University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Masterarbeit D1318

Latent Space Bayesian Optimization with Transfer Learning

Latente Raum Bayes'sche Optimierung mit Transfer Learning

Author: Jagan Shanmugam

Date of work begin: 08.07.2019
Date of submission: 05.01.2020

Supervisor: Dr. Jonathan Spitz
M.Sc. Felix Wiewel
Prof. Dr.-Ing. Bin Yang

Keywords: Bayesian Optimization, Transfer Learning, Multi-Task Learning, Bayesian Linear Regression

Bayesian Optimization (BO) is a popular optimization method for expensive black-box functions, used in a variety of fields including hyperparameter optimization and industrial processes, up to moderate dimensions (10-20). The black box functions are sometimes over-parameterized which results in modeling redundant dimensions in high dimensional spaces. Optimization methods that focus on the most relevant dimensions find optimal solutions faster. Additionally, existing optimization data, e.g. from optimizing similar problems, can be used to further speed up the optimization process on the task of interest i.e. perform Transfer Learning. To warm start BO on the task at hand, it is of utmost importance to model data collected from similar tasks to transfer knowledge. In Transfer Learning BO, models which learn the underlying intrinsic function are essential. We propose a latent space model with Transfer Learning to, 1. learn a transformation from input space to latent space and 2. learn a common set of features from the learned latent space across multiple tasks to perform efficient Transfer Learning. Our model is empirically evaluated against state-of-the-art methods on synthetic benchmarks.

Contents

Acknowledgments

First and foremost, I want to thank Dr. Jonathan Spitz from the Bosch Center for Artificial Intelligence (BCAI) for the support and mentoring throughout this work. The weekly meetings helped me find my balance and enhanced this work.

I would like to thank Felix Wiewel from the Institute for Signal Processing and System Theory (ISS), University of Stuttgart for guiding me in this process. And, I'm profoundly grateful to Prof. Dr.-Ing. Bin Yang for his support during the initial phase.

Also, I would like to thank my colleagues in BCAI for the interesting discussions and valuable inputs. Specially, I would like to thank Stefan Falkner, Anna Eivazi and Lukas Froehlich for the numerous knowledge sharing sessions.

Finally, I would like to thank my friends and family for the constant motivation and support.

Thank you.

Acronyms and Summary of Notation

Acronym	**Meaning**
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
BO	Bayesian Optimization
GPs	Gaussian Processes
NN	Neural Network
BNN	Bayesian Neural Network
RL	Reinforcement Learning
AE	Auto Encoder
VAE	Variational Auto Encoder

Symbol	**Meaning**
$\mid$	conditioning operator
$\mathcal{N}$	normal distribution
$\mathbb{R}$	real valued 1-d domain
$\sim$	sampling operation
$\in$	belongs to
$\mapsto$	maps to

1 Introduction

1.1 Motivation

Nowadays even though there exists a huge volume of data, there are many areas where data is scarce and collecting data is significantly difficult. The cost of collecting data is very high and it cannot be simply ignored in the optimization process, especially in optimizing a resource-intensive task. A typical example of a data deficient case would be fine tuning the hyperparameters of a Deep Learning model to efficiently solve a complex problem instead of settling for a sub-optimal hyperparameter setting. Since Deep Learning models are resource-intensive in terms of computing power required and time, we cannot afford to retrain the models several times and tune the hyperparameters manually in reality. There are also several industrial processes where the cost associated with manufacturing or testing a piece is substantially high and we cannot repeat the process several times owing to our limited resources.

As we train more Deep Learning models and run processes on a daily basis, we optimize the parameters either by tuning them manually or by applying optimization techniques. Yet, on a new day when training a new Deep Learning model or running a new process, our optimization algorithms start to optimize from scratch as we do not feed in the data from previous optimization runs. Data collected from earlier optimization runs are often not utilized. This drives us to draw parallels to the learning process of humans on an unknown task. In the human world, we rarely start to learn about a task from scratch. Rather we relate the task at hand to all the previous tasks we have encountered and shorten the learning process on the target task. Inspired from the human way of solving multiple tasks with minimal effort, we want to design algorithms that extract and transfer knowledge for efficient optimization.

In industrial processes and Deep Learning models, there are several parameters to be tuned in an optimal manner to get the best results out of the process. Often times, the objective function we want to optimize in an industrial setting is parameterized by tens of parameters where not every parameter contributes significantly to the final value of the function. In such cases, the domain expert can assist in identifying the number of relevant parameters thereby reducing the modeling domain from high to low dimensions. The distinction between relevant or important and irrelevant or not important can be done easily by a domain expert in most cases. Even if this distinction cannot be made, only by estimating the number of relevant parameters, we can optimize the objective function faster in a data-efficient manner. This case of over parameterized functions tends to affect the entire optimization chain resulting in a sub-optimal solution. To overcome this problem, several methods have tried to find an effective lower dimensional space which allows us to optimize the function more efficiently. One such pioneering work is Random Embeddings for Bayesian Optimization, *REMBO* [1], where random embeddings are used to project the higher dimensional input space to lower dimensional space on which optimization is carried out. Figure 1.1 shows the efficiency

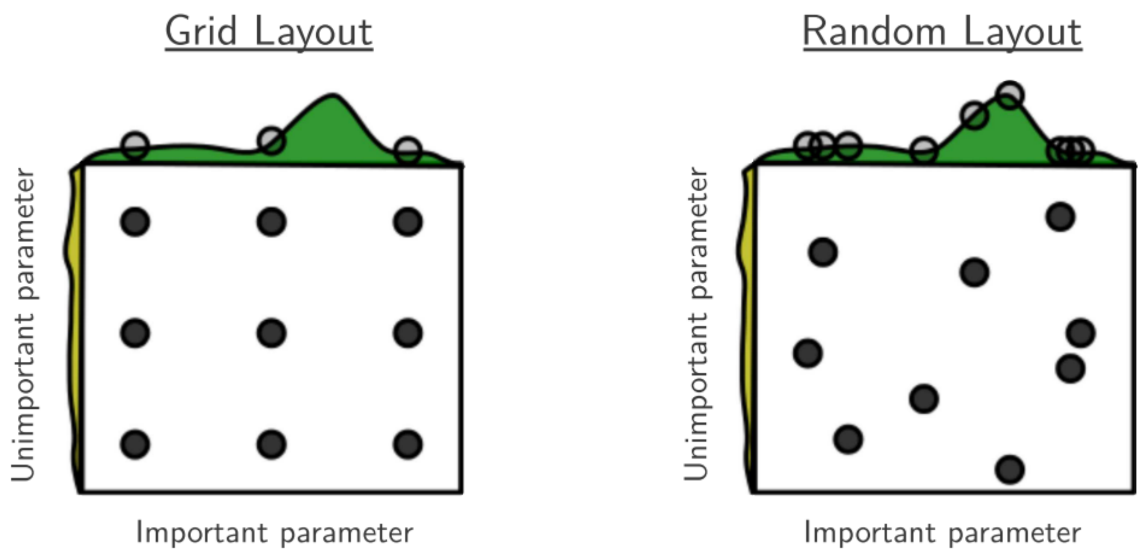


Figure 1.1: Grid search vs Random search from [4]

of random search, when only one of the parameters is important in the two dimensional problem, compared to grid search. Random sampling of parameters is implicitly able to take advantage of the intrinsic dimensionality and finds a better optimum than grid search.

Bayesian Optimization (BO) is an established framework to optimize derivative-free and expensive to evaluate objective functions and has been successfully applied in a variety of fields including Robotics, Deep Learning and Reinforcement Learning. We adopt the BO framework to optimize the objective function. Once we have an estimate of the important parameters, we can optimize in the lower dimensional space to enable faster convergence and overcome the limits of high dimensional BO.

We want to make full use of the existing data and ways for incorporating knowledge from existing data are needed. Especially data resulting from evaluating expensive functions can be utilized to speed up optimization of other related expensive functions. It acts as a warm-up procedure in solving any machine learning related tasks and in literature it is referred to as Transfer Learning. Transfer Learning in optimization has been studied before in settings like hyperparameter tuning [2] and transferring knowledge from a simulator to real experiments [3]. A method which not only scales with the number of data points observed but also transfers the learned knowledge via low dimensional latent and feature spaces is developed in this thesis to tackle the above described problem.

Formally, the problem statement is defined in the next Section 1.2.

1.2 Problem Statement

Black-box global optimization: We treat our unknown function f as a black-box and try to minimize it globally over the parameter space $\mathcal{X}$ as formulated by the Equation 1.1.

$$x^* = \arg\min_{x \in \mathcal{X}} f(x) \tag{1.1}$$

where f has following properties: 1. Non-convexity 2. No analytical form and no gradient information 3. Noisy evaluations 4. Expensive to evaluate.

BO is typically used in optimizing an expensive to evaluate function without any gradient data. BO models the unknown function and guides the search using an acquisition function. In our case, considering M related tasks, $\{f\}_{t=1}^{M}$ with data $D = [D_t]_{t=1}^{M}$ as black-box functions, we take the optimization data from these tasks into account. We refer to them as *meta-tasks* and *metadata*. We want to optimize the unknown target task f_T by transferring knowledge from meta-tasks. The main contributions of this work are as follows:

- A high dimensional BO method that jointly learns the prediction model and low dimensional surface, Section 4.1
- Latent models to learn the low dimensional space on which BO is performed, Sections 4.3.1, 4.3.2.
- Transfer Learning for high dimensional functions with intrinsic low dimensional space by transferring in two stages, Section 4.3.

The following is the structure of this thesis: Chapter 2 describes the related work and Chapter 3 explains the necessary theoretical background which is used in the consecutive chapters. Chapter 4 describes our latent space models and BO in latent space. Formulation of our experiments, results and discussion are presented in Chapter 5. Finally, Chapter 6 concludes this thesis with a future outlook.

2 Related Work

In this Section, earlier works concerning Transfer Learning in the context of Bayesian optimization and High Dimensional Bayesian optimization using low dimensional spaces are discussed. Also, works that use different Bayesian Neural Networks (BNNs) in Bayesian optimization are discussed to portray why BNNs are preferred over Gaussian Processes (GPs) especially when more data is available.

2.1 Bayesian Optimization in Feature Spaces

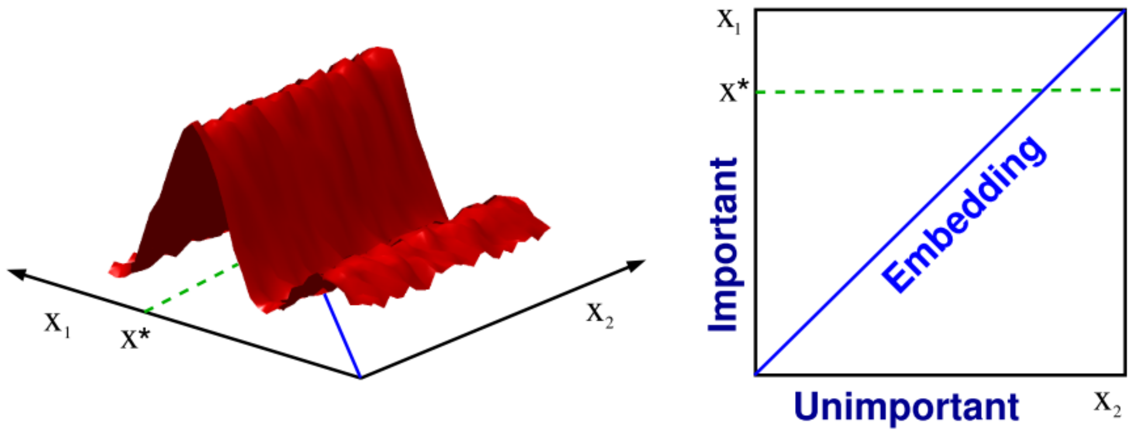


Figure 2.1: Left side shows the function which is 2-dimensional whereas the effective dimensionality is only 1 and the right side displays important dimension and an Embedding which contains the 2D function's maximum [1]

Optimizing in high dimensional space is challenging as the number of data points required to cover the entire space grows exponentially with increasing dimension. To overcome the problem of optimizing in high dimensional space in BO, [1] proposed a method, Random EMbedding Bayesian Optimization (*REMBO*), incorporating the motivation behind Random Search [4] which implicitly covers the low dimensional space by uniform random sampling of the search space. The objective function is assumed to be high dimensional with lower intrinsic dimensionality which is termed as *Effective dimensionality*, d_{eff}.

REMBO uses a random projection matrix, $A \in \mathbb{R}^{D \times d}$ with each element sampled from $\mathcal{N}(0, 1)$ to project the input data from embedding to input space (D dimensional) and $d \geq d_{eff}$. Without box bounds $\mathcal{B}$ on the input space, *REMBO* finds the optimum with the probability of 1 [1]. Practically, all BO problems have box bounds, which reduces the probability of finding the optimum in the chosen random embedding. Furthermore, heuristics such as box-bounds

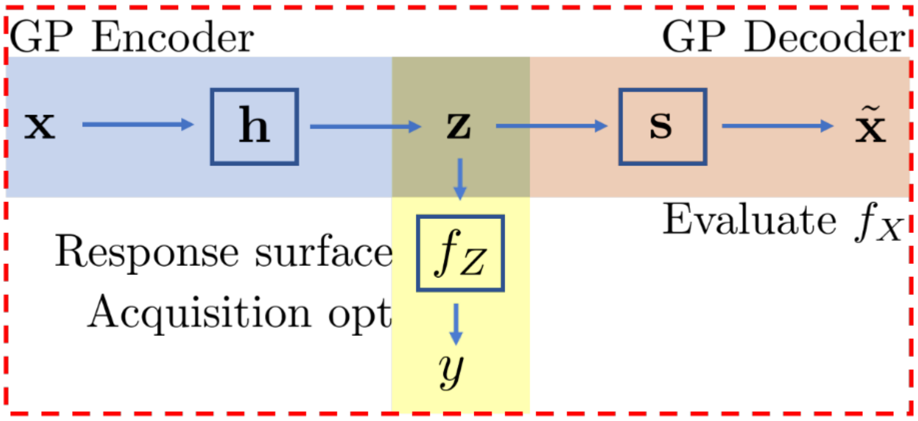


Figure 2.2: General structure used for joint learning of feature space and GP model [6]. Blue shaded box represents the encoder mapping from input x to feature space z, Yellow shaded box indicates the response surface model f_Z on feature space and brown box describes the reconstruction mapping to input space.

for embedding try to prevent the projection of points outside the bounded input space $\mathcal{B}$. The points which are projected outside the bounds of input space $\mathbb{R}^D$ are clipped to boundaries resulting in a nonlinear transformation.

The authors of *REMBO* also proposed a variant referred to as *InterleavedRembo*, which uses multiple random projections and runs BO in an interleaved manner thereby increasing the probability of finding an embedding with optimum. *REMBO* is sample efficient as the only model that needs to be computed with the observed data points is the low dimensional $\mathcal{R}^d$ Gaussian Process (GP) model. Empirically, *InterleavedRembo* performs better than *REMBO* in most cases. However, even in *InterleavedRembo* with multiple random projections and multiple GP models, there is no information shared across all GP models. Figure 2.1 gives an example of a 2-dimensional function with $d_{eff} = 1$ and a random 1-dimensional embedding which comprises of the maximum x^*. It is well suited for problems with very high dimensional input space and very low dimensional known intrinsic space.

Another recent approach is *VAE-BO* proposed in [5] which learns a continuous latent space representation from the existing historical discrete data. Using the learned latent space, the idea is to navigate easily in the continuous latent space for efficient optimization. They train a Variational Auto Encoder and a prediction model jointly on thousands of existing chemical structures to enforce a structure in the latent space. Probabilistic modeling of the latent space by the decoder in *VAE* enables them to generate new structures by sampling randomly from the latent space. This kind of enforcing particular structure in the latent space requires thousands of data points and will not perform efficiently in the traditional BO setting where only a few evaluations are affordable.

In the latest work [6], Manifold Gaussian Processes along with a simple Encoder-Decoder like structure are used to jointly learn the nonlinear feature space and response surface. Gaussian Processes (GPs) on the feature spaces which learn the response surface or the surrogate model are used in the Bayesian optimization framework. Multi-Output Gaussian Processes are used as a decoder to reconstruct the input data from the feature space representation. Manifold GPs are comprised of a GP regression and a nonlinear transformation to the input (a multi-layer perceptron) learned jointly. The model's encoder, decoder and GPs are trained by minimizing the joint log-marginal likelihood. Since it is a GP-based approach, it is not scalable with respect to number of data points as discussed in the Section 3.2. Figure 2.2 shows the general structure of the Encoder-Decoder and manifold GP model, where h is the transformation of input space x to feature space z and s is the reconstruction mapping used in the evaluation of the objective function f_X during BO. f_Z is the low dimensional response surface or model used to compute acquisition function in BO.

All the above methods except [5] are based on the assumption that the function of interest exists in lower dimensional space than the higher input dimensional space and try to adopt the Bayesian optimization framework in the lower dimensional feature space. They also argue for finding better optima by exploring the lower dimensional space and enable the use of Entropy driven acquisition functions such as Entropy Search [7] which perform better in low dimensions.

High dimensional BO methods work on high dimensional input space and make the optimization of high dimensional acquisition functions hard, as discussed in the Section 3.1. *LineBO* [8] addresses the problem of optimizing an acquisition function in high dimensional space and proposes strategy to select and search over one-dimensional subspaces. In [9], an adaptive dropout strategy is developed to work with a subset of variables in input space to solve high dimensional BO.

2.2 Transfer Learning in Bayesian Optimization

Meta-learning or Learning to learn is broadly defined in the Meta-learning chapter [10] of the AutoML book [11] as observing how various machine learning methods perform on learning tasks and learning from this experience or metadata. Transfer learning entails using the model trained on metadata as initialization for modeling the target task. Warm starting the optimization process using metadata also falls under this category. Neural Networks are suitable models for transfer learning as it is easy to transfer the learned information from the meta-data, as discussed in 3.5. Transfer occurs when the weights of all or a few layers of the network are copied from the pre-trained model.

Transferring knowledge in BO has been studied in recent times and has been proven to work well for moderate dimensions. Transfer learning methods in BO can be widely classified into two types based on the models used, namely Gaussian Processes and Bayesian Neural Networks.

One of the pioneering works in modeling multiple tasks using Multi-Task Gaussian Processes is [12]. Their framework uses Multi-Task Gaussian Processes (MTGPs) where each task and its relationship with other tasks is defined by the multiplication of two kernels: one in the traditional way to measure the similarity between data points and another one to capture the similarities between tasks. Also, they extend the acquisition function Entropy Search [7] to the multi-task setting. However, MTGPs do not perform well in problems where the input dimension is greater than 10. Another approach proposed a scalable GP that considers source tasks to arrive in a given order and trains a stack of GP regressors [13]. Every model in the stack is trained with the residuals from the current task's data and predictions from the previous GP model. These works avoid using a single GP and are comparatively scalable.

Closely related to [12], [14] considers multiple information sources that are easier to evaluate than the real experiment or the target task. They define a GP kernel specialized for multi-information sources, specifically the sum of two kernels: one for the target and one for each information source. They also provide a parallelized version of their algorithm and modify their acquisition function to select the most efficient information source at each iteration. Following a similar approach, [3] deals with transferring knowledge from a simulator to real experiments in robotics. Data from the simulator has lower fidelity than data from the real experiment but it is comparatively cheaper. A common GP kernel for two sources, models the cheaply available biased simulator and error between real experiment and simulator. The latter is only active for the evaluations on the real experiment. To select the information source, a cost per information source is defined and the information gain per unit cost is utilized.

[15] performs Multi-Task Bayesian optimization to search for better policies online in a reinforcement learning setting by leveraging offline experiments. From the learning curves of using Multi-Task GPs (instead of single task GPs) for offline simulator and online case, they empirically show the advantage of using Multi-Task modeling.

In cases where the fidelity of the information source can be chosen at various or continuous increments, Multi-Fidelity optimization can be performed. In [16], Multi-Fidelity GPs are combined with the information-theoretic acquisition function Max-value Entropy Search [17] to perform Multi-Fidelity BO. All fidelities are modeled in a single GP, called Multi-Fidelity GP with the difference between discrete fidelities captured in an error kernel which is added to the original GP kernel. Acquisition functions are also extended to the multi-fidelity setting. Along those lines, [18] extends the GP-UCB acquisition function [19] to the multi-fidelity setting. If there are meta-features available during BO for multi-task cases, [20] proposes a ranking-weighted GP ensemble model to utilize the optimization data from past runs and is scalable than a single GP model.

Recent work [2] which uses one Bayesian Linear Regression (BLR) model on top of shared Neural Network basis functions learned from historical metadata is a scalable alternative to GP-based methods. [2] has one Bayesian Linear Regressor for each task and the parameters of the BLR model are integrated out resulting in tractable mean and variance estimates. The theoretical background on Adaptive BLR is provided in the Section 3.3. This work uses the Multi-task ABLR model as described in 4.2 and can be considered as a direct extension of it to higher dimensional problems.

2.3 Bayesian Neural Networks in Bayesian Optimization

In BO, the typical model used is Gaussian Processes owing to its simplicity in definition and maintenance. However, Bayesian Neural Networks (BNNs) is emerging as a competent substitute which scales better when more data is available. Compared to GPs, a domain-specific kernel need not be defined in BNNs.

[21] proposed a deep neural networks based alternative to GPs referred to as *DNGO* (Deep Networks for Global Optimization). They train in two stages: First, a neural network with a linear output layer is fit on the training data. Second, the last layer of the trained network is replaced by a Bayesian linear regressor that captures the uncertainty. The computational

complexity depends on the dimensionality of the feature space, which is the number of units in the last hidden layer rather than the number of observed data points in GPs. They also perform GP-based hyperparameter optimization on a series of benchmarks end up with a 3-layered neural network with $TanH$ activation in the hidden layers.

A full Bayesian treatment of the network parameters is carried out in [22]. Since the posterior of the network parameters is intractable, a Hamiltonian Monte Carlo sampling procedure is used to integrate out the network weights. Well-calibrated uncertainty estimates are obtained due to the full Bayesian treatment. For the multi-task case, a task specific embedding is learned and is considered more expressive than [2].

Multi-Task Adaptive Bayesian Linear Regression (MT-ABLR) [2] which was briefly discussed in the previous subsection 2.2 is an extension of *DNGO* [21] to multi task cases with weights of BLR layer integrated out rather than being sampled. They also follow the 3 layered architecture introduced by *DNGO* [21] and perform Transfer Learning in BO. The Multi-Task ABLR which performs Transfer Learning, with no sampling in the BLR layer is faster than other sampling-based counterparts and is used as the primary model in this thesis 4.2.

Another related work is [23] which learns the linear embeddings by active learning for Gaussian Processes. The active learning method learns about the function and low dimensional embedding associated with the input space which will be later useful for BO. In this work, we propose a method to learn the transformation onto embeddings in the BO loop while simultaneously optimizing the function and modeling the low dimensional latent space.

3 Theoretical Background

This chapter provides a brief theoretical knowledge about algorithms and models used in the later chapters. Firstly, the Section 3.1 discussess Bayesian optimization and its pitfalls. Sections 3.2 and 3.3 introduce Gaussian Processes and Bayesian Neural Networks as models used in BO. In the Section 3.4, dimensionality reduction techniques used in this work are discussed. Lastly, Transfer Learning in general and in BO is discussed in the Section 3.5.

3.1 Bayesian Optimization

Bayesian Optimization [24] is a framework to optimize an unknown, expensive black-box function by using a probabilistic surrogate model and an acquisition function as a heuristic to guide the search. Being a sequential framework, BO updates its probabilistic model with the observed data points and maximizes the acquisition function to get the next point to sample in the input space at every iteration. Typically, BO is repeated for a certain number of iterations or until a specific convergence criterion is met.

Algorithm 1 lists out the steps involved in BO where it is configured to run for a certain number of iterations. Implicitly, optimization of the unknown black-box objective function is shifted to the optimization of the known acquisition function which has an analytic form. The point x_{next} from maximizing the acquisition function on line 2 is evaluated on the objective function on line 3. The model of the objective function is updated using the augmented dataset on lines 4 and 5.

BO has been successfully applied for lower dimensional problems comprising of 20 dimensions or less. In high dimensional problems, BO struggles to find the optimal solution because of primarily two fundamental challenges.

1. Optimizing acquisition function in high dimensional space is difficult as most of the regions are flat which essentially leads to slow or no convergence.
2. As the number of data points needed to reasonably model the function in the high dimensional input space grows exponentially with the dimensions, BO loses its data efficiency.

BO relies on the posterior mean and estimate of the function with the observed data points. Also, a sufficient optimum should be found during the maximization of the possibly non-convex acquisition function to drive BO. The surrogate model typically used in Bayesian optimization is Gaussian Process which will be briefly discussed in the Section 3.2. Random Forests which is a popular Machine Learning method are also adapted to BO with uncertainty estimates [25]. Bayesian Neural Networks with different architectures are used in recent times in BO and is discussed in the Section 3.3. In the context of minimization, at every iteration, we want to sample a point with low value but also explore regions of high uncertainty to look for a better minimum. Acquisition function or Utility function $u(x)$ trades off this exploitation-exploration phenomenon.

Expected Improvement: Out of the acquisition functions, Probability of Improvement (PI) [26], Gaussian Process - Upper Confidence Bound (GP-UCB) [19], Entropy Search (ES) [7] and variants of Entropy Search, Expected Improvement (EI) [24] is simple, robust and a standard acquisition function and will be used in this thesis for all BO experiments. EI quantifies the improvement over the already observed best value f^* for the new point x and is defined by the Equation 3.1.

$$EI(x) = \mathbb{E}[\max(f(x) - f(x^+), 0)] \tag{3.1}$$

where $f(x^+)$ is the best observed function value so far. This can be analytically calculated with mean and standard deviation of the surrogate model $\mu(x)$ and $\sigma(x)$ [27], [28].

$$EI(x) = \begin{cases} (\mu(x) - f(x^+))\Phi(Z) + \sigma(x)\phi(Z), & \text{if } \sigma(x) > 0 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{3.2}$$

where $Z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$ and $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and PDF of the standard normal distribution.

Algorithm 1: Bayesian optimization

```
Input: Objective function ($f$)
Output: Optimum ($x^*$) of objective function and $f(x^*)$
Data: Initial Dataset $D = \{x, y\}$
1 for $t=0,1,2...$ do
2     Maximize acquisition function: $x_{next} = \arg\max_x u(x|D)$
3     Evaluate objective function: $y_{next} = f(x_{next})$
4     Augment the Dataset: $D = \{x, y\} \cup \{x_{next}, y_{next}\}$
5     Update the model: $\hat{f}$ using $D$
```

Figure 3.1 shows two steps of BO on a one dimensional quadratic function with the GP model and EI acquisition function. Mean and two standard deviations of the GP model are plotted in blue along with observed data points in red. The maximum point of the acquisition function at iteration 3 is evaluated on the objective function at next iteration and the GP model is updated. With only 4 points, the GP model and EI acquisition function converge at the area around the minimum as seen in bottom part of the Figure 3.1. During convergence, acquisition function peaks around the minimum which can be seen by observing the difference between acquisition plots in green on steps 3 and 4. Please refer to [28] for a nicely written tutorial on BO and its applications.

3.2 Gaussian Processes

Gaussian Process (GP), fully specified by a mean function $\mu(\cdot)$ and covariance function $k(\cdot, \cdot)$ is a distribution over functions with any finite number of output function values distributed

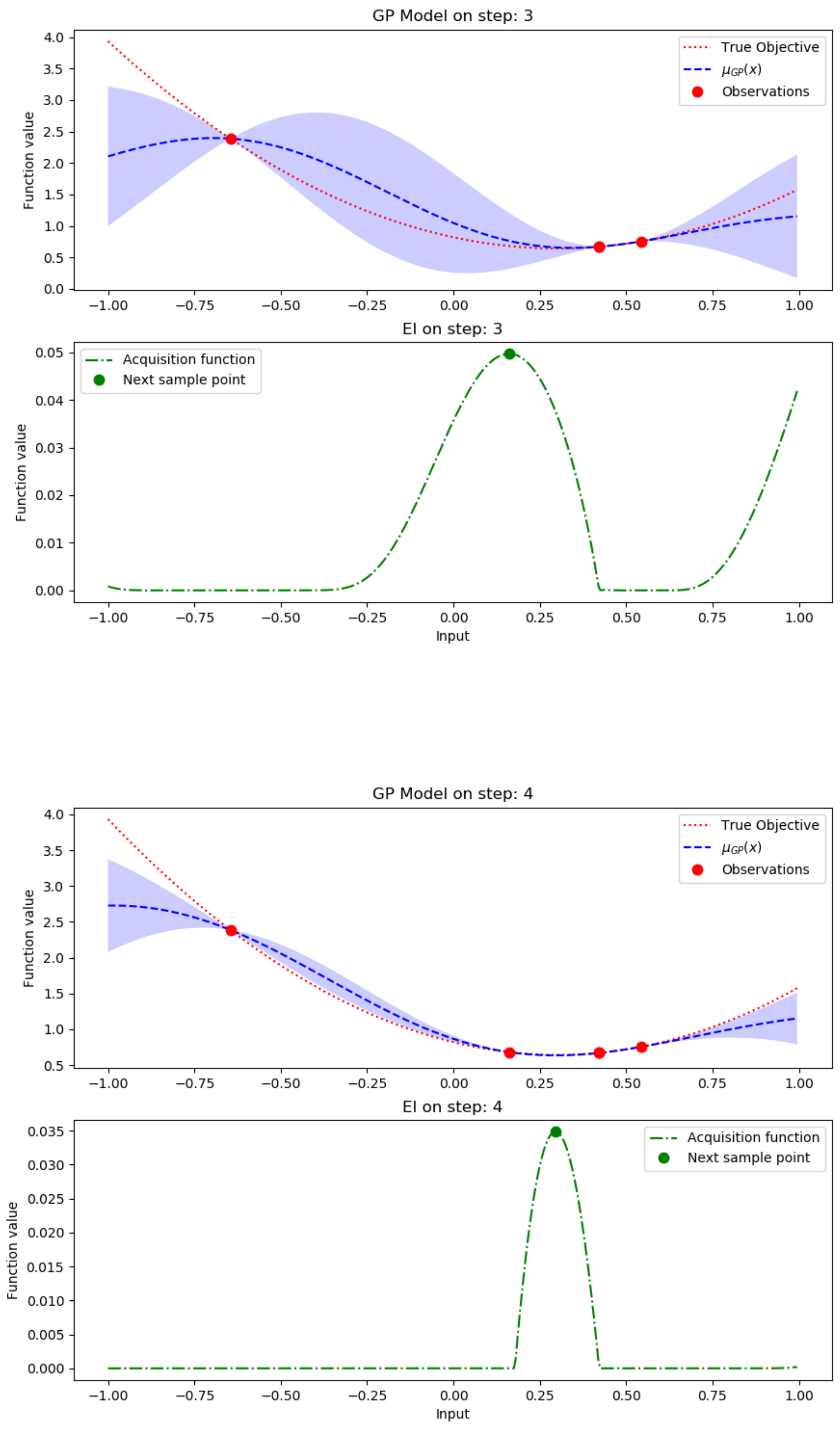


Figure 3.1: BO on steps 3 and 4

in a Gaussian form. It is a natural extension of multivariate Gaussian distribution to infinite dimensions and the function we are modeling is defined as in the Equation 3.3.

$$f(x) \sim GP(\mu(x), k(x, x')) \tag{3.3}$$

where $\mu(x)$ and $k(x, x')$ are the mean function at location x and kernel or covariance function at location x and x'. Prior mean function is commonly set as $\mu(x) = 0$ and the frequently used kernel function, Squared Exponential (SE) or Radial Basis Function (RBF) function is given by,

$$k_{SE}(x_i, x_j) = \sigma^2 exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right) \tag{3.4}$$

where the lengthscale l determines how much it extrapolates away from data and the output variance σ^2 is the scaling factor. It acts as a similarity measure between two input data points and in most cases, the choice of the kernel function is specific to the problem domain.

In the regression case, with the training data $\mathcal{D} = \{(x^i, f^i)\}_{i=1}^N$, GP posterior is computed from prior which is then used to make predictions on new data points. Covariance matrix K with entries $K_{ij} = k_{SE}(x_i, x_j)$ is defined for points in the training data. The joint distribution of observed data $f = (f^i)$ and the predictions is given by the Equation 3.5.

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} k(X, X) + \sigma^2\mathbb{I} & k(X, X_*) \\ k^T(X, X_*) & k(X_*, X_*) \end{pmatrix}\right) \tag{3.5}$$

where X and X_* are training and test points and σ^2 approximates the noise in the observed points. Conditioned on the training data, predictive mean and variance is given by the below Equations 3.6, 3.7.

$$\mu(x) = k^T(X, X_*)(k(X, X) + \sigma^2\mathcal{I})^{-1}y \tag{3.6}$$

$$\Sigma = k(X_*, X_*) - k^T(X, X_*)(k(X, X) + \sigma^2\mathcal{I})^{-1}k(X, X_*) \tag{3.7}$$

Being a non-parametric method, GPs takes all the observed data points during inference. Hence, the GPs scale cubically with the respect to the number of observed data points (N) as a matrix of size $N \times N$ is inverted during prediction. Also, the choice of the kernel which determines the smoothness and shape of the function is a hyperparameter in GPs. We use GPs for modeling the low dimensional space in *REMBO* in our experiments 5.4. Please refer to [29] for complete derivation and details.

3.3 Bayesian Neural Networks

Bayesian Neural Networks (BNNs) are the product of applying Bayesian inference to Neural Networks (NNs) to estimate the posterior distribution of the parameters of NNs based on the assumed prior and observed data. The theoretical relationship between BNNs and GPs have been established two decades ago in [30]. Establishing distribution over the network parameters to yield uncertainty estimates of the output has been considered in the early works [30], [31]. Computing posterior distribution over a larger network is intractable and is approximated using Markov Chain Monte Carlo (MCMC) or approximate inference or variational inference methods. BNNs with well-calibrated uncertainty estimates can be considered as an alternative option to GPs. Recently, these type of Bayesian methods has been applied to small pieces of a bigger network to compute in the context of BO as discussed in 2.3. By making the last layer as Bayesian, BNNs which scale linearly with observations and cubically with the feature map size are used in BO. The last layer can be considered as a Bayesian Linear Regressor on top of the nonlinear basis functions in the form a Neural Network. This form with any nonlinear basis function, in general, is referred to as the Adaptive Bayesian Linear Regression and is briefed in the next Section 3.3.1.

3.3.1 Adaptive Bayesian Linear Regression

Bayesian treatment of linear regression yields us the mean and the variance estimate which are crucial in BO. Comparing with the frequentist perspective of linear regression, Bayesian Linear Regression puts a distribution over the target variable in a supervised learning setting. Linear regression is linear in weights and can have a nonlinear basis represented inputs as features. For simplicity, we assume that the target variable to predict is Gaussian distributed and nonlinear basis representation $\phi(\cdot)$ of the inputs.

$$y \sim \mathcal{N}(w^T\phi(x), \beta^2) \tag{3.8}$$

where β^2 is the noise variance and w is the weight vector of the model with an assumption of zero mean prior with isotropic distribution as,

$$p(w) = \mathcal{N}(w|\mathbf{0}, \alpha^2\mathcal{I}) \tag{3.9}$$

Given the dataset $\mathcal{D} = (x^i, y^i)_{i=1}^N$ of N data points, the posterior distribution of the weights is given by,

$$p(w|\mathcal{D}) = \mathcal{N}(w|m_N, S_N) \tag{3.10}$$

where the design matrix Φ with each row $\Phi = [\phi(x_n)]$ and $m_N = \beta(\alpha\mathcal{I} + \beta(\Phi^T\Phi)^{-1}\Phi^T y)$ and $S_N = \alpha\mathcal{I} + \beta\Phi^T\Phi$.

Since we are interested in prediction for the new point x^*, the predictive distribution given the posterior with the weights marginalized is also Gaussian with the mean and variance given by,

$$\mu(x^*) = m_N^T\phi(x^*) \tag{3.11a}$$

$$\sigma^2(x^*) = \frac{1}{\beta} + \phi(x^*)^T S_N \phi(x^*) \tag{3.11b}$$

With M as the number of features or number of units in the last layer of the NN (nonlinear basis representation) $\phi(\cdot)$, we are inverting the matrix of size $M \times M$ to compute the predictive mean and variance. Typically, later in our method 4.3, we consider the case of $N >> M$, observing more data points than the number of features. This section is based on [32] and please refer to it for the complete derivation.

3.4 Dimensionality Reduction

Reducing the dimensionality of the input space to effectively model and visualize is an age old problem in Machine Learning (ML). Typically, we want to convert the D-dimensional data to d-dimensional space with $d << D$ in an unsupervised manner. In this work, we combine the unsupervised and supervised settings to jointly learn an effective low dimensional representation for both modeling and reconstruction.

Random Projections

One of the popular algorithms, Principal Component Analysis (PCA) finds a low dimensional representation of the high dimensional dataset with the maximum variance. It finds the linear subspace by computing the covariance matrix of the data and selecting the d principal components which forms the projection matrix. PCA requires the entire dataset to compute the projection matrix and that is not the case in optimization. Instead, random projections onto d dimensional subspace offer advantages in terms of time to compute with the stream of data [33]. Direction of projection is dependent on data in PCA and independent when using a random projection matrix. In Linear Algebra, matrices represent linear transformation of the input to the output space [34].

$$z = Ax \tag{3.12}$$

where A is the transformation matrix of size $d \times D$ which takes in a column vector in $\mathbb{R}^d$ and transforms it to space $\mathbb{R}^D$. In the case of having over-parameterized functions as described in the Section 1.1, we transform a higher dimensional input to lower dimensional space where optimization can be performed effectively. In our work as will be discussed in the Section 4.3, we use projection matrix initialized to random values and try to learn the projection matrix simultaneously modeling the function in low dimensional space to perform optimization.

Auto Encoders

Another way to learn general nonlinear transformations is by using stacked multi-layer perceptrons [35]. These are referred to as Auto Encoders, mostly trained in an unsupervised fashion, specifically undercomplete Auto Encoder with three parts,

1. Encoder: Maps the high dimensional input to a low dimensional subspace
2. Bottleneck: Last layer in the encoder which represents the hidden or latent space which should be learned
3. Decoder: Maps the low dimensional latent space variable to high dimensional input space

Figure 3.2 shows the basic schematic structure of Auto Encoder. Encoder and Decoder structure can have multiple hidden layers and the code z is the last layer of the Encoder structure referred to as Bottleneck. Usually, to reconstruct the input x better with the given configuration of the network mean squared error is used as the loss function and the weights of the network are updated by gradient descent. Reconstruction error in this context is mean squared error (MSE) which takes $L2$ norm of the difference between the input x and the reconstructed input $\hat{x}$ at the end of the decoder, $||x - \hat{x}||_2^2$. AEs with linear activation function

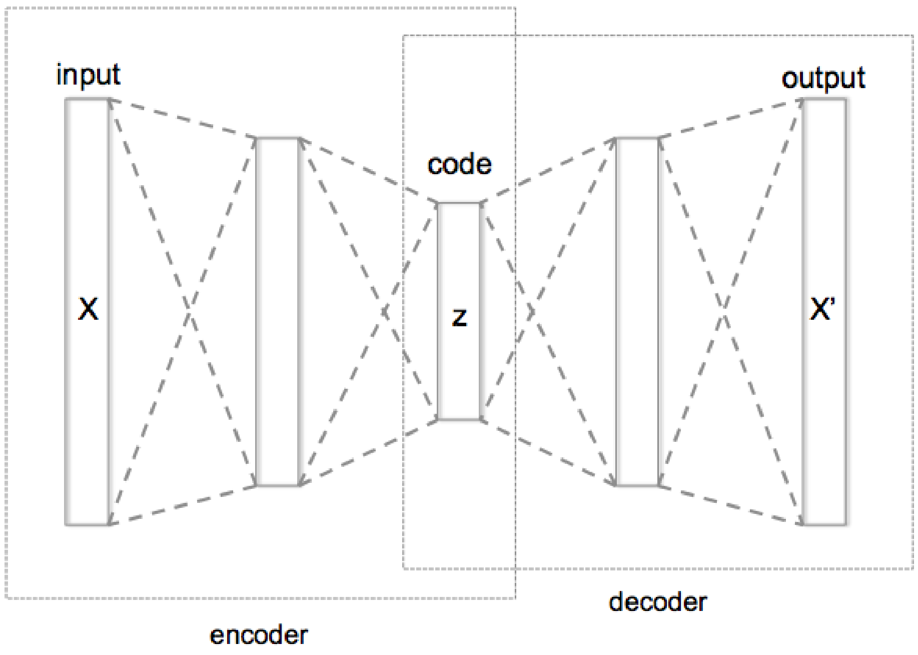


Figure 3.2: Basic schematic structure of undercomplete Auto Encoder [36]

in hidden layers find the same linear subspace as PCA [35]. Whereas, AEs with nonlinear activation functions find nonlinear mappings high to low dimensional space and vice-versa and achieve better compression. With the advent of Deep Learning, Auto Encoders are also used for learning efficient representations of the input dataset, denoising the images [35].

One variant of Auto Encoders which was proposed to enforce the latent space to a particular distribution is Variational Auto Encoder (VAE) where they take a probabilistic perspective that fits the Auto Encoders. In comparison to the standard AEs, VAEs introduce additional loss term to constrain the latent space to represent a distribution. Please refer to [37] for a review on VAEs.

3.5 Transfer Learning

Motivated from psychological factors of how humans learn new tasks, a subpart of Machine Learning (ML) called Transfer Learning is developed [39]. According to [39], traditional supervised learning breaks down when we do not have enough data on the task to be solved. Transfer Learning shows the true potential of ML methods to generalize to unseen scenarios and is regarded as an important driver in the commercial success of ML next to supervised learning [39]. Since labeling data is a time consuming and tedious process, it is also originally motivated to label a few samples in supervised learning setting by doing efficient Transfer Learning.

Figure 3.3 shows the learning process of traditional ML and Transfer Learning methods. In

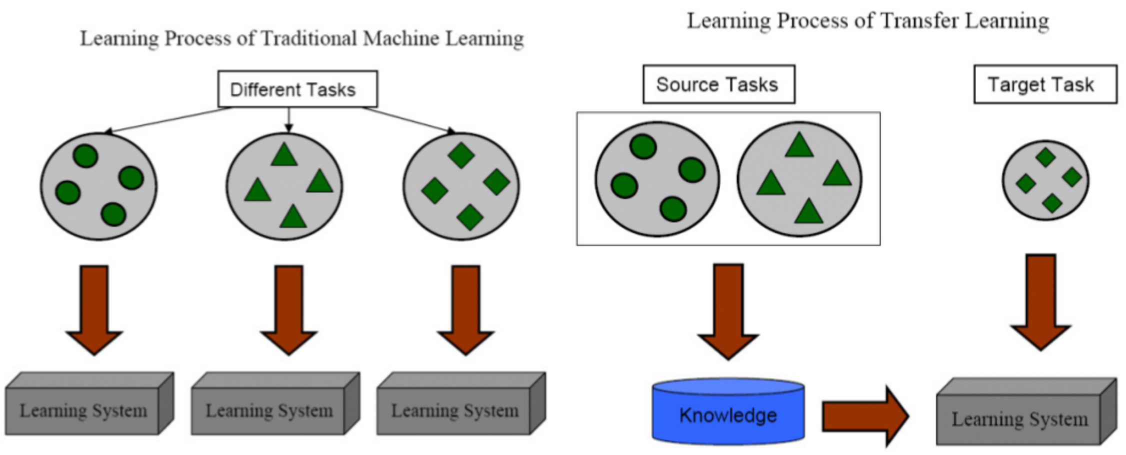


Figure 3.3: Traditional Machine Learning vs Transfer Learning [38]

traditional ML, common information across different tasks is not utilized by learning separate systems. In Transfer Learning, the target task is formulated according to the requirements and the information from other tasks are incorporated while learning a system for the target task. We want to transfer as much knowledge as we can to learn the unseen tasks and share knowledge in various forms. Currently, most of the research [40] is on 1. What to transfer 2. How to transfer and the question, 3. When to transfer. Point 3 is still an open question. For point 2, usually models trained on source tasks are used as an initialization for target task. These pre-trained models which transfer the parameters are common in Deep Learning where the last layer is cut out and remaining part of the network is used as a feature extractor on the target task. If the source and target tasks are not related, brute force transfer learning methods will result in negative transfer.

Formally, given a data set $\mathcal{D}_S$ from source domain defined as the source task $\mathcal{T}_S$ and a target data set $\mathcal{D}_\mathcal{T}$ from target task $\mathcal{T}_\mathcal{T}$, we want to learn the target function f_T conditioned of the data from source and target tasks. Typically, the target task would have exponentially less data than the source task. Application of Transfer Learning has many sub categories including learning from simulations, domain adaptation, meta-learning, multi-task learning, life-long learning etc [38].

Transfer Learning in Bayesian Optimization

To increase the sample efficiency of BO methods, Transfer Learning has been adopted into BO framework recently. BO usually has a flat prior and initially will not be able to suggest good points in the parameter space until the surrogate model is learned to a reasonable extent. This issue is referred to as cold start problem in the literature. To alleviate the cold start problem in BO, Transfer Learning is embraced in BO. As discussed in 2.2, Transfer Learning was achieved by training models on metadata. Bayesian Neural Networks are preferred for its scalability and ease of use in Transfer Learning scenarios. Methods such as Multi-Task Bayesian optimization [12], Multi-Task Adaptive Bayesian Linear Regression [2] which learn common features across tasks are the result of marrying Transfer Learning and BO.

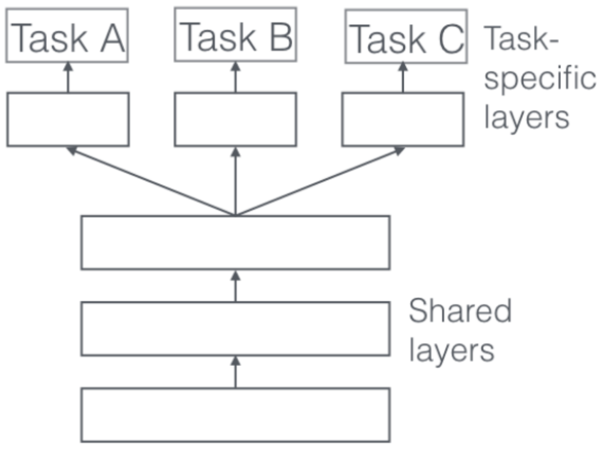


Figure 3.4: Hard parameter sharing in Multi Task Learning [40]

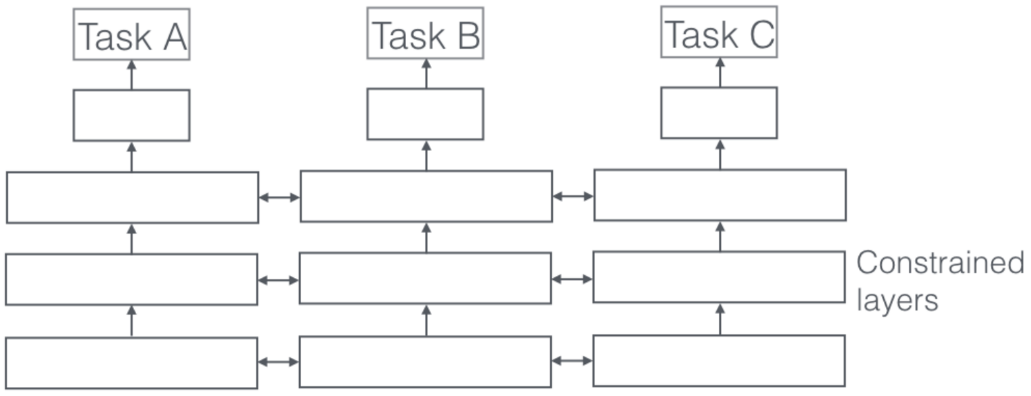


Figure 3.5: Soft parameter sharing in Multi Task Learning [40]

Neural Networks (NNs) in general are favored in Transfer Learning cases since it is modular in nature and parts of the network or layers can be segregated after training on source tasks. In general, based on shared parameters of the network, there are two types of multi task learning networks, hard and soft parameter sharing. Figures 3.4, 3.5 shows a structure of network with hard and soft parameter sharing in multi-task learning. In hard parameter sharing, there is task specific layer on top of the common set of layers for all tasks. This greatly reduces the risk of overfitting to one particular task [40]. In soft parameter sharing, each task is learned using separate network but the parameters are constrained to be similar across tasks. In our work, we use the hard parameter sharing network to learn a shared set of features and a task specific Bayesian Linear Regression layer.

4 Method

This chapter elaborates on the latent space Bayesian optimization algorithm by describing the idea of joint learning in Section 4.1. In Section 4.2, the architecture and training of the Multi-Task Adaptive Bayesian Linear Regression model are explained. Later, in Section 4.3, the setup and details of Projection and Auto Encoder latent models are discussed. Finally, Section 4.4 puts together the latent model in the context of Bayesian optimization.

4.1 Joint Learning of Latent Space and Prediction Model

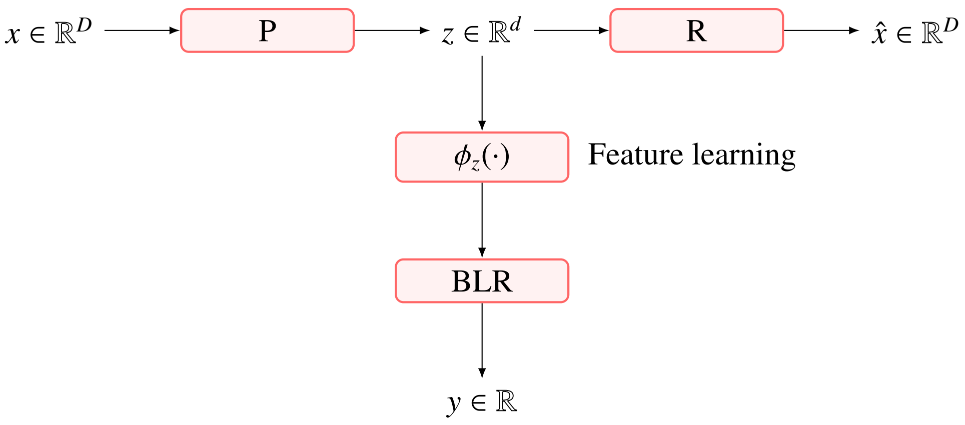
With the idea to reduce the dimensionality of the optimization space, we learn the low dimensional latent space $\mathbb{R}^d$ on which the optimization of the objective function is performed rather than optimizing in the original input space $\mathbb{R}^D$, where $d << D$. This low dimensional latent space is learned by training two orthogonal tasks, namely self-supervised and supervised learning at the same time. The function values y are modelled in the low dimensional latent space $f : \mathbb{R}^d \rightarrow \mathbb{R}$ while learning the mappings $P : \mathbb{R}^D \rightarrow \mathbb{R}^d$ and $R : \mathbb{R}^d \rightarrow \mathbb{R}^D$ from the input space to the latent space and vice versa, simultaneously.

Earlier work uses this type of joint learning, i.e., an Auto Encoder and Model framework to learn low dimensional feature spaces and uses Manifold Gaussian Processes for predictive modeling and Multi-Output Gaussian Processes for reconstruction onto input space [6]. In contrary to joint learning, sequential learning is generally prominent in many areas of ML where the task is to reduce the number of dimensions of our problem so that we can effectively model and predict the target variables. In our case, where data is a scarce commodity, finding a low dimensional latent space which is suitable for optimization can be achieved by joint learning instead of traditional sequential learning.

In this thesis, as we are taking metadata and Transfer Learning into account, it is imperative to use a model that scales with respect to both the number of points and number of meta tasks. Since Gaussian Processes scale cubically with respect to the number of the observed data points, Adaptive Bayesian Linear Regression (ABLR), which scales linearly [2], is used. In the original work [2], they demonstrate learning a set of basis functions from the input space and use it for Bayesian Linear Regression. Here we learn a set of basis functions from the low dimensional latent space and use that for Bayesian Linear Regression.

This type of joint learning of two orthogonal tasks needs an appropriate loss function in the same order to learn the parameters of the models together. Specifically, the loss function for the Bayesian Linear Regression model is negative log-likelihood (NLL) and the typical loss function for reconstructing the input points is Mean Square Error (MSE), $||x - \hat{x}||_2^2$ which can be interpreted as the negative log-likelihood of a Gaussian error between the actual and the reconstructed input. We sum up the two individual losses with equal unit weights and use it to learn the weights of both flows, $\text{Total loss} = \text{NLL} + \text{MSE}$.

Figure 4.1: General structure of joint learning for a single task: Here mappings P and R represent either the Encoder-Decoder structure or Projection and Reconstruction matrices.

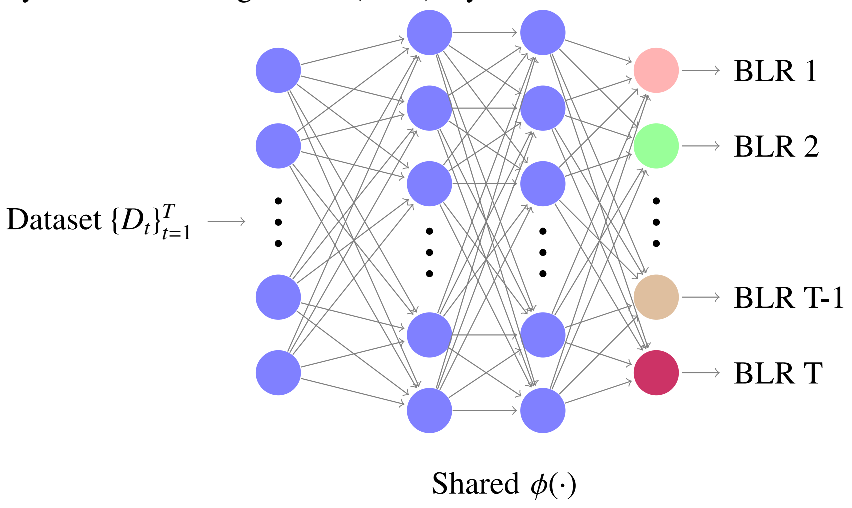


As shown in the Figure 4.1, the mapping, R is trained by the MSE term in the total loss. Similarly, the mapping P and the parameters of the response surface or probabilistic model, i.e. the combination of the feature mapping ϕ_z and Bayesian Linear Regression (BLR) model, are trained by the negative log-likelihood term in the total loss. Hence, this setup is beneficial in typical BO for problems with high dimensional input spaces with an intrinsic lower dimensional space, as the model and acquisition function reside in lower dimensional $\mathbb{R}^d$ space. When we need to evaluate the objective function, we take the optimal point from the acquisition function, reconstruct that point in the input space and then evaluate the objective function on the reconstructed point $\hat{x}$. Intuitively, this allows for two stages of transferring knowledge, one in learning shared mappings and the other being shared basis representation learned as in ABLR [2].

4.2 Multi-Task Adaptive Bayesian Linear Regression

Multi-Task ABLR [2] forms the base of this Section for extending Bayesian Linear Regression described in the earlier Section 3.3.1 to the Multi-Task setup. In the Multi-Task setup, we have T tasks with corresponding black-box functions $\{f_t\}_{t=1}^{T}$. Data is collected from evaluating each black-box function N_t times in the dataset $D_t = \{(x_t^n, y_t^n)\}_{n=1}^{N_t}$ where x_t^n lies on D-dimensional space $\mathbb{R}^D$ and y_t^n on $\mathbb{R}$. In the first stage of the multi-task prediction model for all tasks, a shared feature map $\Phi : \mathbb{R}^D \rightarrow \mathbb{R}^P$ which learns a common set of features across all tasks is implemented. In the original paper, two feature mappings for ϕ_z, RKS (Random Kitchen Sinks) and a feed forward neural network (NN) are tested and based on the results observed, the NN mapping is preferred. Figure 4.2 shows the general picture of Multi-Task ABLR model with the shared feature map $\phi(\cdot)$. Following their results, a Neural Network mapping is implemented to learn a shared set of features from the dataset $\{D_t\}_{t=1}^{T}$. As used in the earlier works [21], [2], three layered NNs with affine layers and 50 units each and *TanH* activation function in the hidden layers is used. Since the number of units in the last layer of the shared Neural Network is 50, P here is 50. For a task t, features for N_t points are collected

Figure 4.2: Pictorial summary of Multi-Task Adaptive Bayesian Linear Regression: Datasets from all tasks {1..T} are fed into the input layer and for each task, a separate Bayesian Linear Regression (BLR) layer is added.



in a matrix $\Phi_t = [\phi_z(x_t^n)]_n \in \mathbb{R}^{N_t \times D}$ which is used later during inference.

In the second stage of the multi-task prediction model for all tasks, a separate Bayesian Linear Regressor (BLR), with weights w_t on top of the shared features for each task, models the output y_t. BLR can be considered as an additional linear layer in the output with their weights integrated out. The parameters of the shared feature map, namely weights and biases are collected in the vector z and are learned along with the precision parameters α_t and β_t of the BLR layer. Because of the linear regression weights in the final layer, Bayesian inference is analytically tractable and computationally efficient [2]. The Bayesian model for y_t is given by the Equation 4.1 with zero mean Gaussian prior over the weights w_t.

$$P(y_t \,|\, w_t, z, \beta_t) = \mathcal{N}(\Phi_t w_t,\, \beta^{-1}\mathbb{I}_{N_t}),\;\; P(w_t \,|\, \alpha_t) = \mathcal{N}(0,\, \alpha^{-1}\mathbb{I}_P) \tag{4.1}$$

where $\beta_t > 0$ and $\alpha_t > 0$.

Posterior Inference: For the new input x_t^* for task t, with the features $\phi_z(x_t^*)$ and function value without noise $f_t^* = w_t^T \phi_t^*$, the predictive distribution is Gaussian given by $P(f_t^* \,|\, x_t^*, D_t) = \int P(f_t^* \,|\, x_t^*, w_t) P(w_t \,|\, D_t) dw_t$. The predictive distribution $P(f_t^* \,|\, x_t^*, D_t) = \mathcal{N}(\mu_t(x_t^*),\, \sigma_t^2(x_t^*))$ is given by the mean and variance below in Equations 4.2a and 4.2b as derived in the supplementary of the work [2]. These are fed into the acquisition function and essentially drive the Bayesian optimization.

$$\begin{aligned} \mu_t(x_t^*) &= \frac{\beta_t}{\alpha_t}(\phi_t^*)^T K_t^{-1} \Phi_t^T y_t \\ &= \frac{\beta_t}{\alpha_t} e_t^T L_t^{-1} \phi_t^* \end{aligned} \tag{4.2a}$$

$$\begin{aligned} \sigma_t^2(x_t^*) &= \frac{1}{\alpha_t}(\phi_t^*)^T K_t^{-1} \Phi_t^T K_t^{-1} \phi_t^* \\ &= \frac{1}{\alpha_t} \left\| L_t^{-1} \phi_t^* \right\|_2^2 \end{aligned} \tag{4.2b}$$

where L_t is the Cholesky factor of the matrix $K_t = L_t L_t^T$ with $K_t = \frac{\beta_t}{\alpha_t}\Phi_t^T\Phi_t + \mathbb{I}_P$ and $e_t = L_t^{-1}\Phi_t^T y_t$. Here, the posterior mean and variance are given for the frequently occurring case in transfer learning where the number of observed points is higher than the feature space dimensionality (number of units in the final hidden layer), $N_t > P$. This reduces computation during inference as a matrix of size $P \times P$ is inverted rather than $N_t \times N_t$.

Negative log likelihood: The Learning criterion (ρ), used to update the parameters of shared NN z, precision parameters α_t and β_t, is the sum of the negative log likelihoods of all tasks given by the Equation 4.3.

$$\begin{aligned} \rho(z, \{\alpha_t, \beta_t\}_{t=1}^T) &= -\sum_{t=1}^{T} log\ P(y_t \,|\, z, \alpha_t, \beta_t) \\ &= \sum_{t=1}^{T}[\frac{-N_t}{2} log\beta_t + \frac{\beta_t}{2}(||y_t||^2 - \frac{\beta_t}{\alpha_t}||e_t||^2) + \sum_{i=1}^{D} log([L_t]_i i)] \end{aligned} \tag{4.3}$$

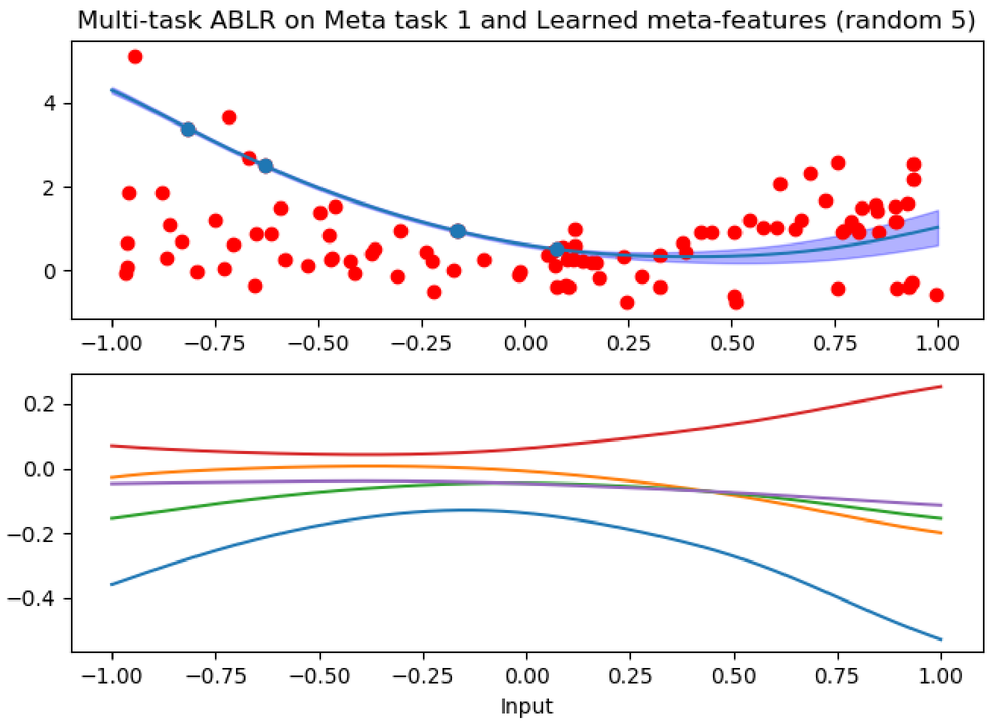
In this work, Multi-Task ABLR is treated as a latent model and models the points in the latent space $\mathbb{R}^d$ rather than directly modeling in the higher dimensional input space $\mathbb{R}^D$. Figure 4.3 shows the working of a Multi-Task ABLR model on meta data from a 1-dimensional parametrized quadratic function $f(x) = (a(x+b))^2 + c$ with three parameters a, b, c. The function parameters are sampled uniformly to generate 20 meta-tasks with 5 evaluations each as explained in the Section 5.2 and later tested on a new target task. Also, Multi-Task ABLR which models directly on the input space is used as a baseline in the experiments in the upcoming Section 5.4.

4.3 Latent Space Models

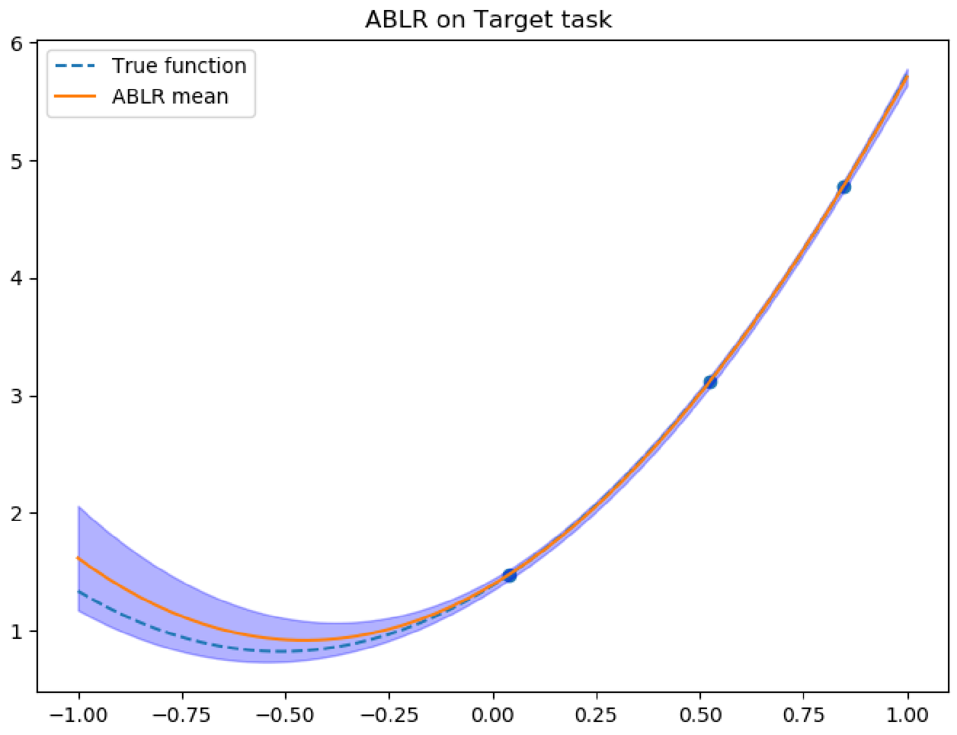
To find a lower dimensional space to perform optimization, a transformation of input data points into the corresponding low dimensional space is learned. Two ways of transforming the input points into low dimensional latent space are considered here. 1. Linear transformation - Projection onto a linear embedding 2. Nonlinear transformation - Mapping onto a nonlinear subspace. In reality, since the transformation is unknown *a priori*, a generic nonlinear transformation that achieves better compression than the linear embedding model can also be used. While learning a linear embedding works well in practice, a nonlinear embedding would provide better compression and learn nonlinear subspaces [6]. If the transformation is known to be linear in advance, learning a linear transformation in the form of a matrix would be an ideal choice. This Section elaborates on the Projection and Auto Encoder models with Adaptive Bayesian Linear Regression acting as a latent model.

4.3.1 Projection ABLR Model

Earlier work [1] uses random projection matrices to find low dimensional linear embeddings and search over them for the optimum. If no prior information on the transformation is given and a high budget is allowed, performing optimization on multiple random embeddings turns out to perform well. Yet, with metadata available at our disposal, we want to make



(a) Top part: Mean function on meta-task 1 with 2 standard deviation. All metadata points are in red and corresponding meta-task points are in blue. Bottom part: 5 Learned features across meta-tasks



(b) Mean prediction on target task with 2 standard deviation

Figure 4.3: Working of Multi-Task ABLR model on a 1-D Quadratic problem (a) Fit on a meta-task and learned features across meta-tasks which are different forms of quadratic curves, (b) Fit on the target task with just 3 points

efficient use of it not just by learning a common set of features as [2] does but also learn the transformation which enables us to optimize faster in a lower dimensional space. If the high dimensional input space can be recovered from an intrinsic low dimensional space by using a linear projection, learning an embedding matrix that projects the input points from high to low dimensional space is sufficient. Hence, the idea is to learn two matrices, one for Projection and one for Reconstruction. At every iteration in the BO loop, the incoming input data points are used to update the learned projection and reconstruction matrices. In the corresponding BO iteration, when a point in latent space is suggested by the acquisition function, it is projected back to the input space by using the learned reconstruction matrix. This model is referred to as *Projection-ABLR* further in this thesis.

Equation 5.2 defines the mapping from a single input point to the latent space.

$$z = Px \tag{4.4}$$

where P is a matrix of size $d \times D$, $x \in \mathbb{R}^D$ and $z \in \mathbb{R}^d$. During the maximization of the acquisition function, we fix the latent space to the hypercube $[-1, 1]^d$. Whenever input points are projected outside these bounds, the latent points are clipped to the boundaries. As this transformation is lossy and only a pseudo-inverse of the matrix P is available to project back the points in latent space to input space, a reconstruction matrix is also learned. The reconstruction matrix R is used in the BO loop to project the points in the latent space to the input space for evaluating the objective function.

$$\hat{x} = Rz \tag{4.5}$$

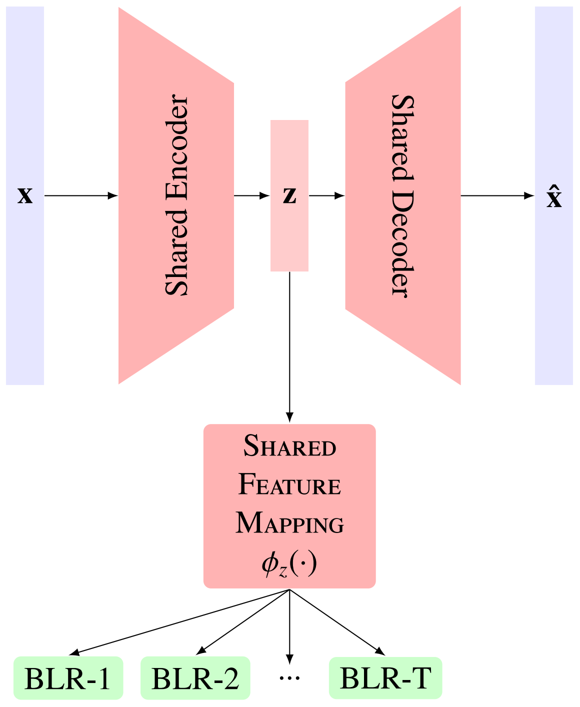
where R is a matrix of size $D \times d$ which maps the vector $z \in \mathbb{R}^d$ to the vector $\hat{x} \in \mathbb{R}^D$. The entries of P and R are learned by gradient descent during the joint learning as described in the Section 4.1.

4.3.2 Auto Encoder ABLR Model

Learning the transformation onto latent space by a sequence of composite nonlinear functions is generally achieved by stacking perceptrons layerwise. Following the work of [6], which uses a Neural Network as an Encoder, we use two Neural Networks in an Encoder-Decoder structure widely known as an Auto Encoder. Such structure is commonly used for dimensionality reduction tasks by configuring fewer units in the last layer of the encoder structure (bottleneck). Nonlinear transformation of the input space to the latent space is achieved by using nonlinear activation functions in at least one hidden layer. In comparison to the Projection model, the Encoder and Decoder networks substitute the projection and reconstruction matrices respectively. Intuitively, this type of nonlinear transformation yields better compression of the input space than linear embedding. This model is referred to as the *AE-ABLR* further in this thesis. Figure 4.4 shows the schematic structure of *AE-ABLR* model for multiple tasks with the shared encoder, decoder, feature mapping and a common latent space z.

Training the latent models: Training the Multi-Task ABLR model together with Projection or Auto Encoder models needs an appropriate loss function as briefed in the Section 4.1.

Figure 4.4: Structure of Multi-Task Auto Encoder Adaptive Bayesian Linear Regression: Red color indicates the shared structures across all tasks and Green represents the task specific Bayesian Linear Regression (BLR) component



Taking the Negative Log-Likelihood (NLL) from the Equation 4.3 and Mean Square Error (MSE) into account, our total loss to be optimized for the multi-task scenario is given by:

$$\text{Total loss} = \sum_{t=1}^{T} \Big[-log\ P(y_t \,|\, z, \alpha_t, \beta_t) + \|x_t - \hat{x}_t\|_2^2\Big] \tag{4.6}$$

Variational Auto Encoder approach: The method proposed by [5], *VAE-BO* is a promising way to do Bayesian optimization in latent space when we have millions of data points in our meta-dataset. As discussed in the Section 2.1, *VAE-BO* trains the Encoder-Decoder structure using a variational approach as in [41] along with the prediction model. Later, they take the trained Encoder-Decoder structure to perform optimization in the latent space without retraining it during optimization.

We compare our method against our adapted version of *VAE-BO* later in the results, Section 5.4. Specifically, we implemented a method which is a hybrid between Variational Auto Encoder and a standard Auto Encoder since the metadata available is assumed to be moderate in size. Driven by the motivation of learning a structured latent space, we constrained the latent space to be Gaussian distributed during meta-training by minimizing the modified ELBO (Evidence Lower Bound) loss as derived in [41]. This, in turn, is achieved by adding a regularization term in the loss which minimizes the KL divergence between the distribution of the latent space and a unit Gaussian distribution.

$$\begin{aligned}\text{VAE Loss} &= \text{Reconstruction loss} + \text{KL divergence}\\ &= \|x_t - \hat{x}_t\|_2^2 - \frac{1}{2}\sum_{i=1}^{d}[1 + log(\sigma_i^2) - \mu_i^2 - \sigma_i^2]\end{aligned} \tag{4.7}$$

where d is the number of units in the last layer of the encoder, μ_i and σ_i are the mean and variance of the latent variables in the last layer of encoder, x_t and $\hat{x}_t$ are the original and reconstructed data point respectively. By minimizing the KL divergence between the latent distribution and a unit gaussian, we enforce the latent variables to follow a Gaussian distribution when training on metadata. Intuitively, this approach will avoid distortion in the latent space and prevent the points in the latent space from collapsing while mapping from input to latent space.

4.4 Latent Space Bayesian Optimization using Metadata

In this section, we describe the usage of latent models in the Bayesian optimization framework. The typical BO framework is explained in detail in the earlier Section 3.1 and this section highlights and provides the differences between typical and Latent space BO. Key differences between typical BO and Latent space BO are listed below.

- In Latent space BO, the probabilistic model resides in low dimensional latent space whereas in typical BO, the model lies in high dimensional input space, i.e, $f : \mathbb{R}^d \rightarrow \mathbb{R}$ in Latent space BO and $f : \mathbb{R}^D \rightarrow \mathbb{R}$ in typical BO with $d << D$.
- As a result of the first item, the acquisition function is maximized in lower dimensional space $\mathbb{R}^d$ in Latent space BO than in $\mathbb{R}^D$ which is comparatively efficient when using global optimization techniques such as DiRECT [42], CMA-ES [43].
- Once we have the maximum from optimizing the acquisition function, in Latent space BO it is projected back to the input space and then the objective function is evaluated at the reconstructed point. If this reconstruction is not learned accurately, it results in slow convergence to the optimum. Whereas in typical BO, this reconstruction is not needed.

The training procedure, in general, is divided and implemented into *Meta-train* and *Target-train* by training on historical tasks (metadata) and the target task respectively.

Meta-train: During meta-train, common structures, namely Encoder, Decoder, Feature mapping Neural Network are created and initialized. Also for every task, a Bayesian Linear Regressor is initialized. The model is trained using a variant of batch gradient descent, ADAM, [44] with data from one task forming a batch. Since this is independent of the optimization loop, we can afford to train slowly and even before we know details about the target task. We can also think of Meta-train as an initialization strategy before we optimize the target task.

Target-train: As this is embedded in the BO loop, it is of paramount importance to train faster to converge to the optimum quicker. Usually, if the target task is closely related to the meta-tasks, it is not required to train the shared feature mapping, encoder and decoder structures but only the task specific BLR parameters. Practically, since it is difficult to determine the similarity between meta-tasks and target task without any evaluation on target task, it is advised to train the shared structures as well.

Algorithm 2 puts together the steps discussed in Latent BO and provides a general overview of the implementation. We define the latent model, *Projection-ABLR* or *AE-ABLR* on line 1 and meta-train offline before the BO loop. BO is configured to run for N_{max} iterations with or without an initial dataset of target task D_T. We begin to optimize the target task by first creating a BLR for target task (line 7). During BO, training the latent model implicitly has two steps as in lines 10 and 11. Also, selecting the next input for evaluation is described with heuristics in lines 13, 14, 15. Rest of the algorithm follows the standard BO framework described in 1.

Algorithm 2: Latent space Bayesian Optimization using Metadata

```
Input: Objective function, $f_T$ (Target task: $T$)
Output: Optimum of Target Task
Data: Metadata - Number of meta tasks $M$, Points per each task $N_t$
1  Define mapping: Projection or AutoEncoder model
2  Meta-train: Train the Latent model on metadata (Offline)
3      Joint training of M-task Adaptive Bayesian Linear Regressor (ABLR) and
         mapping model
4  current_step = 1
5  max_iterations = $N_{max}$
6  Initial data of target task, $D_T$ = {x, y}
7  Create a new BLR for the target task
8  while current_step <= max_iterations do
9      Train the Latent model: Target-train
10         Dimensionality reduction: $z = Mapping(x)$
11         Learn the low dimensional surface: $p(f_z/z, y)$
12     Select next input for evaluation:
13         Maximize Acquisition function over $[-1, 1]^d$: $z_{next}$
14         Project $z_{next}$ to input space: $x_{next}$
15         Clip $x_{next}$ if projected outside bounds
16     Evaluate $x_{next}$ on Target Task $T$
17     Append dataset $D_T = \{x_{next}, f_T(x_{next})\}$
18     Increment current_step by 1
19 Return optimum of $D_T$
```

5 Experiments

In this Chapter, we discuss the experimental evaluation of our method and compare it against the current state-of-the-art baselines with and without the Transfer Learning aspect. We establish the training procedure of our model in the Section 5.1. In the Section 5.2, we define and describe the synthetic functions we have used. In the Section 5.3, we validate our method by visualising the learned intrinsic function. Finally, in the Section 5.4, we present comparisons of our method with the non-Transfer Learning and Transfer Learning methods.

5.1 Model Structure and Training

In this Section, we discuss the performance of our approach in different stages of model development. The specific losses in the total loss as described in the Section 4.1 are Negative log-likelihood (NLL) and Mean Square Error (MSE). They are weighted using two parameters, w_{nll} and w_{mse} and are learned along with other parameters from data. As we believe that it is necessary to model in the latent space effectively than reconstructing input point from latent space, they are initialized to $w_{nll} = 2.0$ and $w_{mse} = 1.0$. The modified loss value which was tried out is given in the Equation 5.1.

$$Modified\ Total\ loss = w_{nll} * (NLL) + w_{mse} * (MSE) \tag{5.1}$$

We observed that both the loss terms should decrease at the same rate for the latent model to learn a meaningful latent space for modeling and reconstruction mapping to input space. However, BO performed using the model trained on this modified total loss did not yield better regret values than the model with fixed unit weights, $w_{nll} = 1$ and $w_{mse} = 1$. All our experiments are carried out with fixed unit weights.

While experimenting with the training procedure of the latent models, we analyzed the optimal way of training the weights of shared structures (Encoder, Decoder, Feature mapping) on the target task. We tried out different approaches by freezing, 1. None of weights of shared structures 2. Weights of Encoder only 3. Weights of Decoder only 4. Weights of Encoder and Decoder 5. Feature mapping weights 6. All the above. However, based on the results and as discussed in the Section 5.5, we stick to training all the parameters of the latent model and did not freeze any part of the network for the experiments.

Estimating intrinsic dimensionality from metadata

An open problem in the formulation of our problem statement is that we know the number of relevant or important parameters in a case with a high number of parameters. However, in reality, the distinction between relevant and not relevant is not clear and a method for estimating the intrinsic or effective dimensionality from the metadata would be ideal. Since we have metadata at our disposal, we tried out two approaches to estimate the number of effective dimensions. We construct multiple models of increasing dimensions in the latent space and train the various models on metadata. For example, if the total number of input parameters is 20, we construct latent models with latent units in *AE-ABLR* from $[1, 2...19]$ and likewise in *Projection-ABLR*.

1. Cross validation by leaving out some tasks from metadata as Target tasks and meta-train on only remaining data
2. Training the multiple latent models on the entire metadata

Later, we plot and compare the MSE and NLL for each latent model to find the intrinsic dimensionality. Even though the NLL plots showed elbow-like behavior for multiple random runs, the estimate was inconsistent and inconclusive. We leave this as future work and discuss in the Chapter 6.

The intrinsic dimensionality or the number of units in the last layer of Encoder in case of *AE-ABLR* or the number of columns of projection matrix in *Projection-ABLR* is assumed to be known in all our experiments.

5.2 Synthetic Functions

A common first step for newly developed models and algorithms for global optimization problems is to test on standard synthetic functions which contain many local optima. We evaluate our method on two standard synthetic functions in the literature by modifying it to reflect our problem statement with the properties:

1. High dimensional problems with intrinsic low dimensionality
2. Transfer Learning from data of related tasks.

In this Section, we discuss the setup we have used to create benchmarks with the above two criteria. Firstly, to define a problem with high dimensions and low effective dimensionality, we define a matrix A which projects the points from D dimensional space to d dimensions with $d << D$. The projected points are passed on to the actual (lower dimensional) function for evaluation. Equation 5.2 shows the relationship between the higher dimensional input variable x and the low dimensional latent variable z.

$$f_X(x) = f(z) = f(Ax) \tag{5.2}$$

where A is a $d \times D$ matrix with orthogonal rows forming $AA^T = \mathbb{I}$. The meta-tasks and the target task are rotated and projected by the same matrix A which is the orthogonal component in the QR decomposition of a random uniformly sampled matrix. For evaluation, only f_X is available to the user whereas f is not available. In the results, benchmarks are denoted as *Function_d_D* where effective dimensionality $d_{eff} = d$ and D is the original dimensionality.

Parameterized Quadratic function

We adapt the parameterized quadratic function in [2] as a high dimensional function with intrinsic low dimensional structure by using the projection technique in the Equation 5.2.

Table 5.1: Search space and parameter ranges of the functions

	Search space	Context/Settings
Quadratic	$[-5,5]^D$	$a, b, c \in [0.1, 10]$
Rosenbrock	$[-2,2]^D$	$s1, s2 \in [1,4]$

The quadratic function is parameterized by three parameters a, b, c and is defined as in the Equation 5.3.

$$f(z) = \frac{1}{2}a||z||^2 + b\mathbf{1}^T z + 3c \tag{5.3}$$

By uniformly sampling the parameters a, b, c in the range as given in the Table 5.1, multiple tasks are generated which are scaled and shifted versions of each other.

Parameterized Rosenbrock function

The Rosenbrock function [45] is a standard global optimization problem which has an additive structure and is difficult to optimize as its global minimum lies in a valley. It is adapted to the transfer learning case by appending extra dimensions called as settings when evaluating the function. Settings differentiate tasks and implicitly increases the dimensionality of problem by number of settings. Each task will have different valued settings and can have any number of settings associated with it. When we need to evaluate this function at a point $x \in \mathbb{R}^{\mathbb{D}}$, with the help of 5.2, we get the corresponding point z_{proj} in $\mathbb{R}^d$ and append it with the settings $(s1, s2)$, two settings in this case, to the form $z = [z_{proj}, s1, s2]$ and evaluate it on the function as defined in the Equation 5.4.

$$f(z) = \sum_{i=1}^{(n_s+d)-1} [100(z_{i+1} - z_i^2)^2 + (z_i - 1)^2] \tag{5.4}$$

where n_s is the number of settings of the task and d is the effective dimension of the function. Scipy's implementation [46] of Rosenbrock is used in our work.

5.3 Meta-Learning the Intrinsic Function

As a first step to validate our method, we need to ensure that our model is able to learn the intrinsic function in the $\mathbb{R}^d$ space from the provided metadata. This will lead to a reasonable model and efficient optimization of the objective function in the low dimensional latent space. As we train our latent model using randomly sampled meta-tasks (see Section 4.3) and visualize the latent space in Figures 5.1, 5.2, the following is interpreted:

- The intrinsic function is learned by jointly training the parameters of Encoder/Projection, Decoder/Reconstruction and the ABLR model.

- The learned mapping from the input space $\mathbb{R}^{\mathbb{D}}$ to low dimensional latent space $\mathbb{R}^d$ and its inverse mapping is linear in most cases as expected.

With fewer points from the target task, our method is able to adapt to the low dimensional structure while simultaneously learning the prediction model. After training on the metadata, we visualized the meta tasks to verify if our model fitted to the training points. We test our method on 2-dimensional quadratic benchmark with 1-dimensional intrinsic space for visualization purposes. Figure 5.1 demonstrates the model fit on the data points from meta tasks and the corresponding intrinsic function of one meta-task in the latent space. To verify if the meta-trained model adapts to the randomly sampled target task, four points from the target task are sampled and fed to the model. The corresponding results from the Figure 5.2 illustrate that the model is able to adapt to the target task. Since the input space of the benchmarks is linearly transformed from $2D$ space to $1D$ space by a matrix as described in the Section 5.2, we expect our model to learn a linear transformation which is a plane in $2D$ space as visualized in the Figure 5.3. To sum up, the general structure represented in the Figure 4.1 is pragmatic in moving the optimization problem from higher dimensional space to a lower dimensional space, provided that the function is intrinsically low dimensional. Thus, Meta-learning the intrinsic function enables us to perform latent space BO.

BO on low dimensional problems for visualization

To gain intuition about our method in the context of BO, BO is performed on a 2-dimensional function with one effective dimension. Figure 5.4 shows the 2D quadratic function and its corresponding contour plot along with the sampled points. Latent space BO using *AE-ABLR* initially does random sampling to figure out the optimal latent space and then is able to sample in an approximate linear subspace (line in 1D case) and thus converging at the minimum. This accurate behavior of finding the right mapping, thereby learning the intrinsic function, is not always observed as the model needs many data points initially.

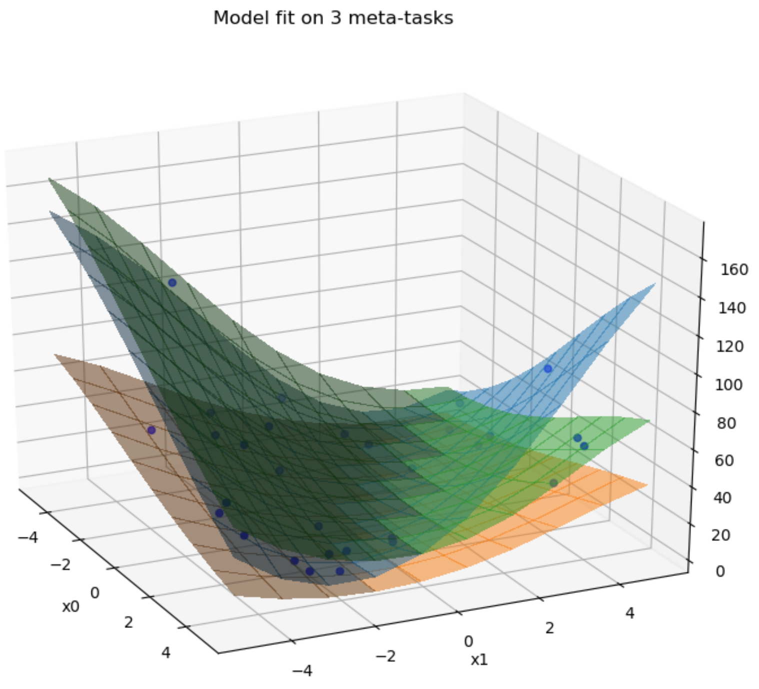
5.4 Results

Primarily, two different cases are studied in this Section: BO on high dimensional functions with intrinsic low dimensional space without Transfer Learning (1) and with Transfer Learning from metadata (2). We compare our methods *Projection-ABLR*, *AE-ABLR* to the BO algorithms *REMBO* and *InterleavedREMBO* in case one. For case two, we make a comparison of our methods against our variant of *VAE-BO* and *ABLR*. Table 5.2 describes the characteristics of all the models used in the experiments. For all our experiments, we use *Simple Regret* which is defined below as an evaluation metric to compare different algorithms.

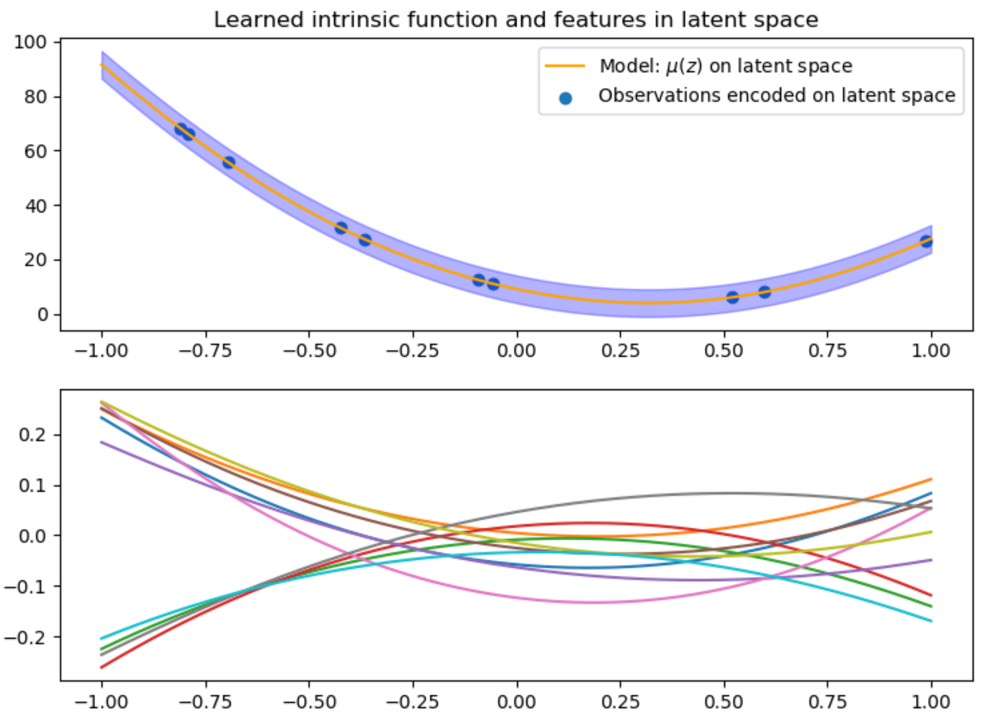
Simple Regret: Simple Regret or Optimality gap is defined as the difference between the best function value found at the current step and the true optimum. It is a relevant and significant measure in global optimization to compare and validate algorithms for varying values of optima.

$$\text{Simple regret} = y_{current\_best} - y_{optimum} \tag{5.5}$$

In the case of synthetic functions, $y_{optimum}$ is typically calculated before the BO loop using popular global optimization algorithms such as DiRECT [42] or CMA-ES [43]. We used



(a) Model fit on 3 random meta-tasks. Points of each meta-task are shown as blue dots



(b) Latent space model and features of ABLR where intrinsic quadratic form is learned

Figure 5.1: Projection-ABLR model trained on metadata (30 tasks and 10 points each) of 2D Quadratic function with 1D intrinsic space

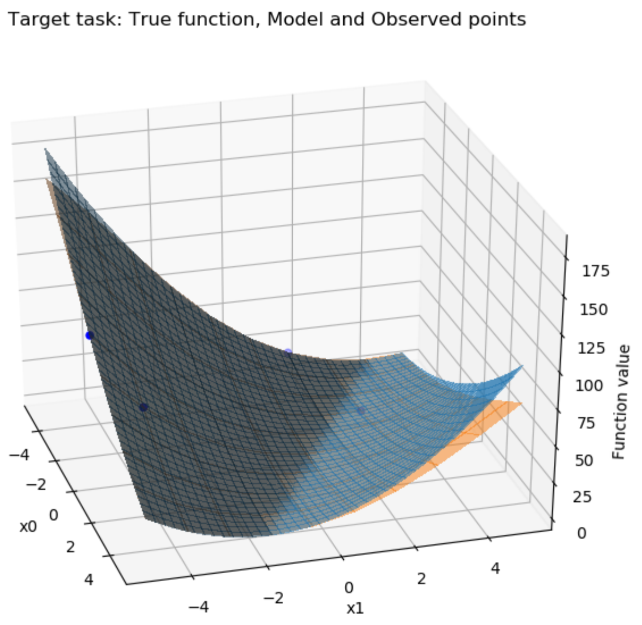


Figure 5.2: Projection-ABLR model on Target task with only 4 randomly sampled points: Original function (Orange), Model fit (Blue)

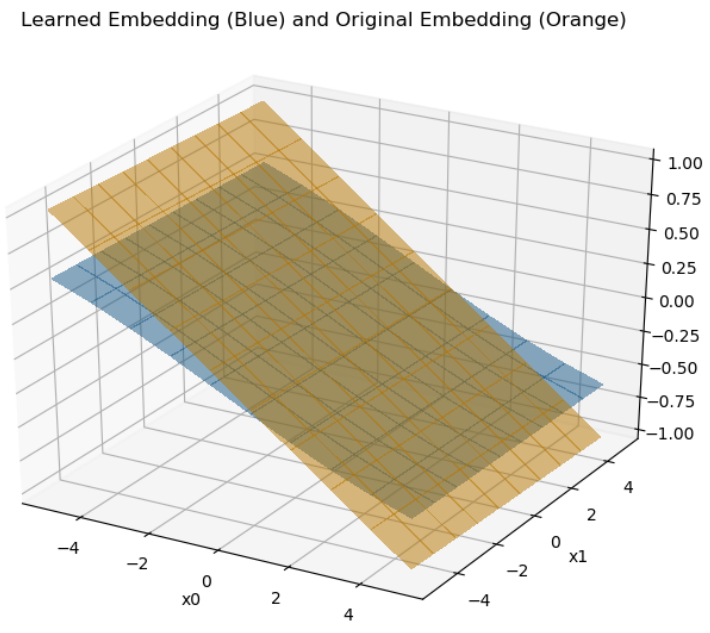
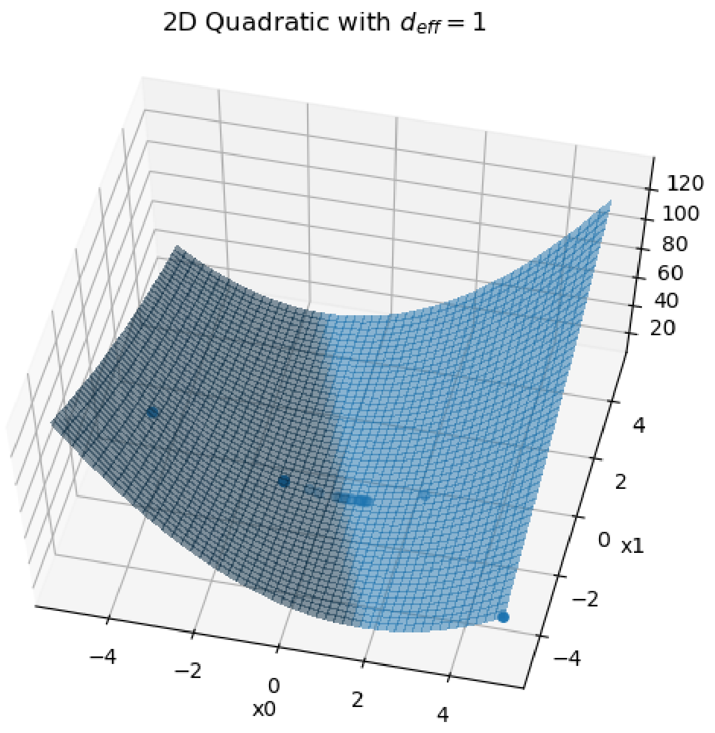
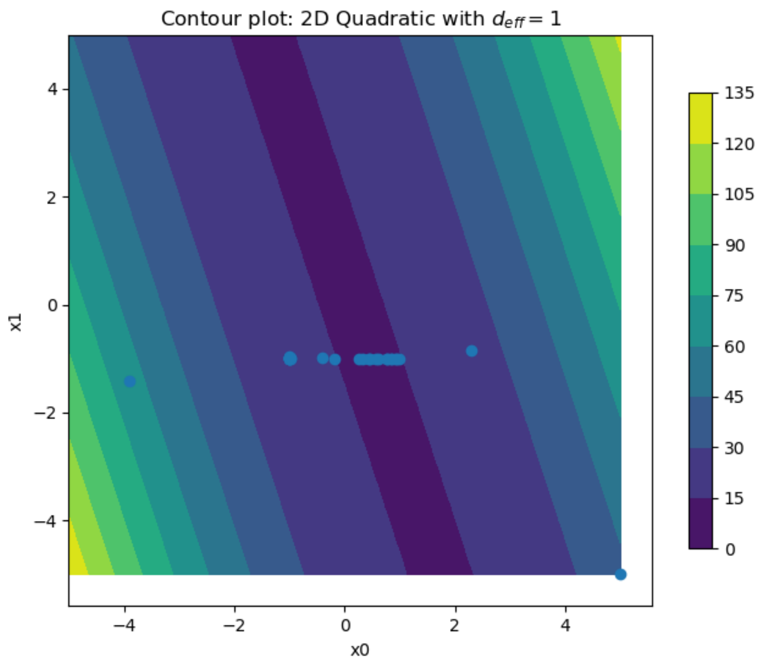


Figure 5.3: Learned and Original linear embedding in the input space: Every point in 2D input space is mapped via learned projection matrix to latent space



(a) 2D Quadratic with intrinsic dimensionality of one and points sampled during BO



(b) Contour plot with the corresponding sampled points

Figure 5.4: Typical BO using *AE-ABLR* on a 2-dimensional Quadratic function with $d = 1$ for 30 iterations

Table 5.2: Models used in experiments

Model	Modeling space	Based on	Transfer Learning
REMBO, InterleavedRembo	Latent space $\mathbb{R}^d$	GP	No
MultiTaskABLR	Original space $\mathbb{R}^{\mathbb{D}}$	BLR	Yes
Projection-ABLR, AE-ABLR, VAE-ABLR	Latent space $\mathbb{R}^d$	BLR	Yes

DiRECT to compute the minimum of our target functions in our experiments. Simple regret is preferred in our case since the true optimum of benchmarks can be computed easily.

Implementation of Baselines

We implemented two baselines *REMBO* and *VAE-BO* to benchmark our model for non-Transfer Learning and Transfer Learning cases respectively.

REMBO: *REMBO* [1] is a popular method for performing high dimensional BO on functions with low effective dimensionality as discussed in the Section 2.1. It uses a random projection matrix to project the points from latent space to input space. We have included it as a baseline in our experiments in this Section. While running BO, the latent space is box bounded by a heuristic $[-\sqrt{d}, \sqrt{d}]^d$ to reduce the chances of projecting outside the input space before function evaluation and for maximization of the acquisition function. If a point is projected outside the input space bounds, it is clipped, thereby inducing a nonlinear behavior. This is one of the reasons why it performs poorly in reality despite strong theoretical guarantees. It also performs poorly on rotated objective functions [47]. A variant of *REMBO* uses multiple random projections and is run in an interleaved manner for increasing the probability of finding an embedding with the optimum and is known as *InterleavedRembo*. Our implementation *REMBO* and *InterleavedRembo* is based on [48]. A random embedding is initialized during the initialization of BO and used to map the points in latent space to the original input space during BO. Our work can be considered as an extension of *ABLR* and *REMBO* to multi-task problems where we want to do Transfer Learning.

VAE-ABLR: As discussed in the Section 2.1, *VAE-BO* [5] is trained on discrete dataset for producing a continuous representation in latent space. In our implementation, as a first step, VAE is trained along with a Multi Task ABLR model on the metadata yielding a structured latent representation. In the orginal work, using the learned latent space, BO is performed in the latent space without modifying the Variational Encoder and decoder. Whereas we relearn the encoder and decoder weights during BO to adapt to the target task. The reparameterization trick suggested by [41] is used to implement a simple variational encoder which outputs mean and log variance in the last layer of the neual network. In our case, we use the variational approach of restricting the latent space to be Gaussian. During meta-training we constrain the latent space to be Gaussian and relax that constraint during target train. We take the output of mean units as the latent encoding of our inputs and ignore the units representing log variance during BO. Simply, we treat the meta-trained VAE as a normal AE during target train.

Another notable Transfer Learning work is [13] which uses a stack of GPs is also imple-

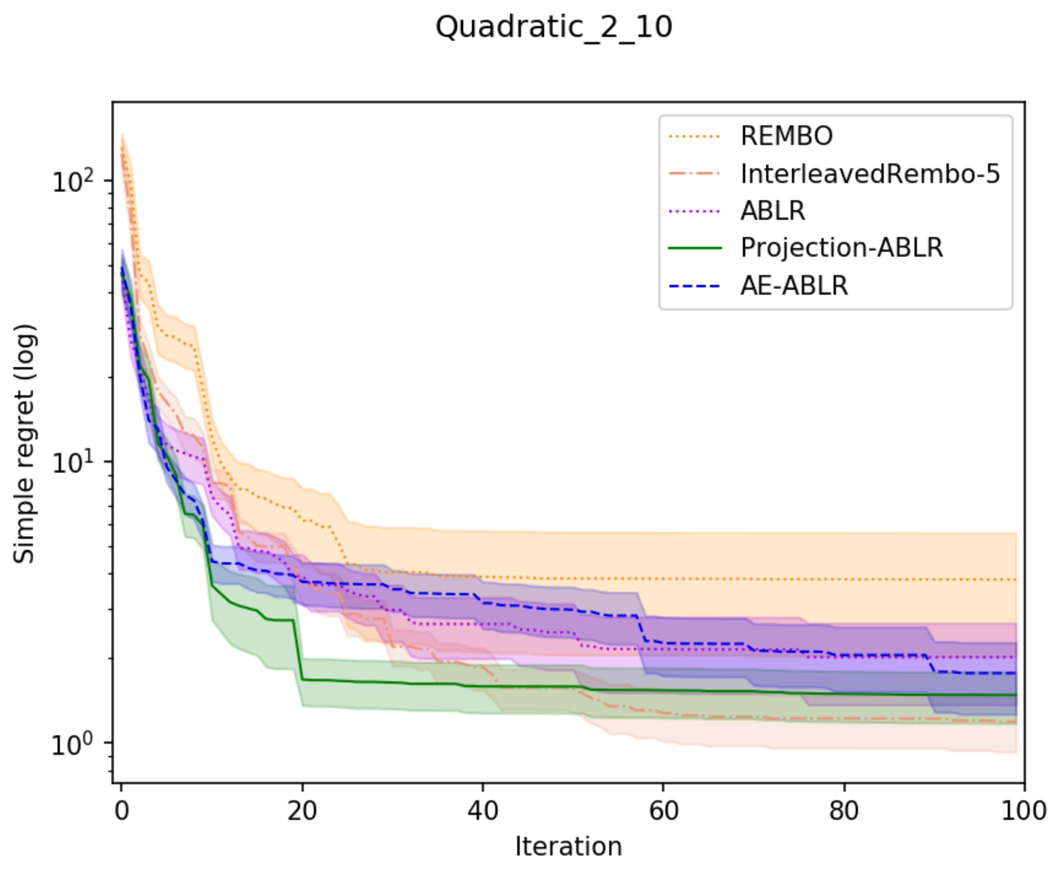


Figure 5.5: Typical BO on 10-dimensional Quadratic with 2D intrinsic dimensionality (without Transfer Learning)

mented. While running experiments, we were not able to run it for 20 trials within predefined the time limit of 9 hours for all models for each trial on 40-dimensional Rosenbrock and Quadratic functions with 10-dimensional intrinsic space. Because of the time out issues, we were only able to run fewer trials and did not include it in the results.

High Dimensional BO - without Transfer Learning

We first study our method when we have no metadata to train on i.e. without the aspect of Transfer Learning in BO. A single task *ABLR* model (with only one BLR) is also included for comparison. *InterleavedRembo-5* indicates that 5 runs of *REMBO* is run in an interleaved manner.

Figure 5.5 plots the regret for different baselines for the traditional BO on a 10-dimensional quadratic function with intrinsic dimensionality 2. *AE-ABLR* has 10-5-2-5-10 number of units in encoder-decoder structure and *Projection-ABLR* has a projection and reconstruction matrices of sizes 10×2 and 2×10 respectively. We can observe that *InterleavedRembo* performs slightly better than *Projection-ABLR* at the end whereas the latter performs much better in the initial 25 iterations. *AE-ABLR* starts off similar to *Projection-ABLR* and at the end of 100 iterations finishes with a regret value similar to *ABLR*. *REMBO* saturates after 40 iterations and even *Projection-ABLR* saturates after 25 iterations after finding a better

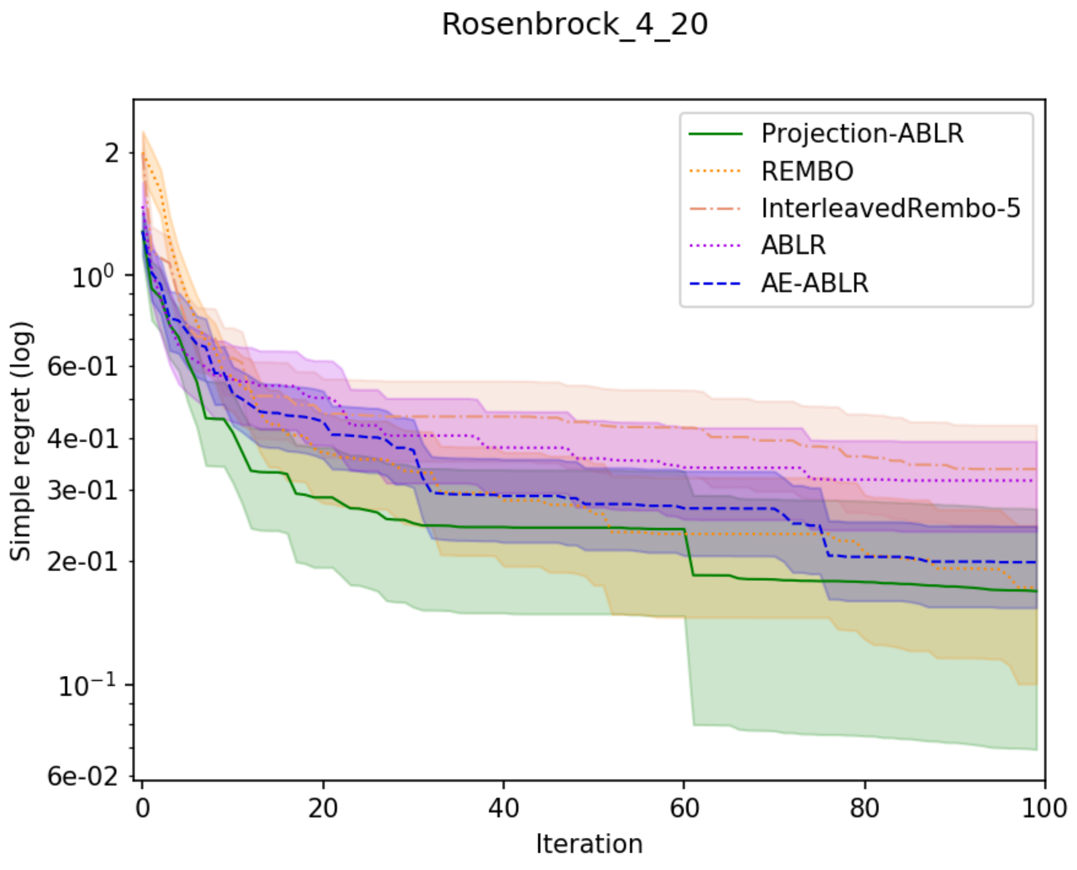


Figure 5.6: Typical BO on 20-dimensional Rosenbrock with 4D intrinsic dimensionality (without Transfer Learning)

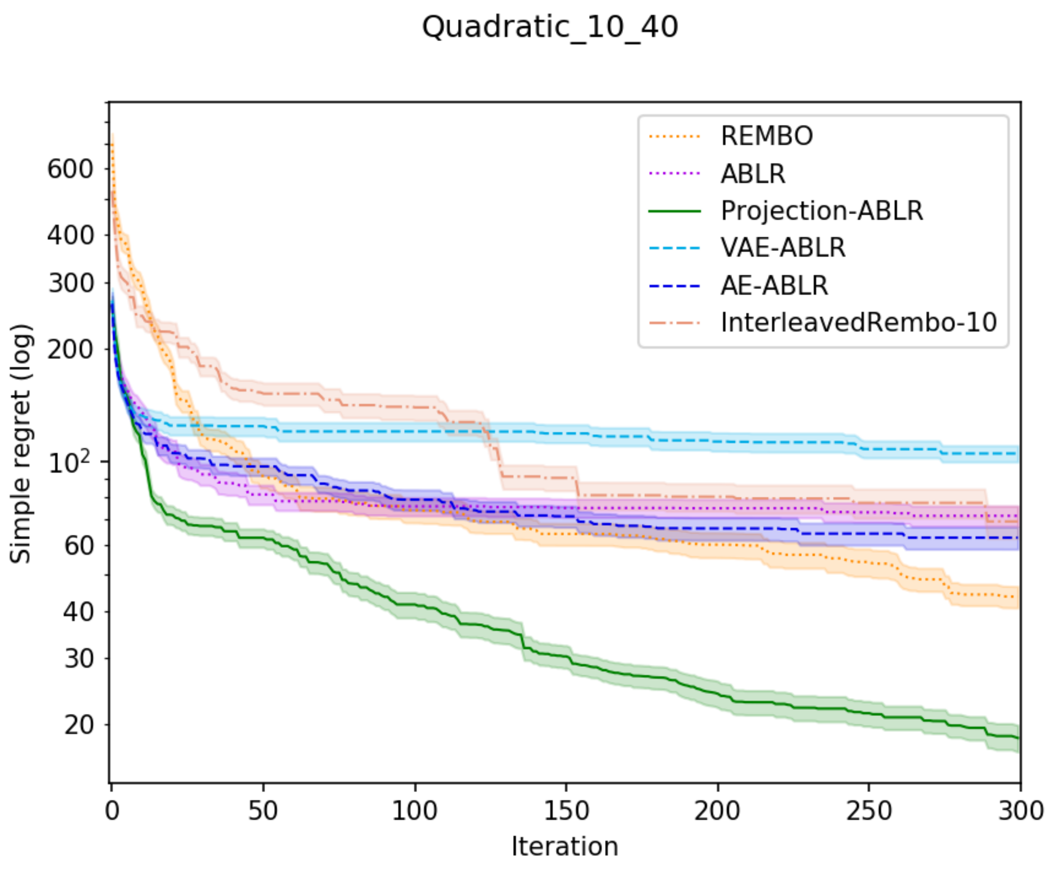


Figure 5.7: Regret for different methods on 40-dimensional Quadratic with 10-dimensional intrinsic space

minimum in the initial iterations. Results for 20-dimensional rosenbrock function with an intrinsic dimensionality of 4 is shown in the Figure 5.6. *Projection-ABLR* achieves lower regret than all the other methods throughout the iterations and *AE-ABLR* finishes as the second best. Contrary to the previous result on quadratic function, *InterleavedREMBO* saturates early in the optimization iteration and *REMBO* achieves a regret similar to *AE-ABLR* at the end of 100 iterations. In both cases, *Projection-ABLR* performs better in the initial steps suggesting that it is able to adapt to the lower dimensional space much faster and find a better optimum.

High dimensional BO - with Transfer Learning

For the Transfer Learning case, we focus on the domain of having multiple related tasks and few data points per task. We generate 30 meta-tasks and 10 points per task (300 points in total) as in [2] for the 40-dimensional Rosenbrock and Quadratic functions with an intrinsic dimensionality of 10.

Meta-training as discussed in the Section 4.4 is performed using the 300 point metadata. After meta-training, BO is run on a randomly sampled target task and the results are shown in Figures 5.7, 5.8. For both synthetic functions, we can observe that our model *Projection-ABLR* performs better than all the other methods. Also, we have compared non-Transfer Learning methods *REMBO*, *InterleavedRembo* to show the importance of Transfer Learning

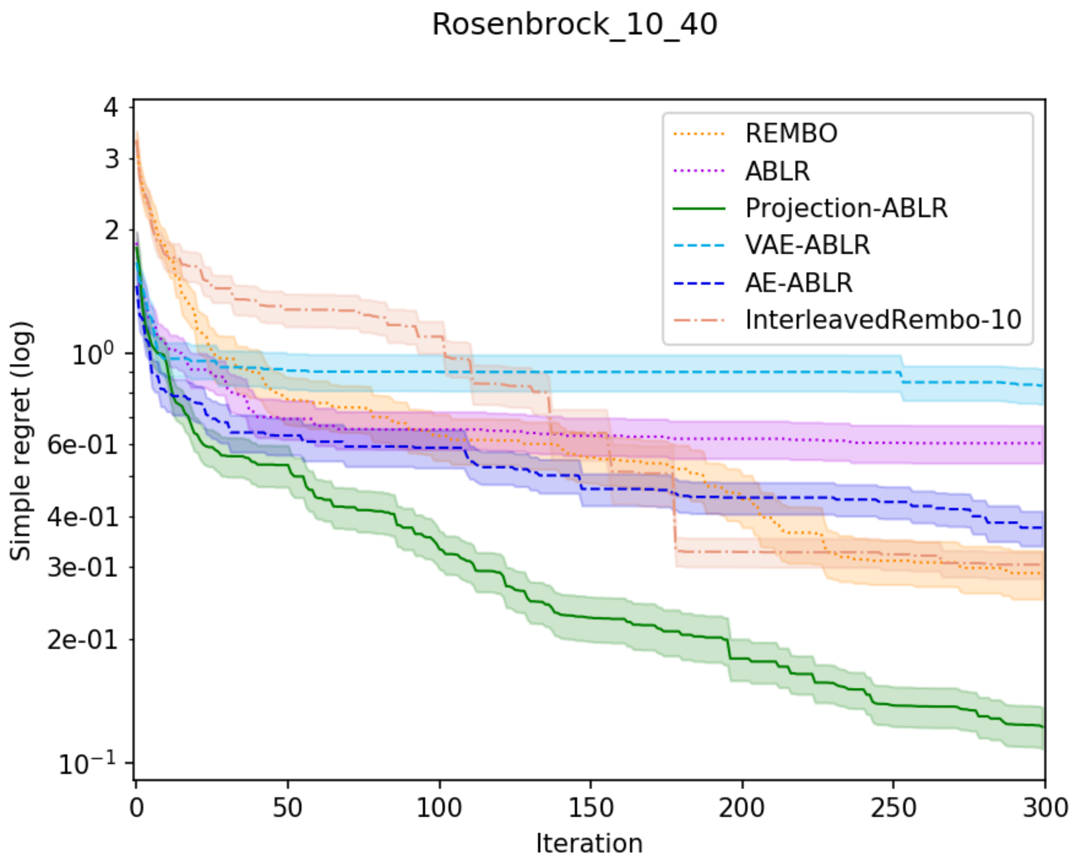


Figure 5.8: Regret for different methods on 40-dimensional Rosenbrock with 10-dimensional intrinsic space

in the given optimization problem. We can clearly notice in the initial iterations, Transfer Learning methods start off with a lower regret value compared to non-Transfer Learning methods. This is due to the Multi-Task ABLR model which learned a shared set of features from metadata during meta-training. Also, *VAE-ABLR* saturates after 25 iterations indicating that the Varitional Encoder-Decoder structure require more data than received in metadata to learn a structured latent space. This behavior is also evident in *AE-ABLR* after 100 iterations where the regret decline rate diminishes and saturates.

Contrary to expectations, a significant result from the Figure 5.7 at the end of 300 iterations is that *REMBO* yields a better regret than the Transfer Learning methods Multi-Task *ABLR* and *AE-ABLR* in 40-dimensional quadratic function. Also, from the Figure 5.8 we can observe that the *REMBO* and *InterleavedRembo* achieves better regret value at the end of 300 iterations. This indicates that the latent space BO methods without Transfer Learning can achieve similar or better performance than high dimensional Transfer Learning methods.

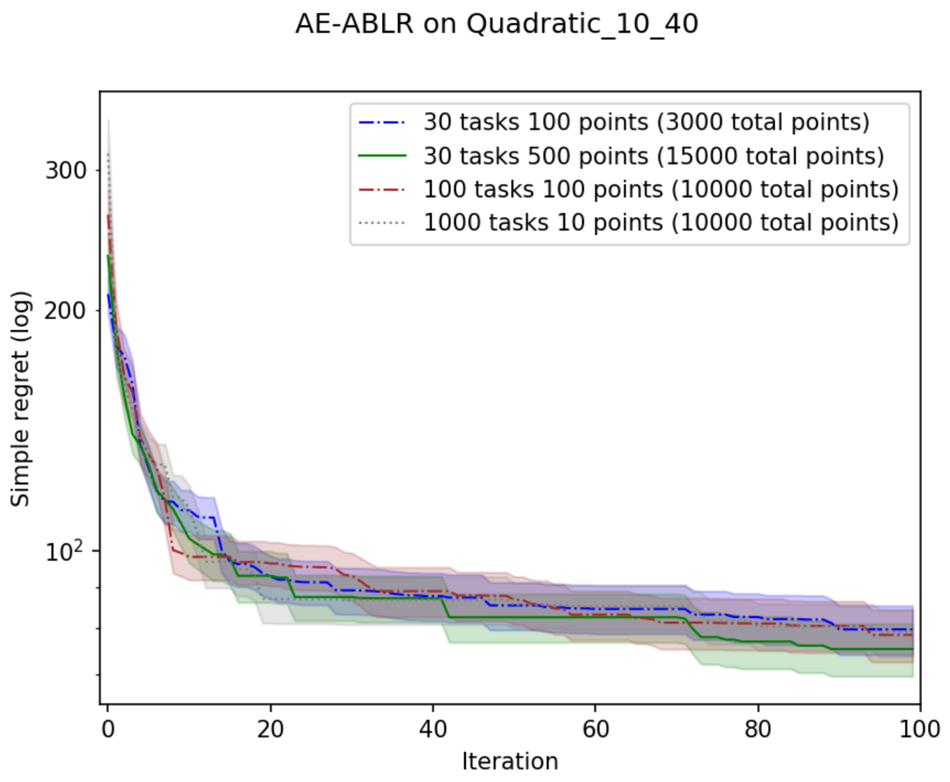
Increasing metadata

One derived advantage of our method from [2] is the scalability of the *ABLR* model with respect to the number of meta tasks and the number of data points. We study the impact of having more metadata and make a comparison within our latent space *ABLR* models to analyze the Transfer Learning capacity of the models by increasing meta tasks and data points per each task.

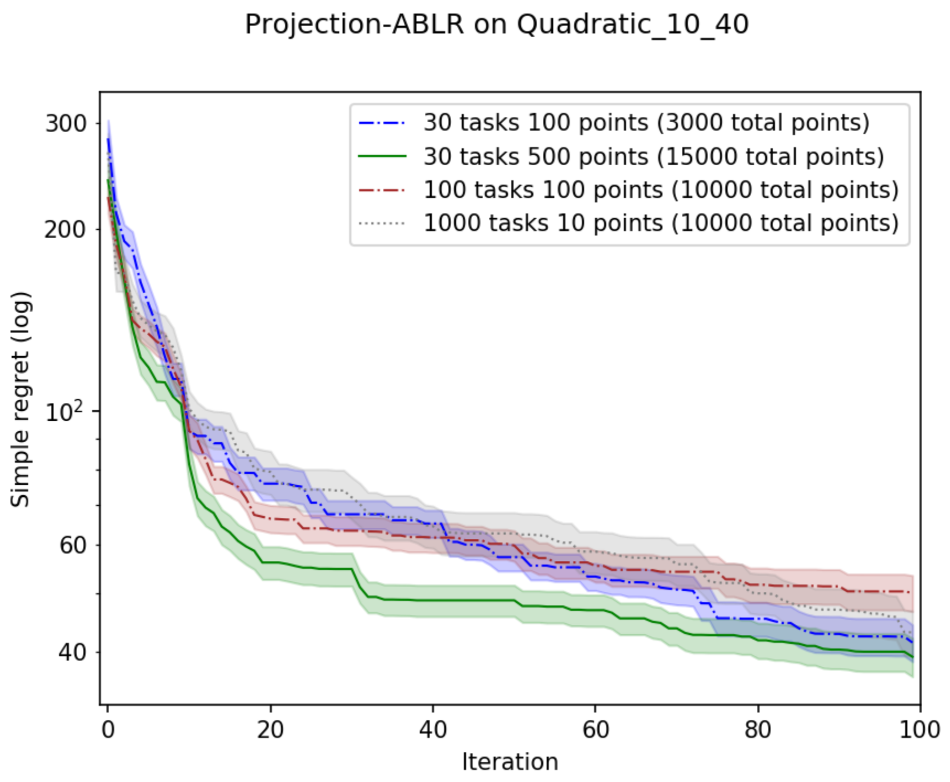
When comparing *Projection-ABLR* and *AE-ABLR* on different sizes of metadata, Figures 5.9 and 5.10 show that *AE-ABLR* does not perform well irrespective of the dataset size. In top part of the Figure 5.9, regret of *AE-ABLR* trained on different sizes of metadata is compared. As all the models converge to the same regret value at the end of 100 iterations, *AE-ABLR* does not take advantage of the metadata size. Even with $15k$ data points during meta-training, it does not increase its Transfer Learning capacity. This might be due to the sub-optimal hyperparameter configuration and the inability to find nonlinear subspace in a linearly projected benchmark. Similar behavior is observed for different metadata sizes of 40-dimensional rosenbrock function in the top part of the Figure 5.10. Regret results of *Projection-ABLR* indicate that it utilizes more metadata to achieve lower regret values during initial iterations. We expected a linear increase (lower regret) in performance when adding more metadata however, the difference in gains is comparatively small at the end of 100 iterations for all sizes of metadata. On the other hand, with 30 meta tasks and 500 data points per task, *Projection-ABLR* performs better than having 1000 meta tasks and 10 points each. And also, the model trained on 30 meta-tasks with 100 points each has higher regret initially as expected and fares off similar to the model trained on 30 meta-tasks and 500 points. This suggests that the model is able to learn better from having more data points per each meta task rather than having fewer points and more meta tasks.

5.5 Discussion

As discussed in 5.1, we carried out initial set of experiments to analyze which parts of the network to freeze during target-train. The difference in the regret value at the end of 100

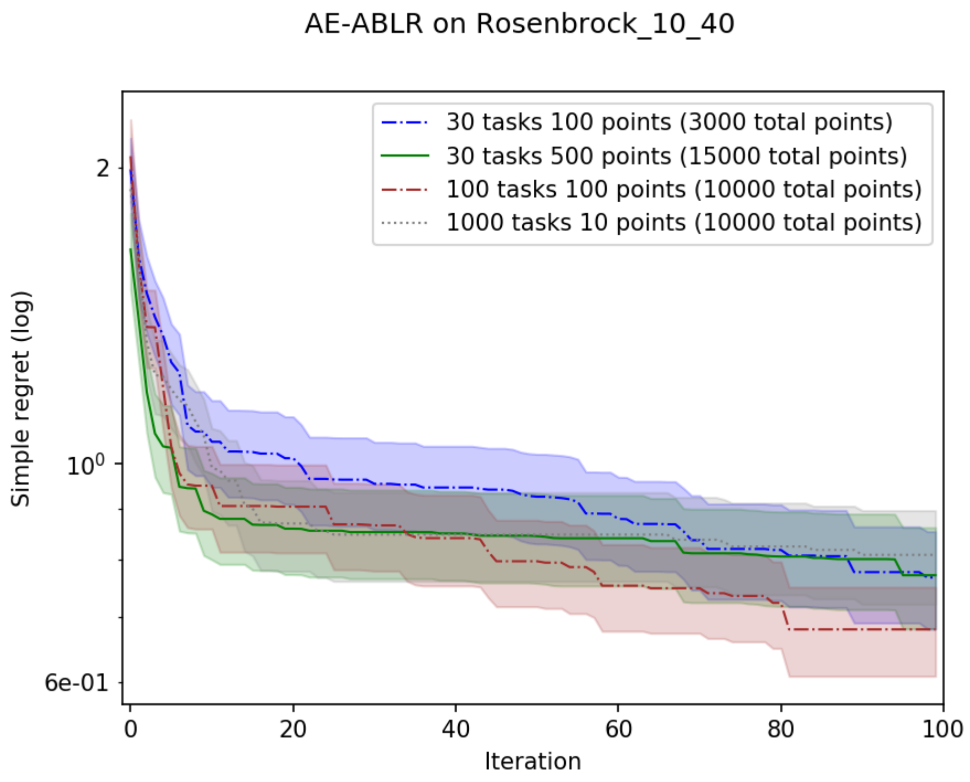


(a) AE-ABLR with four different sizes of metadata

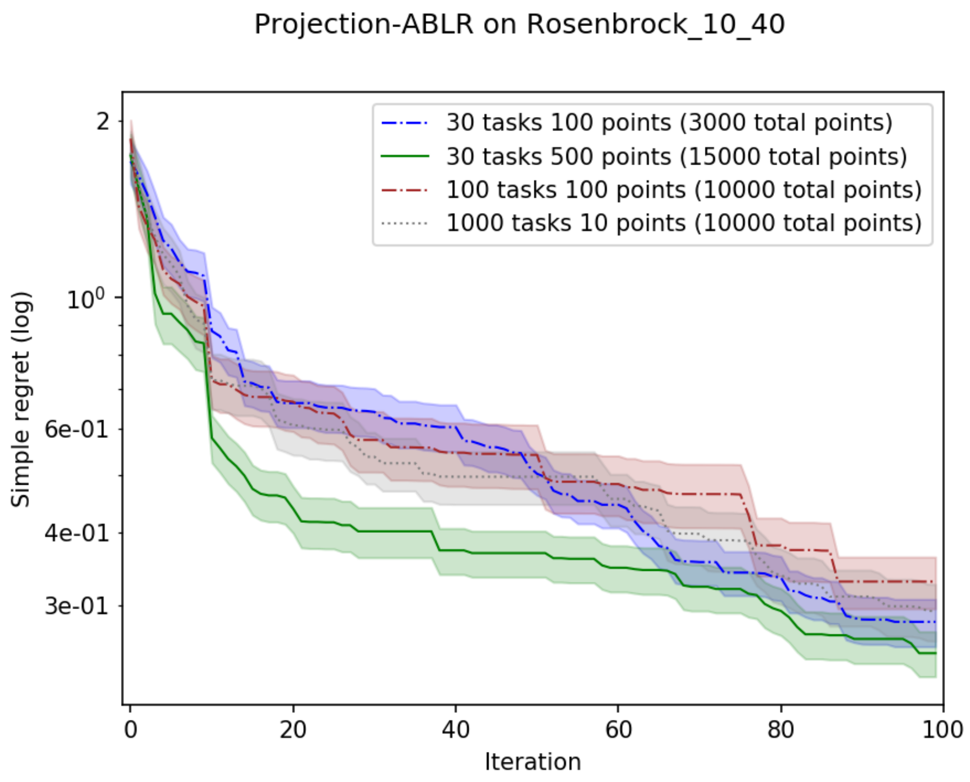


(b) Projection-ABLR with four different sizes of metadata

Figure 5.9: Comparison of our models for increasing metadata on 40-dimensional Quadratic with d=10



(a) AE-ABLR with four different sizes of metadata



(b) Projection-ABLR with four different sizes of metadata

Figure 5.10: Comparison of our models for increasing metadata on 40-dimensional Rosenbrock with d=10

iterations for 20 random trials was manually analyzed. The approach to not freeze any of the weights during target-train worked better than freezing one or multiple parts of the network. We believe that this is due to ability of the model to adapt to the target task if it is entirely different from the meta-tasks. If we freeze at least one part of the shared structure, this behavior cannot be expected and results in saturation of regret value. Hence, all the latent space models used in the results presented in the previous Section 5.4 were trained completely during target-train.

High Dimensional BO - without Transfer Learning: In the traditional BO setting without Transfer Learning (Figures 5.5, 5.6), on Quadratic and Rosenbrock functions with an intrinsic low dimensional space, our methods performs better than random projection based methods in the initial iterations. This is due to the ability to learn the transformation towards finding an optimum instead of random projection in every iteration. However, at the end of 100 iterations, our method *Projection-ABLR* ends up with a similar regret to *InterleavedRembo* in Quadratic case and *REMBO* in Rosenbrock case. We believe that our models learns a transformation to latent space and reconstruction from latent space and saturates faster because of its inability to relearn the transformations. Also, because the latent data points are changed by learning the projection matrix in every iteration, *ABLR* model in the latent space is unable to model effectively.

High Dimensional BO - with Transfer Learning: Having trained the Transfer Learning models with metadata, we compared the performance of models on Quadratic and Rosenbrock functions (Figures 5.8, 5.7). *VAE-ABLR* requires more metadata and saturates within few iterations. As expected in both cases, *Projection-ABLR* is able to find a better minimum as it has already learned the projection to latent space and reconstruction to input space during meta-training. We believe that *AE-ABLR* requires much more data (similar to *VAE-ABLR*) and better hyperparameters to learn a nonlinear transformation from input space to latent space. Even when increasing the metadata 5-fold (Figures 5.9, 5.10), *AE-ABLR* does not learn the optimal transformation as it is trying to find a nonlinear transformation which may not be required based on our formulation of synthetic experiments.

6 Conclusion and Future Work

In this work, we set out to optimize an unknown, expensive to evaluate overparameterized black-box function while making use of available data from previous optimization runs on similar functions. Having formulated our problem statement in the Section 1.2, we adopted the Multi-Task Bayesian optimization framework to optimize a target task with data from other tasks as an initial source. We utilized the joint learning of latent space and probabilistic model in the Section 4.1 and proposed two variants of the latent model namely *Projection-ABLR* and *AE-ABLR*. We implemented the Multi-Task Adaptive Bayesian Linear Regression (MT-ABLR) [2] as a latent space probabilistic model to learn all tasks simultaneously. With three structures shared by all tasks, 1. Encoder/Projection matrix: Mapping from input space to latent space. 2. Decoder/Reconstruction matrix: Mapping from latent space to input space. 3. Feature learning: Common features learned from latent space for probabilistic modeling, we were able to perform Transfer Learning during Latent Bayesian Optimization 4.4 on the target task.

From the experiments (Cf. Section 5.3), we demonstrated that meta-learning of the intrinsic function is feasible with the proposed joint learning which enabled us to efficiently perform latent space BO. Also, we observed that the shared features learned from latent space resemble the common intrinsic structure. By visualizing the MT-ABLR model on latent space and input space, we note that the model is able to adapt to the new target task and joint learning aids in learning the meaningful latent space and the probabilistic model.

We ran BO experiments on high dimensional synthetic functions with low intrinsic dimensionality and compared against the baselines, *REMBO*, *VAE-BO*. The results in 5.4, without the use of metadata (no Transfer Learning aspect), show that our method achieves similar regret value to *REMBO*. As we did not perform hyperparameter optimization, we believe that our method could be improved further.

In the Transfer Learning setting, our model *Projection-ABLR* is able to achieve a lower regret than other baselines. However, *AE-ABLR* regret saturates after some iterations and we believe it is because of the higher number of parameters in the AE network than the *Projection-ABLR* model. Also, adding more data from source tasks in *Projection-ABLR* helps in faster optimization of the target task as shown in increasing metadata subsection 5.4. As we analyze the performance of our method with respect to the number of tasks and points per task, we observe that models with more metadata perform better initially than models trained with less metadata but don't end up with a considerably better performance. We believe that our model seeks to learn tasks better when more points per task are provided. Having tried out two simple approaches to estimate the intrinsic dimensionality of the problem, we note that there is scope for further improvement in it and consider it a hard problem.

Building on this work, we consider the below points as the interesting future directions worth exploring:

- A straight forward extension of our method is to replace ABLR with GPs and try to jointly learn the parameters of the GP and Auto Encoder or VAE. To improve the scalability and to take advantage of GPs, mean and variance functions can be learned by using BNNs from metadata and initialized as a prior for a target task GP instead of using a common GP for all tasks.
- Since we used a common projection matrix for all the tasks in our experiments, it would be interesting to assess our method for a task-specific projection matrix.
- One of the important limitations of our method is that it expects the intrinsic dimensionality to be known beforehand. Reasoning about intrinsic dimensionality by using metadata and meta-loss as it would lead to better modeling of the objective function. There is a need for an effective metric that reflects the meta-objective or meta-loss we want to minimize in estimating the intrinsic dimensionality.
- If we have access to more metadata, we could incorporate the VAE-based approach to learn the intrinsic structure and later use it as a generative model to generate related random tasks from the learned intrinsic latent space.

7 Appendix

The implementation details and the hyperparameters used in the experiments conducted are described here. We used PyTorch [49] to develop all our BLR-based models and GPy [50] for implementing GP-based *REMBO*. During latent BO, we keep track of the best point found at each iteration and at every tenth iteration we check if the best has remained constant over the previous 10 iterations. If so, we choose a random point on the boundary to induce random exploring behavior on boundaries.

We used Adam optimizer [44] to train Multi-task ABLR, Projection-ABLR, AE-ABLR models with cosine annealing scheduler. We list the hyperparameters of used during meta-train and target-train of our models in the Table 7.1. Initialization and bounds of precision parameters of Bayesian Linear Regression layer for Multi-task ABLR, Projection-ABLR and AE-ABLR are given in the Table 7.2. During meta-train and target-train, α and β of the corresponding task are bounded between the given mininum and maximum values.

The results plotted are the mean and variance of 20 trials initiated by random seeds. All the plots show two standard deviation (1.96) normalized by the square root of the number of trials. During meta-training, all the input values are normalized to the range $[-1, 1]$ using MinMaxScaler and the function values y are normalized using RobustScaler from sklearn [51]. Also, weights of the model with the minimum training loss are saved along with the scalers to reuse during the optimization of the target task. For 40-dimensional benchmarks with 10-dimensional intrinsic space, the architecture of Auto Encoder based models is 1 hidden layer Encoder NN with 40-20-10-20-40 units and TanH activation in the hidden layers.

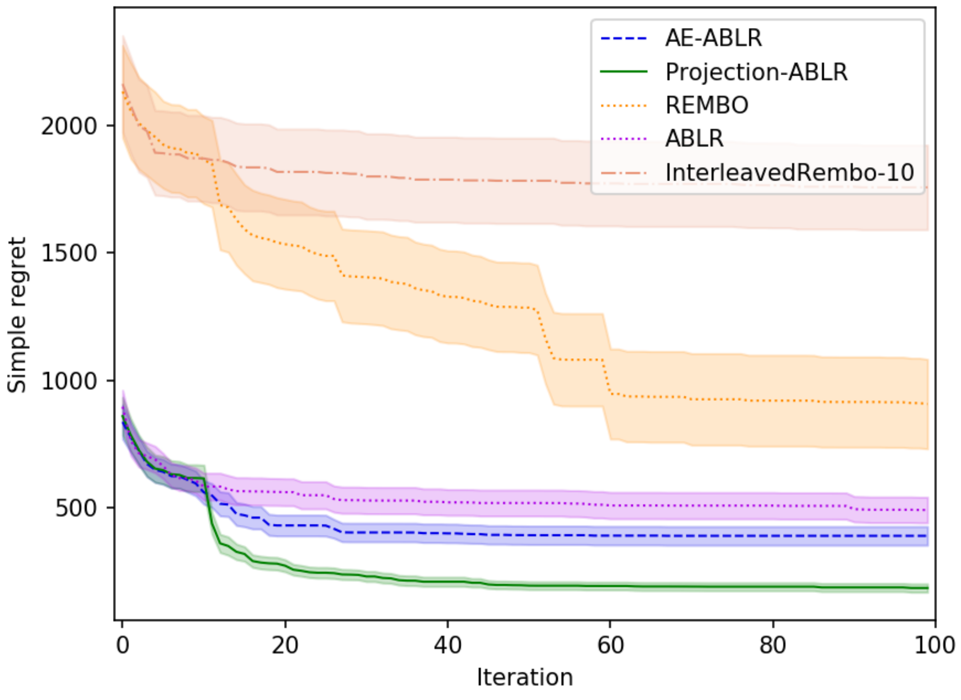
Additional Experiments

One realistic setting is to evaluate our method against a benchmark which is not intrinsically low dimensional. This case occurs if we miscalculate the effective dimensionality of the problem.

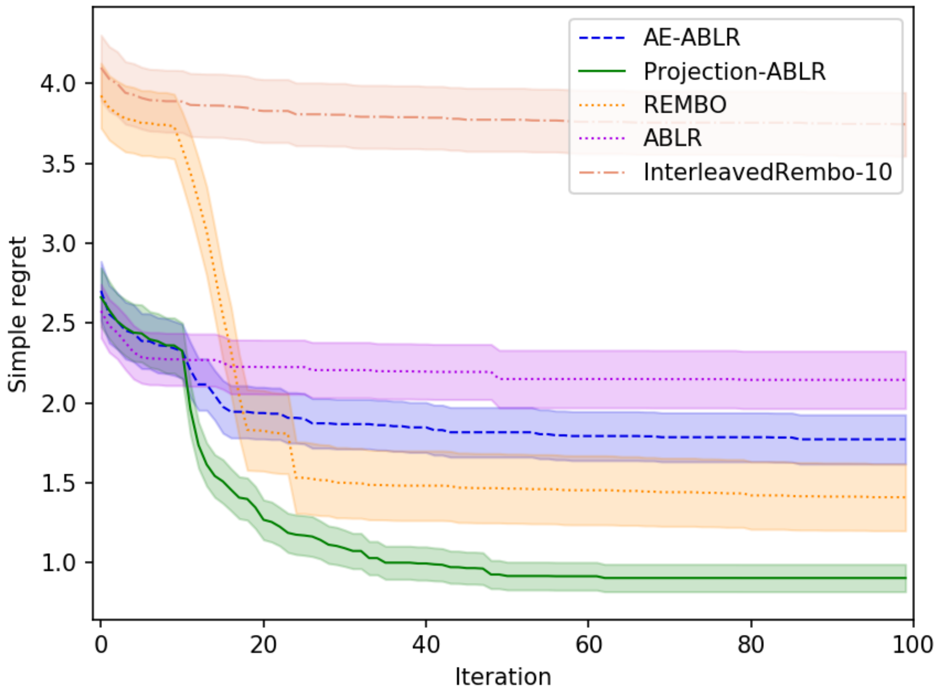
Figure 7.1 displays the result of all methods on 40-dimensional quadratic and rosenbrock functions which is actually 40 dimensional and is not located in a lower dimensional intrinsic

	Meta-train	*get-train*
Number of epochs	2048	1024
Adam: Learning rate (lr)	10^{-3}	10^{-3}
Weight decay	10^{-5}	10^{-3}
Cosine Annealing: Minimum lr	10^{-6}	10^{-5}

Table 7.1: Training hyperparameters of BLR-based models



(a) On 40-dimensional Quadratic function



(b) On 40-dimensional Rosenbrock function

Figure 7.1: Regret for different methods on 40-dimensional benchmarks with no intrinsic structure

	initialization	min	max
α_t	1	1	10^3
β_t	10^3	10^1	10^6

Table 7.2: Initialization and bounds of precision parameters of any task t

space. Latent space models, *Projection-ABLR*, *AE-ABLR*, *REMBO*, *InterleavedRembo* are run on a 20-dimensional space whereas Multi-Task *ABLR* is run on the actual 40-dimensional input space. As we see our latent *Projection-ABLR* model performing better than the input space *ABLR* model in a problem without lower dimensional structure, we emphasize the importance of maximizing acquisition function in lower dimensions.

List of Figures

List of Tables

Bibliography

[1] Z. Wang, F. Hutter, M. Zoghi, D. Matheson and N. De Freitas, "Bayesian optimization in a billion dimensions via random embeddings," *J. Artif. Int. Res.*, vol. 55, no. 1, pp. 361–387, Jan. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=3013558.3013569

[2] V. Perrone, R. Jenatton, M. W. Seeger and C. Archambeau, "Scalable hyperparameter transfer learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 6845–6855. [Online]. Available: http://papers.nips.cc/paper/7917-scalable-hyperparameter-transfer-learning.pdf

[3] A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal and S. Trimpe, "Virtual vs. Real: Trading Off Simulations and Physical Experiments in Reinforcement Learning with Bayesian Optimization," *arXiv e-prints*, p. arXiv:1703.01250, Mar 2017.

[4] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2188395

[5] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *arXiv e-prints*, p. arXiv:1610.02415, Oct 2016.

[6] R. Moriconi, M. P. Deisenroth and K. S. Sesh Kumar, "High-dimensional Bayesian optimization using low-dimensional feature spaces," *arXiv e-prints*, p. arXiv:1902.10675, Feb 2019.

[7] P. Hennig and C. J. Schuler, "Entropy search for information-efficient global optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 1809–1837, Jun. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188385.2343701

[8] J. Kirschner, M. Mutný, N. Hiller, R. Ischebeck and A. Krause, "Adaptive and Safe Bayesian Optimization in High Dimensions via One-Dimensional Subspaces," *arXiv e-prints*, p. arXiv:1902.03229, Feb 2019.

[9] C. Li, S. Gupta, S. Rana, V. Nguyen, S. Venkatesh and A. Shilton, "High Dimensional Bayesian Optimization Using Dropout," *arXiv e-prints*, p. arXiv:1802.05400, Feb 2018.

[10] J. Vanschoren, "Meta-learning," pp. 39–68.

[11] F. Hutter, L. Kotthoff and J. Vanschoren, Eds., *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2018, in press, available at http://automl.org/book.

[12] K. Swersky, J. Snoek and R. P. Adams, "Multi-task bayesian optimization," in *Proceedings of the 26th International Conference on Neural Information Processing*

Systems - Volume 2, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 2004–2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999836

[13] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 1487–1495. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098043

[14] M. Poloczek, J. Wang and P. Frazier, "Multi-information source optimization," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4288–4298. [Online]. Available: http://papers.nips.cc/paper/7016-multi-information-source-optimization.pdf

[15] B. Letham and E. Bakshy, "Bayesian Optimization for Policy Search via Online-Offline Experimentation," *arXiv e-prints*, p. arXiv:1904.01049, Apr 2019.

[16] S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi and M. Karasuyama, "Multi-fidelity Bayesian Optimization with Max-value Entropy Search," *arXiv e-prints*, p. arXiv:1901.08275, Jan 2019.

[17] Z. Wang and S. Jegelka, "Max-value Entropy Search for Efficient Bayesian Optimization," *arXiv e-prints*, p. arXiv:1703.01968, Mar 2017.

[18] K. Kandasamy, G. Dasarathy, J. B. Oliva, J. Schneider and B. Poczos, "Multi-fidelity Gaussian Process Bandit Optimisation," *arXiv e-prints*, p. arXiv:1603.06288, Mar 2016.

[19] N. Srinivas, A. Krause, S. Kakade and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 1015–1022. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104451

[20] M. Feurer, B. Letham and E. Bakshy, "Scalable Meta-Learning for Bayesian Optimization," *arXiv e-prints*, p. arXiv:1802.02219, Feb 2018.

[21] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat and R. P. Adams, "Scalable Bayesian Optimization Using Deep Neural Networks," *arXiv e-prints*, p. arXiv:1502.05700, Feb 2015.

[22] J. T. Springenberg, A. Klein, S. Falkner and F. Hutter, "Bayesian optimization with robust bayesian neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4134–4142. [Online]. Available: http://papers.nips.cc/paper/6117-bayesian-optimization-with-robust-bayesian-neural-networks.pdf

[23] R. Garnett, M. A. Osborne and P. Hennig, "Active learning of linear embeddings for gaussian processes," in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'14. Arlington, Virginia, United States: AUAI Press, 2914, pp. 230–239. [Online]. Available: http://dl.acm.org/citation.cfm?id=3020751.3020776

[24] J. Mockus, “On bayesian methods for seeking the extremum and their application,” in *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, 1977, pp. 195–200.

[25] F. Hutter, H. H. Hoos and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, ser. LION’05. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 507–523. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25566-3_40

[26] H. J. Kushner, “A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise,” *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 03 1964. [Online]. Available: https://doi.org/10.1115/1.3653121

[27] “Bayesian optimization blog,” http://krasserm.github.io/2018/03/21/bayesian-optimization/, accessed: 2019-12-31.

[28] E. Brochu, V. M. Cora and N. de Freitas, “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning,” *arXiv e-prints*, p. arXiv:1012.2599, Dec 2010.

[29] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[30] R. M. Neal, *Bayesian Learning for Neural Networks*. Berlin, Heidelberg: Springer-Verlag, 1996.

[31] D. J. C. MacKay, “A practical bayesian framework for backpropagation networks,” *Neural Comput.*, vol. 4, no. 3, pp. 448–472, May 1992. [Online]. Available: http://dx.doi.org/10.1162/neco.1992.4.3.448

[32] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[33] “Lecture slides - Random projections,” https://web.ma.utexas.edu/users/rachel/madison14.pdf, accessed: 2019-12-31.

[34] “Matrix Transformation wiki,” https://en.wikipedia.org/wiki/Transformation_matrix, accessed: 2019-12-31.

[35] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[36] “Autoencoder wiki,” https://en.wikipedia.org/wiki/Autoencoder, accessed: 2019-12-31.

[37] C. Doersch, “Tutorial on Variational Autoencoders,” *arXiv e-prints*, p. arXiv:1606.05908, Jun 2016.

[38] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010. [Online]. Available: https://doi.org/10.1109/TKDE.2009.191

[39] “Transfer Learning blog,” https://ruder.io/transfer-learning/, accessed: 2019-12-31.

[40] S. Ruder, “An Overview of Multi-Task Learning in Deep Neural Networks,” *arXiv e-prints*, p. arXiv:1706.05098, Jun 2017.

[41] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6114

[42] D. R. Jones, C. D. Perttunen and B. E. Stuckman, “Lipschitzian optimization without the lipschitz constant,” *Journal of Optimization Theory and Applications*, vol. 79, no. 1, pp. 157–181, Oct 1993. [Online]. Available: https://doi.org/10.1007/BF00941892

[43] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evol. Comput.*, vol. 9, no. 2, p. 159–195, Jun. 2001. [Online]. Available: https://doi.org/10.1162/106365601750190398

[44] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv e-prints*, p. arXiv:1412.6980, Dec 2014.

[45] H. H. Rosenbrock, “An Automatic Method for Finding the Greatest or Least Value of a Function,” *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 01 1960. [Online]. Available: https://doi.org/10.1093/comjnl/3.3.175

[46] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and S. . . Contributors, “SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python,” *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[47] B. Letham, R. Calandra, A. Rai and E. Bakshy, “Re-examining linear embeddings for high-dimensional bayesian optimization,” 2020. [Online]. Available: https://openreview.net/forum?id=SJgn3lBtwH

[48] “REMBO implementation,” https://github.com/jmetzen/bayesianoptimization, accessed: 2019-12-31.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[50] GPy, “GPy: A gaussian process framework in python,” http://github.com/SheffieldML/GPy, since 2012.

[51] 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*, vol. 12, pp. 2825–2830, 2011.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 05.01.2020 Jagan Shanmugam