MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Transfer Learning for Bayesian Optimization with Principal Component Analysis

Masui, Hideyuki; Romeres, Diego; Nikovski, Daniel N.

TR2022-169 December 20, 2022

Abstract

Bayesian Optimization has been widely used for black-box optimization. Especially in the field of machine learning, BO has obtained remarkable results in hyperparameters optimization. However, the best hyperparameters depend on the specific task and traditionally the BO algorithm needs to be repeated for each task. On the other hand, the relationship between hyperparameters and objectives has similar tendency among tasks. Therefore, transfer learning is an important technology to accelerate the optimization of novel task by leveraging the knowledge acquired in prior tasks. In this work, we propose a new transfer learning strategy for BO. We use information geometry based principal component analysis (PCA) to extract a low-dimension manifold from a set of Gaussian process (GP) posteriors that models the objective functions of the prior tasks. Then, the low dimensional parameters of this manifold can be optimized to adapt to a new task and set a prior distribution for the objective function of the novel task. Experiments on hyperparameters optimization benchmarks show that our proposed algorithm, called BO-PCA, accelerates the learning of an unseen task (less data are required) while having low computational cost.

International Conference on Machine Learning and Applications (ICMLA) 2022

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Transfer Learning for Bayesian Optimization with Principal Component Analysis

1st Hideyuki Masui
Mitsubishi Electric Corporation
Kamakura, Kanagawa, Japan
Masui.Hideyuki@bc.MitsubishiElectric.co.jp

2nd Diego Romeres
Mitsubishi Electric Research Labs
Cambridge, MA, USA
romeres@merl.com

3rd Daniel Nikovski
Mitsubishi Electric Research Labs
Cambridge, MA, USA
nikovski@merl.com

***Abstract*—Bayesian Optimization has been widely used for black-box optimization. Especially in the field of machine learning, BO has obtained remarkable results in hyperparameters optimization. However, the best hyperparameters depend on the specific task and traditionally the BO algorithm needs to be repeated for each task. On the other hand, the relationship between hyperparameters and objectives has similar tendency among tasks. Therefore, transfer learning is an important technology to accelerate the optimization of novel task by leveraging the knowledge acquired in prior tasks. In this work, we propose a new transfer learning strategy for BO. We use information geometry based principal component analysis (PCA) to extract a low-dimension manifold from a set of Gaussian process (GP) posteriors that models the objective functions of the prior tasks. Then, the low dimensional parameters of this manifold can be optimized to adapt to a new task and set a prior distribution for the objective function of the novel task. Experiments on hyperparameters optimization benchmarks show that our proposed algorithm, called BO-PCA, accelerates the learning of an unseen task (less data are required) while having low computational cost.**

***Index Terms*—Bayesian optimization, Gaussian process, transfer learning, principal component analysis**

I. Introduction

Bayesian Optimization (BO) is a well-established framework for solving black-box optimization problems of the form:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}), \tag{1}$$

where $f(\cdot)$ is an expensive-to-evaluate objective function $f : \mathcal{X} \to \mathbb{R}$ and $\mathcal{X} \subset \mathbb{R}^D$ denotes the configuration space. BO has been applied with remarkable results in a wide range of areas, such as hyperparameter optimization [4], [20], robotics [11], [16], materials design [29], medical science [22] and many others. BO searches for the global optimum $\boldsymbol{x}^*$ by iteratively selecting input configurations and updating a surrogate model of the objective function f. In the case of hyperparameter optimization (HPO) task, $\boldsymbol{x}$ is a vector of hyperparameters and f represents a validation error.

In this work, we focus on transfer learning for black-box optimization [2]. We assume that we have already completed T somewhat similar black-box optimization tasks $\{f_t\}_{t=1}^T$ defined over the same configuration space $\mathcal{X}$. If a new similar black-box optimization task is given, it could be beneficial to leverage the prior tasks as ancillary information when solving the new one. Hence, our goal is to leverage prior source tasks' data sets to accelerate the optimization of the test task f_{T+1}.

Our algorithm accomplishes the knowledge transfer by extracting a low-dimensional manifold and parameter vector from the prior tasks' data, based on information geometry, and updating the parmater vector to solve the new task. This approach has several advantages. First, by using information geometry, a low-dimensional parameter vector is learned for improved stability even though a limited number of source tasks and observations. Second, the optimization problem of these few parameters in each iteration of BO can be converted to one of linear least squares. These advantages are important, because in real situations, it is commonly difficult to collect a lot of data from source tasks to train complex models. Furthermore, we often have to use limited computer resources.

Contributions. Our main contributions are as follow:

1) We introduce a new transfer learning method for BO, called BO-PCA, that uses information geometry-based feature extraction for Gaussian process learning of the objective functions.
2) The proposed algorithm is able to update online the transfer learning model in each iteration of the BO algorithm.
3) We show the effectiveness of our algorithm over s.o.t.a. approaches, using standard HPO benchmark tasks and data sets.

II. Background

A. Bayesian Optimization

BO is a class of machine learning-based algorithms that aim to solve the problem defined in (1), when standard optimization algorithms typically fail due to the complexity of the objective function $f(\boldsymbol{x})$. This class of algorithms is typically composed of iterative procedures that consist of two main computational steps with the objective of exploring the configuration space $\mathcal{X}$ and finding an optimal configuration $\boldsymbol{x}^*$.

The first step consists of constructing a surrogate function of the objective function $f(\cdot)$ using a set of collected observations $\mathcal{D} = (\mathbf{X}, \boldsymbol{y}) = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. We assume that the vector $\boldsymbol{y}$ consists of noisy observations of $f(\cdot)$, i.e.,

$$y_i = f(\boldsymbol{x}_i) + \epsilon, \tag{2}$$

where ϵ is an independent zero-mean Gaussian noise with variance β^{-1}. Typically, the surrogate function is estimated using Gaussian process regression (GPR) [25]. In this case, the objective function is modeled a priori as a Gaussian Process (GP), $f(\cdot) \sim \mathcal{N}(m_0(\cdot), k(\cdot, \boldsymbol{x'}))$, where $m_0(\cdot)$ is an arbitrary mean function and and $k(\cdot, \boldsymbol{x'})$ is the a priori covariance matrix, defined by means of a kernel. Typical choices of the kernel function are the RBF and the Matérn 5/2 kernel function [25]:

$$k_{5/2}(\boldsymbol{x}, \boldsymbol{x}') = \left(1 + \frac{\sqrt{5}r}{\lambda} + \frac{5r^2}{3\lambda^2}\right) \exp\left(-\frac{\sqrt{5}r}{\lambda}\right), \quad (3)$$

where $r = |\boldsymbol{x} - \boldsymbol{x}'|$ and λ is a length scale parameter called a hyperparameter. Every kernel function is parameterized by some hyperparameters and is usually optimized by maximization of the marginal log-likelihood:

$$\log p(\boldsymbol{y}|\mathbf{X}, \boldsymbol{\lambda}) \propto -\log|\mathbf{K}_{\boldsymbol{\lambda}}| - \boldsymbol{y}^T \mathbf{K}_{\boldsymbol{\lambda}}^{-1} \boldsymbol{y}. \quad (4)$$

Remarkably, the posterior distribution of f for an unseen point $\boldsymbol{x}$, namely $\hat{f}(\boldsymbol{x}|\mathcal{D})$, can be computed analytically, and is normally distributed with mean $\mu(\boldsymbol{x})$ and variance $\sigma^2(\boldsymbol{x})$ which are defined below:

$$\mu(\boldsymbol{x}) = m_0(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^T (\mathbf{K} + \beta^{-1}\mathbf{I})^{-1} (\boldsymbol{y} - \boldsymbol{m}_0), \quad (5)$$

$$\sigma^2(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^T (\mathbf{K} + \beta^{-1}\mathbf{I})^{-1} \boldsymbol{k}(\boldsymbol{x}), \quad (6)$$

where $\mathbf{K} = k(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ is defined element-wise through the kernel function $\mathbf{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the a priori covariance matrix, and $\boldsymbol{k}(\boldsymbol{x}) = k(\boldsymbol{x}, \mathbf{X}) \in \mathbb{R}^{N \times 1}$ is a vector with entries $\boldsymbol{k}_i = k(\boldsymbol{x}, \boldsymbol{x}_i)$ and $\boldsymbol{m}_0 := m_0(\mathbf{X}) \in \mathbb{R}^{N \times 1}$ is the mean vector, where each entry is defined as $\boldsymbol{m}_{0,i} = m_0(\boldsymbol{x}_i)$ (typically $m_0(\boldsymbol{x}) \equiv 0$). For later convenience, the posterior distribution $\hat{f}(\boldsymbol{x}|\mathcal{D})$ can also be defined based on the GP posterior of the collected observations, $f(\mathbf{X})$, which is a multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} = \boldsymbol{m}_0 + \mathbf{K}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1} (\boldsymbol{y} - \boldsymbol{m}_0), \quad (7)$$

$$\boldsymbol{\Sigma} = \mathbf{K} - \mathbf{K}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1} \mathbf{K}, \quad (8)$$

After standard computation, an equivalent definition of (5) and (6) based on $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is:

$$\mu(\boldsymbol{x}) = m_0(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^T \mathbf{K}^{-1} (\boldsymbol{\mu} - \boldsymbol{m}_0), \quad (9)$$

$$\sigma^2(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^T \mathbf{K}^{-1} (\boldsymbol{\Sigma} - \mathbf{K}) \mathbf{K}^{-1} \boldsymbol{k}(\boldsymbol{x}). \quad (10)$$

The second step consists in maximizing an acquisition function, which is defined according to an exploration-exploitation criterion. One of the most commonly used function is the expected improvement (EI) [14]:

$$\alpha(\boldsymbol{x}|\mathcal{D}) = \mathbb{E}_{\hat{f}(\boldsymbol{x}|\mathcal{D})}\left[\max(0, \hat{f}(\boldsymbol{x}^+) - \hat{f}(\boldsymbol{x}))\right] \quad (11)$$

$$= \sigma(\boldsymbol{x}) z \Phi(z) + \sigma(\boldsymbol{x}) \phi(z), \quad (12)$$

where $\boldsymbol{x}^+ = \arg\min[\hat{f}(\boldsymbol{x}_1), \hat{f}(\boldsymbol{x}_2), \ldots, \hat{f}(\boldsymbol{x}_N)]$ is the current best configuration, $z = \frac{\hat{f}(\boldsymbol{x}^+) - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$ is a standard normal distribution, and $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and PDF of $\hat{f}(\cdot)$, respectively. W.l.o.g. in this paper, we use the EI function as the acquisition function, but others could be used, e.g. the probability of improvement, the knowledge gradient, etc. The interested reader can refer for example to [6], [19], [21] for a more in depth description of BO.

B. GP-ePCA and GP-mPCA

Principal component analysis (PCA) is a widely used algorithm to extract a low-dimensional linear subspace from high dimensional data that explains that data well. The case in which each of these data points represents a GP distribution was studied in [1], and used in [10] to propose a meta-learning algorithm for GP based on PCA.

Let us assume that there are T data sets $\{\mathcal{D}_t\}_{t=1}^T = \{(\mathbf{X}_t, \boldsymbol{y}_t)\}_{t=1}^T$ with $\mathbf{X}_t \in \mathbb{R}^{N_t \times D}$ and $\boldsymbol{y}_t \in \mathbb{R}^{N_t \times 1}$, where N_t is the number of data points in data set t. With a slight abuse of notation, some of the quantities defined in the previous section for one data set will be redefined for multiple data sets. Let now $\mathbf{X} = \bigcup_{t=1}^T \mathbf{X}_t$ be a union set of the input sets. Following Section II-A, the estimated GP posterior of $\mathbf{X}$ given the t-th data set is a multivariate normal distribution $\hat{f}_t(\mathbf{X}|\mathcal{D}_t) \sim (\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, with

$$\boldsymbol{\mu}_t = \boldsymbol{m}_0(\mathbf{X}) + \mathbf{K}_t(\mathbf{K}_{tt} + \beta^{-1}\mathbf{I})^{-1}(\boldsymbol{y}_t - \boldsymbol{m}_0(\mathbf{X}_t)), \quad (13)$$

$$\boldsymbol{\Sigma}_t = \mathbf{K} - \mathbf{K}_t(\mathbf{K}_{tt} + \beta^{-1}\mathbf{I})^{-1}\mathbf{K}_t^T, \quad (14)$$

where $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_t = k(\mathbf{X}, \mathbf{X}_t)$, $\mathbf{K}_{tt} = k(\mathbf{X}_t, \mathbf{X}_t)$.

Since the multivariate normal distribution belongs to the exponential family, each GP posterior $\hat{f}_t(\mathbf{X}|\mathcal{D}_t)$ is a point on a Riemannian flat manifold, here denoted by $\mathcal{S}$. A flat manifold is spanned by a coordinate system. In particular, there are two coordinate systems, the e-coordinates and m-coordinates, that define two subspaces linear w.r.t. $\boldsymbol{\xi}_t = (\boldsymbol{\theta}_t^T, \text{vec}(\boldsymbol{\Theta}_t)^T)^T$ and $\boldsymbol{\zeta}_t = (\boldsymbol{\eta}_t^T, \text{vec}(\mathbf{H}_t)^T)^T$, respectively, where:

$$\boldsymbol{\theta}_t = \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t, \quad \boldsymbol{\Theta}_t = -\frac{1}{2}\boldsymbol{\Sigma}_t^{-1}, \quad (15)$$

$$\boldsymbol{\eta}_t = \boldsymbol{\mu}_t, \quad \mathbf{H}_t = \boldsymbol{\mu}_t\boldsymbol{\mu}_t^T + \boldsymbol{\Sigma}_t^T. \quad (16)$$

The goal of the GP-ePCA and GP-mPCA methods is to project $\{\boldsymbol{\xi}_t\}_{t=1}^T$ and $\{\boldsymbol{\zeta}_t\}_{t=1}^T$, respectively, into a low dimension linear submanifold $\mathcal{M} \subset \mathcal{S}$ which is expressed by the set of points:

$$\tilde{\boldsymbol{\xi}}_t = (\tilde{\boldsymbol{\theta}}_t^T, \text{vec}(\tilde{\boldsymbol{\Theta}}_t)^T)^T = \mathbf{U}_{\boldsymbol{\xi}}\boldsymbol{w}_t + \boldsymbol{u}_{\boldsymbol{\xi},0}, \quad (17)$$

$$\tilde{\boldsymbol{\zeta}}_t = (\tilde{\boldsymbol{\eta}}_t^T, \text{vec}(\tilde{\mathbf{H}}_t)^T)^T = \mathbf{U}_{\boldsymbol{\zeta}}\boldsymbol{w}_t + \boldsymbol{u}_{\boldsymbol{\zeta},0}. \quad (18)$$

where $\mathbf{U}_{\boldsymbol{\xi}}, \mathbf{U}_{\boldsymbol{\zeta}} \in \mathbb{R}^{(N+N^2) \times L}$ with $L < T$ and $\boldsymbol{w}_t \in \mathbb{R}^{L \times 1}$. When $\{\boldsymbol{\xi}_t\}_{t=1}^T$ or $\{\boldsymbol{\zeta}_t\}_{t=1}^T$ are observed, which is at training time, the task of GP-ePCA and GP-mPCA is to optimize the weights $\mathbf{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_T)^T$, the basis vectors $\mathbf{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_L)$, and the offset $\boldsymbol{u}_0$ by minimizing the sum of the KL divergence:

$$\hat{\mathbf{W}}, \hat{\mathbf{U}}, \hat{\boldsymbol{u}}_0 = \underset{\mathbf{W}, \mathbf{U}, \boldsymbol{u}_0}{\arg\min}\, E(\mathbf{W}, \mathbf{U}, \boldsymbol{u}_0), \quad (19)$$

$$E(\mathbf{W}, \mathbf{U}, \boldsymbol{u}_0) = \begin{cases} \sum_{t=1}^T D_{KL}\left(\boldsymbol{\xi}_t \| \tilde{\boldsymbol{\xi}}_t\right) & \text{(GP-ePCA)} \\ \sum_{t=1}^T D_{KL}\left(\tilde{\boldsymbol{\zeta}}_t \| \boldsymbol{\zeta}_t\right) & \text{(GP-mPCA)} \end{cases}.$$

As a result, GP-ePCA, respectively GP-mPCA, obtain a low-dimensional parameter vector $\boldsymbol{w}_t$ which characterizes each training data set. Therefore, at prediction time, given the weights $\boldsymbol{w}$, the posterior distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}(\boldsymbol{w}), \tilde{\boldsymbol{\Sigma}}(\boldsymbol{w}))$ can be restored by inverting (15) (and (16)). The posterior estimator of a test point $\boldsymbol{x}$ can be computed with mean and variance:

$$\tilde{m}(\boldsymbol{x}, \boldsymbol{w}) = m_0(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^T \mathbf{K}^{-1}(\tilde{\boldsymbol{\mu}}(\boldsymbol{w}) - \boldsymbol{m}_0), \tag{20}$$

$$\tilde{\sigma}^2(\boldsymbol{x}, \boldsymbol{w}) = k(\boldsymbol{x}, \boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^T \mathbf{K}^{-1}(\tilde{\boldsymbol{\Sigma}}(\boldsymbol{w}) - \mathbf{K})\mathbf{K}^{-1}\boldsymbol{k}(\boldsymbol{x}). \tag{21}$$

where $\tilde{\boldsymbol{\mu}}(\boldsymbol{w})$ and $\tilde{\boldsymbol{\Sigma}}(\boldsymbol{w})$ are computed as bellow:

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{w}) = \begin{cases} -0.5\tilde{\boldsymbol{\Theta}}^{-1}\tilde{\boldsymbol{\theta}} & \text{(GP-ePCA)} \\ \tilde{\boldsymbol{\eta}} & \text{(GP-mPCA)} \end{cases}, \tag{22}$$

$$\tilde{\boldsymbol{\Sigma}}(\boldsymbol{w}) = \begin{cases} -0.5\tilde{\boldsymbol{\Theta}}^{-1} & \text{(GP-ePCA)} \\ \tilde{\mathbf{H}} - \tilde{\boldsymbol{\eta}}\tilde{\boldsymbol{\eta}}^T & \text{(GP-mPCA)} \end{cases}. \tag{23}$$

C. Sparse GP-ePCA and GP-mPCA

The major problem of GP-PCA is the calculation cost. If the size of $\mathbf{X} = \bigcup_{t=1}^{T} \mathbf{X}_t$ is $|\mathbf{X}| = N$, the computation cost of GP-PCA becomes $\mathcal{O}(N^3)$. The authors of GP-PCA [10] also proposed sparse GP-PCA to reduce calculation cost which is based on sparse Gaussian process [12]. Let $\mathbf{Z}$ be a set of inducing points $\mathbf{Z} = \bigcup_{t=1}^{T} \mathbf{Z}_t$, $\mathbf{K}_z = k(\mathbf{X}, \mathbf{X}_z)$ and $\mathbf{K}_{zz} = k(\mathbf{Z}, \mathbf{Z})$. The covariance matrix can be approximated as bellow:

$$\mathbf{K} \approx \mathbf{K}_z \mathbf{K}_{zz}^{-1} \mathbf{K}_z^T. \tag{24}$$

In the case of sparse GP-PCA, the posterior distribution of each task corresponds to (13) and (14) is defined as below:

$$\boldsymbol{\mu}_t = \boldsymbol{m}_0(\mathbf{X}_z) + \mathbf{K}_{zz}\mathbf{A}_{zz}^{-1}\mathbf{K}_{zt}(\boldsymbol{y}_t - \boldsymbol{m}_0(\mathbf{X}_t)), \tag{25}$$

$$\boldsymbol{\Sigma}_t = \beta^{-1}\mathbf{K}_{zz}\mathbf{A}_{zz}^{-1}\mathbf{K}_{zz}, \tag{26}$$

where $\mathbf{A}_{zz} = \beta^{-1}\mathbf{K}_{zz} + \mathbf{K}_z^T\mathbf{K}_z$ and $\mathbf{K}_{zt} = k(\mathbf{X}_z, \mathbf{X}_t)$. Let the number of inducing points as $|\mathbf{Z}_t| = M$ and share this for all tasks, $\mathbf{Z}_1 = \mathbf{Z}_2 = \cdots = \mathbf{Z}_T$. In this case, the calculation cost of GP-PCA is reduced from $\mathcal{O}(N^3)$ to $\mathcal{O}(M^3)$.

III. Problem Formulation

Consider T source tasks available at training time, each of this task is defined by an objective function $\{f_t\}_{t=1}^T$ and for each task N_t labeled data are collected $\{\mathcal{D}_t\}_{t=1}^T = \{\{(\boldsymbol{x}_{t,n}, y_{t,n})\}_{n=1}^{N_t}\}_{t=1}^T$. Given the data sets $\{\mathcal{D}_t\}_{t=1}^T$, it is possible to estimate T surrogate functions, using GPR or any othe machine learning function approximator. We are interested in a data-efficient transfer learning BO algorithm capable of transferring the knowledge acquired in the T source tasks to solve a novel test task ($t = T + 1$), i.e.,

$$\boldsymbol{x}_{T+1}^* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} f_{T+1}(\boldsymbol{x} | \{\mathcal{D}_t\}_{t=1}^T). \tag{27}$$

The question we want to answer is: "How can we learn efficiently, in terms of data complexity, a new task, having previously learned T similar source tasks?"

A. Related Work

Standard BO without transfer learning has been already discussed in the previous sections, see [19] for a survey. The use of PCA in BO was proposed in [17] to find a low dimensional space for high dimensional data point where the surrogate of the objective function is learned. This is very different from our approach since PCA was used in the observations and not in the GP posteriors like in GP-PCA. Moreover, in this work we focus on transfer learning of BO from source tasks to a novel unseen task. This problem attracted a lot of research and is strongly connected to the concept of meta-learning [9], [24]. In [2], the authors propose a series of benchmark tasks, data sets as well as metrics to facilitate the comparison among different methods. Moreover, the authors of [5] not only propose a method for transfer learning BO called RGPE, but they also make available the data sets and the implementation of s.o.t.a. methods. We will refer to this implementation and benchmarks to compare our algorithm. The main algorithms for BO transfer learning are summarized in the following. ABLR (adaptive Bayesian linear regression) [15], is a multi-task Bayesian linear regression approach. Transfer learning is achieved by sharing the deep neural network which is trained with source tasks data. Then, separate Bayesian linear regressors are embedded for each task. TST-R [27] (two-stage transfer surrogate using pairwise hyperparameter performance rankings) is an ensemble approach. At the first stage, a surrogate model is trained for each source and test task. Then, at the second stage, the weights are calculated for each model using similarity between the test task and each source task. TAF-R (transfer acquisition function framework using pairwise hyperparameter performance rankings) [28], is a similar ensemble approach to TST-R. The deference between TST-R is that TAF-R forms ensembles of acquisition functions in order to overcome different scales of evaluation scores. Finally, RGPE (ranking-weighted Gaussian process ensemble) [5], is another ensemble surrogate models approach similar to TST-R. The weights are calculated by optimizing a ranking loss using surrogate models and test task data.

IV. BO-PCA

We propose to solve the transfer learning BO problem (27) by learning the surrogate function with a GP-PCA method (both GP-ePCA and GP-mPCA are possible). We call this approach BO-PCA. GP-PCA is a principled way of encoding the GP posteriors in a low-dimensional manifold, and at training time it defines the space where the posterior of the novel $T + 1$ task can be (data) efficiently learned by creating transfer learning model that depends on a low dimensional set of parameters easier to update online. The proposed method is described in Algorithm 1.

During training time, we learn the $\{\hat{f}_t(\mathbf{X}|\mathcal{D}_t)\}_{t=1}^T$ surrogate functions of the source tasks using GP-ePCA (or GP-mPCA), to obtain L basis $\hat{\mathbf{U}}$ and the offset $\hat{\boldsymbol{u}}_0$, as described in Section II-B. Then, in order to optimize an unseen test task, f_{T+1}, we fix the basis functions $\{\hat{\mathbf{U}}, \hat{\boldsymbol{u}}_0\}$, collect a small number of data n_0 from f_{T+1}, and each time a new n-th

Algorithm 1 BO with GP-PCA Prior

Input: source tasks data $\{\mathcal{D}_t\}_{t=1}^{T}$, number of initial points n_0, budget N_{t+1}, GP-PCA manifold dimension L.
Execute GP-ePCA to obtain $\hat{\mathbf{W}}_{\boldsymbol{\xi}}, \hat{\mathbf{U}}_{\boldsymbol{\xi}}, \hat{\boldsymbol{u}}_{\boldsymbol{\xi},0}$ (19)
(Execute GP-mPCA to obtain $\hat{\mathbf{W}}_{\boldsymbol{\zeta}}, \hat{\mathbf{U}}_{\boldsymbol{\zeta}}, \hat{\boldsymbol{u}}_{\boldsymbol{\zeta},0}$ (19))
Observe f_{T+1} at n_0 points $\mathcal{D}_{T+1} = \{\boldsymbol{x}_{T+1,n}, y_{T+1,n}\}_{n=1}^{n_0}$
Set $n = n_0$
while $n < N_{T+1}$ **do**
  Fit weights $\hat{\boldsymbol{w}}$ of GP-ePCA (GP-mPCA) (28)
  Update weight $\boldsymbol{w}_{T+1} = \hat{\boldsymbol{w}}$
  Set prior mean $m_0(\boldsymbol{x}) = \tilde{m}(\boldsymbol{x}, \boldsymbol{w}_{T+1})$ for f_{T+1}
  Learn $\hat{f}_t(\mathbf{X}|\mathcal{D}_{T+1})$ using GPR
  Optimize $\boldsymbol{x}_{T+1,n} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \alpha(\boldsymbol{x}|\mathcal{D}_{T+1})$
  Observe $y_{T+1,n} = f_{T+1}(\boldsymbol{x}_{T+1,n})$
  Update $\mathcal{D}_{T+1} = \mathcal{D}_{T+1} \cup \{(\boldsymbol{x}_{T+1,n}, y_{T+1,n})\}$
  $n = n + 1$
end while
Output: $\hat{\boldsymbol{x}}_{T+1} = \arg\min_{n=1,2,\ldots,N_{T+1}} f_{T+1}(\boldsymbol{x}_{T+1,n})$

data point $(\boldsymbol{x}_{T+1,n}, y_{T+1,n})$ arrives, the weight vector, $\boldsymbol{w}_{T+1}$ is updated by minimizing the least-squares problem:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^L} \|\boldsymbol{y}_{T+1} - \tilde{\boldsymbol{m}}(\mathbf{X}_{t+1}, \boldsymbol{w})\|_2^2, \tag{28}$$

where $\tilde{\boldsymbol{m}}(\mathbf{X}_{t+1}, \boldsymbol{w})$ is a $n_{T+1} \times 1$ vector where each entry $\tilde{\boldsymbol{m}}_i(\mathbf{X}_{t+1}, \boldsymbol{w}) = \tilde{m}(\boldsymbol{x}_{T+1,i}, \boldsymbol{w})$. We use this GP-PCA estimated function $\tilde{m}(\boldsymbol{x}, \hat{\boldsymbol{w}})$ as a prior mean function of f_{T+1}, that is $m_0(\boldsymbol{x}) = \tilde{m}(\boldsymbol{x}, \boldsymbol{w}_{T+1})$. We can therefore use GPR to learn the surrogate objective function of $\hat{f}_t(\mathbf{X}|\mathcal{D}_{T+1})$.The subsequent steps of each iteration follow in accordance with standard BO algorithms.

Remark 1: Note that after computing the optimal vector of weights $\hat{\boldsymbol{w}}$, we could compute both the mean and covariance GP-PCA functions, see equations (22) and (23), respectively. However, the source tasks commonly have many more data points than the test task, and for this reason, the GP-PCA covariance function tends to significantly affect the GP posterior variance (10), which leads to two practical limitations if used in Algorithm 1. First, given the many data points in the source tasks, the estimated variance tends to be narrow, preventing the BO algorithm to explore sufficiently. Second, if some test task configurations are far away from all source task configurations, the posterior variance tends to be erroneously very large, only because that point was not seen during training. For these reasons, only the GP-PCA mean function is used to transfer the knowledge of the source tasks to the novel task.

A. GP-ePCA vs GP-mPCA

Next, we discuss the main differences between GP-ePCA and GP-mPCA that are relevant to implementing our transfer learning BO optimization algorithm. In Algorithm 1, the weights are optimized by solving problem (28) at each iteration (i.e., every time a new data pair becomes available). In the case of GP-ePCA, the estimation vector $\tilde{\boldsymbol{\mu}}(\boldsymbol{w})$ is calculated as shown below, following (22),

$$\tilde{\boldsymbol{\mu}}_{\text{ePCA}}(\boldsymbol{w}) = -0.5\Big(\hat{\mathbf{U}}_{\boldsymbol{\Theta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\Theta}}\Big)^{-1}\Big(\hat{\mathbf{U}}_{\boldsymbol{\theta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\theta}}\Big), \tag{29}$$

where,

$$\begin{bmatrix}\tilde{\boldsymbol{\theta}}(\boldsymbol{w}) \\ \text{vec}\left(\tilde{\boldsymbol{\Theta}}(\boldsymbol{w})\right)\end{bmatrix} = \hat{\mathbf{U}}_{\boldsymbol{\xi}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{\boldsymbol{\xi},0} = \begin{bmatrix}\hat{\mathbf{U}}_{\boldsymbol{\theta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\theta}} \\ \hat{\mathbf{U}}_{\boldsymbol{\Theta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\Theta}}\end{bmatrix}. \tag{30}$$

Analogously, in the case of GP-mPCA:

$$\tilde{\boldsymbol{\mu}}_{\text{mPCA}}(\boldsymbol{w}) = \hat{\mathbf{U}}_{\boldsymbol{\eta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\eta}}, \tag{31}$$

where

$$\begin{bmatrix}\tilde{\boldsymbol{\eta}}(\boldsymbol{w}) \\ \text{vec}\left(\tilde{\mathbf{H}}(\boldsymbol{w})\right)\end{bmatrix} = \hat{\mathbf{U}}_{\boldsymbol{\zeta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_0 = \begin{bmatrix}\hat{\mathbf{U}}_{\boldsymbol{\eta}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\boldsymbol{\eta}} \\ \hat{\mathbf{U}}_{\mathbf{H}}\boldsymbol{w} + \hat{\boldsymbol{u}}_{0,\mathbf{H}}\end{bmatrix}. \tag{32}$$

It follows that $\tilde{\boldsymbol{\mu}}_{\text{mPCA}}(\boldsymbol{w})$ is a linear function of the weights $\boldsymbol{w}$, while $\tilde{\boldsymbol{\mu}}_{\text{ePCA}}(\boldsymbol{w})$ is a non-linear function of the same weights. For this reason, the optimization problem (28) for GP-ePCA is much harder to solve and in practice standard gradient-based methods were failing to obtain optimal weights in many experiments. To make GP-ePCA working we had to apply derivative-free optimization algorithms such as CMA-ES [8], which are in general slower to compute. Therefore, GP-mPCA is superior to GP-ePCA from the point of view of calculation cost.

B. Online GP-mPCA

Following the discussion in the previous section the weight optimization problem (28) for GP-mPCA is linear and is obtained as below by substituting (31) into (20):

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^L} \left\|\boldsymbol{y}_{T+1} - k(\mathbf{X}_{t+1}, \mathbf{X})\mathbf{K}^{-1}\tilde{\boldsymbol{\mu}}(\boldsymbol{w})\right\|_2^2, \tag{33}$$

$$= \arg\min_{\boldsymbol{w}\in\mathbb{R}^L} \left\|\boldsymbol{y}'_{T+1} - \phi(\mathbf{X}_{t+1})\boldsymbol{w}\right\|_2^2, \tag{34}$$

where

$$\boldsymbol{y}'_{T+1} = \boldsymbol{y}_{T+1} - k(\mathbf{X}_{t+1}, \mathbf{X})\mathbf{K}^{-1}\hat{\boldsymbol{u}}_{0,\boldsymbol{\eta}}, \tag{35}$$

$$\phi(\mathbf{X}_{T+1}) = k(\mathbf{X}_{t+1}, \mathbf{X})\mathbf{K}^{-1}\hat{\mathbf{U}}_{\boldsymbol{\eta}}. \tag{36}$$

Hence, (34) is a least squares problem and can be solved recursively by means of the recursive least squares algorithm [13, Chp. 11]. Therefore, the weights at the n-th iteration can be updated exactly based on the weights at the $n-1$-th iteration, without having to solve the whole optimization problem again, and the update does not depend on the number of data, leading to a fast weight update step.

C. Initialization of GP-mPCA

The GP-PCA algorithm uses gradient descent in (19) to optimize $\mathbf{U}$, $\boldsymbol{u}_0$ and $\mathbf{W}$. That is, it is not guaranteed to converge to the global optimum. Therefore, initialization of $\mathbf{U}$, $\boldsymbol{u}_0$ and $\mathbf{W}$ is very important. The authors of m-PCA (e-PCA) [1] suggest to use m-center (e-center) for $\boldsymbol{u}_0$ and to compute

TABLE I
SEARCH SPACE AND HYPERPARAMETERS DEFINITION FOR EACH BENCHMARK.

Quadratic			AdaBoost			SVM			NN		
Name	Range	Log	Name	Range	Log	Name	Range	Log	Name	Range	Log
x_0	$[-5, 5]$	No	# Iterations	$[2, 10^4]$	Yes	Use linear kernel	{No, Yes}	No	Batch size	$[16, 512]$	Yes
x_1	$[-5, 5]$	No	# Terms	$[2, 30]$	Yes	Use polynomial kernel	{No, Yes}	No	Learning rate	$[10^{-4}, 10^{-1}]$	Yes
x_2	$[-5, 5]$	No				Use RBF kernel	{No, Yes}	No	Momentum	$[0.1, 0.99]$	No
						Trade-off parameter	$[2^{-5}, 2^6]$	No	Weight decay	$[10^{-5}, 10^{-1}]$	Yes
						Degree of the kernel	$[2, 10]$	No	# Layers	$[1, 5]$	No
						Width of the kernel	$[10^{-4}, 10^3]$	Yes	# Units per layer	$[64, 1024]$	Yes
									Dropout	$[0.0, 1.0]$	No

standard PCA to initialize $\mathbf{U}$ and $\mathbf{W}$. The definition of m-center (e-center) is given by the arithmetic mean of samples in m-coordinate (e-coordinate),

$$\hat{\boldsymbol{u}}_0 = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\eta}_t, \qquad \left(\hat{\boldsymbol{u}}_0 = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\theta}_t\right). \tag{37}$$

In order to leverage the linearity of GP-mPCA, we modify its initialization. In light of the fact that our algorithm only requires the mean $\tilde{\boldsymbol{\mu}}(\boldsymbol{w})$ of the restored GP distribution and $\tilde{\boldsymbol{\mu}}(\boldsymbol{w})$ is obtained only with $\hat{\mathbf{U}}_{\boldsymbol{\eta}}$ and $\hat{\boldsymbol{u}}_{0,\boldsymbol{\eta}}$ (and not also $\hat{\mathbf{U}}_{\mathbf{H}}$ and $\hat{\boldsymbol{u}}_{0,\mathbf{H}}$) in (31), it is not necessary to extract low-dimension features from $\{\mathbf{H}_t\}_{t=1}^T$, defined in (16).

We propose to expand the basis functions of $\mathbf{U}$ from L to $L+T$ which corresponds to also increase the dimensionality of the weights $\mathbf{W}$ to $L+T$

$$\mathbf{U}\mathbf{W} + \boldsymbol{u}_0 = \begin{bmatrix} \mathbf{U}_{\boldsymbol{\eta},\boldsymbol{\eta}} & \mathbf{U}_{\boldsymbol{\eta},\mathbf{H}} \\ \mathbf{U}_{\mathbf{H},\boldsymbol{\eta}} & \mathbf{U}_{\mathbf{H},\mathbf{H}} \end{bmatrix} \begin{bmatrix} \mathbf{W}_{\boldsymbol{\eta}} \\ \mathbf{W}_{\mathbf{H}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}_{\mathbf{0},\boldsymbol{\eta}} \\ \boldsymbol{u}_{\mathbf{0},\mathbf{H}} \end{bmatrix}, \tag{38}$$

where each matrix and vector is initialized as below.

- $\boldsymbol{u}_{\mathbf{0},\boldsymbol{\eta}}$: m-center
- $\mathbf{U}_{\boldsymbol{\eta},\boldsymbol{\eta}}, \mathbf{W}_{\boldsymbol{\eta}}$: PCA
- $\mathbf{U}_{\mathbf{H},\mathbf{H}} = \left[\text{vec}(\boldsymbol{\Sigma}_1 + \tilde{\boldsymbol{\eta}}_1\tilde{\boldsymbol{\eta}}_1^T), \ldots, \text{vec}(\boldsymbol{\Sigma}_T + \tilde{\boldsymbol{\eta}}_T\tilde{\boldsymbol{\eta}}_T^T)\right]$
- $\mathbf{U}_{\boldsymbol{\eta},\mathbf{H}} = \mathbf{O}$, $\mathbf{U}_{\mathbf{H},\boldsymbol{\eta}} = \mathbf{O}$, $\mathbf{W}_{\mathbf{H}} = \mathbf{I}$, $\boldsymbol{u}_{\mathbf{0},\mathbf{H}} = \mathbf{0}$

This implies that the features $\mathbf{U}_{\boldsymbol{\eta},\boldsymbol{\eta}}$ and the weights $\mathbf{W}_{\boldsymbol{\eta}}$ are extracted to characterize only the mean functions $\{\boldsymbol{\mu}_t\}_{t=1}^T$ and not the covariance functions. Moreover, this expansion theoretically means that the covariance function can be reconstructed exactly i.e., $\{\boldsymbol{\Sigma}_t\}_{t=1}^T = \{\tilde{\boldsymbol{\Sigma}}(\boldsymbol{w}_t)\}_{t=1}^T$.

V. EXPERIMENTS

In this section, we aim to validate the proposed transfer learning BO algorithm w.r.t. state-of-the-art algorithms in terms of data efficiency, accuracy and computational time. BO-PCA is compared against the transfer learning BO algorithms described in Section III-A, namely ABLR [15], TST-R [27], TAF-R [28], RGPE [5] and Vanilla BO. The implementation of these baselines, and of the benchmark datasets was made available in the AutoML/transfer-hpo-framework repositories[1] [5], and we rely on the same implementation for fairness of comparison. The proposed algorithm is implemented in Python and uses GPy[2] for standard GPR. All computation-time based experiments were performed with a 8-core i7-4790K CPU with 16 GB of memory on an Ubuntu machine.

A. HPO Benchmark Problems

We consider 4 well known benchmark functions to evaluate the performance of the algorithms. Table II summarizes the main properties of each benchmark, such as the dimension of the hyperparameters to learn, the number of tasks, the number of sampling data collected for each task, and their respective references.

TABLE II
SUMMARY OF BENCHMARKS

Name	# Dim	N_{tasks}	N_t	Reference
Quadratic Function	3	30	50	[15]
AdaBoost Grid	2	50	50	[18]
SVM Grid	6	50	50	[18]
NN Grid	7	35	50	[30]

The first benchmark consists of parameterized quadratic functions [15]:

$$f_t(\boldsymbol{x}|a_t, b_t, c_t) = a_t \|\boldsymbol{x}\|_2^2 + b_t \mathbf{1}^T\boldsymbol{x} + c_t, \tag{39}$$

where the three coefficients are sampled uniformly from $a, b, c \sim \text{Uniform}(0.1, 10)$. Then, we use two HPO benchmarks called Adaboost and SVM defined in [18]. These benchmarks were created using 50 classification datasets chosen randomly from the UCI repository[3]. They are denominated grid benchmarks, because their test accuracy was pre-computed on a discretized grid of hyperparameters, 2 and 6, respectively. Finally, the last HPO benchmark is composed of deep neural networks defined in [30]. This benchmark was created using training the networks in 35 different tasks in the OpenML datasets [7] and accounting for hyperparameters of dimension 7. The test accuracy was pre-computed for 2,000 configurations, which were sampled randomly. The hyperparameters meaning and search space of each benchmark are shown in Table I. The 'log' column shows whether the hyperparameters are converted into the logarithmic scale for numerical stability purposes.

[1]https://github.com/automl/transfer-hpo-framework

[2]https://sheffieldml.github.io/GPy/

[3]https://archive.ics.uci.edu/ml/index.php

TABLE III
NORMALIZED REGRET

	Quadratic [$\times 10^5$]					AdaBoost [$\times 10^2$]				
	10	20	30	40	50	10	20	30	40	50
Vanilla BO	519	10.7	0.82	0.69	0.59	4.07	2.27	**1.40**	**0.74**	**0.57**
ABLR	776	519	287	200	169	4.29	2.43	1.56	1.11	0.68
TST-R	734	704	677	598	536	4.46	2.76	1.76	1.16	**0.71**
TAF-R	78.8	8.33	1.01	0.84	0.72	**3.91**	2.36	**1.53**	**0.89**	**0.59**
RGPE	141	83.5	58.2	46.3	31.1	4.75	2.51	**1.55**	**0.92**	**0.55**
BO-ePCA	**108**	1.39	**0.48**	0.44	**0.40**	**4.02**	**2.26**	**1.59**	1.23	0.95
BO-mPCA	**76.7**	**0.79**	**0.42**	**0.35**	**0.34**	**3.91**	**2.21**	**1.52**	1.23	1.00
	SVM [$\times 10^2$]					NN [$\times 10^2$]				
	10	20	30	40	50	10	20	30	40	50
Vanilla BO	4.70	2.55	1.56	0.91	0.74	4.64	2.70	1.62	**1.08**	**0.86**
ABLR	4.29	2.43	1.56	1.11	0.68	5.12	3.64	3.30	2.73	2.43
TST-R	5.37	3.97	3.06	2.19	1.77	4.73	3.58	2.50	1.81	1.54
TAF-R	4.69	2.52	1.65	0.95	0.66	4.37	2.39	1.39	**0.98**	**0.85**
RGPE	5.72	3.94	2.91	1.96	1.43	**3.23**	**1.91**	1.50	1.21	1.03
BO-ePCA	**3.97**	1.77	**0.85**	**0.54**	**0.33**	**3.30**	**1.72**	**1.21**	**0.99**	**0.90**
BO-mPCA	**3.97**	**1.54**	**0.80**	**0.40**	**0.27**	**3.25**	**1.67**	**1.23**	**1.02**	**0.88**

B. Experimental Design

For each benchmark, we perform the experiments in a leave-one-task-out style. That is, among all possible tasks, we pick one as the test task, and all the others are considered source tasks. This experiment is repeated in turn, so that each and every task becomes a test task.

From each source task, $N_t = 50$ points were randomly collected and the number of basis functions in BO-PCA was set to $L = 1$. We used sparse GP-PCA for all experiments. The number of inducing points selected was $|\mathbf{Z}| = 30$ for the Quadratic and AdaBoost benchmark, and $|\mathbf{Z}| = 50$ for the SVM and NN benchmark. All the inducing points were collected by Latin hypercube sampling [23]. At test time, all 5 methods started by collecting the same $n_0 = 5$ initial points, which were generated by a sequential model-free hyperparameter optimization [26], from the test task function. The number of total data collected for each test task was set to $N_{T+1} = 50$, corresponding to 45 iterations of the BO transfer learning algorithms.

After performing these experiments, so that each and every task became a test task, we averaged the performance across test tasks for each benchmark. Two evaluation metrics were computed: the average normalized regret and the average rank, that have been proposed as standard metrics for HPO problems in [2]. First, the average normalized regret represents the average of 0-1 range scaled regret on each task. The definition of average normalized regret after N iterations is

$$r_N = \frac{1}{N_{\text{tasks}}} \sum_{t=1}^{N_{\text{tasks}}} \min_{n \in [1,\dots,N]} \frac{y_{t,n} - f_t^{\min}}{f_t^{\max} - f_t^{\min}}, \tag{40}$$

where N_{tasks} is a number of tasks of the benchmark, $y_{t,n}$ is the observation of task t at the $n-$th iteration and $f_t^{\min}$ and $f_t^{\max}$ are the best and worst global values of the function that are available for evaluation purposes only:

$$f_t^{\min} = \min_{\boldsymbol{x}} f_t(\boldsymbol{x}), \;\; f_t^{\max} = \max_{\boldsymbol{x}} f_t(\boldsymbol{x}). \tag{41}$$

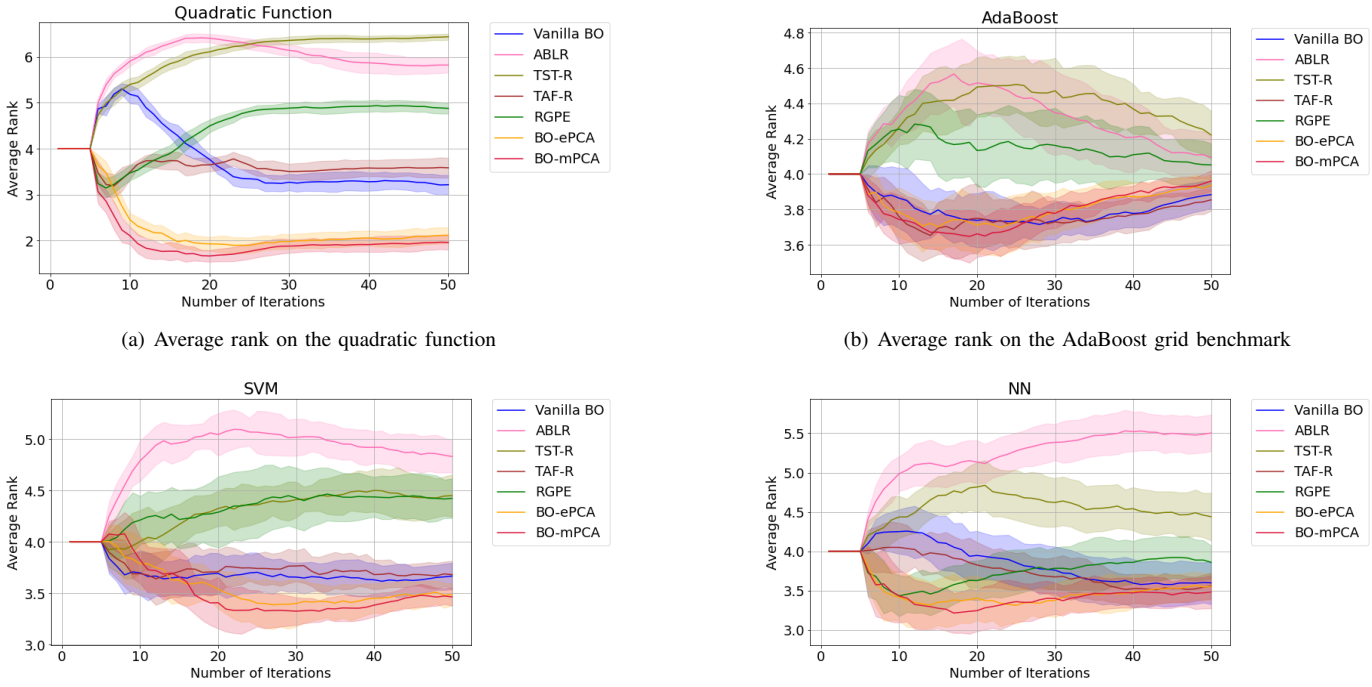
Second, the competing methods are ranked at each iteration accordingly to the normalized regret, going from the best method with score equal 1 to the worst one with increasing score. Every experiment is repeated 15 times and each time a different set of $N_t = 50$ data point is randomly selected for each source task.

TABLE IV
COMPUTATION TIME [MS]

Method	Quadratic 10	Quadratic 30	Quadratic 50	AdaBoost 10	AdaBoost 30	AdaBoost 50
ABLR	33K	14K	35K	83K	31K	2.9K
TST-R	13.7	40.1	155	24.1	68.5	234
TAF-R	13.7	40.3	154	24.2	68.4	233
RGPE	63.5	234	762	86.4	334	1072
BO-ePCA	299	310	473	330	335	464
BO-mPCA	**1.50**	**1.59**	**1.73**	**1.57**	**1.57**	**1.71**
Method	SVM 10	SVM 30	SVM 50	NN 10	NN 30	NN 50
ABLR	60K	82K	61K	32K	30K	9.6K
TST-R	29.4	78.3	259	16.9	50.4	188
TAF-R	27.0	76.6	244	16.5	50.0	192
RGPE	100	375	1764	69.9	261	847
BO-ePCA	367	374	616	322	333	471
BO-mPCA	**2.68**	**3.26**	**2.71**	**2.28**	**2.25**	**2.35**

C. Results

The results of the normalized regret are shown in Table III. For each method, we compared the averaged normalized regret after 10, 20, 30, 40 and 50 iterations of BO. We underline the best value per benchmark at each number of



(a) Average rank on the quadratic function

(b) Average rank on the AdaBoost grid benchmark

(c) Average rank on the SVM grid benchmark

(d) Average rank on the NN grid benchmark

Fig. 1. Average rank

iterations and show in boldface the values which are not significantly different from the best value accordingly to a Wilcoxon signed-rank test [3] with significant level $\alpha = 0.05$. BO-PCA outperformed or performed as well as the best of the other transfer learning algorithms in almost all the benchmarks and all the number of iterations. Indeed, when the average normalized regret of BO-PCA does not have the best value, it is not statistically different from the best method. Only in the AdaBoost benchmark at 40 and 50 iterations, the proposed method was outperformed. This first metric tell us that BO-PCA was empirically the most data-efficient algorithm and the most accurate in most of the cases. These results are confirmed also by the rank shown in Figure 1, where each of the figures displays the average rank and its standard deviation for one of the four benchmarks. Also, the rank shows that BO-PCA is either the best or comparable to the best competitor, which sometimes is RGPE and sometimes plain BO, for each iteration of each task.

Finally, we also compare the computational time needed to update the transfer learning models of the 6 methods. This corresponds to the time to update the Bayesian linear regression for ABLR, and to the time to update the weight vector for the other transfer learning methods. Note that this is the discriminatory step, as all the other steps in the BO iteration are equivalent in all the algorithms (except for ABLR that is anyway performing the worst). The results are shown in Table IV.

The weights of GP-mPCA are updated much faster than any other method. This is because it is the only algorithm that can be updated recursively, as described in Section IV-B. In addition, although the computational time of TST-R, TAF-R and RGPE increases at each iteration, GP-mPCA remains constant, as it does not depend on the number of data. Note that the computational time for ABLR is much slower than the others, because of numerical issues related to matrix inversion that cause long computations for numerical adjustments.

We can conclude that BO-PCA outperforms the s.o.t.a. algorithms in the analyzed benchmarks. Comparing the two variants of this algorithm (using GP-ePCA and GP-mPCA), we obtain similar results in terms of performance, but with GP-mPCA always slightly outperforming GP-ePCA. This supports the efficacy of the proposed initialization method designed specifically for GP-mPCA and described in Section IV-C. Given these results, and the fact that GP-mPCA is much faster in terms of computational time, we can conclude that BO-PCA equipped with GP-mPCA is the most promising algorithm for transfer learning BO.

VI. Discussion and Conclusions

In this work, we proposed an efficient BO algorithm for transfer learning based on GP-PCA, called BO-PCA. It first learns low-dimensional features from a set of Gaussian processes posteriors of the source tasks using GP-PCA. Then, it updates the prior functions by optimizing the low dimensional parameters of the extracted features to fit the target task. The approach could be used with both GP-ePCA and GP-mPCA. In the latter case, the transfer learning procedure becomes a least squares problem which can be solved recursively and independent of the number of data. Therefore, our algorithm requires much lower computational cost compared to previous

methods. We show that our algorithm either works more accurately and is more data efficient or works as well as the previous methods for HPO tasks, even though it incurs a much lower computational cost.

REFERENCES

[1] S. Akaho. e-pca and m-pca: Dimension reduction of parameters by information geometry. In *IEEE international Joint Conference on Neural Networks*, volume 1, pages 129–134, 2004.

[2] S. P. Arango, H. S. Jomaa, M. Wistuba, and J. Grabocka. Hpo-b: A large-scale reproducible benchmark for black-box hpo based on openml. *arXiv preprint arXiv:2106.06257*, 2021.

[3] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

[4] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1437–1446, 2018.

[5] M. Feurer, B. Letham, and E. Bakshy. Scalable meta-learning for bayesian optimization using ranking-weighted gaussian process ensembles. In *AutoML Workshop at ICML*, volume 7, 2018.

[6] P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[7] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[8] N. Hansen, D. V. Arnold, and A. Auger. *Evolution Strategies*, pages 871–898. Springer Berlin Heidelberg, 2015.

[9] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[10] H. Ishibashi and S. Akaho. Principal component analysis for gaussian process posteriors. *Neural Computation*, 34:1189–1219, 2022.

[11] N. Jaquier, V. Borovitskiy, A. Smolensky, A. Terenin, T. Asfour, and L. Rozo. Geometry-aware bayesian optimization in robotics using riemannian matérn kernels. In *Conference on Robot Learning (CoRL)*, 2021.

[12] H. Liu, Y.-S. Ong, X. Shen, and J. Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11):4405–4423, 2020.

[13] L. Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

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

[15] V. Perrone, R. Jenatton, M. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. In *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems*, pages 12751–12761, 2018.

[16] A. Rai, R. Antonova, F. Meier, and C. Atkeson. Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *Journal of Machine Learning Research*, 20:1–24, 2019.

[17] E. Raponi, H. Wang, M. Bujny, S. Boria, and C. Doerr. High dimensional bayesian optimization assisted by principal component analysis. In *International Conference on Parallel Problem Solving from Nature*, pages 169–183. Springer, 2020.

[18] N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Scalable hyperparameter optimization with products of gaussian process experts. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 33–48. Springer, 2016.

[19] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[20] J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems*, pages 2960–2968, 2012.

[21] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[22] N. Sugihara, H. Sugihara, Y. Otake, R. Krems, H. Nakamura, and S. Fuse. Rapid and mild one-flow synthetic approach to unsymmetrical sulfamides guided by bayesian optimization. *Chemistry-Methods*, 1(11):484–490, 2021.

[23] B. Tang. Orthogonal array-based latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993.

[24] J. Vanschoren. Meta-learning. In Hutter et al. [9], pages 39–68.

[25] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[26] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Sequential model-free hyperparameter tuning. In *2015 IEEE international conference on data mining*, pages 1033–1038. IEEE, 2015.

[27] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 199–214. Springer, 2016.

[28] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

[29] Y. Zhang, D. Apley, and W. Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific Reports*, 10(1):1–13, 2020.

[30] L. Zimme, M. Lindauer, and F. Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3079 – 3090, 2021.