FEW-SHOT BAYESIAN OPTIMIZATION WITH DEEP KERNEL SURROGATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter optimization (HPO) is a central pillar in the automation of machine learning solutions and is mainly performed via Bayesian optimization, where a parametric surrogate is learned to approximate the black box response function (e.g. validation error). Unfortunately, evaluating the response function is computationally intensive. As a remedy, earlier work emphasizes the need for *transfer learning* surrogates which learn to optimize hyperparameters for an algorithm from other tasks. In contrast to previous work, we propose to rethink HPO as a *few-shot learning* problem in which we train a shared deep surrogate model to quickly adapt (with few response evaluations) to the response function of a new task. We propose the use of a deep kernel network for a Gaussian process surrogate that is meta-learned in an end-to-end fashion in order to jointly approximate the response functions of a collection of training data sets. As a result, the novel few-shot optimization of our deep kernel surrogate leads to new state-of-the-art results at HPO compared to several recent methods on diverse metadata sets.

Many machine learning models have very sensitive hyperparameters that must be carefully tuned for efficient use. Unfortunately, finding the right setting is a tedious trial and error process that requires expert knowledge. AutoML methods address this problem by providing tools to automate hyperparameter optimization, where Bayesian optimization has become the standard for this task (Snoek et al., 2012). It treats the problem of hyperparameter optimization as a black box optimization problem. Here the black box function is the hyperparameter response function, which maps a hyperparameter setting to the validation loss. Bayesian optimization consists of two parts. First, a surrogate model, often a Gaussian process, is used to approximate the response function. Second, an acquisition function is used that balances the trade-off between exploration and exploitation. In a sequential process, hyperparameter settings are selected and evaluated, followed by an update of the surrogate model.

Recently, several attempts have been made to extend Bayesian optimization to account for a transfer learning setup. It is assumed here that historical information on machine learning algorithms is available with different hyperparameters. This can either be because this information is publicly available (e.g. OpenML) or because the algorithm is repeatedly optimized for different data sets. To this end, several transfer learning surrogates have been proposed that use this additional information to reduce the convergence time of Bayesian optimization.

We propose a new paradigm for accomplishing the knowledge transfer by reconceptualizing the process as a few-shot learning task. Inspiration is drawn from the fact that there are a limited number of black box function evaluations for a new hyperparameter optimization task (i.e. few shots) but there are ample evaluations of related black box objectives (i.e. evaluated hyperparameters on other data sets). This approach has several advantages. First, a single model is learned that is trained to quickly adapt to a new task when few examples are available. This is exactly the challenge we face when optimizing hyperparameters. Second, this method can be scaled very well for any number of considered tasks. This not only enables the learning from large metadata sets but also enables the problem of label normalization to be dealt with in a new way. Finally, we present an evolutionary algorithm that can use surrogate models to get a warm start initialization for Bayesian optimization. All of our contributions are empirically compared with several competitive methods in three different problems. Two ablation studies provide information about the influence of the individual components.

1 RELATED WORK

The idea of using transfer learning to improve Bayesian optimization is being investigated in several papers. The early work suggests learning a single Gaussian process for the entire data (Bardenet et al., 2013; Swersky et al., 2013; Yogatama & Mann, 2014). Since the training of a Gaussian process is cubic in the number of training points, this idea does not scale well. This problem has recently been addressed by several proposals to use ensembles of Gaussian processes where a Gaussian process is learned for each task (Wistuba et al., 2018; Feurer et al., 2018). This idea scales linearly in the number of tasks but still cubically in the number of training points per task. Thus, the problem persists in scenarios where there is a lot of data available for each task.

Bayesian neural networks are a possible more scalable way of learning with large amounts of data. For example, Schilling et al. (2015) propose to use a neural network with task embedding and variable interactions. To obtain mean and variance predictions, the authors propose using an ensemble of models. In contrast, Springenberg et al. (2016) use a Bayesian multi-task neural network. However, since training Bayesian neural networks is computationally intensive, Perrone et al. (2018) propose a more scalable approach. They suggest using a neural network that is shared by all tasks and using Bayesian linear regression for each task. The parameters are trained jointly on the entire data. While our work shares some similarities with the previous work, our algorithm has unique properties. First of all, a meta-learning algorithm is used, which is motivated by recent work on model-agnostic meta-learning for few-shot learning (Finn et al., 2017). This will allow us to integrate all task-specific parameters out such that the model does not grow with the number of tasks. As a result, our algorithm scales very well with the number of tasks. Second, while we are also using a neural network, we combine it with a Gaussian process with a nonlinear kernel in order to obtain uncertainty predictions.

A simpler solution for using the transfer learning idea in Bayesian optimization is initialization (Feurer et al., 2015; Wistuba et al., 2015). The standard Bayesian optimization routine with a simple Gaussian process is used in this case but it is warm-started by a number of hyperparameter settings that work well for related tasks. We also explore this idea in the context of this paper by proposing a simple evolutionary algorithm that can use a surrogate model to estimate a data-driven warm start initialization sequence. The use of an evolutionary algorithm is motivated by its ease of implementation and natural capability to deal with continuous and discrete hyperparameters.

2 PRELIMINARIES

2.1 BAYESIAN OPTIMIZATION

Bayesian optimization is an optimization method for computationally intensive black box functions that consists of two main components, the surrogate model and the acquisition function. The surrogate model is a probabilistic model with mean μ and variance σ^2, which tries to approximate the unknown black box function f, in the following also *response function*. The acquisition function can provide a score for each feasible solution based on the prediction of the surrogate model. This score balances between exploration and exploitation. The following steps are carried out sequentially. First, the feasible solution that maximizes the acquisition function is evaluated. In this way a new observation $(\mathbf{x}, f(\mathbf{x}))$ is obtained. Then, the surrogate model is updated based on the entirety of all observations $\mathcal{D}$. This sequence of steps is carried out until a previously defined convergence criterion occurs (e.g. exhaustion of the time budget). In Figure 1 we provide an example how Bayesian optimization is used to maximize a sine wave.

Expected Improvement (Jones et al., 1998) is one of the most commonly used acquisition functions and will also be used in all our experiments. It is defined as

$$a(\mathbf{x}|\mathcal{D}) = \mathbb{E}\left[\max\left\{f(\mathbf{x}) - y_{\text{max}}, 0\right\}\right] \ , \tag{1}$$

where y_{max} is the largest observed value of f.

2.2 GAUSSIAN PROCESSES

In the regression problem we seek to model the relationship between a dependent variable f_i and multiple independent variables $\mathbf{x}_i$ by a function

$$f : \mathcal{X} \to \mathbb{R} \ . \tag{2}$$

For Gaussian processes (Rasmussen & Williams, 2006) we consider f_i to be a random variable and the joint distribution of all f_i is assumed to be multivariate Gaussian distributed:

$$p\left(\mathbf{f} \,|\mathbf{X}, \boldsymbol{\theta}\right) = \mathcal{N}\left(\mathbf{f}|m\left(\mathbf{X}\right), k\left(\mathbf{X}, \mathbf{X}|\boldsymbol{\theta}\right)\right) \ . \tag{3}$$

A Gaussian process is completely specified by its mean function m, its covariance function k, and may depend on parameters $\boldsymbol{\theta}$. A common choice is to set $m(\mathbf{x}_i) = 0$. At inference time for instances $\mathbf{x}_*$, the assumption is that their ground truth $\mathbf{f}_*$ is jointly Gaussian with $\mathbf{f}$

$$p\left(\mathbf{f}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*, \boldsymbol{\theta}\right) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \middle| \mathbf{0}, \begin{pmatrix} \mathbf{K}_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right) \ , \tag{4}$$

where

$$\mathbf{K}_n = \mathbf{K} + \sigma_n^2 \mathbf{I}, \ \mathbf{K}_* = k\left(\mathbf{X}, \mathbf{X}_*|\boldsymbol{\theta}\right), \ \mathbf{K}_{**} = k\left(\mathbf{X}_*, \mathbf{X}_*|\boldsymbol{\theta}\right) \tag{5}$$

for brevity. In this notation, σ_n^2 is a noise hyperparameter added to the diagonal of the kernel matrix $\mathbf{K} = k\left(\mathbf{X}, \mathbf{X}|\boldsymbol{\theta}\right)$. Then, the posterior predictive distribution has mean and covariance

$$\mathbb{E}\left[f_*\right] = \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}, \ \mathrm{cov}\left[f_*\right] = \mathbf{K}_{**} - \mathbf{K}^T \mathbf{K}^{-1} \mathbf{K}_* \tag{6}$$

Examples for covariance functions are the linear or squared exponential kernel. However, these kernel functions are designed by hand. The idea of *deep kernel learning* (Wilson et al., 2016) is to learn the kernel function. They propose to use a neural network φ to transform the input $\mathbf{x}$ to a latent representation which serves as input for the kernel function.

$$k_{\text{deep}}(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}, \mathbf{w}) = k(\varphi(\mathbf{x}, \mathbf{w}), \varphi(\mathbf{x}', \mathbf{w})|\boldsymbol{\theta}) \tag{7}$$

3 FEW-SHOT BAYESIAN OPTIMIZATION

Hyperparameter optimization is traditionally either tackled as an optimization problem without prior knowledge about the response function or alternatively as a multi-task or transfer learning problem. In the former, every search basically starts from scratch. In the latter, one or multiple models are trained that attempt to reuse knowledge of the source tasks for the target task.

In this work we will address the problem as a few-shot problem. Given T related source tasks and very few examples of the target task, we want to make reliable predictions. For each of the source tasks we have observations $\mathcal{D}^{(t)} = \{(\mathbf{x}_i^{(t)}, y_i^{(t)})\}_{i=1\ldots n^{(t)}}$, where $\mathbf{x}_i^{(t)}$ is a hyperparameter setting and $y_i^{(t)}$ is the noisy observations of $f^{(t)}(\mathbf{x}_i^{(t)})$, i.e. a validation score of a machine learning model on data set t. In the following we will denote the set of all data points by $\mathcal{D} := \bigcup_{t=1}^{T} \mathcal{D}^{(t)}$. Model-agnostic meta-learning (Finn et al., 2017) has become a popular choice for few-shot learning and we propose to use an adaption for Gaussian processes (Patacchiola et al., 2019) as a surrogate model within the Bayesian optimization framework. The idea is simple. A deep kernel φ is used to learn parameters across tasks such that all its parameters $\boldsymbol{\theta}$ and $\mathbf{w}$ are task-independent. All task-dependent parameters are kept separate which allows to marginalize its corresponding variable out when solely optimizing for the task-independent parameters. If we assume that the posteriors over $\boldsymbol{\theta}$ and $\mathbf{w}$ are dominated by their respective maximum likelihood estimates $\hat{\boldsymbol{\theta}}$ and $\hat{\mathbf{w}}$, we can approximate the posterior predictive distribution by

$$p\left(f_*|\mathbf{x}_*, \mathcal{D}\right) = \int p\left(f_*|\mathbf{x}_*, \boldsymbol{\theta}, \mathbf{w}\right) p\left(\boldsymbol{\theta}, \mathbf{w}|\mathcal{D}\right) \mathrm{d}\boldsymbol{\theta}, \mathbf{w} \approx p\left(f_*|\mathbf{x}_*, \mathcal{D}, \hat{\boldsymbol{\theta}}, \hat{\mathbf{w}}\right) \tag{8}$$

We estimate $\hat{\boldsymbol{\theta}}$ and $\hat{\mathbf{w}}$ by maximizing the log marginal likelihood for all tasks.

$$\log p\left(\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(T)}|\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(T)}, \boldsymbol{\theta}, \mathbf{w}\right) = \sum_{t=1}^{T} \log p\left(\mathbf{f}^{(t)}|\mathbf{X}^{(t)}, \boldsymbol{\theta}, \mathbf{w}\right) \tag{9}$$

$$\propto - \sum_{t=1}^{T} \left(\mathbf{f}^{(t)^{\mathrm{T}}} \mathbf{K}_n^{(t)^{-1}} \mathbf{f}^{(t)} + \log \left|\mathbf{K}_n^{(t)}\right|\right) \tag{10}$$

Algorithm 1: Few-Shot GP Surrogate

Input: Learning rates α and β, training data $\mathcal{D}$, kernel k, and neural network φ.
while *not done* **do**
  Sample task $t \sim \mathcal{U}(\{1, \ldots, T\})$;
  Estimate l and u (Equation 11);
  for b_n *times* **do**
    Sample batch $\mathcal{B} = \{(\mathbf{x}_i, y_i)\}_{i=1,\ldots,b} \sim \mathcal{D}^{(t)}$ and scale labels $\mathbf{y}$ using l and u;
    Compute marginal likelihood $\mathcal{L}$ on $\mathcal{B}$. (Equation 9);
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}$;
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \nabla_{\mathbf{w}} \mathcal{L}$;
  end
end

We maximize the marginal likelihood with stochastic gradient ascent. Each batch contains data for one task. It is important to note that this batch data does not correspond to the data in $\mathcal{D}^{(t)}$, but rather a subset of it. With small batch sizes, this is an efficient way to train the Gaussian process, regardless of how many tasks knowledge is transferred from or how large $\mathcal{D}^{(t)}$ is.

The training works as follows. In each iteration, a task t is sampled uniformly at random from all T tasks. Then we sample a batch of training instances uniformly at random from $\mathcal{D}^{(t)}$. Finally, we calculate the log marginal likelihood for this batch (Equation 9) and update the kernel parameters with one step in the direction of the gradient.

In hyperparameter optimization, we are only interested in the hyperparameter setting that works best according to a predefined metric of interest. Therefore, only the ranking of the hyperparameter settings is important, not the actual value of the metric. In practice, however, the minimum / maximum score and the range of values differ significantly between the tasks. The most common way to address this problem of label normalization is to z-normalize the labels or to scale them to $[0, 1]$ per data set. However, this does not fully solve the problem. The main problem is that this label normalization must also be applied to the target task. This is not possible with a satisfactory degree of accuracy, especially when only a few examples are available, since the approximate values for minimum / maximum or mean / standard deviation are insufficiently accurate.

Since our proposed method scales easily to any number of tasks, we can afford to consider another option. We propose a task augmentation strategy that addresses the label normalization issue by randomly scaling the labels for each batch. Since y_{min} and y_{max} are the minimum and maximum values for all T tasks, we can generate augmented tasks for each batch $\mathcal{B} = \{\mathbf{x}_i, y_i\}_{i=1,\ldots,b} \sim \mathcal{D}^{(t)}$, where b is the batch size, as follows. A lower and upper limit is sampled for each sample batch,

$$l \sim \mathcal{U}(y_{\text{min}}, y_{\text{max}}), u \sim \mathcal{U}(y_{\text{min}}, y_{\text{max}}) \ , \tag{11}$$

such that $l < u$ holds. Then the labels for that batch are scaled to this range,

$$\mathbf{y} \leftarrow \frac{\mathbf{y} - l}{u - l} \ . \tag{12}$$

No further changes are required. We summarize this procedure in Algorithm 1. The idea here is to learn a representation that is invariant with respect to various offsets and ranges so that the target task does not require normalization. This strategy is not possible for other hyperparameter optimization methods, since this either increases the training data set size even further and thus becomes computationally impossible (Bardenet et al., 2013; Swersky et al., 2013; Yogatama & Mann, 2014) or thousands of new tasks would have to be generated, which is also is not practical (Springenberg et al., 2016; Perrone et al., 2018; Wistuba et al., 2018; Feurer et al., 2018).

At test time, the posterior predictive distribution is computed according to Equation 6. In practice, the target data set can be very different to the learned prior. For this reason, the deep kernel parameters are fine-tuned on the target task's data for few epochs. We discuss the usefulness of this step in more detail when we discuss the empirical results.

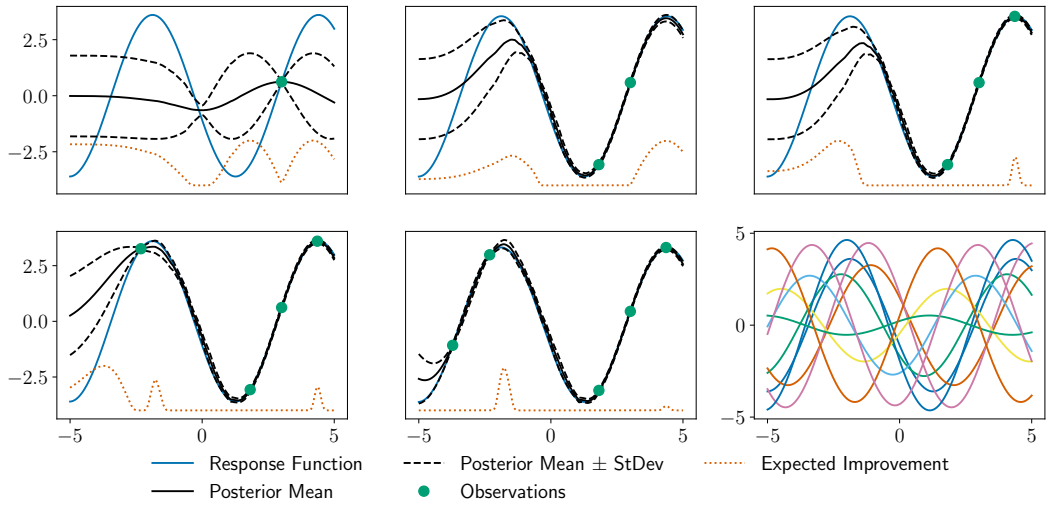

Figure 1: Demonstration of five steps of Bayesian Optimization with FSBO for maximizing a sine wave (blue). One maximum is discovered within only three steps. Expected Improvement has been scaled to improve readability. In black the predictions of the surrogate model. Bottom right are examples of the source tasks. The deep kernel consists of a spectral kernel (Wilson & Adams, 2013) combined with a two-layer neural network ($1 \to 64 \to 64$).

4 A Motivating Example

We would like to first motivate our method, FSBO, with an example, similar to the one described in Nichol et al. (2018). The aim is to find the argument x that maximizes a one-dimensional sine wave of the form $f^{(t)}(x) = a^{(t)} \sin(x + b^{(t)})$. It is not known that the function is a sine wave and that one only has to determine the amplitude a and phase b. However, information about other sine waves is available to the optimizer which are considered to be related. These source tasks $t = 1, \dots, T$ are generated randomly with $a \sim \mathcal{U}(0.1, 5)$ and $b \sim \mathcal{U}(0, 2\pi)$. 50 examples $(x_i^{(t)}, f^{(t)}(x_i^{(t)}))$ are available for each task t, whereby the points $x_i^{(t)} \in [-5, 5]$ are evenly spaced.

It should be noted at this point that the expected value for each x_i is 0. Thus, training on the data described above leads to a model predicting 0 for every x_i. However, the used meta-learning procedure allows for accurately reconstructing the underlying sine wave after only a few examples (see Figure 1). Provided that the response function of the original task is similar to the target task, this is a very promising technique for accelerating the search for good hyperparameter settings.

5 A Data-Driven Initialization

The proposed few-shot surrogate model requires only a few examples before reliable predictions can be made for the target data set. How do you choose the initial hyper parameter settings? The simplest idea would be to pick them at random or use Latin Hypercube Sampling (LHS) (McKay et al., 1979). Since we already have data from various tasks and a surrogate model that can impute missing values, we propose a data-driven warm start approach. If we evaluate the performance of a hyperparameter optimization algorithm with a loss function $\mathcal{L}$, we use an evolutionary algorithm to find a set with I hyperparameter settings which minimizes this loss on the source tasks.

$$\mathbf{X}^{(\text{init})} = \underset{\mathbf{X} \in \mathcal{X}^I}{\arg\min} \sum_{t=1}^{T} \mathcal{L}\left(f^{(t)}, \mathbf{X}\right) \tag{13}$$

The loss of a set of hyperparameters depends on the response function values for each of the elements. Since these are not available for every $\mathbf{x}$, we approximate $f^{(t)}(\mathbf{x})$ with the prediction of the surrogate model described in the last section whenever this is necessary. The evolutionary algorithm

works as follows. We initialize the population with sets containing I random settings, the settings being sampled in proportion to their performance according to the predictions. The best sets are then selected to be refined using an evolutionary algorithm. The algorithm randomly chooses either to mutate a set or to perform a crossover operation between two sets. When mutating a set, a setting is removed uniformly at random and a new setting is added proportional to its predicted performance. The crossover operation creates a new set and adds elements from the union of the parent sets until the new set has I settings. The new set is added to the population. After 100,000 steps, the algorithm is stopped and the best set is used as the set of initial hyperparameter settings. In our experiment we use the normalized regret as a loss function which is defined in the next section. In Figure 3 we compare this warm start initialization with the simple random or LHS initialization.

6 EXPERIMENTS

We used three different optimization problems to compare the different hyperparameter optimization methods: AdaBoost, GLMNet, and SVM. We created the GLMNet and SVM metadata set by downloading the 30 data sets with the most reported hyperparameter settings from OpenML for each problem. The AdaBoost data set is publicly available (Wistuba et al., 2018). The number of settings per data set varies, the number of settings across all tasks for the GLMNet problem is approximately 800,000. See the appendix for more details about the metadata sets. We compare the following list of hyperparameter optimization methods.

Random Search This is a simple but strong baseline for hyperparameter optimization (Bergstra & Bengio, 2012). Hyperparameters settings are selected uniformly at random from the search space.

Gaussian Process (GP) Bayesian optimization with a Gaussian process as a surrogate model is a standard and strong baseline (Snoek et al., 2012). We use a Matérn 5/2 kernel with ARD and rely on the scikit-optimize implementation. We compare with two variations. The first is the vanilla method, which uses Latin Hypercube Sampling (LHS) with design size of 10 to initialize. The second method uses the warm start (WS) method described in Section 5 to find an initial set of 5 settings to use the knowledge from other data sets.

MetaBO Using the acquisition function instead of the surrogate model for transfer learning is another option. MetaBO is the state-of-the-art transfer acquisition function and was presented last year at ICLR (Volpp et al., 2020). We use the implementation provided by the authors.

ABLR The state-of-the-art surrogate model for scalable hyperparameter transfer learning (Perrone et al., 2018). This method uses a multi-task GP which combines a linear kernel with a neural network and scales to very large data sets. Transfer learning is achieved by sharing the neural network parameters across different tasks.

Multi-Head GPs This closely resembles ABLR but uses the same deep kernel as our proposed method. The main difference from our proposed method is that it is using a GP for every task, only shares the neural network across tasks, and uses standard stochastic gradient ascent.

Few-Shot Bayesian Optimization (FSBO) Our proposed method described in Section 3. The deep kernel is composed of a two-layer neural network ($128 \rightarrow 128$) with ReLU activations and a squared-exponential kernel. We use the Adam optimizer with learning rate 10^{-3} and a batch size of fifty. The warm start length is five.

The experiments are repeated ten times and evaluated in a leave-one-task-out cross-validation. This means that all transfer learning methods use one task as the target task and all other tasks as source tasks. For AdaBoost we use the same train/test split as used by Volpp et al. (2020) instead. We report the aggregated results for all tasks within one problem class with respect to the mean of normalized regrets. The normalized regret for a task is obtained by first scaling the response function values between 0 and 1 before calculating the regret (Wistuba et al., 2018).

6.1 HYPERPARAMETER OPTIMIZATION

We conduct experiments on three different metadata sets and report the aggregated mean of normalized regrets in Table 1. The best results are in bold. Results that are not significantly worse than the best are in italics. We determined the significance using the Wilcoxon signed rank test with a confi-

Method	**AdaBoost**			**GLMNet**			**SVM**		
	15	33	50	33	67	100	33	67	100
RANDOM	*4.87*	3.02	2.16	0.85	0.40	0.29	2.01	1.12	0.83
GP (LHS)	3.87	2.49	2.23	1.32	1.01	0.73	1.94	1.40	1.14
GP (WS)	*3.25*	*1.66*	*1.02*	*0.86*	0.36	0.24	*1.10*	0.81	0.65
METABO	5.27	3.52	1.96	11.02	10.97	10.96	12.39	12.39	11.93
ABLR	*4.56*	1.44	1.24	1.77	0.50	0.40	1.81	1.23	0.84
MULTI-HEAD GPS (WS)	6.78	3.45	1.80	2.83	2.80	2.68	11.20	9.82	8.72
FSBO	**3.10**	**1.13**	**0.80**	**0.42**	**0.22**	**0.16**	**0.79**	**0.51**	**0.36**

Table 1: FSBO obtains better results for all hyperparameter optimization problems. The best results are in **bold**. Results that are not significantly worse than the best are in *italics*. Used initialization in parentheses, (LHS) - Latin Hypercube Sampling, (WS) - Warm Start.

dence level of 95%. Results are reported after every 33 trials of Bayesian optimization for GLMNet and SVM. Since AdaBoost has fewer test examples, we report its results in shorter intervals.

Our proposed FSBO method outperforms all other transfer methods in all three tasks. Results are significantly better but in the case of AdaBoost where GP (WS) achieves similar but on average worse results. The results obtained for MetaBO are the worst. We contacted Volpp et al. (2020) and they confirmed that our results are obtained correctly. The problem is apparently that the Reinforcement Learning method does not work well for larger number of trials. We provide a longer discussion in the appendix. Also ABLR and the very related Multi-Head GP are not performing very well. As we show in the next section, one possible reason for this might be because the neural network parameters are fixed which will prohibit a possibly required adaptation to the target task. The vanilla GP and its transfer variant that uses a warm start turn out to be among the strongest baselines. This is in particular true for the warm start version which is the second best method. This simple baseline of knowledge transfer is often not taken into account when comparing transfer surrogates, although it is easy to implement. Due to the very competitive results, we recommend using it as a standard baseline to assess the usefulness of new transfer surrogates.

6.2 COMPONENT CONTRIBUTIONS

In this section we highlight the various components that make a significant difference to ABLR. The results for each setup are reported in Figure 2. The Multi-Head GP is a surrogate model that shares the neural network between tasks but uses a separate Gaussian process for each task. It closely resembles the idea of ABLR but is closer to our proposed implementation of FSBO. Starting from this configuration, we add various components that will eventually lead to our proposed model FSBO. We consider Multi-Head GP (WS), a version that additionally uses the warm start method described in Section 5 instead of a random initialization. Multi-Head GP (fine-tune) not only updates the kernel parameters when a new observation is received but also fine-tunes the parameters of the neural network. Finally, FSBO is our proposed method, which uses only one Gaussian process for all tasks. We see that all Multi-Head GP versions have a problem adapting to the target tasks efficiently. Fine-tuning the deep kernel is an important part of learning FSBO. Although FSBO outperforms all Multi-Head GP versions without fine-tuning, the additional use of it makes for a significant improvement. We analyzed the reason for this in more detail. We observed that the learned neural network provides a strong prior that leads to strong exploitation behavior. Fine-tuning prevents this and thus ensures that the method does not get stuck in local optima.

6.3 DATA-DRIVEN WARM START

The use of a warm start is not the main contribution of this paper but it is interesting to explore further nonetheless. First of all, for some methods this is the only differentiating factor. We compare our suggested warm start initialization with a random and a Latin hypercube sampling (LHS) initialization in Figure 3. Considering that GP (WS) always performed better than GP (LHS) in Table 1, one would expect that the warm start itself also performs better than LHS. While this is the case for GLMNet and SVM, there are no significant differences for the AdaBoost problem. Our assumption

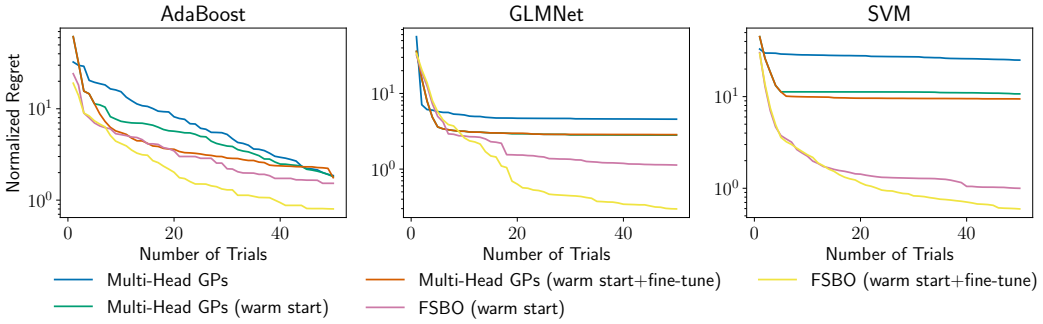

Figure 2: Comparison of the contribution of the various FSBO components to the final solution. Each component makes its own orthogonal contribution.

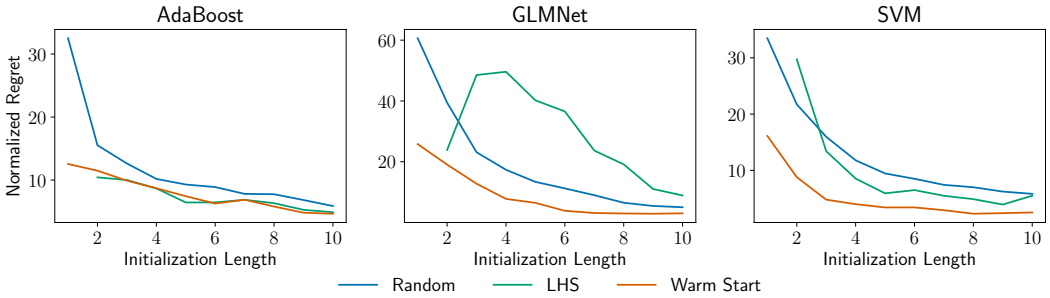

Figure 3: The warm start initialization yields the best results on GLMNet and SVM for all initialization lengths. For AdaBoost it is comparable to LHS.

is that in this case the warm start might not have always found good settings as part of the initialization, it still explored areas in the search space close to the optimum. This would facilitate finding a better solution in one of the subsequent steps of Bayesian optimization.

7 Conclusions

In this work, we propose to rethink hyperparameter optimization as a few-shot learning problem in which we train a shared deep surrogate model to quickly adapt (with few response evaluations) to the response function of a new task. We propose the use of a deep kernel network for a Gaussian process surrogate that is meta-learned in an end-to-end fashion in order to jointly approximate the response functions of a collection of training data sets. This few-shot surrogate model is used for two different purposes. First, we use it in combination with an evolutionary algorithm in order to estimate a data-driven warm start initialization for Bayesian optimization. Second, we use it directly for Bayesian optimization. In our empirical evaluation on three hyperparameter optimization problems, we observe significantly better results than with state-of-the-art methods that use transfer learning.

References

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 199–207, 2013. URL `http://proceedings.mlr.press/v28/bardenet13.html`.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012. URL `http://dl.acm.org/citation.cfm?id=2188395`.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pp. 1128–1135, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029.

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization. *CoRR*, abs/1802.02219, 2018. URL http://arxiv.org/abs/1802.02219.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1126–1135, 2017. URL http://proceedings.mlr.press/v70/finn17a.html.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. Global Optimization*, 13(4):455–492, 1998. doi: 10.1023/A:1008306431147. URL https://doi.org/10.1023/A:1008306431147.

M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2): 239–245, 1979. ISSN 00401706. URL http://www.jstor.org/stable/1268522.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018.

Massimiliano Patacchiola, Jack Turner, Elliot J. Crowley, and Amos Storkey. Deep kernel transfer in gaussian processes for few-shot learning. *arXiv preprint arXiv:1910.05199*, 2019.

Valerio Perrone, Rodolphe Jenatton, Matthias W. Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 6846–6856, 2018. URL http://papers.nips.cc/paper/7917-scalable-hyperparameter-transfer-learning.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

Nicolas Schilling, Martin Wistuba, Lucas Drumond, and Lars Schmidt-Thieme. Hyperparameter optimization with factorized multilayer perceptrons. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II*, pp. 87–103, 2015. doi: 10.1007/978-3-319-23525-7_6. URL https://doi.org/10.1007/978-3-319-23525-7_6.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 2960–2968, 2012. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4134–4142, 2016. URL http://papers.nips.cc/paper/6117-bayesian-optimization-with-robust-bayesian-neural-networks.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2004–2012, 2013. URL http://papers.nips.cc/paper/5086-multi-task-bayesian-optimization.

Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *8th International Conference on Learning Representations, ICLR 2020, Addis*

Ababa, Ethiopia, April 26-30, 2020, 2020. URL https://openreview.net/forum?id=ryeYpJSKwr.

Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 1067–1075, 2013. URL http://proceedings.mlr.press/v28/wilson13.html.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pp. 370–378, 2016. URL http://proceedings.mlr.press/v51/wilson16.html.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, pp. 1–10, 2015. doi: 10.1109/DSAA.2015.7344817. URL https://doi.org/10.1109/DSAA.2015.7344817.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Mach. Learn.*, 107(1):43–78, 2018. doi: 10.1007/s10994-017-5684-y. URL https://doi.org/10.1007/s10994-017-5684-y.

Dani Yogatama and Gideon Mann. Efficient transfer learning method for automatic hyperparameter tuning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pp. 1077–1085, 2014. URL http://proceedings.mlr.press/v33/yogatama14.html.

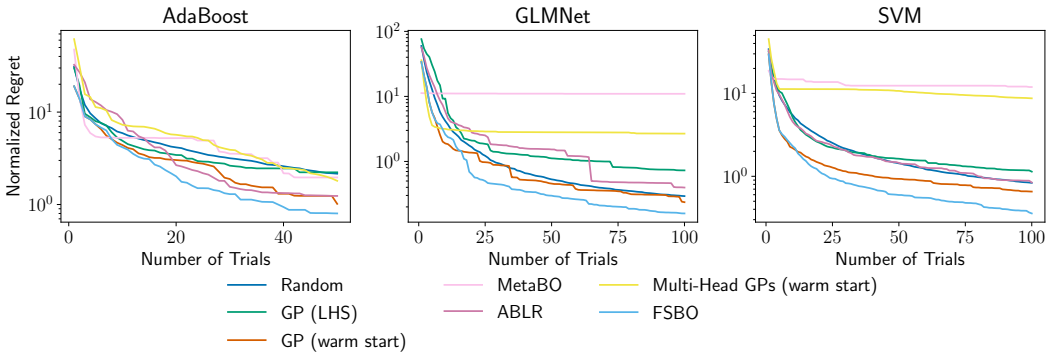

Figure 4: FSBO is the best of all considered methods.

A Hyperparameter Optimization

In the main paper we report the normalized regret after every 33 trials of Bayesian optimization in a table. This allows us to also report statistical significance. In Figure 4 we show results for all 100 trials. The conclusions remain unchanged. FSBO provides consistently the best results.

B Challenges with MetaBO

The results reported in the main paper for MetaBO seem not to align with the numbers reported by Volpp et al. (2020), in particular for AdaBoost. In order to understand this behavior, we conducted a deeper analysis for the AdaBoost metadata set which was used in the evaluation of Volpp et al. (2020) as well. Using the authors' code[1], we executed the same scripts used by the authors to report their results for $T = 15$ trials. Furthermore, we changed two lines in the scripts to train MetaBO for $T = 50$, the number of trials considered for AdaBoost in our experiment. We report the outcome of these two experiments in Figure 5. In the two left plots we show how the learned policies for $T = 15$ and $T = 50$ perform on validation and test during training. The authors' default setting for the total number of iterations is 2000. However, we noticed a sudden drop in regret for the setting $T = 50$ around 2000 iterations. For that reason we assumed that further training might help to further improve MetaBO and increased it to 5000 iterations. As can be seen in the middle plot, that was not the case. We observe that the policy learned for $T = 15$ has a lower regret than the one for $T = 50$ even though it is only trying 15 compared to 50 trials. In the right plot we show the results compared to a simple random search baseline. The results for $T = 15$ are in line with those reported by Volpp et al. (2020). We contacted the authors and they confirmed that we use their code correctly. Their explanation is that the increase of episode length makes this a harder problem for Reinforcement Learning. This results in worse results for $T = 50$ and even worse for $T = 100$ in case of the GLMNet and SVM problem. They proposed to randomly vary T between 5 and 50 during the training phase. We also considered this training protocol and report the results in Figure 5. This improves over training only with $T = 50$ but does not improve over a simple random search.

C Metadata

We created the GLMNet and SVM metadata using OpenML. We chose the 30 OpenML flows with the most observations. We also limited ourselves to the uploaded results from user with ID 2702 (OpenML_Bot R), who uploaded the majority of all runs, to ensure that the metadata was generated under similar circumstances. In some cases the choice of a single hyperparameter is not indicated. In such a case, we assume that the default value was used.

We further use the AdaBoost metadata which has been used to evaluate MetaBO (Volpp et al., 2020). It was created by running a grid search on 50 different classification data sets. Compared to the

[1] `https://github.com/boschresearch/MetaBO`

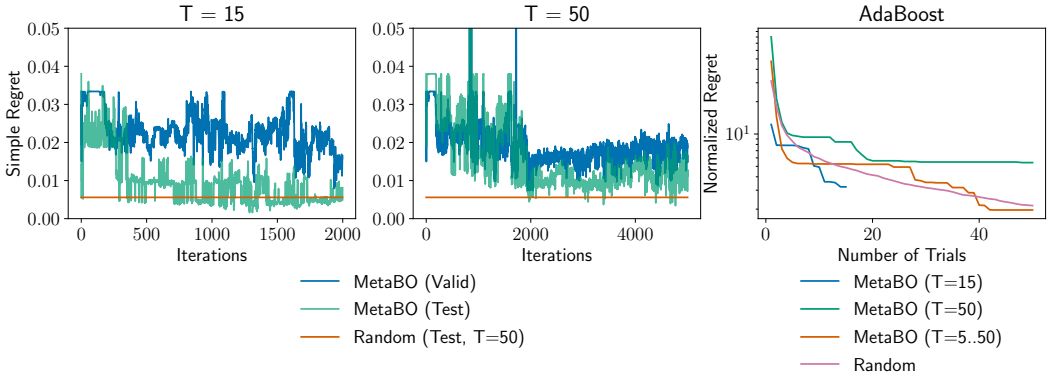

Figure 5: Left two plots: Training MetaBO for different number of trials T. Right plot: comparing the final policy against a random search. Results for $T = 15$ match the results reported by Volpp et al. (2020).

Method	**AdaBoost**			**GLMNet**			**SVM**		
	15	33	50	33	67	100	33	67	100
FSBO	**3.10**	**1.13**	**0.80**	*0.42*	0.22	*0.16*	**0.79**	**0.51**	**0.36**
FSBO (REPTILE)	*3.80*	2.13	*1.21*	**0.36**	**0.19**	**0.14**	1.48	0.89	0.72

Table 2: Comparing different model-agnostic meta-learning approaches. Again, the best results are in **bold** and results that are not significantly worse than the best are in *italics*.

other two metadata sets, this one is significantly smaller. Its use is mainly motivated by comparing to MetaBO under the same circumstances as used to evaluate MetaBO. For that reason, we do not use the leave-one-data-set-out cross-validation protocol but instead use the same fixed split created by Volpp et al. (2020).

Statistics about these metadata sets are provided in Table 3.

D META-LEARNING WITH REPTILE

In principle our few-shot surrogate can be combined with any model-agnostic meta-learning approach. In this section we compare our proposed meta-learning approach against Reptile (Nichol et al., 2018), a first-order approximation of MAML. We use the same hyperparameters and data augmentation as described in Algorithm 1. Reptile introduces a new hyperparameter, an outer learning rate. We set it to 0.1 and decay it linearly to 0. As summarized in Table 2, FSBO with the meta-learning approach described in Algorithm 1 is either better or not significantly worse than its variation with Reptile.

AdaBoost		**GLMNet**		**SVM**	
DATA SET	RUNS	OPENML ID	RUNS	OPENML ID	RUNS
A9A	108	3	12796	37	2429
W8A	108	31	60721	3485	1000
ABALONE	108	37	17950	3492	3217
APPENDICITIS	108	219	13963	3493	1828
AUSTRALIAN	108	3492	16968	3494	1854
AUTOMOBILE	108	3493	19931	3889	1951
BANANA	108	3889	13942	3891	3083
BANDS	108	3891	13917	3899	2487
BREAST-CANCER	108	3899	15954	3902	1105
BUPA	108	3903	39155	3903	1018
CAR	108	3913	22949	3913	1448
CHESS	108	3917	16960	3918	1459
COD-RNA	108	3918	20984	3950	1998
COIL2000	108	9914	37635	6566	2702
COLON-CANCER	108	9946	25532	9889	2485
CRX	108	9952	25854	9914	1498
DIABETES	108	9967	21994	9946	1291
ECOLI	108	9978	19956	9952	2483
GERMAN-NUMER	108	9980	31961	9967	1492
HABERMAN	108	9983	37134	9971	1381
HOUSEVOTES	108	10101	66277	9976	1473
IJCNN1	108	125923	35555	9978	1503
KR-VS-K	108	145847	29927	9980	2943
LED7DIGIT	108	145857	31180	9983	987
LETTER	108	145862	12954	10101	1871
LYMPHOGRAPHY	108	145872	17835	14951	1765
MAGIC	108	145953	47439	34536	1499
MONK-2	108	145972	19972	34537	1489
PENDIGITS	108	145979	12979	145878	1244
PHONEME	108	146064	43785	146064	1435
PIMA	108				
RING	108				
SAHEART	108				
SEGMENT	108				
SEISMIC	108				
SHUTTLE	108				
SONAR-SCALE	108				
SPAMBASE	108				
SPECTFHEART	108				
SPLICE	108				
TIC-TAC-TOE	108				
TITANIC	108				
TWONORM	108				
USPS	108				
VEHICLE	108				
WDBC	108				
WINE	108				
WINEQUALITY-RED	108				
WISCONSIN	108				
YEAST	108				
TOTAL	5400	TOTAL	804159	TOTAL	54418

Table 3: Metadata set statistics. OpenML ID refers to a task on openml.org.