Experienced Optimization with Reusable Directional Model for Hyper-Parameter Search

Yi-Qi Hu, Yang Yu, Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{huyq, yuy, zhouzh}@lamda.nju.edu.cn

Abstract

Hyper-parameter selection is a crucial yet difficult issue in machine learning. For this problem, derivative-free optimization has been playing an irreplaceable role. However, derivative-free optimization commonly requires a lot of hyper-parameter samples, while each sample could have a high cost for hyper-parameter selection due to the costly evaluation of a learning model. To tackle this issue, in this paper, we propose an experienced optimization approach, i.e., learning how to optimize better from a set of historical optimization processes. From the historical optimization processes on previous datasets, a directional model is trained to predict the direction of the next good hyper-parameter. The directional model is then reused to guide the optimization on learning new datasets. We implement this mechanism within a state-of-the-art derivative-free optimization method SRACOS, and conduct experiments on learning the hyper-parameters of heterogeneous ensembles and neural network architectures. Experimental results verify that the proposed approach can significantly improve the learning accuracy within a limited hyper-parameter sample budget.

1 Introduction

Machine learning has become a main driving force for the development of artificial intelligence. Open source packages such as Weka [Hall et al., 2009] and Scikit-learn [Pedregosa et al., 2011] with a variety of learning algorithms provide users an easy way to apply machine learning. But the algorithm performance highly depends on the hyper-parameter setting. For example, the performance of neural network model is sensitive to the architecture, learning rate, and so on [Orr and Müller, 2012]. The hand-crafted hyper-parameter tuning requires expert domain knowledge and tremendous manpower. Thus, automatic machine leaning (autoML), with the aim of choosing the best hyper-parameter without human interference, is appealing. AutoML is usually studied as algorithm selection problems [Adankon and Cheriet, 2009; Biem, 2003; Brazdil et al., 2003], hyper-parameter tuning problems [Bengio, 2000; Bergstra and Bengio, 2012], and more recently, as the combined algorithm selection and hyper-parameter optimization problem (CASH) [Thornton et al., 2013; Feurer et al., 2015]. Despite the different formulations, derivative-free optimization methods are often employed to solve them. Unlike gradient-based optimization that relies on derivatives, these methods work through sampling from the search space. Derivative-free optimization has shown great potential in solving sophisticated problems, such as non-differentiable and non-convex functions.

However, derivative-free optimization commonly needs to sample a lot of hyper-parameters before finding a good one. This is due to the sampling nature of these methods, i.e., a sufficient exploration of the search space by many trial-and-errors is inevitable. The low sample efficiency issue is even severer in autoML tasks. Evaluation of a sampled hyper-parameter in autoML is highly expensive because of, e.g., the high time consumption of the k-fold cross validation on large datasets. Therefore, improving the sample effectiveness of derivative-free optimization, and thus reducing the required sample amount, is a key issue for autoML.

In this paper, we propose an experienced optimization approach to improve the sample effectiveness, which extracts information from previous tasks to guide the optimization on new tasks. Although the tasks can be various, we observed that the search direction of different optimization processes can be aligned. Therefore, we propose to learn a directional model from a set of experienced optimization tasks, and reuse the directional model to guide the search in new optimization tasks. The directional model can help reduce unnecessary explorations and thus save wasted samples. We then incorporate the idea into the recently proposed derivative-free optimization method SRACOS [Hu et al., 2017], and implement the ExpSRACOS approach. Experiments on synthetic functions, hyper-parameter search for heterogeneous ensembles, and architecture design for deep neural networks disclose that the directional model can effectively capture the search direction information, and help ExpSRACOS improve the sample effectiveness significantly within limited sample budget.
2 Background

In this paper, we focus on the hyper-parameter optimization problems. Let \( C \) denote a machine learning model and \( \delta \in \Delta \) denote a hyper-parameter setting, where \( \Delta \) is its hyper-parameter space. Taking \( k \)-fold cross validation as the evaluation criterion of \( C^\delta \), the evaluation can be formulated as follows:

\[
f(\delta) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(C^\delta, \mathcal{D}^i_{\text{train}}, \mathcal{D}^i_{\text{valid}}),
\]

where \( \mathcal{L}(\cdot) \) is a loss function, \( \mathcal{D}^i_{\text{train}} \) and \( \mathcal{D}^i_{\text{valid}} \) are the training and validation data in the \( i \)-th fold. The hyper-parameter optimization problem is to find the best hyper-parameter setting as \( \delta^* = \arg\min_{\delta \in \Delta} f(\delta) \).

Since the hyper-parameter optimization objectives are often non-differentiable, they are commonly solved by derivative-free optimization. Derivative-free optimization is designed for sophisticated problems such as non-convex, non-continuous, and the like. We consider the minimization problems, and let \( X \) denote the compact searching space and \( f: X \to \mathbb{R} \) denote the objective function. The minimization problem can be presented as to find \( x^* \in X \) s.t. \( \forall x \in X : f(x^*) \leq f(x) \). The processes of derivative-free optimization algorithms [Shahriari et al., 2015; Munos, 2011; Yu et al., 2016] share the same idea of optimization from samples. Without any gradient information, derivative-free optimization explores objective space according to the samples and their evaluations. The most important part of derivative-free optimization is how to generate new samples. For example, Bayesian optimization [Shahriari et al., 2015] connects the searching space and the function values with a surrogate function modeled by the Gaussian process (GP), and chooses the sample with the best acquisition function value which is based on the GP model. However, autoML often faces integer and categorical searching spaces. SMAC [Hutter et al., 2011] was proposed to adapt this situation by replacing GP with the random forest model during building the surrogate function. SRACOS [Hu et al., 2017] is a recently proposed classification-based derivative-free optimization method. It shows outstanding efficiency and scalability in some applications.

Employing derivative-free optimization approaches, autoML has achieved successful outcomes. AutoWeka and AutoSklearn, open source autoML projects using SMAC, have received outstanding performance in some autoML competitions. More recent studies focused on searching hyper-parameters for deep neural networks. In [Baker et al., 2017], researchers employed deep reinforcement learning to model architecture designing process. The architectures of layers was considered as actions. The performance of designed architecture was considered as the reward. An reinforcement learning algorithm was then employed to train a policy to predict the actions.

Despite existing successes, evaluation of hyper-parameters in autoML tasks is commonly resource consuming, due to the large training data, repeated evaluations, large models, etc. To reduce the total sample cost, some previous studies have considered how to speed up the convergence by reusing the historical optimization experience for optimizing new tasks. Most of these studies are based on Bayesian optimization, e.g., [Swersky et al., 2013; Lindauer and Hutter, 2018]. To transfer the experience, the previous Bayesian process models can be directly re-used into new optimization processes. For example, in [Lindauer and Hutter, 2018], historical models are linearly combined as a warmstart for new tasks. In our work, we don’t reuse the historical optimization models directly as they may not be well aligned across different optimization tasks, instead, we learn and reuse a directional model for the optimization process that can be better aligned and transferable.

3 Proposed Method

This section describes the proposed experienced optimization approach. We firstly present the overall framework, and then present the implementation details of this framework based on a state-of-the-art derivative-free optimization algorithm.

3.1 Overall Framework

This paper considers an optimization problem set \( F = \{f\} \), where \( f \sim \mathcal{F} \) and \( \mathcal{F} \) is an underlying problem distribution. For example, optimizing hyper-parameters for a certain learning model on different datasets can be seen as a problem distribution. Given \( F_e \) as the experienced problem set, the experienced optimization aims at optimizing future problems \( F_i = F - F_e \) more efficiently.

We have observed that, in different optimization processes, search direction can be aligned and can generalize across difference optimization processes. Therefore, our approach learns a directional model from the historical optimization processes, and guides new optimization processes.

The framework of the experienced optimization with directional model is presented in Algorithm 1. It mainly consists of three steps:

- Organizing the experience dataset \( \mathcal{D}_{F_e} \) from the historical optimization processes (line 1 to 4). Derivative-free optimization methods often store some historical samples during optimization. The instances in \( \mathcal{D}_{F_e} \) can be

Input:
\( F_e, F_i \): Experienced and target problem sets;
\( \mathcal{A} \): The optimization approach;
Log&Assign: Log and assign experience dataset;
Train: Train directional model.

Procedure:
1: \( \mathcal{D}_{F_e} = \emptyset \)
2: for \( f \in F_i \) do
3: \( \mathcal{D}_{F_e}^f = \text{Log&Assign}(\mathcal{A}, f) \)
4: \( \mathcal{D}_{F_e} = \mathcal{D}_{F_e} \cup \mathcal{D}_{F_e}^f \)
5: end for
6: \( \Phi = \text{Training}(\mathcal{D}_{F_e}) \)
7: for \( f \in F_i \) do
8: \( x^*_f = \mathcal{A}(f, \Phi) \)
9: end for

Algorithm 1: Framework of Experienced Optimization by Directional Model
extracted from the snippet of the stored samples. For each instance, we extract features and assign a label that is the direction to a later found better solution. In line 3, the Log&Assign sub-process is used to collect the labeled instances from the optimization processes.

- Learning directional model \( \Phi \) on \( D_{Fe} \) (line 6). With the labeled experience dataset, training \( \Phi \) is a supervised learning problem. Note that \( \Phi \) can be trained by any state-of-the-art learning algorithm.

- Utilizing \( \Phi \) to predict the direction of the next sample during optimizing in new problems (line 7 to 9). The directional models can be embedded in optimization method by adding a pre-sampling step, which generates a set of candidate samples. Among the candidate samples, the one most close to the direction predicted by \( \Phi \) is selected as the next sample.

3.2 ExPSRACOS: Experienced SRACOS

We implement the idea within SRACOS, and propose the ExPSRACOS method for experienced optimization.

**SRACOS.** To explain ExPSRACOS clearly, it is necessary to introduce SRACOS briefly. SRACOS also follows the trial-error process. Throughout the optimization process, SRACos maintains two sets of solutions \( B^+ \) and \( B^- \). \( B^+ \) contains the best \( k \) solutions, and \( B^- \) contains a collection of the rest solutions. They are initialized from random samples of solutions. In every iteration of SRACOS, it learns a classifier from \( B^+ \) as the positive samples and \( B^- \) as the negative samples. It then samples a new solution from the positive area of the classifier, and updates \( B^+ \) or \( B^- \) according to the evaluation result of the new solution. More details of SRACOS can be found in [Hu et al., 2017].

**Collect experience dataset.** We extract the experience dataset from SRACOS optimization processes on previous tasks. In the \( t \)-th iteration, the sampling area of SRACOS is learned on \( (x_t^+, B_t^-) \) during optimizing, where \( x_t^+ \in B_t^+ \). \( B_t^- \) stores the remaining solutions. Therefore, we organize the depending dataset \( (x_t^+, B_t^-) \) as a context matrix:

\[
\kappa_t = \begin{bmatrix}
x_{t,1} - x_t^+ \\
x_{t,2} - x_t^+
\vdots \\
x_{t,m} - x_t^+
\end{bmatrix}, \text{where } x_{t,i} \in B_t^-, i = 1, 2, \ldots, m.
\]

Note that \( \kappa_t \) is a matrix with size \( m \times n \), where \( m = |B_t^-| \) and \( n \) is dimension size of searching space. Each row of \( \kappa_t \) is a sample of \( B_t^- \) which is centralized around \( x_t^+ \). With the centralization, the search behavior at different time and in different optimization tasks can be aligned, thus the model trained on \( \kappa_t \) could be reused to other problems more easily.

Suppose \( x_t' \) is the generated sample using the context matrix \( \kappa_t \). We combine the two to compose an experience instance, \( [\kappa_t; x_t'] \). We assign a label to this instance according to the quality of \( x_t' \). If it improves the objective value, the label is positive, otherwise, the label is negative, i.e.,

\[
\ell_t ([\kappa_t; x_t']) = \begin{cases}
1, & f(x_t') < f(\bar{x}_t) \\
0, & f(x_t') \geq f(\bar{x}_t).
\end{cases}
\]

**Input:**
- \( f \): Objective function to be minimized;
- \( P \): The number of pre-sampling;
- \( r \): The number of samples in initialization;
- \( N \): The evaluation budget;
- \( \Phi \): Directional model;
- **Initialize:** Initialization steps;
- **Sample:** Get a new sample by SRACOS.

**Procedure:**

1. \((B^+, B^-,(\bar{x}, \bar{y})) = \text{Initialize}(\mathcal{H}_X)\)
2. for \( t = r + 1 \) to \( N \) do
3. \( P = \emptyset \)
4. for \( i = 1 \) to \( P \) do
5. \((\kappa, x) = \text{Sample}(B^+, B^-, \lambda, C)\)
6. \( p = \Phi(\kappa, x)\)
7. \( P = P \cup \{(\kappa, x), p\}\)
8. end for
9. \((\bar{x}, \bar{y}) = \text{argmax}_{((\kappa, x), p) \in P} p\)
10. \( \bar{y} = f(\bar{x})\)
11. \((B^+, B^-) = \text{Update}((\bar{x}, \bar{y}), B^+, B^-)\)
12. \((\bar{x}, \bar{y}) = \text{argmin}_{(x, y) \in B^+ \cup (\bar{x}, \bar{y})} y\)
13. end for
14. return \((\bar{x}, \bar{y})\)

**Algorithm 2:** Experienced SRACOS (ExPSRACOS)

where \( \bar{x}_t \) is the best-so-far solution. Putting all experience instances into a dataset, we obtain the experience dataset as \( D_{Fe} = \{(\kappa_1; x_1^+), \ell_1), (\kappa_2; x_2^+), \ell_2), \ldots\} \). **Training directional model.** Because the \( D_{Fe} \) is a labeled dataset, any classifier can be employed. From the previous subsection, we notice that an instance in \( D_{Fe} \) consists of two parts a matrix \( \kappa \) and a vector \( x' \). Thus, we should re-organize the instance from \( [\kappa; x'] \) by reshaping \( \kappa \) as a vector and combining it with \( x' \). In our work, we just apply a simple multilayer perceptron (MLP) as the directional model \( \Phi \). By the last layer of \( \Phi \), we map the directional model output to \([0,1]\) to reflect the goodness of the sample.

**ExPSRACOS.** ExPSRACOS follows the algorithm framework of SRACOS. Before evaluating a sample, a pre-sampling step is added. It generates a set of samples which will be filtered by the directional model. Algorithm 2 presents the pseudo code of ExPSRACOS. Line 1 is the initialization step. Line 4 to 8 is pre-sampling process. The directional model \( \Phi \) will predict goodness of each pre-sampling solution (line 6). Only the solution with highest prediction will be evaluated (line 9 and 10). And then the solution and its evaluation value will be used to update \((B^+, B^-)\). An implementation of ExPSRACOS can be found at https://github.com/eyounx/ExPSRACOS.

**Why ExPSRACOS works.** We discuss why experienced optimization works under the following two assumptions:

- We assume that optimization tasks \( f \in F_e \) and \( F_t \) share the same search space \( X \);
- For any two instances \((\kappa_a; x_a^+), \ell_a)\) and \((\kappa_b; x_b^+), \ell_b)\) in \( D_{Fe} \), we assume \( \ell_a = \ell_b \) if \( \kappa_a; x_a^+ = \kappa_b; x_b^+ \).

The centralization process of \( \kappa \) makes sure that the second as-
similar with each other. We investigate how the relationship among different functions depends on the size of \( x \) that influences the performance of \( \text{EXP}_{\text{SRACOS}} \). We re-used to predict the goodness of new samples on new tasks. In \( \text{EXP}_{\text{SRACOS}} \), only the sample with the highest prediction of \( \Phi \) will be evaluated. Compared with SRACOS that wastes many samples for exploration, \( \text{EXP}_{\text{SRACOS}} \) avoids evaluating many inferior samples.

4 Experiments

In this section, we empirically investigate the effectiveness of the proposed experienced optimization framework (using its implementation \( \text{EXP}_{\text{SRACOS}} \)) on some tasks. We firstly study \( \text{EXP}_{\text{SRACOS}} \) on a synthetic function. Then, \( \text{EXP}_{\text{SRACOS}} \) is employed to tackle two autoML tasks, tuning hyper-parameters on ensemble classifier and optimizing architecture of deep neural network. In these tasks, the directional models are all structures of MLP. But the detail settings are slightly different according to tasks.

We choose the state-of-the-art derivative-free optimization methods to compare with \( \text{EXP}_{\text{SRACOS}} \) including SRACOS (code from \texttt{https://github.com/eyounz/ZOOpt}), CMAES [Hansen et al., 2003] (code from \texttt{https://pypi.python.org/pypi/cma}), SMAC (code from \texttt{https://github.com/automl/SMAC3}) and TPE [Bergstra et al., 2011] (code from \texttt{http://jaberg.github.io/hyperopt/}).

4.1 On Synthetic Functions

We firstly compare \( \text{EXP}_{\text{SRACOS}} \) with SRACOS, CMAES, SMAC and TPE on a highly non-convex function Ackley: 
\[
\text{Ackley}(x) = -20e^{-\frac{1}{2} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i^2 - \cos(2\pi x_i))}} + e + 20.
\]

\( x^* = (x_1^*, x_2^*, \ldots, x_n^*) \) denotes the optimal solution, where \( n \) is dimension size. The search domain is \( X = [-1, 1]^n \). Because Ackley function has many local optima, it provides an extreme situation to test the effectiveness of the experienced optimization.

**Problem settings.** The problem distribution \( \mathcal{F} \) is constructed by randomly shifting \( x^* \) of Ackley function. The relationship among different functions depends on the size of the space \( X^{\text{shift}} \). If it defines a tiny space, problems are more similar with each other. We investigate how \( X^{\text{shift}} \) influences the performance of \( \text{EXP}_{\text{SRACOS}} \) by setting \( X^{\text{shift}} = [-0.1, 0.1]^n \) and \( [-0.5, 0.5]^n \), where we set \( n = 10 \) and 20.

![Figure 1: The convergence rate on Ackley functions with \( x_i^* = 0.04 \) and 0.2, \( n = 10 \) and 20, where \( i = 1, 2, \ldots, n \).](image)

```
Table 1: Average target-problem performance for each problem setting of Ackley function. \( X^{\text{shift}} \) is the optimal solution shifting region and \( n \) denotes the search space dimension. A number in bold means the best function value in its setting.
```

<table>
<thead>
<tr>
<th>Settings</th>
<th>( X^{\text{shift}} )</th>
<th>( n )</th>
<th>( \text{EXP}_{\text{SRACOS}} )</th>
<th>SRACOS</th>
<th>CMAES</th>
<th>SMAC</th>
<th>TPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 1]</td>
<td>10</td>
<td>1.67 ± 0.42</td>
<td>2.37 ± 0.36</td>
<td>2.85 ± 0.36</td>
<td>3.01 ± 0.21</td>
<td>2.69 ± 0.26</td>
<td></td>
</tr>
<tr>
<td>[0, 1]</td>
<td>20</td>
<td>2.76 ± 0.26</td>
<td>2.87 ± 0.20</td>
<td>3.41 ± 0.25</td>
<td>3.31 ± 0.13</td>
<td>3.09 ± 0.17</td>
<td></td>
</tr>
<tr>
<td>[0, 0.5]</td>
<td>10</td>
<td>2.26 ± 0.41</td>
<td>2.42 ± 0.36</td>
<td>2.93 ± 0.39</td>
<td>2.95 ± 0.25</td>
<td>2.71 ± 0.29</td>
<td></td>
</tr>
<tr>
<td>[0, 0.5]</td>
<td>20</td>
<td>2.09 ± 0.27</td>
<td>2.98 ± 0.22</td>
<td>3.56 ± 0.28</td>
<td>3.35 ± 0.18</td>
<td>3.18 ± 0.18</td>
<td></td>
</tr>
</tbody>
</table>

Considering combination, there are 4 different problem distributions on synthetic function experiments.

**Training \( \Phi \) for \( \text{EXP}_{\text{SRACOS}} \).** For each \( X^{\text{shift}} \) setting, we sample a batch of problems to form \( F_c \) and \( F_l \), where \( F_c \cap F_l = \emptyset \). Because \( X^{\text{shift}} = [-0.5, 0.5]^n \) is much larger than \( X^{\text{shift}} = [-0.1, 0.1]^n \), we set \( |F_c| = 200 \) for \([-0.1, 0.1]^n \) and \( |F_l| = 400 \) for \([-0.5, 0.5]^n \). We collect experience dataset \( D_{F_c} \) by applying SRACOS to optimize \( f \in F_c \) with 400 evaluation budget and repeat running for 5 times. We set \( |F_l| = 100 \) target problems for all 4 settings.

Because of the inferior sample effectiveness of SRACOS, the dataset \( D_{F_c} \) is highly imbalanced in classes. The number of positive instances is much less than negative instances. The upsampling strategy is used to adjust the balance, on which the directional model is trained.

**On convergence rate.** We test the convergence rate of each method on Ackley with a specific optimal solution \( x^* \) and the budget of 100 samples. We set \( x^* = (0.2, 0.2, \ldots) \) for \( X^{\text{shift}} = [-0.5, 0.5]^n \) and \( x^* = (0.04, 0.04, \ldots) \) for \( X^{\text{shift}} = [-0.1, 0.1]^n \). Ackley functions with these two \( x^* \) are not in \( F_c \), so it can verify the transfer performance of \( \Phi \) for each problem distribution. Each method runs for five times independently on each function. In Figure 1, two settings of the optimal solutions are presented. It can be observed that the reuse of the directional model is generally less effective for the larger change of the optimal solution. Meanwhile, the convergence speed of \( \text{EXP}_{\text{SRACOS}} \) is the fastest on all settings. Especially compared with SRACOS, in Figure 1 (a), (b), the convergence of \( \text{EXP}_{\text{SRACOS}} \) is significantly faster.

**On average performance.** We report the average performance of the compared methods on all problems in \( F_l \) with 50 evaluation budget. Each method runs five times on each problem independently. The average performance is presented in Table 1. \( \text{EXP}_{\text{SRACOS}} \) shows the best performance among all.
problem settings. The conclusion of this experiment is similar with that of the convergence experiment. 

### On AutoML Tasks

In this section, we apply ExPSRACOS to solve autoML tasks: the hyper-parameter optimization on heterogeneous weighted voting ensemble classifier and the architecture optimization on deep neural networks. These two tasks with the k-fold cross validation error as the criterion have been formulated in Section 2. For ExPSRACOS, the datasets are split as source and target datasets. We organize the experiment dataset on source datasets, and test the reusing performance of Φ on the target datasets. ExPSRACOS is compared with SRAcos, SMAC, TPE. In the experiment results, the optimization and generalization performance mean the validation error on training dataset when optimization and test error on testing dataset. We run each method for 3 times independently and the best solution is selected for testing generalization performance. For each optimal solution, we test the generalization performance for 5 times and report its average performance.

### On Ensemble Classifier Hyper-Parameter Optimization

This task is optimizing hyper-parameters for heterogeneous weighted voting ensemble classifier Cv. We choose ten base classifiers as follows: decision tree, multilayer perceptron, logistic regression, SVM, Gaussian processes, passive aggressive classifier, Gaussian naive Bayes, linear classifiers with SGD training, random forest, k-nearest neighbors classifier. All the 24 hyper-parameters (17 continuous, 4 integer and 3 categorical) need to be optimized including voting weights and hyper-parameters in the base classifiers. Our codes are based on scikit-learn [Pedregosa et al., 2011]. We report the generalization performance of Cv with default hyper-parameters (noted as Default Cv) on each dataset as base-
Table 3: The optimization and generalization performance of neural network architecture optimization. The value in bold means the best error rate on its dataset. The dataset in bold is the target dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>optimization error</th>
<th>generalization error</th>
<th>HC-Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source dataset (MNIST)</td>
<td>.0069</td>
<td>.0079</td>
<td>.0103</td>
</tr>
<tr>
<td>Target dataset (SVHN)</td>
<td>.0557</td>
<td>.0617</td>
<td>.0782</td>
</tr>
</tbody>
</table>

Figure 2: The neural network architecture for the datasets.

A tricky method: Best $C$ is choosing a base classifier with best 5-fold cross validation error. Best $C$ is employed as the baseline on optimization performance.

We collected 34 UCI classification datasets. Among these, 24 datasets are the source datasets (top 24 data sets in Tabel 2). The rest of them are the target datasets (dataset name in bold in Tabel 2). We organize the experience dataset by running SRACOS on training datasets. For organizing the experience dataset, SRACOS has 200 evaluation budget and repeat for 10 times. The label imbalanced problem exists on this task too. We still use upsampling to balance positive and negative instances. EXPSRACOS is then employed to optimize the hyper-parameters. We set 50 sample budget for all the compared methods.

The results are presented in Table 2. The generalization performance by optimization methods is better than the baseline (Default $C_v$) in most cases. It indicates that hyper-parameter optimization can improve generalization performance effectively. EXPSRACOS receives the best optimization and generalization error (the avg. rank is the lowest compared with other methods). EXPSRACOS outperforms SRACOS in most cases. Especially, EXPSRACOS beats SRACOS on 9 target datasets. It indicates the effectiveness of reusing directional model in EXPAROS. On the generalization performance, similar results have been obtained. EXPAROS outperforms other methods on both source and target datasets, and its performance on the target datasets is significantly better. All these results indicate that the experienced optimization can improve optimization efficiency significantly.

5 Conclusions

AutoML tasks such as hyper-parameter optimization and neural network architecture optimization attract attentions incrementally. AutoML tasks are often solved by derivative-free optimization, which, however, commonly suffers from the high evaluation cost. This paper proposes an experienced optimization approach. It utilizes the experience of historical optimization processes to guide the optimization on new tasks. Specifically, the directional model is used to predict the direction of the next samples generated in every algorithm iteration. A pre-sample step is added in the optimization to generate candidate samples and filter them with the directional model. In this way, this framework can eliminate many unnecessary explorations and thus improve the sample effectiveness. Incorporating with a state-of-the-art derivative-free optimization method SRACOS, we implement this framework as EXPAROS. We empirically compare EXPAROS with other state-of-the-art methods on Ackley function and two autoML tasks. The experiment results show that the experienced optimization with directional model can effectively reduce the sample budget and improve the optimization. Recently Zhou [Zhou, 2016] proposed the new concept of learnware, with properties of reusability, evolvability and comprehensibility. The effort of this paper can be viewed as an effort towards reusability from a new perspective.
References


