
Lifelong Bayesian Optimization

Yao Zhang
 University of Cambridge
 yz555@cam.ac.uk

James Jordon
 University of Oxford
 james.jordon@wolfson.ox.ac.uk

Ahmed M. Alaa
 University of California, Los Angeles
 ahmedmalaa@ucla.edu

Mihaela van der Schaar
 University of Cambridge
 mv472@cam.ac.uk

Abstract

Automatic Machine Learning (Auto-ML) systems tackle the problem of automating the design of prediction models or pipelines for data science. In this paper, we present Lifelong Bayesian Optimization (LBO), an online, multitask Bayesian optimization (BO) algorithm designed to solve the problem of model selection for datasets arriving and evolving over time. To be suitable for “lifelong” Bayesian optimization, an algorithm needs to scale with the ever increasing number of acquisitions and should be able to leverage past optimizations in learning the current best model. In LBO, we exploit the correlation between black-box functions by using components of previously learned functions to speed up the learning process for newly arriving datasets. Experiments on real and synthetic data show that LBO outperforms standard BO algorithms applied repeatedly on the data.

1 Introduction

Designing machine learning (ML) models or pipelines is often a tedious job, requiring experience with, and a deep understanding of, both the methods and the data. For this reason, Auto-ML methods are becoming ever more necessary to allow for broader adoption of machine learning in real-world applications, allowing non-experts to use machine learning methods easily “off-the-shelf”. The problem of pipeline design is a one of model selection and hyper-parameter optimization (which is itself a type of model selection). To perform this model selection, the mapping from model to performance on a given dataset is treated as a black-box function that needs to be optimized. Several AutoML frameworks [1, 2, 3, 4] have been proposed for the black-box optimization.

In this paper, we address a general problem of (related) datasets arriving over time, in which we want to perform a model selection for each dataset as it arrives. This problem presents itself in many settings. For example, in medicine, data collection is an ongoing process; every hospital visit a patient makes generates new data, and the models we wish to use on the data need to be optimized to perform well on the most recent patient population. Moreover, hospital practices might change, thereby creating a potential shift in the distribution or structure of the data—some features may stop being measured and new ones introduced because of a technological or medical advancement. It is key that past data not simply be discarded and a new optimization run every time new data arrives, but instead past data should be leveraged to guide the optimization of each new pipeline.

We cast model selection as a black-box function optimization problem. In particular, we assume there is a sequence of *related* black-box functions to be optimized (corresponding to a sequence of related datasets). Bayesian Optimization (BO) [5] aims to find an input optimization $\mathbf{x}_* \in \arg \max_{\mathbf{x} \in \mathbb{X}} f(\mathbf{x})$, corresponding to an optimal model and hyperparameter configuration for the dataset being modelled by f . In this setting a sequence of datasets arrives over time, hence we have a *sequence* of black-box

functions to optimize, f_1, \dots, f_T . If the dataset at time T is similar to some previous datasets, their optimal hyperparameter configuration will be similar, or the corresponding black-box functions will be correlated. In the literature, Meta Learning and Multitask Bayesian Optimization are two classes of methods to speed up the hyperparameter optimization process for new datasets using the information from past datasets. Our method LBO is closely related to the latter.

Meta Learning. Using additional meta-features to identify which past datasets are likely to be similar, Meta Learning warm starts a BO optimizer with the hyperparameter configurations that are optimal for the most similar previous datasets. There are three different kinds of meta-features: PCA, Statistical and Landmarking meta-features [6, 7, 8]. In the space of meta-features, algorithms and distance metrics are proposed to measure the distance between datasets [2, 6, 9, 10]. Instead of using meta-features, our method LBO learns the correlation between black-box functions from the acquisition data.

Multitask Bayesian Optimization. In Neural Network based BO, the acquisition function is constructed by performing a Bayesian linear regression in the output layer [11]. However, this approach gives rise to two technical challenges: (1) Marginal Likelihood Optimization requires inverting a $D^* \times D^*$ matrix in every gradient update where D^* is the size of the last hidden layer; (2) Poor variance estimate due to not measure the uncertainty using the posterior distribution of all the neural network parameters. The work [12] proposes an efficient training algorithm in marginal likelihood maximization. The work [13] uses Bayesian Neural Network [14] to improve the variance estimate by approximating the posterior distribution of all the neural network parameters using Stochastic HMC [15]. Both works [12, 13] assume all the tasks share one single neural network. Assuming there exists a shared model among the tasks, Multitask Learning (MTL) can improve the model generalization with increasing total sample size over all the tasks [16]. Multitask BO is more sample efficient in optimizing f_T after we have obtained sufficient acquisition data of f_1, \dots, f_{T-1} to learn the shared model. However, when f_T is different from the previous black-box functions, enforcing them to share the model may mislead the estimate of f_T when there are insufficient samples of f_T . In LBO, the black-box function learns not only the parameters to be shared but also which datasets should share these parameters, thus retaining the benefits of MTL. In GP based BO, the existing multitask BO [17, 18, 19] or Bandit algorithms [20, 21] only consider using a fixed number of latent GPs to model the black-box functions. Fixing the number of tasks requires rerunning the Multitask GPs every time a dataset arrives. The complexity of exact inference of Multitask GPs is $O((T \times N)^3)$ where N is the total number of acquisitions and T is the number of black-box functions. Sparse GP approximation [22] can reduce the complexity to $O(T \times K^3)$ where K is the number of inducing points used to approximate each latent Gaussian process. However, the approximation method using inducing points [23] is usually only shown to be useful when there are abundant observations in a low-dimensional space. When the number of inducing points is much smaller than the number of observations, the predictive performance deteriorates severely in practice [24]. Sparse GPs for single task BO is attempted in [25] where the experiment only shows the results within 80 function evaluations. In LBO, we use neural networks to model the black-box functions, without assuming the data is abundant for a sequence of black-box functions.

Contribution. In LBO, we go beyond restricting the black-box functions to share one single neural network or a fixed number of latent GPs. Inspired by Indian Buffet Process (IBP) [26], as datasets arrive over time, we treat the black-box function on each dataset as a new customer arriving in a restaurant; we apply IBP to generate dishes (neural networks) to approximate the new black-box function. LBO learns a suitable number of neural networks to span the black-box functions such that the correlated functions can share information, and the modelling complexity for each function is restricted to ensure a good variance estimate in BO. We develop a simple and efficient variational algorithm to optimize our model. We demonstrate that LBO improves the computation time of model selection on three datasets collected from United Network for Organ Sharing (UNOS) registry.

2 Lifelong Bayesian Optimization

Let \mathbb{X} be some input space¹. Let $f : \mathbb{X} \rightarrow \mathbb{R}$ denote some black-box function and $a_f : \mathbb{X} \rightarrow \mathbb{R}$ denote the acquisition function that quantifies the utility of evaluating f at x . In this paper, we

¹Typically this will be the product of all hyperparameter spaces for the models being considered along with an additional dimension that will be used to indicate the model.

extend this formulation by assuming that we have datasets \mathcal{D}_t arriving sequentially for time steps $t \in \mathbb{Z}^+$. The datasets may overlap if the user chooses to combine the newly arriving data with the old ones. In this case, we let \mathcal{D}_t denote the updated dataset at time t . Assume there is a corresponding black-box function $f_t : \mathbb{X} \rightarrow \mathbb{R}$, each mapping from model-and-hyperparameter-settings (\boldsymbol{x}) to performance on the corresponding dataset. Our goal at each time step t is then to find the best model-and-hyperparameter-setting, \boldsymbol{x}_t^* , that maximises f_t . More specifically, we look to find a maximiser of the J -fold cross-validation performance given by:

$$\boldsymbol{x}_t^* \in \arg \max_{\boldsymbol{x}_t \in \mathbb{X}} \frac{1}{J} \sum_{j=1}^J \mathcal{L}(\boldsymbol{x}_t, \mathcal{D}_{t,train}^{(j)}, \mathcal{D}_{t,valid}^{(j)}) \quad (1)$$

where \mathcal{L} is some given performance metric (e.g. AUC-ROC, Model Likelihood, etc), and $\mathcal{D}_{t,train}^{(j)}$ and $\mathcal{D}_{t,valid}^{(j)}$ are training and validation splits of \mathcal{D}_t in the j th fold, respectively. The objective in (1) has no closed form solution (both the performance metric and the input space \mathbb{X} are allowed to be anything), and so we must treat this as a black-box optimization problem. For each time step, we denote by $\mathcal{A}_t = (\mathbf{X}_t, \mathbf{Y}_t) = \{(\boldsymbol{x}_{i,t}, y_{i,t})\}_{i=1}^{N_t}$ the acquisition set for f_t where N_t is the number of acquisitions made for the t -th black-box function. We define the set of all acquisitions until time t to be $\mathcal{A}_{\leq t} = \bigcup_{u=1}^t \mathcal{A}_u$. In a setting where datasets that are being collected are similar, our goal is to leverage past acquisitions, $\mathcal{A}_{\leq t-1}$, when learning f_t .

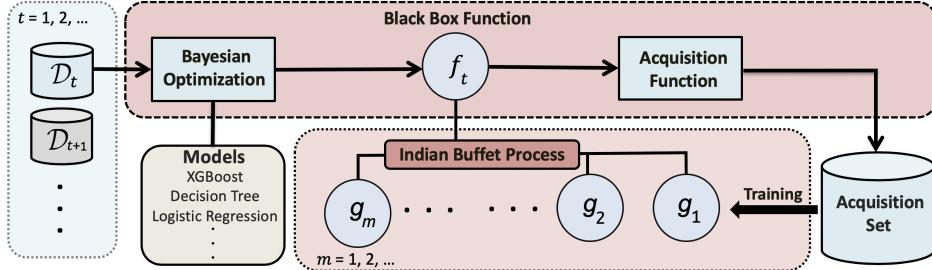


Figure 1: A pictorial depiction of Lifelong Bayesian Optimization. As the datasets arrive over time, the cross-validation performance is treated as a black-box function f_t . Some latent functions g_m are generated in an Indian Buffet Process and trained on the acquisition set to fit f_t .

In Linear Model of Coregionalization (LMC) [27, 28, 29], each black-box function f_t is a linear combination of some latent functions, $g_1, \dots, g_M : \mathbb{X} \rightarrow \mathbb{R}$ where each $g_m(\cdot)$ is drawn from a one-dimensional Gaussian process $\mathcal{GP}_m(0, K_m(\cdot, \cdot))$. The number of latent functions M and the number of black-box functions T are fixed. In LBO, T and M can be arbitrarily large and change over time. To allow for a suitable number of latent functions in our model, we set M to be some large number and introduce an additional binary variable $z_{t,m}$ to determine whether the latent function $g_m(\cdot)$ is used in the modelling of $f_t(\cdot)$. f_t is then defined by

$$f_t(\cdot) = \sum_{m=1}^M z_{t,m} s_{t,m} g_m(\cdot)$$

where $s_{t,m}$ is the weight (to be learned) of the latent function g_m in the spanning of f_t . The correlation between f_t and $f_{t'}$ is given by $\text{Cov}(f_t(\cdot), f_{t'}(\cdot)) = \sum_{m=1}^M z_{t,m} z_{t',m} s_{t,m} s_{t',m} K_m(\cdot, \cdot)$. This correlation is exploited to speed up the optimization process in Multitask BO [17, 18]. The number of latent functions that have been used until a given time-step t is given by $\tilde{M}_t = \sum_{m=1}^M \min(\sum_{u=1}^t z_{u,m}, 1)$. In LBO, as shown in Figure 1, the latent functions are generated by an Indian Buffet Process in the modelling of black-box functions. When a new black-box function f_t can be spanned by the existing (i.e. previously used) latent functions, no new latent functions will be added to the model (i.e. $z_{t+1,m} = 0$ for $m > \tilde{M}_t$). We aim to learn the number of latent functions \tilde{M}_t , which allows us to identify the actual number of unknown functions in a sequence of black-box functions and avoid spending computational power in re-solving similar optimization problems that we encounter over time. Furthermore, when there is concept drift in the new arriving data set, we can model the corresponding black-box function independently by a new latent function, without enforcing it to be correlated with the previous black-box functions.

2.1 Approximating latent processes using Neural Networks

To enable lifelong Bayesian optimization, the algorithm needs to be scalable to a large number of acquisitions over time. We consider the approach of approximating the infinite-dimensional feature map corresponding to the kernel of each latent Gaussian process $\mathcal{GP}_m(0, K_m(\cdot, \cdot))$, with a finite-dimensional feature map learned by Deep Neural Networks [11, 12, 13]. Note that this approximation varies every time we optimize a new black-box function. When modelling f_t , we obtain a D^* -dimensional feature map $\phi_{t,m} : \mathbb{X} \rightarrow \mathbb{R}^{D^*}$, where D^* is the output dimension of the feature map. We set the dimension D^* to be the same for all $\phi_{t,m}, m = 1, \dots, M$. We then compose this with a final, fully connected layer $\mathbf{h}_{t,m} : D^* \rightarrow \mathbb{R}$ to define $g_{t,m}$ (i.e. $g_{t,m} = \mathbf{h}_{t,m}^\top \phi_{t,m}$). We collect the parameters of each feature map, $\phi_{t,1}, \dots, \phi_{t,M}$ into a single vector Θ_t . In our approximation $f_t(\cdot) \approx \sum_{m=1}^M z_{t,m} s_{t,m} \mathbf{h}_{t,m}^\top \phi_{t,m}(\cdot)$, we can treat $s_{t,m}$ as part of $\mathbf{h}_{t,m}$ in learning and discard the parameters $s_{t,m}$ from our model. Let Φ_t be the $N_t \times D^* M$ matrix s.t. $[\Phi_t]_{u,m} = z_{u,m} \phi_{t,m}(\mathbf{x}_u)$. We will also write $\phi_{t,1:M}(\mathbf{x})$ to denote $[z_{t,1} \phi_{t,1}^\top(\mathbf{x}), \dots, z_{t,M} \phi_{t,M}^\top(\mathbf{x})]^\top$. Let $\mathbf{H}_t = [\mathbf{h}_{t,1}^\top, \dots, \mathbf{h}_{t,M}^\top]^\top$, $\mathbf{Z}_t = [z_{t,1}, \dots, z_{t,M}]^\top$. Then the posterior distribution of \mathbf{H}_t is given by

$$P(\mathbf{H}_t | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t) = \mathcal{N}(\mathbf{m}_{\mathbf{H}_t}, \mathbf{K}_{\mathbf{H}_t}^{-1}) \quad (2)$$

where the mean function, $\mathbf{m}_{\mathbf{H}_t}$, and the kernel, $\mathbf{K}_{\mathbf{H}_t}$, are given by

$$\mathbf{m}_{\mathbf{H}_t} = \frac{\beta_t}{\lambda_t} \mathbf{K}_{\mathbf{H}}^{-1} \Phi_t^\top \mathbf{Y}_t, \quad \mathbf{K}_{\mathbf{H}_t} = \frac{\beta_t}{\lambda_t} \Phi_t^\top \Phi_t + \mathbf{I}_{D^* \times D^*}$$

where λ_t is the precision parameter in the prior distributions $P(\mathbf{h}_{t,m} | \lambda_t) = \mathcal{N}(\mathbf{0}, \lambda_t^{-1} \mathbf{I}_{D^* \times D^*})$, and β_t is the precision parameter in the likelihood function

$$P(\mathbf{Y}_t | \mathbf{f}_t, \beta_t) = P(\mathbf{Y}_t | \mathbf{Z}_t, \mathbf{X}_t, \mathbf{H}_t, \Theta_t) = \prod_{i=1}^{N_t} \mathcal{N}(y_{i,t}; \sum_{m=1}^M z_{t,m} \mathbf{h}_{t,m}^\top \phi_{t,m}(\mathbf{x}_{i,t}), \beta_t^{-1}) \quad (3)$$

In [11], the neural network parameters Θ_t, \mathbf{H}_t are optimized end-to-end by stochastic gradient descent. This approach is shown to be disadvantageous in the setting of Multitask BO [12]. In LBO, we only learn the parameter Θ_t by marginal likelihood optimization. The parameter \mathbf{H}_t is integrated out w.r.t its posterior distribution from (2) when we derive the predictive distribution for a testing data point \mathbf{x} as

$$f_t(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x} | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t), \sigma^2(\mathbf{x} | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t))$$

where

$$\mu(\mathbf{x} | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t) = \mathbf{m}_{\mathbf{H}_t}^\top \phi_{t,1:M}(\mathbf{x}), \quad \sigma^2(\mathbf{x} | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t) = \phi_{t,1:M}(\mathbf{x})^\top \mathbf{K}_{\mathbf{H}_t}^{-1} \phi_{t,1:M}(\mathbf{x}) \quad (4)$$

Complexity. In marginal likelihood optimization, we switch between the primal and dual form of the log marginal likelihood for computational efficiency and numerical stability. In primal form, the log marginal likelihood is

$$\mathcal{L}(\Theta_t) = -\frac{N_t}{2} \log(2\pi\beta_t^{-1}) - \frac{\beta_t}{2} \|\mathbf{Y}_t\|^2 + \frac{\beta_t^2}{\lambda_t} \mathbf{Y}_t^\top \Phi_t \mathbf{K}_{\mathbf{H}_t}^{-1} \Phi_t^\top \mathbf{Y}_t - \frac{1}{2} \log |\mathbf{K}_{\mathbf{H}_t}|$$

In dual form, it is given as the logarithm of $\mathcal{N}(\mathbf{Y}_t; \lambda_t^{-1} \Phi_t \Phi_t^\top + \beta_t^{-1} \mathbf{I}_{N_t \times N_t})$. When $N_t > \tilde{M}_t D^*$, we optimize the log marginal likelihood in primal form, otherwise in dual form. Recall that \tilde{M}_t denotes the number of active neural networks. The dimension of the feature map $\phi_{t,1:M}(\mathbf{x})$ is equal to $\tilde{M}_t D^*$ since some $z_{t,m}$ are zero. Despite the fact that we set a large value for M , the complexity only depends on the lesser of the number of active neural networks and the size of the acquisition data. Overall, optimizing the log marginal likelihood has complexity $O(\max(\tilde{M}_t D^*, N_t)(\min(\tilde{M}_t D^*, N_t))^2)$. When we initialize LBO by activating a relatively large number of neural networks, the computational overhead is still affordable since the acquisition data is small. When we acquire more and more data, we use an Indian Buffet Process to keep a small number of activated neural networks.

2.2 Modelling Z as an Indian Buffet Process

We put an Indian Buffet Process (IBP) prior on the binary matrix \mathbf{Z} , where $[\mathbf{Z}]_{t,m} = z_{t,m}$ and $z_{t,m} = 1$ if the m th neural network $g_m(\cdot)$ is used to span the t th black box function and $z_{t,m} = 0$

otherwise. In the Stick-breaking Construction of IBP [30], a probability $\pi_m \in [0, 1]$ is first assigned to each column of \mathbf{Z} . In the t th column \mathbf{Z}_t , each $z_{t,m}$ is sampled independently from the distribution $\text{Bern}(\pi_m)$. The sampling order of the rows does not change the distribution. The probability π_m is generated according to

$$\pi_m = \prod_{k=1}^m v_k \text{ where } v_k \sim \text{Beta}(\alpha, 1) \quad (5)$$

In this construction, $\mathbb{E}[\pi_m]$, the expected probability of using the m th neural network, decreases exponentially as m increases. The parameter α represents the expected number of active neural networks for each black-box function. It controls how quickly the probabilities π_m decay. Larger values of α correspond to a slower decrease in π_m , and thus we would expect to use more neural networks to represent the black box functions. Using IBP we are able to limit the number of neural networks used at each time-step, while also introducing new neural networks when the new black-box function is distinct to the previous ones.

2.2.1 Mean Field Approximation

We use the variational method [31, 32] to infer \mathbf{Z}_t , which involves a mean-field approximation to the posterior distribution of \mathbf{Z}_t and learning the variational parameters through the evidence lower bound (ELBO) maximization. The joint distribution of the acquisitions $(\mathbf{X}_t, \mathbf{Y}_t)$ and latent variables \mathbf{Z}_t is given by

$$P(\mathbf{Y}_t, \mathbf{Z}_t, \mathbf{X}_t, \Theta_t, \mathbf{v}) = P(\mathbf{Y}_t | \mathbf{Z}_t, \mathbf{X}_t, \Theta_t) P(\mathbf{Z}_t | \mathbf{v})$$

where $\mathbf{v} = [\pi_1, \dots, \pi_M]$. The prior distribution of \mathbf{Z}_t is given by $P(\mathbf{Z}_t | \mathbf{v}) = \prod_{m=1}^M \text{Bern}(\pi_m)$. Since computing the true posterior distribution $P(\mathbf{Z}_t | \mathbf{X}_t, \mathbf{Y}_t, \Theta_t)$ is intractable, we use a fully factorized mean-field approximation given by $Q(\mathbf{Z}_t) = \prod_{m=1}^M Q_{\gamma_{t,m}, \tau_{t,m}}(z_{t,m})$ where $Q_{\gamma_{t,m}, \tau_{t,m}}(z_{t,m})$ is a Binary Concrete distribution, the continuous relaxation of the discrete Bernoulli distribution,

$$\text{BinConcrete}(\gamma, \tau) = \frac{\gamma \tau z^{-\tau-1} (1-z)^{-\tau-1}}{(\gamma z^{-\tau} + (1-z)^{-\tau})^2} \quad (6)$$

where $z \in (0, 1)$, $\gamma \in (0, \infty)$ is a probability ratio and $\tau \in (0, \infty)$ is the temperature hyperparameter. As $\tau \rightarrow 0$, samples from the Concrete distribution are binary and identical to the samples from a discrete Bernoulli distribution. A realization of the Concrete variable z has a convenient and differentiable parametrization:

$$z = \text{Sigmoid}\left(\frac{1}{\tau}(\log \gamma - \log(u) - \log(1-u))\right) \quad (7)$$

where $u \sim \text{Uniform}(0, 1)$. We can learn Θ_t and $\{\gamma_{t,m}\}_{m=1}^M$ jointly by maximizing the ELBO of the log-marginal likelihood given by

$$\mathcal{L}_e(Q, \Theta_t) = \langle \log P(\mathbf{Y}_t | \mathbf{X}_t, \mathbf{Z}_t, \Theta_t) \rangle_{Q(\mathbf{Z}_t)} - D_{KL}[Q(\mathbf{Z}_t) || P(\mathbf{Z}_t | \mathbf{v})] \quad (8)$$

where $D_{KL}[\cdot || \cdot]$ is the Kullback–Leibler divergence. We approximate the expectation and KL divergence in (8) by drawing a sample $z_{t,m}$ in (7) and π_m in (5). Since $z_{t,m}$ is an approximate binary sample, we cannot evaluate it in the Binary distribution $\text{Bern}(\pi_m)$. Therefore, we also approximate each $\text{Bern}(\pi_m)$ in $P(\mathbf{Z}_t | \mathbf{v})$ by the corresponding Concrete distribution $\text{BinConcrete}(\frac{\pi_m}{1-\pi_m}, \tau)$ when evaluating the KL divergence in (8). Intuitively, ELBO maximization has two objectives: (1) Tuning the feature map parameter Θ_t to model the data $(\mathbf{X}_t, \mathbf{Y}_t)$; and (2) Learning $Q(\mathbf{Z}_t)$ to infer how many neural networks are required to achieve the first objective with a small KL divergence from $P(\mathbf{Z}_t | \mathbf{v})$.

2.3 Online Training of Neural networks

In the update of Θ_t , we use only the data (acquisitions) from the current black-box function f_t . To prevent catastrophic forgetting and leverage the past data, we can use the neural network weights from previously learned neural networks as a graph regularizer [33, 34, 35, 36]. In LBO, we use the unweighted graph regularizer in [34]. Let $\mathbf{W}_{u,m}$ denote the weight parameters in $\phi_{u,m}(\cdot)$ when we

finish optimizing the u th black-box function. Then, at time t , we optimize the ELBO in (8) with the graph regularizer on $\mathbf{W}_{t,1:M}$ being given by

$$\Omega(\mathbf{W}_{t,1:M}; \mathbf{W}_{1:t-1,1:M}, \mathbf{Z}) = \sum_{m=1}^M \left(\sum_{u=1}^{t-1} z_{u,m} z_{t,m} \|\mathbf{W}_{t,m} - \mathbf{W}_{u,m}\|_2^2 \right) \quad (9)$$

The regularizer enforces that the current weights $\mathbf{W}_{t,1:M}$ for each neural network are similar to the previous weights $\mathbf{W}_{1:t-1,1:M}$ with the term $z_{u,m} z_{t,m}$ ensuring that similarity is only enforced between two time-steps that actually use the neural networks m in their decomposition of f . Our primary interest is f_t rather than the previous black-box functions that have already been optimized and so we allow each neural network to deviate slightly from the optimal solution for previous black-box functions. If the ELBO can be optimized under this constraint, f_t is said to neighbour some of the previous black-box functions. If the existing neural networks cannot fit the new black-box function well, then when updating \mathbf{Z}_t , and a new untrained neural network will be used in addition to the existing ones. This new network will not have any similarity constraints since its corresponding penalty term in (9) will be zero. Pseudo-code for the algorithm can be found in the Supplementary Materials.

2.4 Hyperparameters

In Section 2.1, the posterior distribution $P(\mathbf{H}_t | \mathcal{A}_t, \Theta_t, \mathbf{Z}_t)$ has two precision parameters λ_t and β_t . We learn these two hyperparameters by ELBO maximization i.e. Empirical Bayes [37]. In Section 2.2.1, the Binary Concrete distribution has a temperature parameter of τ . As shown in the works [38, 39], Setting τ to 0.1 is sufficient to obtain approximate one-hot samples from a discrete Bernoulli distribution. In the Indian Buffet Process, α is the expected number of active neural networks. Recall that D_* is the input dimension of each feature map $\phi_{t,m}$ defined in Section 2.1. The parameter α is set to be a small positive number such that it is computationally affordable to invert $\alpha D_* \times \alpha D_*$ matrices when optimizing the ELBO in (8). In the following section of experiments, we set α to be 2 when the feature maps are fifty-dimensional. Following the work [34], we select the regularization parameter for (9) by a grid search over $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, choosing the parameter with the best cross-validation error on the acquisition data.

3 Experiments

In this section, we assess the ability of LBO to optimize a sequence of black-box functions. We first examine the effectiveness and robustness of the algorithm on a simple synthetic data set. We compare LBO with Random Search, Multitask GP (MTGP) [17], Multitask Neural Network model ABLR [12], Single-task GP model (SGP) [40] and Single-task Neural Network model DNGO [11]. We then compare the BO algorithms on three publicly available UNOS datasets².

3.1 Synthetic dataset

In the synthetic dataset, we consider five sequences of five Branin functions that need to be minimized. The standard Branin function is $f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - r) \cos(x_1) + s$ where $a = 1, b = 5.1/(4\pi^2), c = 5/\pi, r = 6, s = 10$ and $t = 1/(8\pi)$. This function is evaluated on the square $x_1 \in [-5, 10], x_2 \in [0, 15]$. Each sequence consists of five Branin functions $f_{t,1}, \dots, f_{t,5}$. We create each $f_{t,i}$ by adding a small constant to each parameter of the standard Branin function. The constant is drawn from $\mathcal{N}(0, \sigma_t^2)$, where $\sigma_1 = 0.01, \sigma_2 = 0.05, \sigma_3 = 0.1, \sigma_4 = 0.5$ and $\sigma_5 = 1$. The correlation between functions decreases with increasing σ_t . Each of the sequences are optimized independently. We aim to test the robustness of different BO algorithms when they are applied to optimize sequences of functions at different levels of correlation. In LBO, we set the maximum number of neural networks, M , to 10. Each neural network g_1, \dots, g_5 has three layers with fifty tanh units. LBO is optimized using Adam [41]. The benchmark DNGO is a single neural network, with three layers each with fifty tanh units. ABLR is a Multitask neural network, also with three layers with tanh activations. We make sure the neural network is sufficiently wide that it can learn the shared feature map for the black-box functions. We conduct the experiments for ABLR with fifty and one-hundred units, denoted by ABLR-50 and ABLR-100 respectively. The same architectures

²<https://unos.org/data/>

are used in the original paper [12] after investigating the computation time of ABLR. The benchmark SGP is a Gaussian process implemented using the GPyOpt library [42]. The benchmark MTGP is implemented with a full-rank task covariance matrix, using the GPy library [43]. Both GP models use a Matérn-5/2 covariance kernel and automatic relevance determination hyperparameters, optimized by Empirical Bayes [37]. For every algorithm, we use Expected Improvement (EI) [44] as the acquisition function a_f . We set the maximum number of function evaluations to 200. All the BO optimizers are initialized with the same five random samples in the domain of the Branin function.

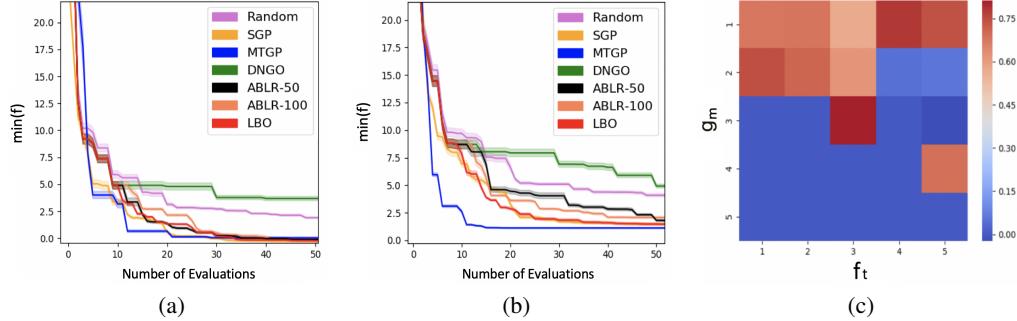


Figure 2: (a) The average minimum function values in the first fifty evaluations for the sequence $t = 1, 2, 3$; (b) The average minimum function values in the first fifty evaluations for the sequence $t = 3, 4, 5$; (c) Correlation between neural networks g_1, \dots, g_5 and black-box functions f_1, \dots, f_5 .

The experiments of each sequence are averaged over 10 random repetitions. Figure 2a shows the minimum function values averaged over the sequence $t = 1, 2, 3$. It represents the performance of the BO algorithms when the function correlation is strong. Figure 2b shows the same results for the sequence $t = 3, 4, 5$, which compares the BO algorithms when the function correlation is weak. As expected, for a smooth function like Branin, MTGP and SGP take fewer samples to approximate it and find its minimum because GP models start with the right kernel to model the smooth function while neural networks take more samples to learn the kernel from scratch. In ABLR, all the black-box function share one single neural network. The size of the shared network and the function correlation determine the sample complexity of neural network based BO. Comparing the results of ABLR-50 and ABLR-100, we find the former converges faster than the latter when function correlation is strong (as in Figure 2a), while the converse is true when function correlation is weak (as in Figure 2b). Selecting the size of the neural networks requires prior knowledge of the correlation between the black-box functions. In Figure 2a, the performance of LBO is comparable to ABLR-50 when the correlated functions share the same feature map. In Figure 2b, LBO outperforms ABLR-50 and ABLR-100, by learning the latent variable $z_{t,m}$ to infer a suitable number of neural networks in modelling the black-box functions, and which functions should share the same feature maps. To further understand the results of LBO, we compute the correlation matrix between the neural networks g_m and the noisy observation y_t of f_t . The sample correlation is estimated as

$$\hat{\text{Corr}} = \frac{\sum_{j=1}^{N_t} (g_m(\mathbf{x}_{j,t}) - \bar{g}_m)(y_{j,t} - \bar{y}_t)}{\sqrt{\sum_{j=1}^{N_t} (g_m(\mathbf{x}_{j,t}) - \bar{g}_m)^2 \sum_{j=1}^{N_t} (y_{j,t} - \bar{y}_t)^2}} \quad (10)$$

where \bar{y}_t and \bar{g}_m are the sample means. In Figure 2c, we show the correlation multiplied by $z_{t,m}$. We see from Figure 2c, that g_1 is used by all the black-box functions. We also see that both of f_1 and f_2 can be fit using the same two neural networks, g_1 and g_2 but the third function, f_3 requires a new neural network, g_3 to be introduced in order to find a good fit. This indicates that there were no neighbours of g_1 and g_2 that would suitably fit f_3 .

3.2 Real-world datasets

We extracted the UNOS-I, UNOS-II and UNOS-III data between the years 1985 to 2004 from the United Network for Organ Sharing (UNOS). Cohort UNOS-I is a pre-transplant population of cardiac patients who were enrolled in a heart transplant wait-list. Cohort UNOS-II is a post-transplant population of patients who underwent a heart transplant. Cohort UNOS-III is a post-transplant population of patients who underwent a lung transplant. We divide each of the datasets into subsets

\mathcal{D}_t , $t = 1, 2, \dots, T$, varying the number of T throughout the experiments. The model is tasked with predicting survival at the time horizon of three years. As each \mathcal{D}_t arrives over time, we treat the 5-fold cross-validation AUC of the models on \mathcal{D}_t as a black-box function f_t . In our experiment, the models we consider for selection are XGBoost [45], Logistic Regression [46], Bernoulli naive bayes [47], Multinomial naive bayes [48]. XGBoost is implemented with the Python package xgboost. Other models are implemented with the scikit-learn library [49]. The input space \mathbb{X} is the product of all hyperparameter spaces for all the models together with a categorical variable indicating which model is selected. The details of the full hyperparameter space are provided in the Supplementary Material.

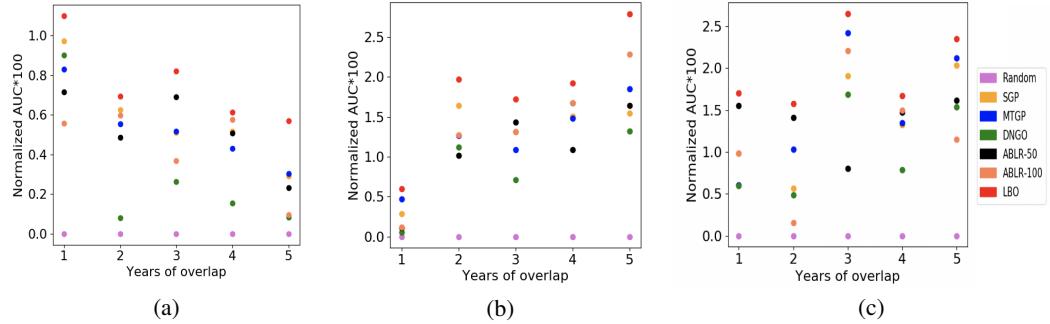


Figure 3: Normalized AUC after fifty function evaluations: (a) UNOS-I; (b) UNOS-II; (c) UNOS-III.

To compare lifelong learning with multitask learning, we create varying amounts of “overlap” between each dataset. To do this, we set a year “width” (from 4, 5, 6) and then set the first dataset to be all data from 1985 until 1985+width. Each subsequent dataset is created by removing the oldest year from the previous dataset and adding in the next year (for example, the second dataset consists of all data from 1986 until 1986+width). The total number of years of overlap between each subsequent dataset is therefore width minus one. Note that this set-up corresponds to several standard data collection mechanisms in which data is collected regularly and “old data” is phased out (because it is believed to no longer be indicative of the current distribution, for example).

We compare LBO against Random Search, SGP, MTGP, DGNO, ABLR-50 and ABLR-100. The models LBO, DGNO, ABLR-50 and ABLR-100 have the same architecture as in the Section 3.1. We set the maximum number of function evaluation to 200. All the BO optimizers are initialized with the same five random samples. Since the point of LBO is to increase the speed at which we learn the best model, the primary metric we report is maximum AUC after the first fifty function evaluations. In order to visualize and compare the results on the datasets with different overlaps, we normalized the AUC of each BO algorithm by the AUC of Random Search (i.e. the AUC is first subtracted, then divided by the AUC of Random Search). Figure 3 shows the experimental results averaged over 10 random repetitions. In the experiments with different amounts of “overlap”, LBO is the most consistent method over all the experiments. LBO outperforms the benchmarks in the UNOS-I, II and III datasets for all overlaps. LBO shows the gain from learning a suitable number of neural networks to model the black-box functions, rather than enforcing them to share one single neural network as in ABLR or assuming the number of latent functions is equal to the number of black-box functions as in MTGP. Our method LBO offers extra robustness when leveraging past data to speed up the current optimization. In the Supplementary Materials, we provide the details of overlapping over the years and the results of 200 function evaluations for all datasets and overlaps.

4 Conclusion

This paper developed Lifelong Bayesian Optimization (LBO) to solve the model selection problem in the setting where datasets are arriving over time. LBO improves over standard BO algorithms by capturing correlations between black-box functions corresponding to different datasets. We used an Indian Buffet Process to control the introduction of new neural networks to the model. We applied a variational method to optimize our model such that the neural network parameters and the latent variables can be learned jointly. Through synthetic and real-world experiments we demonstrated that LBO can improve the speed and robustness of BO.

References

- [1] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5, 2017.
- [2] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.
- [3] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74, 2016.
- [4] Ahmed M Alaa and Mihaela van der Schaar. Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning. *arXiv preprint arXiv:1802.07207*, 2018.
- [5] Ryan Prescott Adams, Jasper Snoek, and Hugo Larochelle. Practical bayesian optimization of machine learning algorithms. 2012.
- [6] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207, 2013.
- [7] Bill Fulkerson. Machine learning, neural and statistical classification, 1995.
- [8] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *ICML*, pages 743–750, 2000.
- [9] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.
- [10] Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87(3):357–380, 2012.
- [11] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.
- [12] Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cedric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pages 6845–6855, 2018.
- [13] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 4134–4142, 2016.
- [14] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.
- [15] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691, 2014.
- [16] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. The benefit of multitask representation learning. *The Journal of Machine Learning Research*, 17(1):2853–2884, 2016.
- [17] Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013.
- [18] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.
- [19] Matthias Poloczek, Jialei Wang, and Peter Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, pages 4288–4298, 2017.
- [20] Kirthevasan Kandasamy, Gautam Dasarathy, Junier B Oliva, Jeff Schneider, and Barnabás Póczos. Gaussian process bandit optimisation with multi-fidelity evaluations. In *Advances in Neural Information Processing Systems*, pages 992–1000, 2016.
- [21] Jialin Song, Yuxin Chen, and Yisong Yue. A general framework for multi-fidelity bayesian optimization with gaussian processes. *CoRR*, abs/1811.00755, 2018.
- [22] Trung V Nguyen, Edwin V Bonilla, et al. Collaborative multi-output gaussian processes. In *UAI*, pages 643–652, 2014.
- [23] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.
- [24] Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable gaussian processes. *arXiv preprint arXiv:1511.01870*, 2015.

- [25] Mitchell McIntire, Daniel Ratner, and Stefano Ermon. Sparse gaussian processes for bayesian optimization. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 517–526. AUAI Press, 2016.
- [26] Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12:1185–1224, 2011.
- [27] Andre G. Journel and Charles J. Huijbregts. *Mining Geostatistics*. Academic Press, London, 1978.
- [28] Pierre Goovaerts. *Geostatistics For Natural Resources Evaluation*. Oxford University Press, USA, 1997.
- [29] Matthias Seeger, Yee-Whye Teh, and Michael Jordan. Semiparametric latent factor models. Technical report, 2005.
- [30] Yee Whye Teh, Dilan Grür, and Zoubin Ghahramani. Stick-breaking construction for the indian buffet process. In *Artificial Intelligence and Statistics*, pages 556–563, 2007.
- [31] Rachit Singh, Jeffrey Ling, and Finale Doshi-Velez. Structured variational autoencoders for the beta-bernoulli process. In *Proceedings of NIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.
- [32] Eric Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders. *arXiv preprint arXiv:1605.06197*, 2016.
- [33] Charles A Micchelli and Massimiliano Pontil. Kernels for multi-task learning. In *Advances in neural information processing systems*, pages 921–928, 2005.
- [34] Daniel Sheldon. Graphical multi-task learning. 2008.
- [35] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.
- [36] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR.org, 2017.
- [37] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.
- [38] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.
- [39] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [40] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- [41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [42] J González. Gpyopt: A bayesian optimization framework in python, 2016.
- [43] GPy. GPy: A gaussian process framework in python. <http://github.com/SheffieldML/GPy>, since 2012.
- [44] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.
- [45] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.
- [46] Peter McCullagh and John A Nelder. *Generalized linear models*, volume 37. CRC press, 1989.
- [47] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.
- [48] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *Australasian Joint Conference on Artificial Intelligence*, pages 488–499. Springer, 2004.
- [49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Supplementary Material

Overlap Datasets

In Section 3.2, we subsampled the UNOS datasets to create sequences of datasets with different amount of overlaps. In this section, we provide the details of the “overlaps” experiments, including histograms of sample sizes and tables of “overlaps” for each UNOS dataset. In all the UNOS datasets, we remove the samples with missing features. In Figure 4, we show the histograms of sample sizes from 1985 to 2004. In UNOS-I, the samples are distributed between 1985 and 2004. In UNOS-II and III, the majority of samples are distributed between 1989 and 2004. Therefore, the “overlaps” experiments of UNOS-I start from 1985 while the experiments of UNOS-II and III begin with 1989. The details of “overlaps” are provided in Tables 1, 2 and 3.

In each overlap experiment (i.e. each row of the tables), we move the windows five times, which creates a sequence of five overlapped datasets. For example, in the row “1 year” of Table 1, we create a sequence of five datasets with one-year overlap. We create the first dataset in this sequence by sub-sampling the UNOS-I data in the years between 1985 and 1988. The complexity of MTGP [17] and ABLR [12] increases with an increasing number of black-box functions in the sequence. Especially, we implemented MTGP without using a low-rank approximation of the task covariance matrix. It is expensive to extend our experiments to optimize a long sequence of functions since we need to retrain the Multitask BO models with all the previous acquisitions data every time we optimize a new black-box function. In this paper, we focus on comparing the BO algorithms in terms of their speed in finding the function maximizers or minimizers. Not only can LBO speed up the optimization process of each black-box function, but it is also scalable with an increasing number of black-box functions and their acquisition data. This is an important advantage of LBO when we use it to keep updating machine learning models or pipelines over time with new arriving data. With LBO, we can always leverage the past acquisition data in the optimization process of the current black-box function, despite the ever increasing size of the acquisition data we accumulated over time.

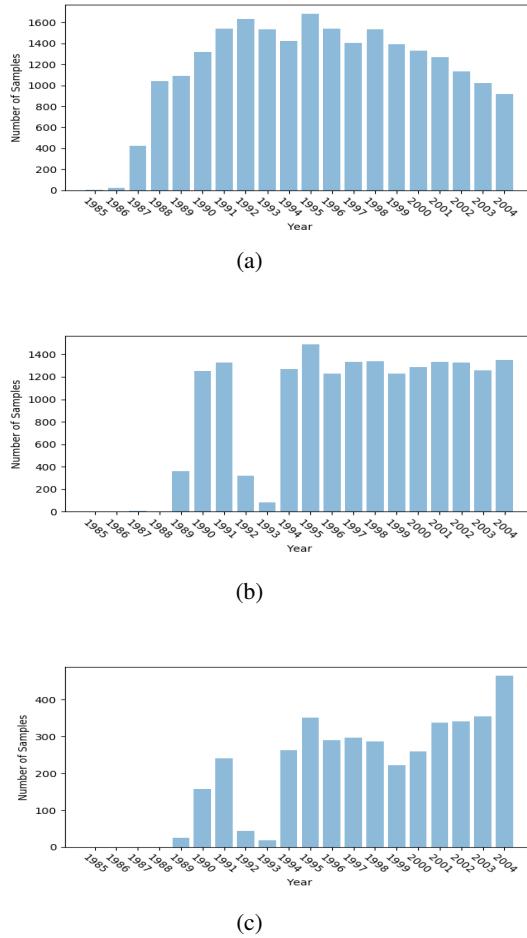


Figure 4: Histograms of sample sizes in years: (a) UNOS-I; (b) UNOS-II; (c) UNOS-III.

Years Overlaps	UNOS-I				
1 year	1985-88	1988-91	1991-94	1994-97	1997-00
2 years	1985-88	1987-90	1989-92	1991-94	1993-96
3 years	1985-88	1986-89	1987-90	1988-91	1989-92
4 years	1985-89	1986-90	1987-91	1988-92	1989-93
5 years	1985-90	1986-91	1987-92	1988-93	1989-94

Table 1: Overlapped UNOS-I datasets from 1985 to 2000

Years Overlaps	UNOS-II				
1 year	1989-92	1992-95	1995-98	1998-01	2001-04
2 years	1989-92	1991-94	1993-96	1995-98	1997-00
3 years	1989-92	1990-93	1991-94	1992-95	1993-96
4 years	1989-93	1990-94	1991-95	1992-96	1993-97
5 years	1989-94	1990-95	1991-96	1992-97	1993-98

Table 2: Overlapped UNOS-II datasets from 1989 to 2004

Years Overlaps	UNOS-II				
1 year	1989-92	1992-95	1995-98	1998-01	2001-04
2 years	1989-92	1991-94	1993-96	1995-98	1997-00
3 years	1989-92	1990-93	1991-94	1992-95	1993-96
4 years	1989-93	1990-94	1991-95	1992-96	1993-97
5 years	1989-94	1990-95	1991-96	1992-97	1993-98

Table 3: Overlapped UNOS-III datasets from 1989 to 2004

Computation Time of LBO

In this section, we show the accumulated computation time of LBO in optimizing a sequence of black-box functions, with different values of the IBP hyperparameter α . We use LBO to optimize the model selection annually for the UNOS-II data from 1994 to 2008. We choose this period because the data in every year consists of samples in both classes. The AUC performance on the data of each year is treated as a black-box function. We optimize a sequence of fifteen black-box functions. In LBO, the truncated number of neural networks, M , is set to 25. Each neural network has three layers each with fifty tanh units. The experimental result is averaged over 10 runs. In Figure 5, we show the accumulated running time of LBO with $\alpha = \{0.1, 0.5, 1, 2, 3\}$. The accumulated running time only takes into account the time spent on ELBO optimization but not the process of evaluating the black-box functions with the hyperparameter selected by LBO. We can see the times increase linearly with an increasing number of black-box functions. Updating a large truncated number of neural networks by marginal likelihood optimization is computationally expensive since it needs to invert a $MD^* \times MD^*$ matrix at every iteration. In LBO, we reduce the computation to an affordable level by using an IBP prior, in which only a small number of neural networks is activated by sampling at each iteration of ELBO optimization. Therefore, LBO is lifelong in speeding up the hyperparameter optimization process for new datasets using the information from past datasets.

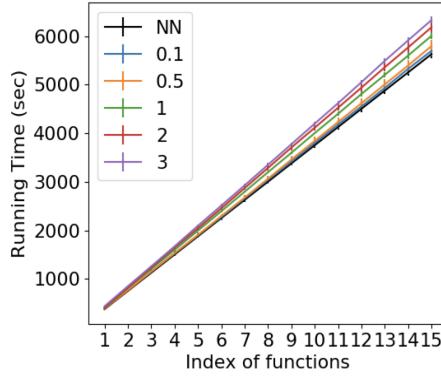


Figure 5: The accumulated running time of LBO with $\alpha = \{0.1, 0.5, 1, 2, 3\}$. As a comparison, the black line, denoted by “NN”, is the accumulated running time by a 3-layers neural network with 100 units at each layer, trained by marginal likelihood optimization. The neural network “NN” optimizes each black-box function independently.

Hyperparameter Space in the UNOS Experiments

In the experiments of the UNOS datasets, we considered the model-and-hyperparameter selection problem over the algorithms XGBoost [45], Logistic Regression [46] and Bernoulli naive bayes [47], Multinomial naive bayes [48]. XGBoost is implemented with the Python package `xgboost`. Other models are implemented with the `scikit-learn` library [49]. The hyperparameter space \mathbb{X} is nine-dimensional.

The XGBoost model consisted of the following 3 hyperparameters:

- Number of estimators (type: int, min: 10, max: 500)
- Maximum depth (type: int, min: 1, max: 10)
- Learning rate (type: float, min: 0.005, max: 0.5)

The Logistic Regression model consisted of the following 3 hyperparameters:

- Inverse of regularization strength (type: float, min: 0.001, max: 10) ,
- Optimization solver (type: categorical, solvers: Newton-Conjugate-Gradient, Limited-memory BFGS, Lbfgs, Sag, Saga)
- Learning rate (type: float, min: 0.005, max: 0.5)

Both Bernoulli naive bayes and Multinomial naive bayes model have one single hyperparameter:

- Additive smoothing parameter (type: float, min: 0.005, max: 5)

In addition, we have a categorical variable indicating which model is selected:

- Selected Model (type: categorical, algorithms: XGboost, Logistic Regression, Bernoulli naive bayes, Multinomial naive bayes)

UNOS-I

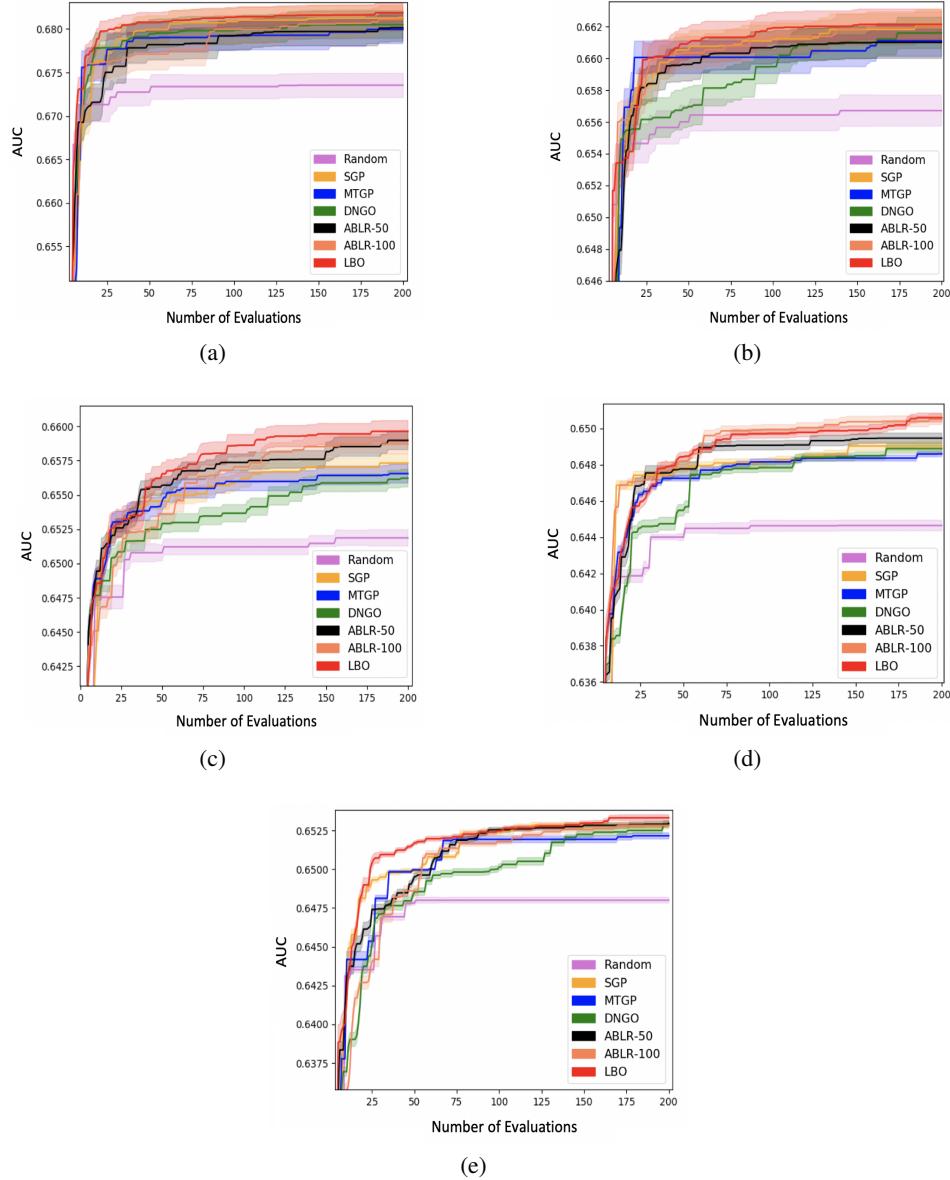


Figure 6: 200 function evaluations of BO algorithms on UNOS-I: (a) 1-year Overlap; (b) 2-years Overlap; (c) 3-years Overlap; (d) 4-years Overlap; (e) 5-years Overlap.

UNOS-II

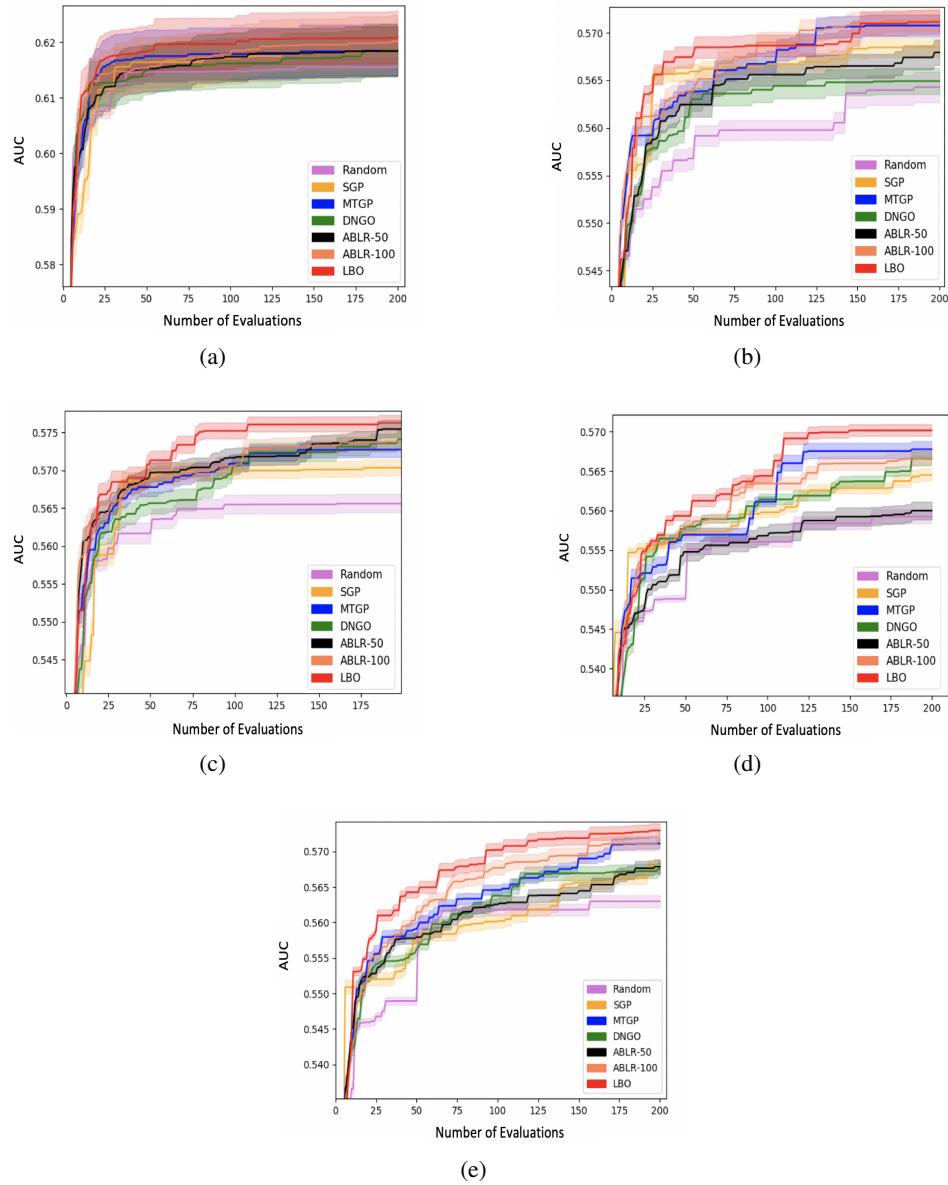


Figure 7: 200 function evaluations of BO algorithms on UNOS-II: (a) 1-year Overlap; (b) 2-years Overlap; (c) 3-years Overlap; (d) 4-years Overlap; (e) 5-years Overlap.

UNOS-III

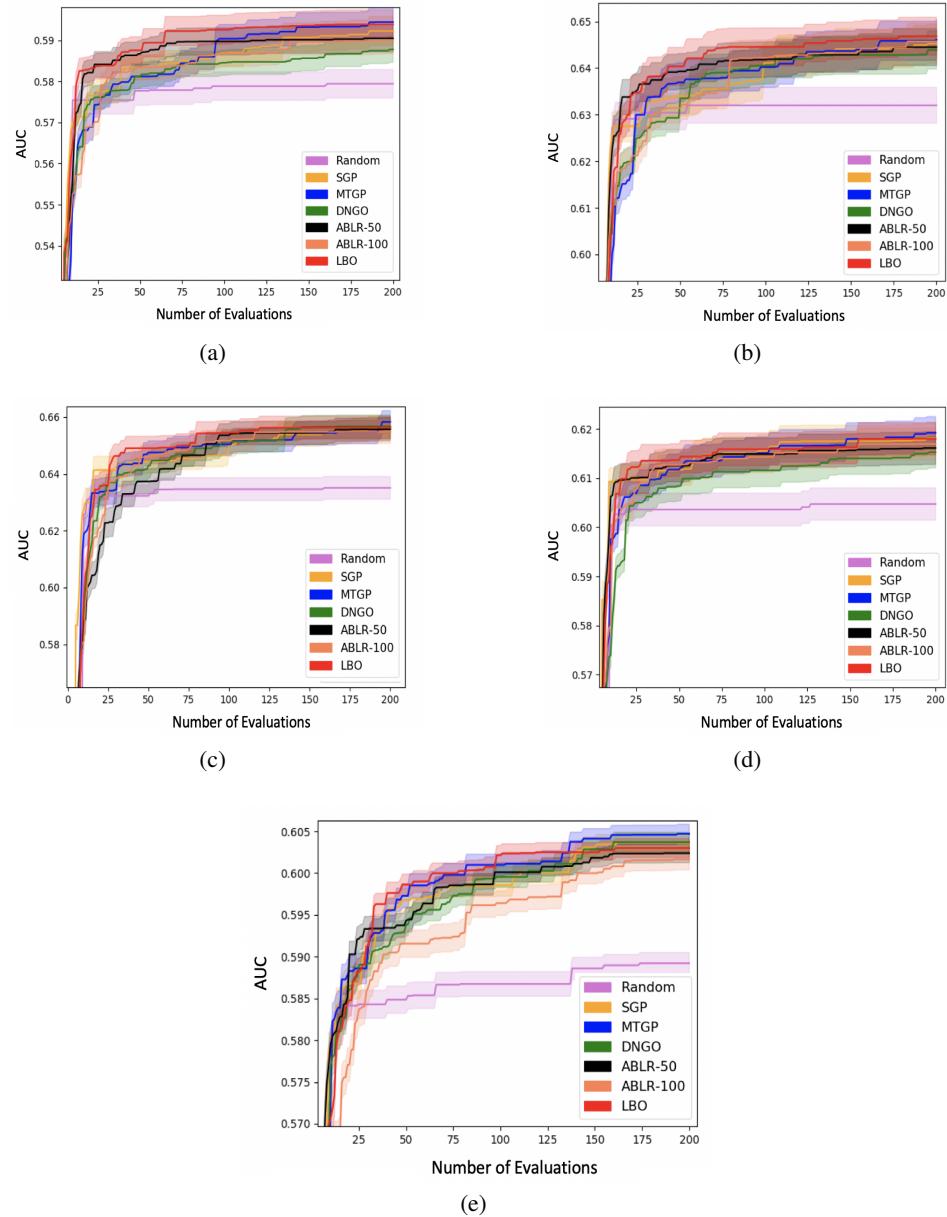


Figure 8: 200 function evaluations of BO algorithms on UNOS-III: (a) 1-year Overlap; (b) 2-years Overlap; (c) 3-years Overlap; (d) 4-years Overlap; (e) 5-years Overlap.

Pseudo-code for Lifelong Bayesian Optimization (LBO)

Algorithm 1 Lifelong Bayesian Optimization

Hyperparameters: Maximum number of acquisitions N_1 , Maximum number of training iterations N_2 , Indian Buffet Process parameter α , Truncated number of neural network M

Input: New dataset \mathcal{D}_t (or equivalently new black-box function f_t), acquisition function a_{f_t}

Initialize: \mathcal{A}_t by performing some random acquisitions of f_t

for $i = 1$ to N_1 **do**

for $j = 1$ to N_2 **do**

Sampling $\pi_m z_{t,m}$ using (5) and (7)

Update Θ_t and $\{\gamma_{t,m}\}_{m=1}^M$ by optimizing the ELBO in (8) regularized by (9)

end for

Update $a_{f_t}(\mathbf{x})$ using (4)

Solve $\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathbb{X}} a_{f_t}(\mathbf{x})$,

$\mathcal{A}_t \leftarrow \mathcal{A}_t \cup (\mathbf{x}^*, y^*)$

end for

Output: (\mathbf{x}, y) with the maximum value of y in \mathcal{A}_t

Table of mathematical notations

Notation	Definition
D	The input dimension of the hyperparameter space \mathbb{X}
D^*	The dimension of the feature map $\phi_{t,m}$
f_t	The t th black-box function
$s_{t,m}$	The weight of m th latent function in f_t
$z_{t,m}$	Indicator of whether g_m is used in f_t
g_m	The m th latent function (neural network)
$\phi_{t,m}$	the m th feature map in the spanning of f_t
$\mathbf{W}_{t,m}$	the parameters in the feature map $\phi_{t,m}$
Θ_t	the parameters of all the feature maps $\phi_{t,m}, m = 1, \dots, M$
\mathbf{h}_m	The output layer of the m th neural network
a_f	Acquisition function
\mathcal{A}_t	Acquisition set for f_t
$\mathcal{A}_{\leq t}$	The union of acquisition sets up to time t
\mathbf{X}_t	Input data in \mathcal{A}_t
\mathbf{Y}_t	Output data in \mathcal{A}_t
N_t	The total number of acquisitions of f_t
M	The truncated number of latent functions in IBP
α	the parameter of Indian Buffet Process
v_m	the m th Beta sample in the Stick breaking Process
π_m	the probability of using the m th neural network
$\gamma_{t,m}$	the variational parameter in the BinConcrete distribution $Q_{\gamma_{t,m}, \tau_{t,m}}$
$\tau_{t,m}$	the temperature hyperparameter in the BinConcrete distribution $Q_{\gamma_{t,m}, \tau_{t,m}}$
λ_t	precision parameter in the prior distribution $P(\mathbf{h}_{t,m} \lambda_t)$
β_t	precision parameter in the likelihood distribution $P(\mathbf{Y}_t \mathbf{f}_t, \beta_t)$

Table 4: Table of mathematical notations