META-LEARNING ACQUISITION FUNCTIONS FOR TRANSFER LEARNING IN BAYESIAN OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Transferring knowledge across tasks to improve data-efficiency is one of the open key challenges in the area of global optimization algorithms. Readily available algorithms are typically designed to be universal optimizers and, thus, often suboptimal for specific tasks. We propose a novel transfer learning method to obtain customized optimizers within the well-established framework of Bayesian optimization, allowing our algorithm to utilize the proven generalization capabilities of Gaussian processes. Using reinforcement learning to meta-train an acquisition function (AF) on a set of related tasks, the proposed method learns to extract implicit structural information and to exploit it for improved data-efficiency. We present experiments on a sim-to-real transfer task as well as on several simulated functions and two hyperparameter search problems. The results show that our algorithm (1) automatically identifies structural properties of objective functions from available source tasks or simulations, (2) performs favourably in settings with both scarse and abundant source data, and (3) falls back to the performance level of general AFs if no particular structure is present.

1 INTRODUCTION

Global optimization of black-box functions is highly relevant for a wide range of real-world tasks. Examples include the tuning of hyperparameters in machine learning, the identification of control parameters or the optimization of system designs. Such applications oftentimes require the optimization of relatively low-dimensional ($\lesssim 10D$) functions where each function evaluation is expensive in either time or cost. Furthermore, there is typically no gradient information available.

In this context of data-efficient global black-box optimization, Bayesian optimization (BO) has emerged as a powerful solution (Mockus, 1975; Brochu et al., 2010; Snoek et al., 2012; Shahriari et al., 2016). BO's data efficiency originates from a probabilistic surrogate model which is used to generalize over information from individual data points. This model is typically given by a Gaussian process (GP), whose well-calibrated uncertainty prediction allows for an informed exploration-exploitation trade-off during optimization. The exact manner of performing this trade-off, however, is left to be encoded in an acquisition function (AF). There is wide range of AFs available in the literature which are designed to yield universal optimization strategies and thus come with minimal assumptions about the class of target objective functions.

To achieve optimal data-efficiency on new instances of previously seen tasks, however, it is crucial to incorporate the information obtained from these tasks into the optimization. Therefore, transfer learning (or warm-starting) is an important and active field of research. Indeed, in many practical applications, optimizations are repeated numerous times in similar settings, underlining the need for specialized optimizers. Examples include hyperparameter optimization which is repeatedly done for the same machine learning model on varying datasets or the optimization of control parameters for a given system with varying physical configurations.

Following recent approaches (Swersky et al., 2013; Feurer et al., 2018; Wistuba et al., 2018), we argue that it is beneficial to perform transfer learning for global black-box optimization in the framework of BO to retain the proven generalization capabilities of its underlying GP surrogate model. To not restrict the expressivity of this model, we propose to implicitly encode the task structure in a specialized AF, i.e., in the optimization strategy. We realize this encoding via a novel method which meta-learns a neural AF, i.e., a neural network representing the AF, on a set of training tasks. The meta-training is performed using reinforcement learning, making the proposed approach applicable to the standard BO setting, where we do not assume access to objective function gradients.

Our contributions are (1) a novel transfer learning method allowing the incorporation of implicit structural knowledge about a class of objective functions into the framework of BO through learned neural AFs to increase data-efficiency on new task instances, (2) an automatic and practical meta-learning procedure for training such neural AFs which is fully compatible with the black-box optimization setting, i.e, not requiring gradients, and (3) the demonstration of the efficiency and practical applicability of our approach on a challenging hardware control task, hyperparameter optimization problems, as well as a set of synthetic functions.

2 Related Work

The general idea of improving the performance or convergence speed of a learning system on a given set of tasks through experience on similar tasks is known as learning to learn, meta-learning or transfer learning and has attracted a large amount of interest in the past while remaining an active field of research (Schmidhuber, 1987; Hochreiter et al., 2001; Thrun and Pratt, 1998; Lake et al., 2016).

In the context of meta-learning optimization, a large body of literature revolves around learning local optimization strategies. One line of work focuses on learning improved optimizers for the training of neural networks, e.g., by directly learning update rules (Bengio et al., 1991; Runarsson and Jonsson, 2000) or by learning controllers for selecting appropriate step sizes for gradient descent (Daniel et al., 2016). Another direction of research considers the more general setting of replacing the gradient descent update step by neural networks which are trained using either reinforcement learning (Li and Malik, 2016; 2017) or in a supervised fashion (Andrychowicz et al., 2016; Metz et al., 2019). Finn et al. (2017), Nichol et al. (2018), and Flennerhag et al. (2019) propose approaches for initializing machine learning models through meta-learning to be able to solve new learning tasks with few gradient steps.

We are currently aware of only one work tackling the problem of meta-learning global black-box optimization (Chen et al., 2017). In contrast to our proposed method, the authors assume access to gradient information and choose a supervised learning approach, representing the optimizer as a recurrent neural network operating on the raw input vectors. Based on statistics of the optimization history accumulated in its memory state, this network directly outputs the next query point. In contrast, we consider transfer learning applications where gradients are typically not available.

A number of articles address the problem of increasing BO's data-efficiency via transfer learning, i.e., by incorporating data from similar optimizations on *source tasks* into the current *target task*. A range of methods accumulate all available source and target data in a single GP and make the data comparable via a ranking algorithm (Bardenet et al., 2013), standardization or multi-kernel GPs (Yogatama and Mann, 2014), multi-task GPs (Swersky et al., 2013), the GP noise model (Theckel Joy et al., 2016), or by regressing on prediction biases (Shilton et al., 2017). These approaches naturally suffer from the cubic scaling behaviour of GPs, which can be tackled for instance by replacing the GP model, e.g., with Bayesian neural networks (Springenberg et al., 2016) with task-specific embedding vectors or adaptive Bayesian linear regression (Perrone et al., 2018) with basis expansions shared across tasks via a neural network. Other approaches retain the GP surrogate model and combine individual GPs for source and target tasks in an ensemble model with the weights adjusted according to the GP uncertainties (Schilling et al., 2016), dataset similarities (Wistuba et al., 2016), or estimates of the GP generalization performance on the target task (Feurer et al., 2018). Similarly, Golovin et al. (2017) form a stack of GPs by iteratively regressing onto the residuals w.r.t. the most recent source task. In contrast to our proposed method, many of these approaches rely on hand-engineered dataset features to measure the relevance of source data for the target task. Such features have also been used to pick promising initial configurations for BO (Feurer et al., 2015a;b;c).

The method being closest in spirit and capability to our approach is proposed by Wistuba et al. (2018). It is similar to the aforementioned ensemble techniques with the important difference that the source and target GPs are not combined via a surrogate model but via a new AF, the so-called transfer acquisition function (TAF). This AF is defined to be a weighted superposition of the predicted improvements according to the source GPs and the expected improvement according to the target GP. The weights are adjusted either according to the GP's uncertainty prediction (mixture-of-experts, TAF-ME) or using a ranking-based approach (TAF-R). Viewed in this context, our method also combines knowledge from source and target tasks in a new AF. Our weighting is determined in a meta-learning phase and can automatically be regulated during the optimization on the target task to adapt on-line to the specific objective function at hand. Furthermore, our method does not store and evaluate many source GPs during optimization as the knowledge from the source datasets is encoded directly in the network weights of the learned neural AF. This allows MetaBO to incorporate large amounts of source data while the applicability of TAF is restricted to a comparably small number of source tasks.

3 Preliminaries

We are aiming to find a global optimum $\mathbf{x}^* \in \arg\max_{\mathbf{x}\in\mathcal{D}} f(\mathbf{x})$ of some unknown bounded real-valued objective function $f : \mathcal{D} \to \mathbb{R}$ on the compact domain $\mathcal{D} \subset \mathbb{R}^D$. The only means of acquiring information about f is via (possibly noisy) evaluations at points in $\mathcal{D}$. Thus, at each optimization step $t \in \{1, 2, \dots\}$, the optimizer has to decide for the iterate $\mathbf{x}_t \in \mathcal{D}$ based on the *optimization history* $\mathcal{H}_t \equiv \{\mathbf{x}_i, f(\mathbf{x}_i)\}_{i=1}^{t-1}$. In particular, the optimizer does not have access to gradient information. To assess the performance of global optimization algorithms, it is natural to use the *simple regret* $R_t \equiv f(\mathbf{x}^*) - f\left(\mathbf{x}_t^+\right)$ where $\mathbf{x}_t^+$ is the best input vector found by an algorithm up to and including step t. The proposed method relies on the framework of BO and is trained using reinforcement learning. Therefore, we give a short overview of the notation used in both frameworks.

Bayesian Optimization In Bayesian optimization (BO) (Shahriari et al., 2016), one specifies a prior belief about the objective function f and at each step t builds a probabilistic surrogate model conditioned on the current optimization history $\mathcal{H}_t$. Typically, a Gaussian process (GP) (Rasmussen and Williams, 2005) is employed as surrogate model in which case the resulting posterior belief about f is given in terms of a *mean function* $\mu_t(\mathbf{x}) \equiv \mathbb{E}\{f(\mathbf{x}) \mid \mathcal{H}_t\}$ and a *variance function* $\sigma_t^2(\mathbf{x}) \equiv \mathbb{V}\{f(\mathbf{x}) \mid \mathcal{H}_t\}$, for which closed-form expressions are available. To determine the next iterate $\mathbf{x}_t$ based on the belief about f given $\mathcal{H}_t$, a sampling strategy is defined in terms of an *acquisition function* (AF) $\alpha(\cdot \mid \mathcal{H}_t) : \mathcal{D} \to \mathbb{R}$. The AF outputs a score value at each point in $\mathcal{D}$ such that the next iterate is defined to be given by $\mathbf{x}_t \in \arg\max_{\mathbf{x}\in\mathcal{D}} \alpha(\mathbf{x} \mid \mathcal{H}_t)$. The strength of the resulting optimizer is largely based upon carefully designing the AF to trade-off exploration of unknown versus exploitation of promising areas in $\mathcal{D}$.

Reinforcement Learning Reinforcement learning (RL) allows an agent to learn goal-oriented behavior via trial-and-error interactions with its environment (Sutton and Barto, 1998). This interaction process is formalized as a Markov decision process: at step t the agent senses the environment's state $s_t \in \mathcal{S}$ and uses a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ to determine the next action $a_t \in \mathcal{A}$. Typically, the agent explores the environment by means of a probabilistic policy, i.e., $\mathcal{P}(\mathcal{A})$ denotes the probability measures over $\mathcal{A}$. The environment's response to a_t is the next state s_{t+1}, which is drawn from a probability distribution with density $p(s_{t+1} \mid s_t, a_t)$. The agent's goal is formulated in terms of a scalar reward $r_t = r(s_t, a_t, s_{t+1})$, which the agent receives together with s_{t+1}. The agent aims to maximize the expected cumulative discounted future reward $\eta(\pi)$ when acting according to π and starting from some state $s_0 \in \mathcal{S}$, i.e., $\eta(\pi) \equiv \mathbb{E}_\pi\left[\sum_{t=1}^T \gamma^{t-1} r_t \,\middle|\, s_0\right]$. Here, T denotes the episode length and $\gamma \in (0, 1]$ is a discount factor.

4 Meta-Learning Acquisition Functions for Bayesian Optimization

We devise a global black-box optimization method that is able to automatically identify and exploit structural properties of a given class of objective functions for improved data-efficiency. We stay within the framework of BO, enabling us to exploit the powerful generalization capabilities of a GP surrogate model. The actual optimization strategy which is informed by this GP is classically encoded in a hand-designed AF. Instead, we meta-train on a set of source tasks to replace this AF by a neural network but retain all other elements of the proven BO-loop (middle panel of Fig. 1). To distinguish the learned AF from a classical AF α, we call such a network a *neural acquisition function* and denote it by α_θ, indicating that it is parametrized by a vector θ.

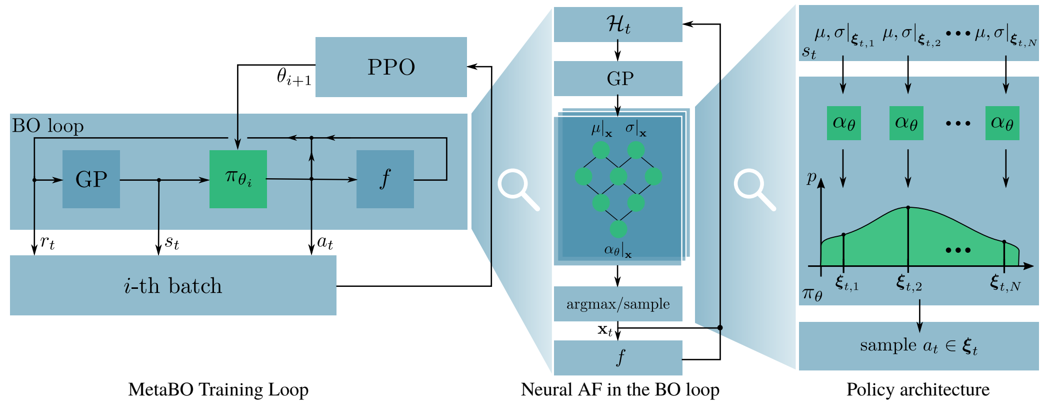

PPO
θ_{i+1}
BO loop
GP
π_{θ_i}
f
r_t
s_t
a_t
i-th batch
MetaBO Training Loop
$\mathcal{H}_t$
GP
$\mu|_{\mathbf{x}}$ $\sigma|_{\mathbf{x}}$
$\alpha_\theta|_{\mathbf{x}}$
argmax/sample
$\mathbf{x}_t$
f
Neural AF in the BO loop
$\mu,\sigma|_{\xi_{t,1}}$ $\mu,\sigma|_{\xi_{t,2}}$ ••• $\mu,\sigma|_{\xi_{t,N}}$
s_t
α_θ α_θ ••• α_θ
p
π_θ
$\xi_{t,1}$ $\xi_{t,2}$ $\xi_{t,N}$
sample $a_t \in \xi_t$
Policy architecture

Figure 1: Different levels of the MetaBO framework. Left panel: structure of the training loop for meta-learning neural AFs using RL (PPO). Middle panel: the classical BO loop with a neural AF α_θ. At test time, there is no difference to classical BO, i.e., $\mathbf{x}_t$ is given by the $\arg\max$ of the AF output. During training, the AF corresponds to the RL policy evaluated on an adaptive set $\boldsymbol{\xi}_t$ in the optimization domain. The outputs are interpreted as logits of a categorical distribution from which the actions $a_t = \mathbf{x}_t$ are sampled. This sampling procedure is detailed in the right panel.

As inputs to the neural AF we use the pointwise GP posterior prediction, i.e., $\alpha_\theta(\mathbf{x}) = \alpha_\theta[\mu_t(\mathbf{x}), \sigma_t(\mathbf{x})]$. This architecture allows learning a scalable neural AF that can be evaluated at arbitrary points $\mathbf{x} \in \mathcal{D}$ and which can be used as a plug-in feature in any state-of-the-art BO framework. In particular, if smooth activation functions are chosen, a neural AF constitutes a smooth mapping $\mathcal{D} \to \mathbb{R}$, allowing to utilize gradient-based optimization strategies to find its maximum when used in the BO loop during evaluation. We further emphasize that after the training phase the resulting neural AF is fully defined, i.e., there is no need to calibrate any hyperparameters.

Let $\mathcal{F}$ be the class of objective functions for which we want to learn a neural acquisition function α_θ. For instance, let $\mathcal{F}$ be the set of objective functions resulting from different physical configurations of a laboratory experiment or the set of loss functions used for training a machine learning model evaluated on different data sets. Often, such function classes have structure which which we aim to exploit for data-efficient optimization on further instances of related tasks. In many relevant cases, it is straightforward to obtain approximations to $\mathcal{F}$, i.e., a set of functions $\mathcal{F}'$ which capture the most relevant properties of $\mathcal{F}$ (e.g., through numerical simulations or from previous hyperparameter optimization tasks (Eggensperger et al., 2013; Wistuba et al., 2018)) but are much cheaper to evaluate. During the offline meta-training phase, the proposed algorithm makes use of such cheap approximations to identify the implicit structure of $\mathcal{F}$ and to adapt θ to obtain a data-efficient optimization strategy customized to $\mathcal{F}$.

Since transfer learning relies on the identification and comparison of relevant structure between tasks, the incorporation of additional information in the sampling strategy is crucial. In our setting, this is achieved via extending s_t by additional features to enable the neural AF to evaluate sample locations. Thus, in addition to assessing the mean $\mu_t = \mu_t|_{\mathbf{x}}$ and variance $\sigma_t = \sigma_t|_{\mathbf{x}}$ at potential sample locations, the neural AF also considers the input location $\mathbf{x}$ itself. Furthermore, the state s_t is extended by the current optimization step t and the optimization budget T, as these features can be valuable for adjusting the exploration-exploitation trade-off (Srinivas et al., 2010).

Training Procedure In the general BO setting, gradients of $\mathcal{F}$ are assumed not to be available. This is oftentimes also true for the functions in $\mathcal{F}'$, for instance, if $\mathcal{F}'$ comprises numerical simulations. Therefore, we resort to RL (PPO, (Schulman et al., 2015a)) as the meta-algorithm, as it does not require gradients of the objective functions. Tab. 1 translates the MetaBO-setting into RL parlance.

During training, the meta-algorithm requires access to the global information contained in the GP posterior prediction. Thus, the state s_t at optimization step t formally corresponds to the *entire functions* μ_t and σ_t. RL explores the state space using a parametrized stochastic policy π_θ from which the actions $a_t = \mathbf{x}_t$ are sampled, i.e., $a_t \sim \pi_\theta(\cdot \mid s_t)$. To arrive at a practical implementation, we evaluate μ_t and σ_t on a discrete set of points $\boldsymbol{\xi} \equiv \{\boldsymbol{\xi}_n\}_{n=1}^N \subset \mathcal{D}$ and feed these evaluations through the neural AF α_θ one at a time, yielding one scalar output value $\alpha_\theta(\boldsymbol{\xi}_i) = \alpha_\theta[\mu_t(\boldsymbol{\xi}_i), \sigma_t(\boldsymbol{\xi}_i)]$ for each point $\boldsymbol{\xi}_i$. These outputs are interpreted as the logits of a categorical distribution, i.e., we arrive at the policy architecture $\pi_\theta(\cdot \mid s_t) \equiv \mathrm{Cat}[\alpha_\theta(\boldsymbol{\xi}_1), \ldots, \alpha_\theta(\boldsymbol{\xi}_N)]$, cf. Fig. 1, right panel. Thus, the proposed policy evaluates the *same* neural acquisition function α_θ at arbitrarily many input points $\boldsymbol{\xi}_i$ and preferably samples points $\boldsymbol{\xi}_i$ with high $\alpha_\theta(\boldsymbol{\xi}_i)$.

Table 1: RL $\leftrightarrow$ MetaBO

RL	**MetaBO**
Policy π_θ	Neural AF α_θ
Episode	Opt. run on $f \in \mathcal{F}'$
Ep. length T	Opt. budget T
State s_t	μ_t, σ_t on $\boldsymbol{\xi}_t$, $\mathbf{x}$, t, T
Action a_t	Sampling point $\mathbf{x}_t$
Reward r_t	E.g., neg. regret $-R_t$
Trans. prob. p	Noisy eval. of f, GP-update

As for all BO approaches, calculating a sufficiently fine static set of of evaluation points $\boldsymbol{\xi}$ is challenging for higher dimensional settings. Instead, we build on the approach proposed by (Snoek et al., 2012) and continuously adapt $\boldsymbol{\xi}$ to the current state of α_θ. At each step t, α_θ is first evaluated on a static and relatively coarse Sobol grid $\boldsymbol{\xi}_{\mathrm{global}}$ spanning the whole domain $\mathcal{D}$. Subsequently, local maximizations of α_θ from the k points corresponding to the best evaluations are started. We denote the resulting set of local maxima by $\boldsymbol{\xi}_{\mathrm{local},t}$. Finally, we define the state s_t to consist of the functions μ_t and σ_t evaluated on the union $\boldsymbol{\xi} = \boldsymbol{\xi}_t \equiv \boldsymbol{\xi}_{\mathrm{local},t} \cup \boldsymbol{\xi}_{\mathrm{global}}$. The adaptive local part of this set enables the agent to exploit what it has learned so far by picking points which look promising according to the current neural AF while the static global part maintains exploration.

The final characteristics of the learned AF are controlled through the choice of reward function during meta-training. For the presented experiments we emphasized fast convergence to the optimum by setting $r_t \equiv -R_t$, i.e., the negative simple regret (or a logarithmically transformed version, $r_t \equiv -\log_{10} R_t$). This choice does not penalize explorative evaluations which do not yield and immediate improvement and serves as normalization of the functions $f \in \mathcal{F}'$. We emphasize that the knowledge about the true maximum is only needed during training and that cases in which it is not even known at training time do not limit the applicability of our method, as a cheap approximation (e.g., by evaluating the function on a coarse grid) can also be utilized.

The left panel of Fig. 1 depicts the resulting training loop graphically. The outer loop corresponds to the RL meta-training iterations, each performing a policy update step $\pi_{\theta_\mathrm{i}} \to \pi_{\theta_{i+1}}$. To approximate the gradients of the meta-training loss function, in the inner loop we record a batch of episodes, i.e., a set of (s_t, a_t, r_t)-tuples, by rolling out the current policy π_{θ_i}. At the beginning of each episode, we draw some function f from the training set $\mathcal{F}'$ and fix an optimization budget T. In each iteration of the inner loop we determine the adaptive set $\boldsymbol{\xi}_t$ and feed the state s_t, i.e., the GP posterior evaluated on this set, through the policy which yields the action $a_t = \mathbf{x}_t$. We then evaluate f at $\mathbf{x}_t$ and use the result to compute the reward r_t and to update the optimization history: $\mathcal{H}_t \to \mathcal{H}_{t+1} = \mathcal{H}_t \cup \{\mathbf{x}_t, f(\mathbf{x}_t)\}$. Finally, the GP is conditioned on the updated optimization history $\mathcal{H}_{t+1}$ to obtain the next state s_{t+1}.

5 Experiments

We trained MetaBO on a wide range of function classes and compared the performance of the resulting neural AFs with the general-purpose AF expected improvement (EI) (Mockus, 1975)[1] as well as the transfer acquisition function framework (TAF) which proved to be the current state-of-the-art solution for transfer learning in BO in an extensive experimental study (Wistuba et al., 2018). We tested both the ranking-based version (TAF-R) and the mixture-of-experts version (TAF-ME). We refer the reader to App. A for a more detailed experimental investigation of MetaBO's performance.[2]

To give quantities such as lengthscales a consistent meaning, we scaled all objective functions to an optimzation domain of unit size. If not stated differently, we report performance in terms of the median simple regret R_t over 100 optimization runs on unseen test functions as a function of

[1]We also evaluated the general-purpose AFs probability of improvement (PI) (Kushner, 1964) as well as GP-upper confidence bound (GP-UCB) (Srinivas et al., 2010) but do not present the results here to avoid clutter, as EI performed better on all our experiments.

[2]Unfortunately, we were not able to benchmark against the end-to-end approach proposed in Chen et al. (2017), as the authors could not provide the source code until the submission deadline. Furthermore, this approach is not directly applicable to our main experiments on hyperparameter optimization and simulation-to-real tasks, as gradients of the objective functions are not available in these cases.

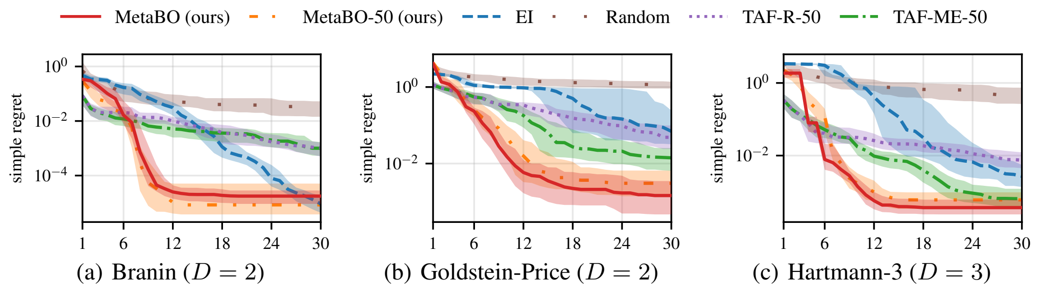


Figure 2: Performance on three global optimization benchmark functions with random translations sampled uniformly from $[-0.1, 0.1]^D$ and scalings from $[0.9, 1.1]$. For TAF, we evaluated both the ranking-based version (TAF-R-50) and the mixture-of-experts version (TAF-ME-50). Results for TAF-20 are moved to App. A, Fig. 6. MetaBO-50 outperformed EI by clear margin, especially in early stages of the optimization. After few steps to identify the objective function, MetaBO-50 also outperforms both flavors of TAF over wide ranges of the optimization budget. Note that MetaBO-50 was trained on the same set of source tasks as TAF. However, MetaBO can naturally learn from the whole set of available source tasks (MetaBO).

the optimization step t together with $30\%/70\%$ percentiles (shaded areas). We emphasize that all experiments use the same MetaBO hyperparameters, making our method easily applicable in practice. To foster reproducibility, we provide a detailed exposition of the experimental settings in App. B and make the source code of MetaBO available online.[3]

Global Optimization Benchmark Functions We evaluated our method on a set of synthetic function classes based on the standard global optimization benchmark functions Branin ($D = 2$), Goldstein-Price ($D = 2$), and Hartmann-3 ($D = 3$) (Picheny et al., 2013). To construct the training set $\mathcal{F}'$, we applied translations in $[-0.1, 0.1]$ for all dimensions as well as scalings in $[0.9, 1.1]$.

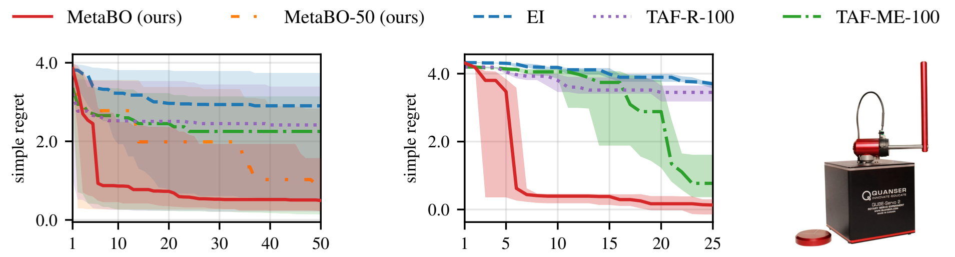
As TAF stores and evaluates one source GP for each source task, its applicability is restricted to a relatively small amount of source data. For the evaluations of TAF and MetaBO, we thus picked a set of $M = 50$ source tasks from the continuously parametrized family $\mathcal{F}'$ of available objective functions and spread these tasks uniformly over the whole range of translations and scalings (MetaBO-50, TAF-R-50, TAF-ME-50). We used $N_{\text{TAF}} = 100$ data points for each source GP of TAF. We also tested both flavors of TAF for $M = 20$ source tasks ($N_{\text{TAF}} = 50$) and observed that TAF's performance does not necessarily increase with more source data, rendering the choice of source tasks cumbersome. To avoid clutter, we moved the results for TAF-20 to App. A, cf. Fig. 6.

Fig. 2 shows the performance on unseen functions drawn randomly from $\mathcal{F}'$. MetaBO-50 outperformed EI by large margin, in particular at early stages of the optimization, by making use of the structural knowledge acquired during the meta-learning phase. Furthermore, after about 5 initial steps used to identify the objective function at hand, MetaBO-50 outperformed both flavors of TAF-50 over wide ranges of the optimization budget. This is due to its ability to learn to regulate the weighting between source and target data to adapt online to the specific objective function at hand.

We emphasize, however, that MetaBO does not require the user to choose a suitable set of source tasks but it can naturally learn from the whole set of available source tasks by randomly picking a new source task from $\mathcal{F}'$ at the beginning of each BO iteration and aggregating this information in the neural AF weights to obtain optimal optimization strategies. We also trained this full version of MetaBO (labelled MetaBO) on the global optimization benchmark functions, obtaining performance comparable with MetaBO-50. We demonstrate below that for more complex experiments, such as the simulation-to-real task, MetaBO's ability to learn from the full set of available source tasks is crucial for efficient transfer learning.

As a final test on synthetic functions, we evaluated the neural AFs on objective functions outside of the training distribution. Fig. 3 shows the number of optimization steps required by MetaBO and EI to achieve a given performance when applying translations in $[-a, a]$ and

[3]https://github.com/metabo-iclr2020/MetaBO



(a) Evaluation in simulation. (b) Evaluation on hardware in (c). (c) Exp. setup.[3]

Figure 4: Performance on a simulation-to-real task (cf. text). MetaBO and TAF used data from a cheap simulation. (a) Performance on an extended training set in simulation. (b) Transfer to the hardware depicted in (c), 10 BO runs. MetaBO learns robust neural AFs with very strong early-time performance and on-line adaption to the target objectives, reliably yielding stabilizing controllers after less than ten BO iterations while TAF-ME, TAF-R, and EI explore too heavily. We move the results for $M_{\text{TAF}} = 50$ to App. A, Fig. 8. Here MetaBO benefits from its ability to learn from the whole set of available source data, while TAF's applicability is restricted to a comparably small number of source tasks.

varying the maximal translation $a \in [0.0, 0.8]$ on the Branin function. The results show that MetaBO behaves robustly and generalizes well to functions outside of the training distribution, demonstrating that it is indeed capable of combining the knowledge about promising areas in $\mathcal{D}$ with a sophisticated sampling strategy built upon the GP surrogate model. The corresponding results on Goldstein-Price and Hartmann-3 were similar and moved to App. A (Fig. 7).

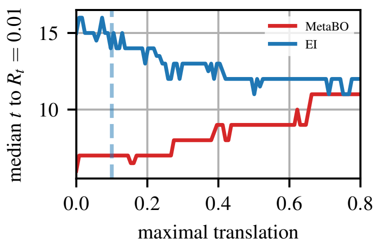


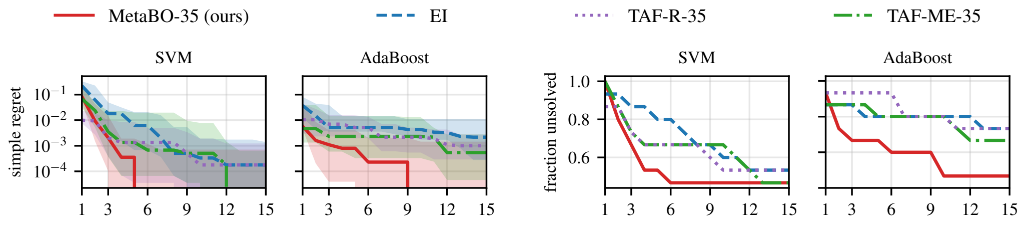
Figure 3: MetaBO generalizes robustly beyond the training distribution on the Branin function. The dashed blue line indicates the maximal translation seen during training. Additional results are moved to App. A, Fig. 7

Simulation-to-Real Task Sample efficiency is of special interest for the optimization of real world systems. In cases where an approximate model of the system can be simulated, the proposed approach can be used to improve the data-efficiency on the real system. To demonstrate this, we evaluated MetaBO on a $4D$ control task on a Furuta pendulum (Furuta et al., 1992). We applied BO to tune the four feedback gains of a linear state-feedback controller used to stabilize the pendulum in the upward equilibrium position. To assess the performance of a given controller, we employed a logarithmic quadratic cost function (Bansal et al., 2017) with a penalty term if no stabilization could be achieved. We emphasize that the cost function is rather sensitive to the control gains, resulting in a challenging black-box optimization problem.

To meta-learn the neural AF, we employed a simple numerical simulation of the Furuta pendulum which models only the most basic physical effects. The training distribution was then generated by sampling the free physical parameters (two lengths, two masses), uniformly on a range of $75\% - 125\%$ around the true parameters of the hardware (Quanser QUBE – Servo 2,[4] Fig. 4(c)). We also used this simulation to generate source tasks for TAF-50 and TAF-100 ($N_{\text{TAF}} = 200$). We move the results for TAF-50 to App. A, Fig. 8.

Fig. 4(a) show the performance on objective functions from simulation with physical parameters sampled uniformly on a wider range than during training ($10\% - 200\%$ around the true parameters). MetaBO learned robust neural AFs, combining knowledge about the class of objective functions with an on-line adaption to the target objective function based on the GP model, yielding extremely strong early-time performance and allowing further improvements over time. In contrast, TAF does not adapt the weighting of source and target tasks online to the specific objective function at hand which leads to excessive explorative behaviour on this complex class of objective functions.

[4]`https://www.quanser.com/products/qube-servo-2`



(a) Simple regret (MetaBO's performance signal). (b) Fraction of unsolved test tasks.

Figure 5: Performance on two $2D$ hyperparameter optimization tasks (SVM and AdaBoost). We trained MetaBO on precomputed data for 35 randomly chosen datasets and also used these as source tasks for TAF. The remaining 15 datasets were used for this evaluation. MetaBO learned extremely data-efficient sampling strategies on both experiments, outperforming the benchmark methods by clear margin. Note that the optimization domain is discrete and therefore tasks can be solved exactly, corresponding to zero regret.

By comparing the performance of MetaBO and MetaBO-50 in simulation, we find that on this challenging optimization problem, our architecture's ability to automatically incorporate large amounts of source data is indeed beneficial. This is underlined by the results on the hardware, on which we applied the corresponding AFs from simulation without any changes. Fig. 4(b) shows that MetaBO learned a neural AF which generalizes well from the simulated objectives to the hardware task and was thereby able to rapidly adjust to its specific properties. This resulted in very data-efficient optimization on the target system, consistently yielding stabilizing controllers after less than ten BO iterations. In comparison, the benchmark AFs required many samples to identify promising regions of the search space and therefore did not reliably find stabilizing controllers within the budget of 25 optimization steps.

Hyperparameter Optimization We also considered two hyperparameter optimization (HPO) problems on RBF-based SVMs and AdaBoost. As proposed in Feurer et al. (2018) and Wistuba et al. (2018), we used precomputed results of training these models on 50 datasets with 144 parameter configurations (RBF kernel parameter, penalty parameter C) for the SVMs and 108 configurations (number of product terms, number of iterations) for AdaBoost. We randomly split this data into 35 source datasets used for training of MetaBO as well as for TAF and evaluated the resulting optimizers on the remaining 15 datasets.[5] We emphasize that MetaBO did not use more source data than TAF in this experiment, underlining again its broad applicability in practice to application scopes with both scarse and abundant source data. The results (Fig. 5) show that MetaBO learned extremely data-efficient neural AFs which surpassed EI und TAF on both experiments.

General Function Classes Finally, we evaluated the performance of MetaBO on function classes without any particular structure structure. In such cases, it is desirable to obtain neural AFs which fall back on the performance level of general-purpose AFs such as EI. We generated the training set by sampling functions from a GP prior with squared exponential kernel with varying lengthscales. The results of our experiment show that MetaBO indeed performs at least on-par with the general-purpose AFs EI, GP-UCB and PI, cf. App. A, Fig. 9.

6 Conclusion and Future Work

We introduced MetaBO, a novel approach for transfer learning in the framework of BO. Via a flexible meta-learning approach we inject prior knowledge directly into the optimization strategy of BO using neural AFs. Our experiments on several real-world optimization tasks show that our method consistently outperforms the popular general-purpose AF EI as well as the state-of-the-art solution TAF for warmstarting BO, for instance in simulation-to-real settings or on hyperparameter search tasks. Our approach is broadly applicable to a wide range of practical problems, covering both the cases of scarse and abundant source data. The resulting neural AFs generalize well beyond the training distribution, allowing our algorithm to perform robustly unseen problems. In future work, we aim to tackle the multi-task multi-fidelity setting (Valkov et al., 2018), where we expect MetaBO's sample efficiency to be of high impact.

[5]Visualizations of the objective functions can be found on `http://www.hylap.org`

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J. Tomlin. Goal-driven Dynamics Learning via Bayesian Optimization. In *IEEE Annual Conference on Decision and Control*, pages 5168–5173, 2017.

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michèle Sebag. Collaborative Hyperparameter Tuning. In *International Conference on Machine Learning*, pages 199–207, June 2013.

Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. Learning a Synaptic Learning Rule. In *International Joint Conference on Neural Networks*, pages II–A969, 1991.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR*, abs/1012.2599, 2010.

Yutian Chen, Matthew W. Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P. Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to Learn without Gradient Descent by Gradient Descent. In *International Conference on Machine Learning*, pages 748–756, 2017.

Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning Step Size Controllers for Robust Neural Network Training. In *AAAI Conference on Artificial Intelligence*, pages 1519–1525, 2016.

Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger H Hoos, and Kevin Leyton-Brown. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, page 5, 2013.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems*, pages 2755–2763, 2015a.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Methods for Improving Bayesian Optimization for AutoML. *Journal of Machine Learning Research*, 2015b.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. *AAAI Conference on Artificial Intelligence*, pages 1128–1135, 2015c.

Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable Meta-Learning for Bayesian Optimization. *CoRR*, abs/1802.02219, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

Sebastian Flennerhag, Pablo Garcia Moreno, Neil Lawrence, and Andreas Damianou. Transferring Knowledge across Learning Processes. In *International Conference on Learning Representations*, 2019.

K. Furuta, M. Yamakita, and S. Kobayashi. Swing-up Control of Inverted Pendulum Using Pseudo-State Feedback. *Journal of Systems and Control Engineering*, 206(4):263–269, 1992.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google Vizier: A Service for Black-Box Optimization. In *International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495, 2017.

GPy. GPy: A Gaussian Process Framework in Python, 2012.

Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to Learn Using Gradient Descent. In *Artificial Neural Networks - ICANN*, pages 87–94, 2001.

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*, 86(1):97–106, March 1964.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building Machines That Learn and Think Like People. *CoRR*, abs/1604.00289, 2016.

Ke Li and Jitendra Malik. Learning to Optimize. *CoRR*, abs/1606.01885, 2016.

Ke Li and Jitendra Malik. Learning to Optimize Neural Nets. *CoRR*, abs/1703.00441, 2017.

Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Learning Unsupervised Learning Rules. In *International Conference on Learning Representations*, 2019.

J. Mockus. On Bayesian Methods for Seeking the Extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404, 1975.

Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *arXiv:1803.02999 [cs]*, March 2018. arXiv: 1803.02999.

Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cedric Archambeau. Scalable Hyperparameter Transfer Learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 6845–6855, 2018.

Victor Picheny, Tobias Wagner, and David Ginsbourger. A Benchmark of Kriging-based Infill Criteria for Noisy Optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, September 2013.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

T. P. Runarsson and M. T. Jonsson. Evolution and Design of Distributed Learning Rules. In *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pages 59–63, May 2000.

Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Scalable Hyperparameter Optimization with Products of Gaussian Process Experts. *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 33–48, September 2016.

Jurgen Schmidhuber. *Evolutionary Principles in Self-Referential Learning. On Learning how to Learn: The Meta-Meta-Meta...-Hook.* Diploma Thesis, Technische Universitat Munchen, Germany, May 1987.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR*, abs/1506.02438, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2016.

Alistair Shilton, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Regret Bounds for Transfer Learning in Bayesian Optimisation. In *International Conference on Artificial Intelligence and Statistics*, volume 54, pages 307–315, April 2017.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, pages 2960–2968, 2012.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian Optimization with Robust Bayesian Neural Networks. In *Advances in Neural Information Processing Systems*, page 9, 2016.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias W. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *International Conference on Machine Learning*, pages 1015–1022, 2010.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning - an Introduction*. Adaptive computation and machine learning. MIT Press, 1998.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-Task Bayesian Optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012, 2013.

Tinu Theckel Joy, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Flexible Transfer Learning Framework for Bayesian Optimisation. In *Advances in Knowledge Discovery and Data Mining*, April 2016.

Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

Lazar Valkov, Rodolphe Jenatton, Fela Winkelmolen, and Cédric Archambeau. A simple transfer-learning extension of Hyperband. *NeurIPS Workshop on Meta-Learning*, page 6, 2018.

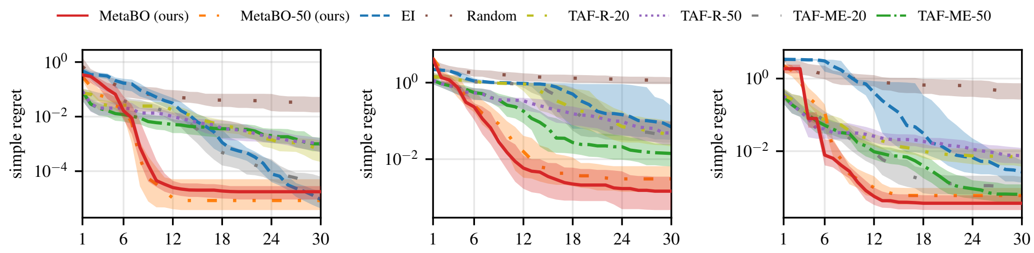
Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-Stage Transfer Surrogate Model for Automatic Hyperparameter Optimization. In *European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 199–214, 2016.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable Gaussian Process-based Transfer Surrogates for Hyperparameter Optimization. *Journal of Machine Learning Research*, 107(1):43–78, January 2018.

Dani Yogatama and Gideon Mann. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In *International Conference on Artificial Intelligence and Statistics*, pages 1077–1085, April 2014.

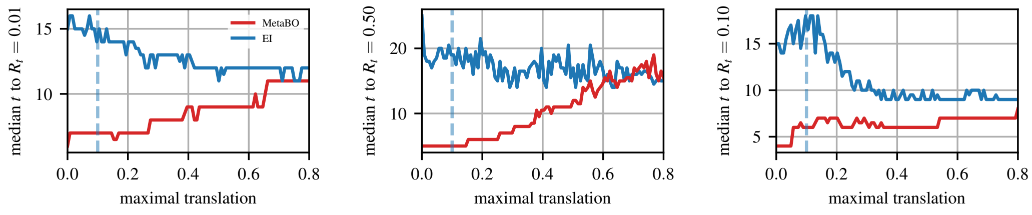
A Additional Experimental Results

Global Optimization Benchmark Functions We provide the full set of results for the experiment on the global optimization benchmark functions. In Fig. 6 we also include results for TAF with $M = 20$, showing that TAF's performance does not necessarily increase with more source data. Furthermore, we present the full set of results of the experiment investigating the generalization capabilities of MetaBO to functions outside of the training distribution (Fig. 7).



(a) Branin ($D = 2$) (b) Goldstein-Price ($D = 2$) (c) Hartmann-3 ($D = 3$)

Figure 6: Performance on three global optimization benchmark functions with random translations sampled uniformly from $[-0.1, 0.1]^D$ and scalings from $[0.9, 1.1]$. For TAF, we evaluated both the ranking-based version (TAF-R-50) and the mixture-of-experts version (TAF-ME-50). MetaBO-50 outperformed EI by clear margin, especially in early stages of the optimization. After few steps to identify the objective function, MetaBO-50 also outperforms both flavors of TAF over wide ranges of the optimization budget. Note that MetaBO-50 was trained on the same set of source tasks as TAF. However, MetaBO can naturally learn from the whole set of available source tasks (MetaBO).

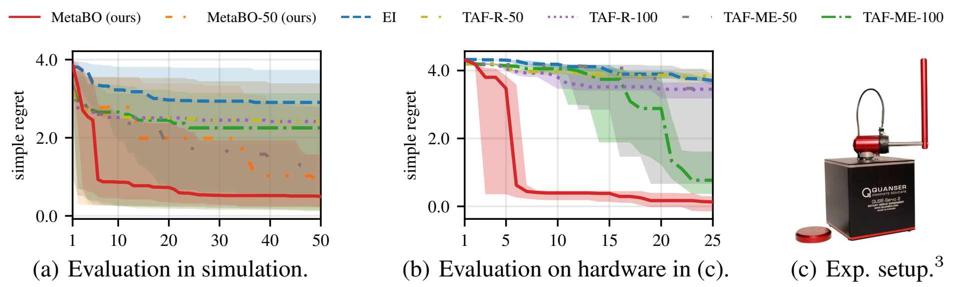


(a) Branin function ($2D$) (b) Goldstein-Price function ($2D$) (c) Hartmann-3 function ($3D$)

Figure 7: Investigation of the robustness and generalization capabilities of neural AFs with respect to changes in the distribution of objective functions. The neural AFs were trained on global optimization benchmark functions with random translations sampled uniformly from $[-a, a]^D$ and scalings from $[0.9, 1.1]$. The value a of the maximal translation along each axis used during training is indicated by a vertical blue dashed line. During evaluation, we varied the ranges of translations and report the median number of steps the neural AFs required to achieve a given simple regret and compare this with EI. 100 optimization runs were performed for each value of a. We observe robust behavior of the neural AFs, outperforming EI by large margin for small to medium values of a and performing at least on-par with EI for larger values of a. Note that EI's performance does not necessarily have to be constant, as, for instance, local optima may be pushed outside of the optimization domain for larger values of a.

Simulation-to-Real Experiment We provide the full set of results for the experiment on the global optimization benchmark functions, including the results for TAF-50, cf. Fig. 8.

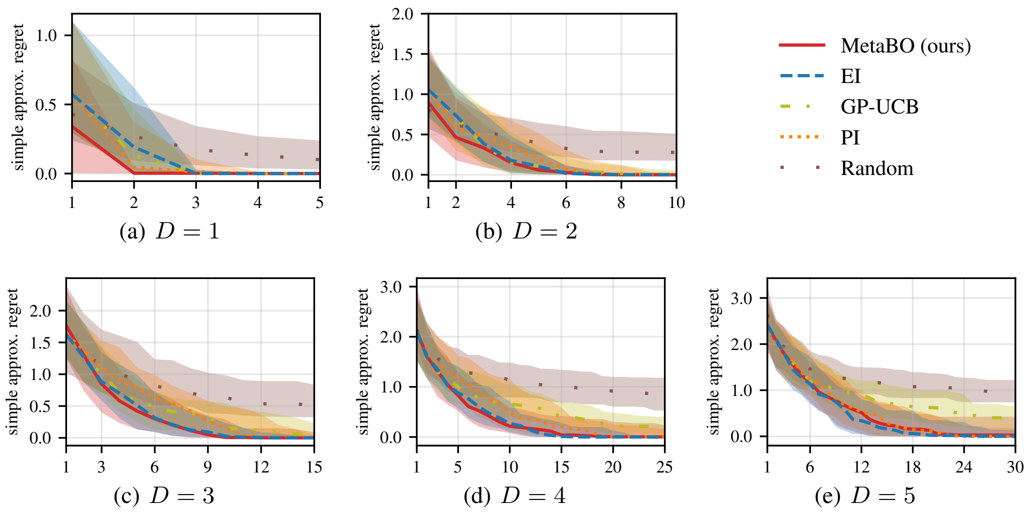
General Function Classes We investigated MetaBO's performance on general function classes without any specific structure except some correlation lengthscale. We present this experiment as a sanity check to show that MetaBO falls back at least on the performance of general-purpose optimization strategies which is to be expected when there is no special structure present in the objective functions which could be exploited in a transfer learning setting. We sampled such objective functions from a GP prior with squared-exponential kernel with lengthscales drawn uniformly from



(a) Evaluation in simulation. (b) Evaluation on hardware in (c). (c) Exp. setup.[3]

Figure 8: Performance on a simulation-to-real task (cf. text). MetaBO and TAF used data from a cheap simulation. (a) Performance on an extended training set in simulation. (b) Transfer to the hardware depicted in (c), 10 BO runs. MetaBO learns robust neural AFs with very strong early-time performance and on-line adaption to the target objectives, reliably yielding stabilizing controllers after less than ten BO iterations while TAF-ME, TAF-R, and EI explore too heavily. Here MetaBO benefits from its ability to learn from the whole set of available source data, while TAF's applicability is restricted to a comparably small number of source tasks.

$\ell \in [0.1, 1.0]$. The results (Fig. 9) show that MetaBO is capable of learning neural AFs which perform slightly better than or on on-par with the benchmark AFs on these general function classes.



(a) $D = 1$ (b) $D = 2$ (c) $D = 3$ (d) $D = 4$ (e) $D = 5$

Figure 9: Performance on functions sampled from a GP prior with lengthscales sampled uniformly from $\ell \in [0.1, 1.0]$ for different dimensions D. We trained one single dimensionality-agnostic neural AF on the 5-dimensional task and evaluated it for different dimensionalities without retraining. MetaBO performs slightly better than or on par with the benchmark AFs on this function class.

B Experimental Details

To foster reproducibility, we provide a detailed explanation of the settings used in our experiments and make source code available online.[6]

B.1 General Implementation Details

In what follows, we explain all hyperparameters used in our experiments and summarize them in Tab. 2. We emphasize that we used the same MetaBO hyperparameters for all our experiments, making our method easily applicable in practice.

Gaussian Process Surrogate Models We used the implementation GPy (GPy, 2012) with squared-exponential kernels with automatic relevance determination and a Gaussian noise model and tuned the corresponding hyperparameters (noise variance, kernel lengthscales, kernel signal variance) off-line by fitting a GP to the objective functions in the training and test sets using type-2 maximum likelihood. We also used the resulting hyperparameters for the source GPs of TAF. We emphasize that our method is fully compatible with other (on-line) hyperparameter optimization techniques, which we did not use in our experiments to arrive at a consistent and fair comparison with as few confounding factors as possible.

Baseline AFs As is standard, we used the parameter-free version of EI. For TAF, we follow Wistuba et al. (2018) and evaluate both the ranking-based as well as the product-of-experts versions. We detail the specific choices for the number of source tasks M and the number of datapoints N_{TAF} contained in each source GP in the main part of this paper.

For EI we used the midpoint of the optimization domain $\mathcal{D}$ as initial design. For TAF we did not use an initial design as it utilizes the information contained in the source tasks to warmstart BO. Note that MetaBO also works without any initial design.

Maximization of the AFs Our method is fully compatible with any state-of-the-art method for maximizing AFs. In particular, if smooth activation functions are used, our neural AFs can be optimized using gradient-based techniques. We chose to switch off any confounding factors related to AF maximization and used a hierarchical gridding approach for all evaluations as well as during training of MetaBO. For the experiments with continuous domains $\mathcal{D}$, i.e. all experiments except the HPO task, we first put a multistart Sobol grid with N_{MS} points over the whole optimization domain and evaluated the AF on this grid. Afterwards, we implemented local searches from the k maximal evaluations via centering k local Sobol grids with N_{LS} points, each spanning one unit cell of the uniform grid, around the k maximal evaluations. The AF maximum is taken to be the maximal evaluation of the AF on these k Sobol grids. For the HPO task, the AF maximum can be determined exactly, as the domain is discrete.

Reinforcement Learning Method We use the trust-region policy gradient method Proximal Policy Optimization (PPO) (Schulman et al., 2017) as the algorithm to train the neural AF.

Reward Function If the true maximum of the objective functions is not known at training time, we compute R_t with respect to an approximate maximum and define the reward to be given by $r_t \equiv -R_t$. This is the case for the experiment on general function classes (GP samples) where we used grid search to approximate the maximum as well as for the simulation-to-real task on the Furuta pendulum where we used the performance of a linear quadratic regulator (LQR) controller as an approximate maximum. For the experiments on the global optimization benchmark functions as well as on the HPO tasks, we do know the exact value of the global optimum. In these cases, we use a logarithmic transformation of the simple regret, i.e., $r_t = -\log_{10} R_t$ as the reward signal. Note that we also consistently plot the logarithmic simple regret in our evaluations for these cases.

Neural AF Architecture We used multi-layer perceptrons with relu-activation functions and four hidden layers with 200 units each to represent the neural AFs.

[6] `https://github.com/metabo-iclr2020/MetaBO`

Table 2: Parameters of the MetaBO framework used in our experiments.

Description	**Value in experiments**
BO/AF parameters	
Cardinality N_{MS} of multistart grid	
Branin, Goldstein-Price	1000
Hartmann-3	2000
Sim2Real	10000
GPs ($D = 1, 2, 3, 4, 5$)	$500, 1000, 2000, 3000, 4000$
Cardinality N_{LS} of local search grid	N_{MS}
Number k of multistarts	5
MetaBO parameters	
Cardinality of $\boldsymbol{\xi}_{\text{global}}$	N_{MS}
Cardinality of $\boldsymbol{\xi}_{\text{local}}$	k
Neural AF architecture	200 - 200 - 200 - 200, relu activations
PPO parameters (Schulman et al., 2017)	
Batch size	1200
Number of epochs	4
Number of minibatches	20
Adam learning rate	$1 \cdot 10^{-4}$
CPI-loss clipping parameter	0.15
Value network architecture	same as for neural AF
Value coefficient in loss function	1.0
Entropy coefficient in loss function	0.01
Discount factor γ	0.98
GAE-λ (Schulman et al., 2015b)	0.98

Value Function Network To reduce the variance of the gradient estimates for PPO, a value function $V_\pi(s_t)$, i.e., an estimator for the expected cumulative reward from state s_t, can be employed (Schulman et al., 2015b). In this context, the optimization step t and the budget T are particularly informative features, as for a given sampling strategy on a given function class they allow quite reliable predictions of future regrets. Thus, we propose to use a separate neural network to learn a value function of the form $V_\pi(s_t) = V_\pi(t, T)$. We used the same network architecture to learn the value functions as we used for the neural AFs.

Computation Time For training MetaBO, we employed ten parallel CPU-workers to record the data batches and one GPU to perform the policy updates. Depending on the complexity of the objective function evaluations, training a neural AF for a given function class took between approximately $30\,\text{min}$ and $10\,\text{h}$ on this moderately complex architecture.

B.2 Experiment-specific Details

We shortly list some additional details specific to the experiments presented in this article.

Simulation-to-real Task The task was to stabilize a Furuta pendulum (Furuta et al., 1992) for $5\,\text{s}$ around the upper equilibrium position using a linear state-feedback controller. If the controller was not able to stabilize the system or if the voltage applied to the motor exceeded some safety limit, we added a penalty term to the cost function proportional to the remaining time the pendulum would have had to be stabilized for successfully completing the task.

The numerical simulation we used to train MetaBO was based on the nonlinear dynamics equations of the Furuta pendulum and did only contain the most basic physical effects. In particular, effects like friction and stiction were not modeled.

As the true maximum of the objective functions is not known, we used the cost accumulated by a linear quadratic regulator (LQR) controller as an approximate maximum to compute the simple regret.

Hyperparameter Optimization Tasks We performed 7-fold cross validation on the training datasets to determine at which iteration to stop the meta-training.

General Function Classes We determined approximate maxima on the objective functions sampled from a GP prior via grid search.