ZOOpt: Toolbox for Derivative-Free Optimization

Yu-Ren Liu LIUYR@LAMDA.NJU.EDU.CN
Yi-Qi Hu HUYQ@LAMDA.NJU.EDU.CN
Chao Qian QIANC@NJU.EDU.CN
Yang Yu* YUY@NJU.EDU.CN
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China

Hong Qian HQIAN@CS.ECNU.EDU.CN
School of Computer Science and Technology
East China Normal University, Shanghai 200062, China

Abstract

Recent advances of derivative-free optimization allow efficient approximating the global optimal solutions of sophisticated functions, such as functions with many local optima, non-differentiable and non-continuous functions. This article describes the ZOOpt (**Z**eroth **O**rder **Opt**imization) toolbox that provides efficient derivative-free solvers and are designed easy to use. ZOOpt provides single-machine parallel optimization on the basis of python core and multi-machine distributed optimization for time consuming tasks by incorporating with the Ray framework — a famous platform for building distributed applications. ZOOpt particularly focuses on optimization problems in machine learning, addressing high-dimensional and noisy problems such as hyper parameter tuning and direct policy search. The toolbox is maintained toward a ready-to-use tool in real-world machine learning tasks.

Keywords: Software, Derivative-free optimization, Hyper-parameter optimization, Non-convex optimization, Subset selection, Distributed optimization

1. Derivative-Free Optimization

Optimization, taking $x^* = \operatorname{argmin}_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$ as a general representative, is fundamental in machine learning. Derivative-free optimization, also termed as zeroth-order or black-box optimization, involves a class of optimization algorithms that do not rely on gradient information. In recent years, derivative-free optimization has achieved remarkable applications in machine learning, including hyper-parameter optimization (Thornton et al., 2013; Feurer et al., 2015), direct policy search (Salimans et al., 2017; Hu et al., 2017), subset selection (Qian et al., 2015), image classification (Real et al., 2017), etc. Representative derivative-free algorithms include evolutionary algorithms (Hansen et al., 2003), Bayesian optimization (Shahriari et al., 2016), optimistic optimization (Munos, 2014), model-based optimization (Yu et al., 2016), etc.

2. Classification-based Optimization

Model-based derivative-free optimization algorithms share a framework that iteratively learns a model for promising search areas and samples solutions from the model. Different kinds of meth-

$*$. Correspondence author

ods usually vary in the design of the model. For example, cross-entropy methods (de Boer et al., 2005) may use Gaussian distribution as the model, Bayesian optimization methods (Snoek et al., 2012) employ Gaussian process to model the joint distribution, and the estimation of distribution algorithms have incorporated many kinds of learning models. Classification-based optimization algorithms learn a particular type of model: classification model, leading to theoretical grounded properties of optimization performance. A classification model learns to classify solutions into two categories, $good$ or bad. Then solutions are sampled from the $good$ areas. SRACOS (Hu et al., 2017) is a recently proposed classification-based optimization algorithm. Unlike other model-based optimization algorithms, the sampling region of SRACOS is learned by a simple classifier, which maintains an axis-parallel rectangle to cover all the positive but no negative solutions. SRACOS shows outstanding performance in empirical studies. With the aim of supporting machine learning tasks, ZOOpt implements a set of classification-based methods that are efficient and performance-guaranteed, with add-ons handling noise and high-dimensionality.

3. Methods in ZOOpt

Algorithms in ZOOpt		Search space			Parallelization			Noise Handler	High-dimensionality Handler	Suitable Tasks
		Continuous	Discrete	Hybrid	Single-machine	Multi-machine	Asynchronous			
Classification-based Optimization	Racos	✓	✓	✓	✓			✓	✓	Non-differential, non-convex, noisy and high-dimensional functions
	SRacos	✓	✓	✓	✓			✓	✓	
	ASRacos	✓	✓	✓	✓	✓	✓	✓	✓	
Pareto Optimization for Subset Selection	POSS		✓					✓		Subsect selection problems
	PPOSS		✓		✓			✓		

Table 1: Algorithms implemented in the ZOOpt toolbox. For each algorithm, we conclude its support on different kinds of search space, parallelization and the compatibility with the noise handler and the high-dimensional handler.

Optimization in the continuous/discrete/hybrid space. We implement SRACOS (Hu et al., 2017) as the default optimization method, which has shown high efficiency in a range of learning tasks. Optional methods are RACOS (Yu et al., 2016) and ASRACOS (Liu et al., 2019), respectively are the batch and asynchronous version of SRACOS. A routine is in place to setup the default parameters of the two methods, while users can override them. Benefit from the compatibility of the classifier with multiple data types, classification-based optimization supports optimization in the continuous, discrete (categorical), or hybrid space naturally.

Optimization in the binary vector space with constraint. If the optimization task is in a binary vector space with constraints, such as the subset selection problem, POSS (Qian et al., 2015) is the default optimization method. POSS treats subset selection task as a bi-objective optimization problem that simultaneously optimizes some given criterion and the subset size. POSS has been proven with the best-so-far approximation quality on these problems. PPOSS (Qian et al., 2016a) is the parallel version of the POSS algorithm.

Noise handling. Noise has a great impact on the performance of derivative-free optimization. Resampling is the most straightforward method to handle noise, which evaluates one sample several times to obtain a stable mean value. Besides resampling, more efficient methods including value suppression (Wang et al., 2018) and threshold selection (Qian et al., 2017) are implemented.
High-dimensionality handling. Increase of the search space dimensionality badly injures the performance of derivative-free optimization. When a high dimensional search space has a low effective-dimension, random embedding (Wang et al., 2016) is an effective way to improve the efficiency. Also, the sequential random embeddings (Qian et al., 2016b) can be used when there is no clear low effective-dimension.
Distributed Optimization. Evaluation of a sampled solution is usually time consuming for many real-world optimization tasks, such as hyperparameter tuning in large-scale machine learning projects. Incorparating with the Ray framework (Moritz et al., 2018), ZOOpt implements an efficient distributed optimization module that enables users to parallelize single-machine code, with little to zero code changes.

4. Usage

In this section, we will briefly introduce the single-machine optimization, distributed optimization, optimization under noise and optimization in the high dimensional space through a few examples. For the full tutorial, including the detailed API introduction, hyper-parameter tuning tricks and all examples, we refer readers to `https://zoopt.readthedocs.io/en/latest/`.
Single-machine optimization. The core architecture of ZOOpt includes three parts: $Objective$, $Parameter$ and $Opt.min$. The $Objective$ object defines the function expression and the search space. The $Parameter$ object defines all parameters used by the optimization algorithm. $Opt.min$ is the interface for performing optimization. After defining a user-specified objective function and the corresponding search space, only one line of code is needed to perform optimization by using $Opt.min$. A quick-start example is provided as follows.

```
import numpy as np
from zoopt import ValueType, Dimension2, Objective, Parameter, Opt

def ackley(solution):
        x = solution.get_x()
        bias = 0.2
        value = -20 * np.exp(-0.2 * np.sqrt(sum([(i - bias) * (i - bias) for i
                                                 in x]) / len(x))) - \
                np.exp(sum([np.cos(2.0*np.pi*(i-bias)) for i in x]) / len(x)) +
                                                  20.0 + np.e
        return value

dim_size = 100  # dimension size
dim = Dimension2([(ValueType.CONTINUOUS, [-1, 1], 1e-6)]*dim_size)
obj = Objective(ackley, dim)
# perform optimization
solution = Opt.min(obj, Parameter(budget=100*dim_size))
# print the solution
print(solution.get_x(), solution.get_value())
# parallel optimization for time-consuming tasks
solution = Opt.min(obj, Parameter(budget=100*dim_size, parallel=True, server_num
                                        =3))
```

Distributed optimization. Distributed optimization in ZOOpt is implementd by incorparating with Ray. Currently, ZOOpt is an optional optimization tool in $Ray.tune$ – a library for fast hyperparameter tuning at any scale. Through $Ray.tune$, users can easily distribute the optimization without caring about the communication infrastructure. We provide an example as follows.

```
import time
from ray import tune
from ray.tune.suggest.zoopt import ZOOptSearch
from ray.tune.schedulers import AsyncHyperBandScheduler
from zoopt import ValueType  # noqa: F401

def evaluation_fn(step, width, height):
    time.sleep(0.1)
    return (0.1 + width * step / 100)**(-1) + height * 0.1

def easy_objective(config):
    # Hyperparameters
    width, height = config["width"], config["height"]

    for step in range(config["steps"]):
        # Iterative training function - can be any arbitrary training procedure
        intermediate_score = evaluation_fn(step, width, height)
        # Feed the score back back to Tune.
        tune.report(iterations=step, mean_loss=intermediate_score)

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--smoke-test", action="store_true", help="Finish quickly for testing")
    parser.add_argument(
        "--server-address",
        type=str,
        default=None,
        required=False,
        help="The address of server to connect to if using "
        "Ray Client.")
    args, _ = parser.parse_known_args()

    if args.server_address:
        import ray
        ray.init(f"ray://{args.server_address}")
    num_samples = 10 if args.smoke_test else 1000
    # Optional: Pass the parameter space yourself
    # space = {
    #     # for continuous dimensions: (continuous, search_range, precision)
    #     "height": (ValueType.CONTINUOUS, [-10, 10], 1e-2),
    #     # for discrete dimensions: (discrete, search_range, has_order)
    #     "width": (ValueType.DISCRETE, [0, 10], True)
    #     # for grid dimensions: (grid, grid_list)
    #     "layers": (ValueType.GRID, [4, 8, 16])
    # }
    zoopt_search_config = {
        "parallel_num": 8,
    }
```

```
    zoopt_search = ZOOptSearch(
        algo="Asracos",  # only support ASRacos currently
        budget=num_samples,
        # dim_dict=space,  # If you want to set the space yourself
        **zoopt_search_config)
    scheduler = AsyncHyperBandScheduler()
    analysis = tune.run(
        easy_objective,
        metric="mean_loss",
        mode="min",
        search_alg=zoopt_search,
        name="zoopt_search",
        scheduler=scheduler,
        num_samples=num_samples,
        config={
            "steps": 10,
            "height": tune.quniform(-10, 10, 1e-2),
            "width": tune.randint(0, 10)
        })
    print("Best config found: ", analysis.best_config)
```

Optimization under noise. The noise handler can be enabled through adding some attributes to the definition of the $Parameter$ object. Three kinds of noise handlers are implemented in ZOOpt. Naive re-sampling reduces the noise by evaluating the same solution for many times and taking their mean value as the final result. Value suppression (Wang et al., 2018) reduces the noise with a higher efficiency by re-evaluating the best solution when it isn't updated for a pre-defined number of times. Threshold selection (Qian et al., 2017) is a noise handler customized for the POSS algorithm, where the solution x is better than y only if $f(x)$ is smaller than $f(y)$ by at least a threshold. We provide simplified cases on how to use these noise handlers as follows. Their full versions can be found in the tutorial.

```
from zoopt import Parameter
from sparse_mse import SparseMSE
import numpy as np

# naive resampling
parameter = Parameter(budget=200000, noise_handling=True, resampling=True,
                                        resample_times=10)
# value suppression
parameter = Parameter(budget=200000, noise_handling=True, suppression=True,
                                        non_update_allowed=500, resample_times=
                                        100, balance_rate=0.5)
# threshold selection
mse = SparseMSE('sonar.arff')
mse.set_sparsity(8)
parameter = Parameter(algorithm='poss', noise_handling=True, ponss=True,
                                        ponss_theta=0.5, ponss_b=mse.get_k(),
                                        budget=2 * np.exp(1) * (mse.get_sparsity
                                        () ** 2) * mse.get_dim().get_size())
```

Optimization in the high-dimensional space. ZOOpt contains a high-dimensionality handling algorithm called sequential random embedding (SRE) (Qian et al., 2016b). SRE runs the optimization algorithms in the low-dimensional space, where the function values of solutions are evaluated

via the embedding into the original high-dimensional space sequentially. SRE is effective for the function class that all dimensions may affect the function value but many of them only have a small bounded effect, and can scale RACOS, SRACOS and ASRACOS (the main optimization algorithm in ZOOpt) to 100,000-dimensional problems. The high-dimensionality handler can be enabled through adding attributes to the definition of the $Parameter$ object. An example is provided as follows.

```
from simple_function import sphere_sre
from zoopt import Dimension, ValueType, Dimension2, Objective, Parameter, ExpOpt
dim_size = 10000  # dimension size
dim_regs = [[-1, 1]] * dim_size  # search space
dim_tys = [True] * dim_size  # continuous
dim = Dimension(dim_size, dim_regs, dim_tys)  # form up the dimension object
objective = Objective(sphere_sre, dim)  # form up the objective function
budget = 2000  # number of calls to the objective function
parameter = Parameter(budget=budget, high_dimensionality_handling=True,
                                        reducedim=True, num_sre=5, low_dimension
                                        =Dimension(10, [[-1, 1]] * 10, [True] *
                                        10))
solution_list = ExpOpt.min(objective, parameter, repeat=1, plot=True)
```

5. Experiments

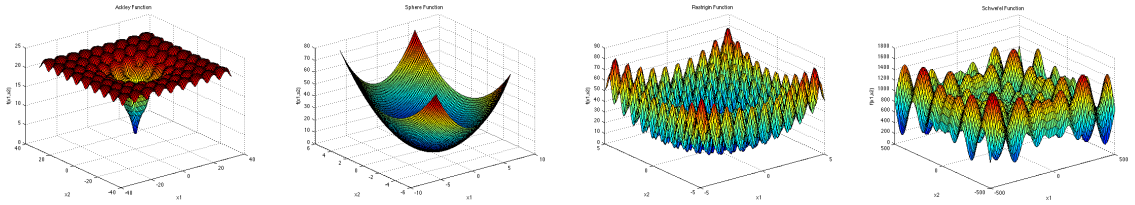
In our experiments, we aim to answer following questions: (1) How does ZOOpt compare to prior derivative-free optimization toolboxes in classic optimization benchmarks? (2) Can ZOOpt scale better than other toolboxes when the dimension size of the optimization task increases? (3) Can ZOOpt have better robustness against noise than other toolboxes? (4) How does ZOOpt compare to other toolboxes in machine learning tasks?

To answer those questions, we compare ZOOpt to several prior derivative-free optimization toolboxes, incuding pycma [1], DEAP [2], pygad [3] and Hyperopt [4]. Pycma (Hansen et al., 2019) is a Python implementation of the CMA-ES (Hansen et al., 2003) algorithm. DEAP (Fortin et al., 2012) is a evolutionary computation framework. Pygad (Gad, 2021) is an open-source Python library of genetic algorithms. Hyperopt (Bergstra et al., 2013) implements the state-of-the-art Bayesian optimization algorithms for hyperparameter tuning. For all toolboxes, we choose the default algorithm and the recommended parameters according to their tutorials. It's worth noting that each toolbox actually implements many optimization algorithms. However we don't exhaust the algorithm-level comparison in this paper, instead, we choose the default algorithm and focus more on the toolbox itself. We refer readers who are interested in the algorithm-level comparision to the paper RACOS (Yu et al., 2016), SRACOS (Hu et al., 2017), ASRACOS (Liu et al., 2019), POSS (Qian et al., 2015) and PPOSS (Qian et al., 2016a). Source code of the experiments can be found from `https://github.com/AlexLiuyuren/ZOOpt_experiment`.

Experiments are conducted on three kinds of tasks. To answer question (1), (2), (3), we conduct experiments on optimizing benchmark synthetic functions. We empirically evaluate the performance of the tested toolboxes, including the convergence rate, the scalability and the robustness against noise, on four benchmark synthetic functions. To answer question (4), we then conduct experiments

1. `https://github.com/CMA-ES/pycma`
2. `https://github.com/DEAP/deap`
3. `https://github.com/ahmedfgad/GeneticAlgorithmPython`
4. `https://github.com/hyperopt/hyperopt`

on two machine learning tasks. We study on a classification task with Ramploss, where the objective function is similar to that of support vector machines (SVM) but the loss funciton of SVM is the hinge loss. We then study on the direct policy search for controlling tasks, where the policy is a fully connected feedforward neural network and its weights are optimized directly by derivative-free optimization algorithms.



(a) Ackley (b) Sphere (c) Rastrigin (d) Schwefel

Figure 1: 3-d graphs of four Benchmark synthetic functions (from `http://www.sfu.ca/%7Essurjano/optimization.html`). Among them, the Ackley, Rastrigin and Schwefel functions are highly non-convex while the Sphere function is conve.

5.1 Results on optimizing synthetic functions

To answer question (1), (2), (3), we conduct experiments on optimizing benchmark synthetic functions. Among them, the Ackley, Rastrigin and Schwefel functions are highly non-convex while the Sphere function is convex. The optimal values of four functions are all zero. The Ackley and Sphere functions are minimized within the seach space $X = [-1, 1]^d$, where d is the dimension size. The Rastrigin function is minimied within $[-5, 5]^d$. The Schwefel function is minimized within $[-500, 500]^d$. The optimal position of each function (except the Schwefel function, which is fixed to $[420.97, ..., 420.97]$) is shifted from $[0, ..., 0]$ to a random point sampled from $[0.2 * l, 0.2 * u]^d$, where l and u respectively refer to the lower and upper bound of the search space on that dimension. This is to avoid a possible optimization bias to the origin point. The 3-d graphs of these functions are shown in Figure 1. Each experiment is repeated for 30 times. Mean values and 95% confidence intervals are recorded. Results are shown in Figure 2.

On convergence rate. We set the dimension size to be 20 for each objective function and number of evaluations to be 2000. We study the convergence rate with regard to the number of function evaluations by recording the best-so-far solution value during the optimization. As shown in the top row of Figure 2, ZOOpt reduces the objective function value with the highest rate in all tasks.

On Scalability The scalability of derivative-free optimization methods is critical on solving large-scale problems. In this experiment, we quantitatively study the scalability of ZOOpt. We set the dimension size d to be 20, 200, 400, 600, 800, 1000 and the number of function evaluations to be $100 \times d$. The confidence interval is omitted for clarity. The middle row of Figure 2 shows that ZOOpt has the lowest growth rate on the function value in all tasks as the dimension size increases, indicating that ZOOpt has better scalability than other toolboxes.

On robustness against noise. To study the performance of ZOOpt on optimizing noisy object, we add the Gaussian noise to original functions to simulate the noisy environment. The new objective functions are defined as $f^N(x) = f(x) + N(0, \sigma^2)$. The number of function evaluations is set to

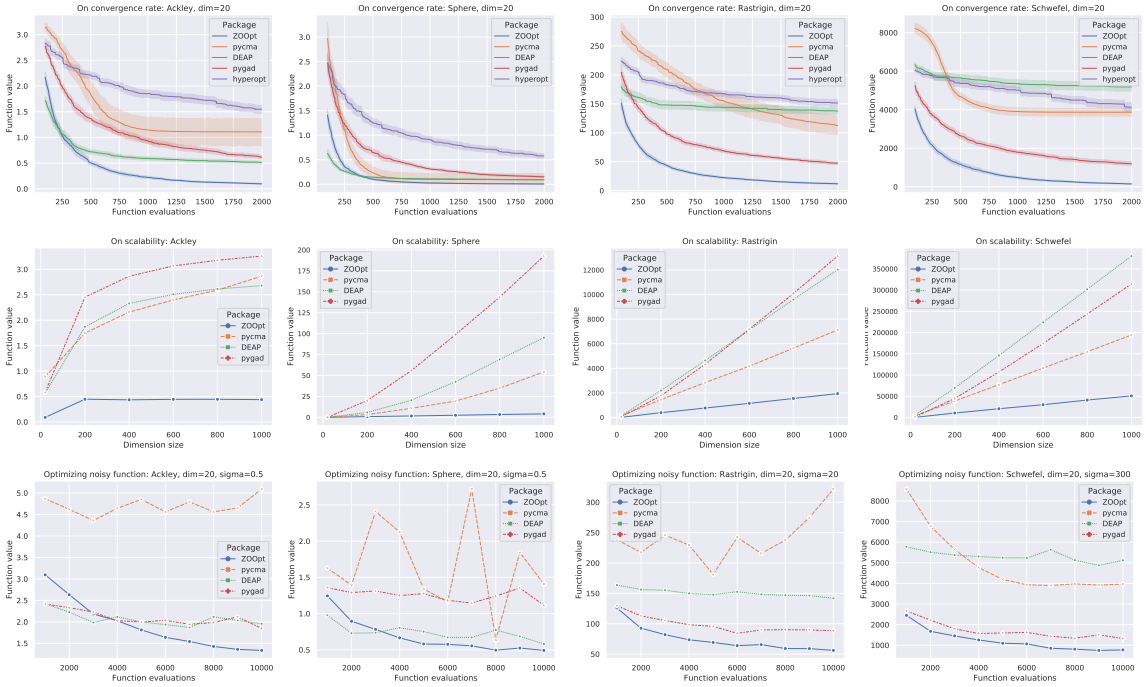


Figure 2: Evaluate the optimization (minimization) performance of ZOOpt on four benchmark synthetic functions. The top row shows the convergence rate of the tested toolboxes. The middle row shows the scalability of the tested toolboxs as the dimension size increases. The bottom row demonstrates the performance on optimizing noisy functions. It can be observed that ZOOpt achieves the best performance in all tasks.

be 10000. For all tasks, ZOOpt and pycma use their built-in noise handler while DEAP and pygad not. It can be observed that ZOOpt reduces the function value at a steady pace as the number of evaluation increases despite the noise.

5.2 Results on classification tasks with Ramploss.

The Ramp loss is defined as $R_s(z) = H_1(z) - H_s(z)$ with $s < 1$, where $H_s(z) = \max\{0, s - z\}$ is the Hinge loss with s being the Hinge point. The task is to find a vector w and a scalar b to minimize $f(w, b) = \frac{1}{2}\|w\|_2^2 + C\sum_\ell^L R_s\left(y_\ell\left(w^\top v_\ell + b\right)\right)$, where v_l is the training instance and $y_l \in \{-1, +1\}$ is its label. Due to the convexity of the Hinge loss, the number of support vectors increases linearly with the number of training instances in SVM, which is undesired with respect to scalability. While this problem can be relieved by using the Ramp loss (Collobert et al., 2006).

We employ two binary class UCI datasets, Adult and Bank, for the classification task. Discrete variables of the original features are preprocessed by one-hot encoding. Continuous variables are normalized into $[-1, 1]$. The result feature dimension (excluding the label) is expanded to 108 for Adult and 51 for Bank. Since we focus on the optimization performance, we only compare the results on the complete data set. Two hyper-parameters, i.e. C and s, are adjustable in the optimization formulation. We set $s \in \{-1, 0\}$ and $C \in \{0.1, 0.5, 1, 2, 5, 10\}$ to study the effectiveness of the

S	Package \ C	0.1	0.5	1	2	5	10
-1	ZOOpt	1642.07 ± 79.33	6331.05 ± 147.10	12002.56 ± 287.74	**23098.68** ± 435.40	**55151.49** ± 772.15	**108896.69** ± 1944.12
	pycma	**1414.25** ± 154.10	**6028.83** ± 495.39	**11537.06** ± 120.91	23259.40 ± 2184.85	55576.41 ± 762.45	109422.90 ± 944.75
	DEAP	2005.05 ± 88.32	6822.13 ± 157.85	12625.33 ± 257.62	23909.87 ± 303.31	57152.50 ± 845.67	111093.63 ± 1454.88
	pygad	3315.09 ± 146.83	8643.11 ± 276.07	14456.62 ± 240.14	26048.55 ± 381.98	59147.50 ± 440.53	113461.05 ± 840.28
0	ZOOpt	1001.56 ± 29.79	3585.84 ± 160.28	**6665.13** ± 408.27	**12451.74** ± 247.92	**29583.38** ± 1886.19	57042.13 ± 751.92
	pycma	**780.45** ± 32.60	**3406.22** ± 345.54	6668.67 ± 776.18	12715.15 ± 1493.09	29639.06 ± 2585.68	**56650.23** ± 509.49
	DEAP	1297.46 ± 41.37	4159.68 ± 201.96	7185.12 ± 457.99	13124.33 ± 859.58	30400.34 ± 1767.03	58898.11 ± 3797.57
	pygad	2531.69 ± 164.29	5588.36 ± 146.79	8846.75 ± 300.44	14949.72 ± 710.44	32167.27 ± 474.29	60436.24 ± 600.72

S	Package \ C	0.1	0.5	1	2	5	10
-1	ZOOpt	128.31 ± 6.69	545.45 ± 7.45	1068.09 ± 12.00	**2075.12 ± 40.36**	**5045.72 ± 98.89**	**9957.51 ± 306.85**
	pycma	**114.24 ± 5.82**	**531.11 ± 4.59**	**1056.25 ± 6.63**	2088.15 ± 30.01	5185.14 ± 89.46	110236.24 ± 280.79
	DEAP	248.58 ± 22.72	670.73 ± 21.44	1191.32 ± 24.96	2234.39 ± 19.57	5307.23 ± 67.89	10316.18 ± 226.76
	pygad	627.27 ± 69.33	1055.97 ± 61.62	1564.35 ± 77.59	2618.18 ± 66.00	5753.68 ± 86.63	10893.56 ± 119.12
0	ZOOpt	73.69 ± 6.61	285.04 ± 9.02	545.82 ± 4.82	1064.49 ± 6.32	**2618.38 ± 52.69**	**5091.75 ± 124.31**
	pycma	**60.84 ± 4.08**	**270.39 ± 3.49**	**532.24 ± 5.70**	**1053.18 ± 3.72**	2620.21 ± 11.10	5221.21 ± 31.35
	DEAP	192.68 ± 16.94	415.67 ± 24.26	673.14 ± 21.71	1187.59 ± 16.59	2763.42 ± 17.70	5329.71 ± 44.41
	pygad	543.22 ± 60.52	798.79 ± 73.10	1037.72 ± 81.79	1573.16 ± 89.69	3145.45 ± 80.23	5787.98 ± 71.72

Table 2: Results on the Adult (upper) and Bank (lower) data sets. Comparing the achieved objective function values against the parameter C of the classification with Ramp loss.

tested toolboxes under different hyper-parameters. We set the total number of calls to the objective function to be 40n for all toolboxes. The achieved objective values are reported in Table 2.

It can be observed that ZOOpt is comparable with pycma and dominate DEAP and pygad in all cases. Notice that the smaller the C is, the closer the objective function is to convexity. Therefore, the optimization difficulty increases with C. Although the results of ZOOpt and pycma are close, ZOOpt achieves the better results when C is large, i.e., the objective function is further from the convexity. Pycma is better when the objective function is closer to the convexity.

5.3 Results on direct policy search for OpenAI controlling tasks.

Gym tasks. In the OpenAI Gym environment, we use six existing controlling tasks, 'Acrobot', 'MountainCar', 'HalfCheetah', 'Hopper', 'Humanoid' and 'Swimmer', to test the toolboxes. We apply the feedforward neural network as the policy. The task information and neural network structures are showed in Table 3. For example, in 'Acrobot': $|S| = 6$, $|A| = 3$; the neural network has two hidden layers with 5 and 3 neurons each; $|w| = 48$; the activation function for hidden layers and the output layer are respectively relu and softmax; the maximum number of steps is 500. We will give a summary of each task. More details can be found in the homepage of OpenAI Gym. In 'Acrobot', system includes two joints and two links, where the joint between the two links is actuated.

Task name	d_{State}	Action type	Action size	NN nodes	#Weights	Activation (hidden)	Activation (output)	Horizon
Acrobot-v1	6	Discrete	3	5, 3	54	relu	softmax	500
MountainCar-v0	2	Discrete	3	5	25	relu	softmax	200
HalfCheetah-v2	17	Continuous	6	10	230	relu	tanh	1000
Hopper-v2	11	Continuous	3	9, 5	159	relu	tanh	1000
Humanoid-v2	376	Continuous	17	25	9825	relu	tanh	1000
Swimmer-v2	8	Continuous	2	5, 3	61	relu	tanh	1000

Table 3: The parameters of the direct policy search for OpenAI controlling tasks.

Initially, the links are hanging downwards and the goal of this task is to swing the end of the low link up to a given height. In 'MountainCar', a car is positioned in a valley between two mountains and wants to drive up the mountain on the right by building up momentum. 'HalfCheetah', 'Hopper', 'Humanoid' and 'Swimmer' are simulation tasks. In those tasks, policy control simulated objects to achieve a goal. For example, in 'HalfCheetah', policy should control a cheetah with half body running forward as fast as possible. The tasks of 'Acrobot' and 'MountainCar' are finding policies with smallest step number when goals are met. The tasks except for 'Acrobot' and 'MountainCar' are finding policies to control object getting scores from the environment as high as possible. Therefore, in Table 4, columns of 'Acrobot' and 'MountainCar' are step numbers, the smaller the better. The other rows are the cumulative rewards from environments, the larger the better.

The average cumulative rewards of 10 simulations is used as the evaluation value of a neural network to reduce noise. The solution space X is set to be $[-10, 10]^{\#\text{Weight}}$. The output of the neural network is scaled to be within the action space, which is defined by the environment. All toolboxes use 2,000 evaluations for each task. The best solution will be re-evaluated for 30 times to reduce the noise further and their mean value will be recorded as the final result. Each experiment is repeated for 10 times. The mean value and the standard deviation are recorded in Table 4. It can be observed that ZOOpt obtained the best results on 5/6 tasks.

Package	Acrobot-v1 ↓	MountainCar-v0 ↓	HalfCheetah-v2 ↑	Hopper-v2 ↑	Humanoid-v2 ↑	Swimmer-v2 ↑
ZOOpt	**82.02 ± 3.05**	**128.23 ± 12.41**	1295.39 ± 731.71	**738.86 ± 391.06**	**448.93 ± 80.33**	**138.05 ± 107.95**
pycma	314.40 ± 186.55	197.81 ± 6.56	465.50 ± 492.81	305.27 ± 358.43	398.30 ± 111.12	35.39 ± 32.06
DEAP	144.26 ± 121.82	200.00 ± 0.00	**1409.11 ± 437.10**	224.53 ± 259.22	303.29 ± 110.66	75.05 ± 104.25
pygad	207.66 ± 146.10	174.85 ± 33.98	188.32 ± 809.55	181.45 ± 230.35	293.49 ± 102.77	50.03 ± 102.21

Table 4: The mean scores and the standard deviation of the best found policy by each toolbox. The numbers in bold mean the best scores in each column. The mark ↓ means the score is the smaller the better, and ↑ means the larger the better.

6. Conclusion

In this paper, we introduce the toolbox ZOOpt which provides efficient derivative-free solvers and is designed easy to use. By combining several state-of-the-art classification-based optimization methods, noise-handlers and high-dimensionality handlers, ZOOpt is particularly good at optimization problems in machine learning. By incorporating with Ray, the optimization in ZOOpt can be easily distributed across multiple machines. In empirical studies, we firstly study the convergence rate, the scalability and the robustness against noise of ZOOpt on optimizing synthetic functions. ZOOpt achieves the best performance in all of these experiments. We then test ZOOpt on two machine learning tasks. Results on classification tasks with Ramploss show that ZOOpt is comparable with pycma and dominates other toolboxes. Results on direct policy search for OpenAI controlling tasks show that ZOOpt achieved the best performance on 5/6 tasks. For a detailed tutorial of the usage of ZOOpt, we refer readers to the project homepage `https://github.com/polixir/ZOOpt`.

Acknowledgments

We would like to acknowledge the support for this project from the National Key Laboratory for Novel Software Technology at Nanjing University (No. KFKT2021B14), the Natural Science Foundation of Shanghai (No. 21ZR1420300), Shanghai Key Laboratory of Multidimensional Information Processing at East China Normal University (No. MIP202101), and the Fundamental Research Funds for the Central Universities.

References

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 115–123, Atlanta, GA, 2013.

R. Collobert, F. H. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, volume 148, pages 201–208, Pittsburgh, Pennsylvania, 2006.

P. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28*, pages 2962–2970, Montreal, Canada, 2015.

F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

A. F. Gad. Pygad: An intuitive genetic algorithm python library. *CoRR*, abs/2106.06158, 2021.

N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11 (1):1–18, 2003.

N. Hansen, Y. Akimoto, and P. Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, 2019.

Y.-Q. Hu, H. Qian, and Y. Yu. Sequential classification-based optimization for direct policy search. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2029–2035, San Francisco, CA, 2017.

Y. Liu, Y. Hu, H. Qian, and Y. Yu. Asynchronous classification-based optimization. In *Proceedings of the First International Conference on Distributed Artificial Intelligence*, pages 9:1–9:8, Beijing, China, 2019.

P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, pages 561–577, Carlsbad, CA, 2018.

R. Munos. From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–130, 2014.

C. Qian, Y. Yu, and Z.-H. Zhou. Subset selection by pareto optimization. In *Advances in Neural Information Processing Systems 28*, pages 1765–1773, Montreal, Canada, 2015.

C. Qian, J. Shi, Y. Yu, K. Tang, and Z. Zhou. Parallel pareto optimization for subset selection. In S. Kambhampati, editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1939–1945, New York, NY, 2016a. IJCAI/AAAI Press.

C. Qian, J.-C. Shi, Y. Yu, K. Tang, and Z.-H. Zhou. Subset selection under noise. In *Advances in Neural Information Processing Systems 30*, pages 3563–3573, Long Beach, CA, 2017.

H. Qian, Y.-Q. Hu, and Y. Yu. Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1946–1952, New York, NY, 2016b.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2902–2911, Sydney, Australia, 2017.

T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, Lake Tahoe, NV, 2012.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855, Chicago, IL, 2013.

H. Wang, H. Qian, and Y. Yu. Noisy derivative-free optimization with value suppression. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, New Orleans, LA, 2018.

Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. D. Freitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

Y. Yu, H. Qian, and Y.-Q. Hu. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2286–2292, Phoenix, AZ, 2016.