-MFBO	0.182 ± 0.022	0.973 ± 0.013
BMBO-DARN	$\mathbf{0.158 \pm 0.016}$	$\mathbf{0.965 \pm 0.005}$

Table 1: Surrogate learning performance on *Branin* function with three-fidelity training examples and *Levy* function with two-fidelity examples: normalized root-mean-square-error (nRMSE) and mean-negative-log-likelihood (MNLL). The results were averaged over five runs.

6.2 Real-World Applications

Next, we used BMBO-DARN to optimize the hyperparameters of four popular machine learning models: Convolutional Neural Networks (CNN) (Fukushima and Miyake, 1982; LeCun et al., 1990) for image classification, Online Latent Dirichlet Allocation (LDA) (Hoffman et al., 2010) for text mining, XGBoost (Chen and Guestrin, 2016) for diabetes diagnosis, and Physics-Informed Neural Networks (PINN) (Raissi et al., 2019) for solving partial differential equations (PDE).

Methods and Setting. We compared with the state-of-the-art multi-fidelity BO algorithms mentioned in Sec. 6.1, (1) MF-GP-UCB, (2) MF-MES, (3) SHTL, and (4) DNN-MFBO. In addition, we compared with (5) MF-MES-Batch (Takeno et al., 2019), the (asynchronous) parallel version of MF-MES, (6) SF-Batch (Kandasamy et al., 2017b) (`https://github.com/kirthevasank/gp-parallel-ts`), a single-fidelity GP-based BO that optimizes posterior samples of the objective function to obtain a batch of queries, (7) SMAC3 (`https://github.com/automl/SMAC3`), BO based on random forests, (8) Hyperband (Li et al., 2017) (`https://github.com/automl/HpBandSter`) that conducts multi-fidelity random search over the hyperparameters, (9) BOHB (Falkner et al., 2018) that uses Tree Parzen Estimator (TPE) (Bergstra et al., 2011) to generate hyperparameter candidates in Hyperband iterations. We also tested our method that queries at one input and fidelity each time ($B = 1$), which we denote by BMBO-DARN-1. We used



(a) *CNN* (b) *Online LDA* (c) *XGBoost* (d) *PINN*

Figure 1: Performance *vs.* accumulated cost (running time) in Hyperparameter optimization tasks. The results were averaged over five runs. Note that MF-GP-UCB, MF-MES and MF-MES-Batch often obtained very close results and their curves overlap much.

the same setting as in Sec. 6.1 for all the multi-fidelity methods, except that for SHTL, we ran 2K epochs in surrogate training to ensure the convergence. For our method, we set the maximum number of iterations in optimizing the batch acquisition function (see Algorithm 8) to 100 and tolerance level to 10^{-3}. For the remaining methods, *e.g.*, SMAC3 and Hyperband, we used their original implementations and default settings. For all the batch querying methods, we set the batch size to 5. All the single fidelity methods queried at the highest fidelity.

Convolutional Neural Network (CNN). Our first application is to train a CNN for image classification. We used CIFAR10 dataset (`https://www.cs.toronto.edu/~kriz/cifar.html`), from which we used 10K images for training and another 10K for evaluation. To optimize the hyperparameters, we considered three fidelities, *i.e.*, training with 1, 10, 50 epochs. We used the average negative log-loss (nLL) to evaluate the prediction accuracy of each method. We considered optimizing the following hyperparameters: # convolutional layers ranging from [1,4], # channels in the first filter ([8, 136]), depth of the dense layers ([1, 8]), width of the dense layers ([32, 2080]), pooling type ([“max”, “average”]), and dropout rate ([10^{-3}, 0.99]). We optimized the dropout rate in the log domain, and used a continuous relaxation of the discrete parameters.

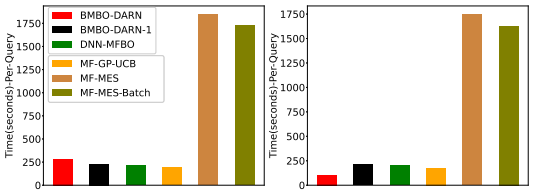
Initially, we queried at 10 random hyperparameter settings at each fidelity. All the methods started with these evaluation results and repeatedly identified new hyperparameters. We used the average running time at each training fidelity as the cost: $\lambda_1 : \lambda_2 : \lambda_3 = 1 : 10 : 50$. After each query, we evaluated the performance of the new hyperparameters at the highest training level. We ran each method until 100 queries were issued. We repeated the experiment for 5 times and in Fig. 1a report the average accuracy (nLL) and its standard deviation for the hyperparameters found by each method throughout the optimization procedure.

Online Latent Dirichlet Allocation (LDA). Our second task is to train online LDA (Hoffman et al., 2010) to extract topics from 20NewsGroups corpus (`http://qwone.com/~jason/20Newsgroups/`). We used 5K documents for training, and 2K for evaluation. We used the implement from the scikit-learn library (`https://scikit-learn.org/stable/`). We considered optimizing the following hyperparameters: document topic prior $\alpha \in [10^{-3}, 1]$, topic word prior $\eta \in [10^{-3}, 1]$, learning decay $\kappa \in [0.51, 1]$, learning offset $\tau_0 \in [1, 2, 5, 10, 20, 50, 100, 200]$, E-step stopping tolerance $\epsilon \in [10^{-5}, 10^{-1}]$, document batch size in $[2, 4, 8, 16, 32, 64, 128, 256]$, and topic number $K \in [1, 64]$. We optimized α, η, κ and ϵ in the log domain, and used a continuous

relaxation of the discrete parameters. We considered three fidelities — training with 1, 10 and 50 epochs, and randomly queried 10 examples at each fidelity to start each method. We evaluated the performance of the selected hyperparameters in terms of perplexity (the smaller, the better). In Fig. 1 b, we reported the average perplexity (and its standard deviation) of each method after five runs of the hyperparameter optimization.

XGBoost. Third, we trained an XGBoost model (Chen and Guestrin, 2016) to predict a quantitative measure of the diabetes progression (`https://archive.ics.uci.edu/ml/datasets/diabetes`). The dataset includes 442 examples. We used two-thirds for training and the remaining one-third for evaluation. We used the implementation from the scikit-learn library. We optimized the following hyperparameters: Huber loss parameter $\alpha \in [0.01, 0.1]$, the non-negative complexity pruning parameter ($[0.01, 100]$), fraction of samples used to fit individual base learners ($[0.1, 1]$), fraction of features considered to split the tree ($[0.01, 1]$), splitting criterion ([“MAE”, “MSE”]), minimum number of samples required to split an internal node ($[2, 9]$), and the maximum depth of individual trees ($[1, 16]$). The hyperparameter space is 12 dimensional. We considered three fidelities — training XGBoost with 2, 10 and 100 weak learners (trees). The querying cost is therefore $\lambda_1 : \lambda_2 : \lambda_3 = 1 : 5 : 50$. We started with 10 random queries at each fidelity. We used the log of nRMSE to evaluate the performance. We ran 5 times and report the average log-nRMSE of the identified hyperparameters by each method in Fig. 1c.

Physics-informed Neural Networks (PINN). Our fourth application is to learn a PINN to solve PDEs (Raissi et al., 2019). The key idea of PINN is to use boundary points to construct the training loss, and meanwhile use a set of collocation points in the domain to regularize the NN solver to respect the PDE. With appropriate choices of hyperparameters, PINNs can obtain very accurate solutions. We used PINNs to solve Burger's equation (Morton and Mayers, 2005) with the viscosity $0.01/\pi$. The solution becomes sharper with bigger time variables (see the Appendix) and hence the learning is quite challenging. We followed (Raissi et al., 2019) to use fully connected networks and L-BFGS for training. The hyperparameters include NN depth ($[1, 8]$), width ($[1, 64]$), and activations (8 choices: `Relu`, `tanh`, `sigmoid`, their variants, *etc.*). Following (Raissi et al., 2019), we used 100 boundary points as the training set and 10K collocation points for regularization. We used 10K points for evaluation. We chose 3 training fidelities, running L-BFGS with 10, 100, 50K maximum iterations. The querying cost (average training time) is $\lambda_1 : \lambda_2 : \lambda_3 = 1 : 10 : 50$. Note that in fidelity 3, L-BFGS usually converged before running 50K iterations. We initially issued 10 random queries at each fidelity. We ran each method for 5 times and reported the average log nRMSE after each step in Fig. 1d.



(a) *CNN* (b) *Online LDA*

Figure 2: Average time to generate queries (including surrogate training).

Results. As we can see, in all the applications, BMBO-DARN used the smallest cost (*i.e.*, running time) to find the hyperparameters that gives the best learning performance. In general, BMBO-DARN identified better hyperparameters with the same cost, or equally good hyperparameters with the smallest cost. BMBO-DARN-1 outperformed all the one-by-one querying methods, except that for online LDA (Fig. 1b) and PINN (Fig. 1d), it was worse than DNN-MFBO and Hyperband at the early stage, but finally obtained better learning performance. We observed that the GP based baselines (MF-MES, MF-GP-UCB, SF-Batch, *etc.*) are often easier to be stuck in suboptimal hyperparameters, this might because these models are not effective enough to integrate information of multiple fidelities to obtain a good surrogate. Together these results have shown the advantage of our method, especially in our batch querying strategy. Finally, we show the average query generation time of BMBO-DARN for CNN and Online LDA in Fig. 2 (including surrogate training). It turns out BMBO-DARN spends much less time than MF-MES using the global optimization method DIRECT (Jones et al., 1998), and comparable to MF-GP-UCB and DNN-MFBO. Therefore, BMBO-DARN is efficient to update the surrogate model and generate new queries.

7 Conclusion

We have presented BMBO-DARN, a batch multi-fidelity Bayesian optimization method. Our deep auto-regressive model can serve as a better surrogate of the black-box objective. Our batch querying method not only is efficient, avoiding combinatorial search over discrete fidelities, but also significantly reduces the cost while improving the optimization performance.

References

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. Journal of Machine Learning Research, 3(Nov):397–422.

Baptista, R. and Poloczek, M. (2018). Bayesian optimization of combinatorial structures. In International Conference on Machine Learning, pages 462–471.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems (NIPS 2011), volume 24. Neural Information Processing Systems Foundation.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo. In International conference on machine learning, pages 1683–1691. PMLR.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm SigKDD international conference on knowledge discovery and data mining, pages 785–794.

Daxberger, E., Makarova, A., Turchetta, M., and Krause, A. (2019). Mixed-variable bayesian optimization. arXiv preprint arXiv:1907.01329.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Twenty-fourth international joint conference on artificial intelligence.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In International Conference on Machine Learning, pages 1437–1446. PMLR.

Forrester, A., Sobester, A., and Keane, A. (2008). Engineering design via surrogate modelling: a practical guide. John Wiley & Sons.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and cooperation in neural nets, pages 267–285. Springer.

Garrido-Merchán, E. C. and Hernández-Lobato, D. (2020). Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. Neurocomputing, 380:20–35.

Hennig, P. and Schuler, C. J. (2012). Entropy search for information-efficient global optimization. Journal of Machine Learning Research, 13(Jun):1809–1837.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In Advances in neural information processing systems, pages 918–926.

Hoffman, M., Bach, F. R., and Blei, D. M. (2010). Online learning for latent dirichlet allocation. In advances in neural information processing systems, pages 856–864. Citeseer.

Huang, D., Allen, T. T., Notz, W. I., and Miller, R. A. (2006). Sequential kriging optimization using multiple-fidelity evaluations. Structural and Multidisciplinary Optimization, 32(5):369–382.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. Journal of Global optimization, 13(4):455–492.

Kandasamy, K., Dasarathy, G., Oliva, J. B., Schneider, J., and Póczos, B. (2016). Gaussian process bandit optimisation with multi-fidelity evaluations. In Advances in Neural Information Processing Systems, pages 992–1000.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. (2017a). Multi-fidelity bayesian optimisation with continuous approximations. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1799–1808. JMLR. org.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Poczos, B. (2017b). Asynchronous parallel Bayesian optimisation via Thompson sampling. arXiv preprint arXiv:1705.09236.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017a). Fast bayesian optimization of machine learning hyperparameters on large datasets. In Artificial Intelligence and Statistics, pages 528–536. PMLR.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2017b). Learning curve prediction with bayesian neural networks. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

Laguna, M. and Martí, R. (2005). Experimental testing of advanced scatter search designs for global optimization of multimodal functions. Journal of Global Optimization, 33(2):235–255.

Lam, R., Allaire, D. L., and Willcox, K. E. (2015). Multifidelity optimization using statistical surrogate modeling for non-hierarchical information sources. In 56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, page 0143.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems, pages 396–404.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research, 18(1):6765–6816.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. (2018). Massively parallel hyperparameter tuning. arXiv preprint arXiv:1810.05934.

Li, S., Xing, W., Kirby, R., and Zhe, S. (2020). Multi-fidelity bayesian optimization via deep neural networks. In Advances in Neural Information Processing Systems.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1-3):503–528.

McLeod, M., Osborne, M. A., and Roberts, S. J. (2017). Practical bayesian optimization for variable cost objectives. arXiv preprint arXiv:1703.04335.

Mockus, J. (2012). Bayesian approach to global optimization: theory and applications, volume 37. Springer Science & Business Media.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. Towards global optimization, 2(117-129):2.

Morton, K. W. and Mayers, D. F. (2005). Numerical solution of partial differential equations: an introduction. Cambridge university press.

Neal, R. M. et al. (2011). Mcmc using Hamiltonian dynamics. Handbook of markov chain monte carlo, 2(11):2.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc.

Perdikaris, P., Raissi, M., Damianou, A., Lawrence, N., and Karniadakis, G. E. (2017). Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473(2198):20160751.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. (2018). Scalable hyperparameter transfer learning. In Advances in Neural Information Processing Systems, pages 6845–6855.

Picheny, V., Ginsbourger, D., Richet, Y., and Caplin, G. (2013). Quantile-based optimization of noisy computer experiments with tunable precision. Technometrics, 55(1):2–13.

Poloczek, M., Wang, J., and Frazier, P. (2017). Multi-information source optimization. In Advances in Neural Information Processing Systems, pages 4288–4298.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707.

Rasmussen, C. E. and Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. MIT Press.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems, pages 2951–2959.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable bayesian optimization using deep neural networks. In International conference on machine learning, pages 2171–2180.

Song, J., Chen, Y., and Yue, Y. (2019). A general framework for multi-fidelity bayesian optimization with gaussian processes. In The 22nd International Conference on Artificial Intelligence and Statistics, pages 3158–3167.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust bayesian neural networks. In Advances in neural information processing systems, volume 29, pages 4134–4142.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In Proceedings of the 27th International Conference on International Conference on Machine Learning, pages 1015–1022.

Swersky, K., Snoek, J., and Adams, R. P. (2014). Freeze-thaw Bayesian optimization. arXiv preprint arXiv:1406.3896.

Takeno, S., Fukuoka, H., Tsukada, Y., Koyama, T., Shiga, M., Takeuchi, I., and Karasuyama, M. (2019). Multi-fidelity bayesian optimization with max-value entropy search. arXiv preprint arXiv:1901.08275.

Wang, Z. and Jegelka, S. (2017). Max-value entropy search for efficient bayesian optimization. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 3627–3635. JMLR. org.

Wu, J. and Frazier, P. I. (2017). Continuous-fidelity bayesian optimization with knowledge gradient. In NIPS Workshop on Bayesian Optimization.

Zhang, Y., Hoang, T. N., Low, B. K. H., and Kankanhalli, M. (2017). Information-based multi-fidelity bayesian optimization. In NIPS Workshop on Bayesian Optimization.

Checklist

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

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

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

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]