Meta Learning Black-Box Population-Based Optimizers

- 作者：Hugo Siqueira Gomes, Benjamin Léger, Christian Gagné
- 机构：laval 大学, Canada CIFAR AI Chair
- 会议：arxiv（*5 Mar 2021提交*）
- 地址：https://arxiv.org/abs/2103.03526
- 代码：https://github.com/optimization-toolbox/meta-learning-population-based-optimizers

论文主要内容

摘要

No free lunch定理引发思考：如何能够为具体不同任务量身定制优化器从而达到SOTA性能？本文提出Learning-to-Optimize POMDP (LTO-POMDP), 基于 partially observable Markov decision process (POMDP) 的一个元学习框架。将优化器用神经网络的权重参数化表示，训练得到data-driven的优化器。相比SOTA的通用优化算法（CMA-ES），具有更好的泛化能力、sample efficiency。

贡献

1. 一种通用的方法论用来训练、评估 可学习的黑盒算法
2. 给定一个优化任务的distribution，能够为这个任务量身定制优化算法from scratch

优势

1. 并行性支持较好（population based）
2. 训练集和测试集可以是不同configspace？（coordinate wise）

研究内容

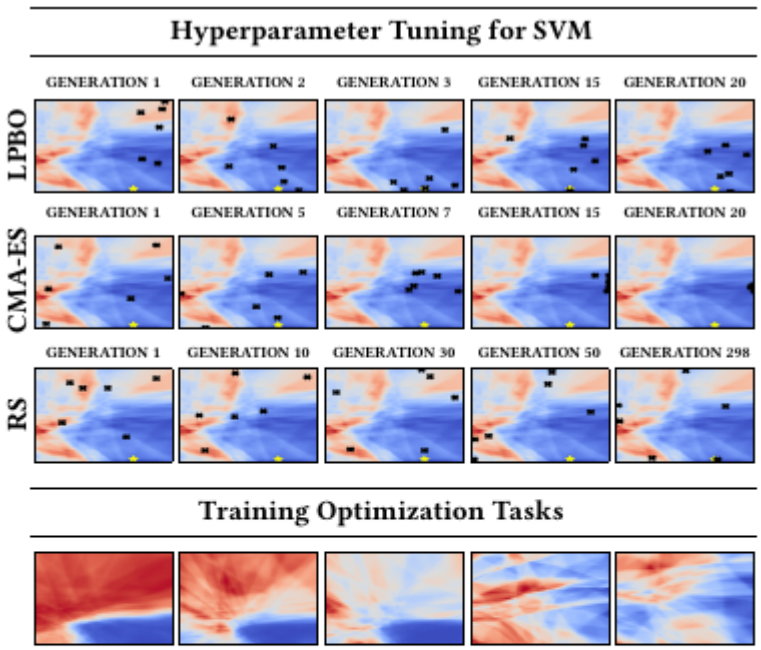


Figure 1: An example run of a learned optimizer. Most algorithms may fail to locate the global optimum (yellow star) due to the function's deceptive nature. The learned population-based optimizer (LPBO) solution learns an inductive bias over related training tasks (bottom). It enables a better initialization of the population (black cross, generation 1), a faster adaptation of the search behavior to exploit the function (generation 5), and encourages the maintaining of diversity in the population to reach other regions efficiently without the use of external mechanisms (e.g., restart strategies or diversity preservation)

Learned optimizer：

- 从相关任务中学到归纳偏置（inductive bias）
- 更好的初始化种群（better initialize）
- 搜索行为能更快适应函数（faster adaptive）
- 自动维持种群多样性（diversity）

通用群体智能算法的一般描述：

Algorithm 1 General Template of Population-Based Algorithms

procedure OPTIMIZE(f)
  $P_X^0 \leftarrow$ generate initial population
  $P_Y^0 = f(P_X^0)$
  for g = 1, 2, ..., G **do**
    $P_X^g \sim \pi(P_X | \{P_X^i, P_Y^i\}_{i=0}^{g-1})$
    $P_Y^g = f(P_X^g)$
  end for
end procedure

- 不同群体智能算法**实例**的区别仅在生成下一代种群的策略 π （算法第5行，第2行也是）
- 论文思路就是把之前手工设计的 π 替换成 π_θ
- θ 通过learning确定，能够考虑初始化种群策略、stopping准则、搜索空间representation等

方法

Learning to optimize能看作一个双层优化问题：

- base loop（内循环）
  - 在一个task下，给定optimizer（ θ 固定）的群体优化算法循环
  - base loop 最终给出在该task、该 π_θ 下的performance measure（外循环的reward）
- outer loop 根据reward 更新 θ ，sample task

> 📍 π_θ 对于外循环来说就是强化学习中的policy
>
> 学习 θ 的方式是deep GA

整体框架

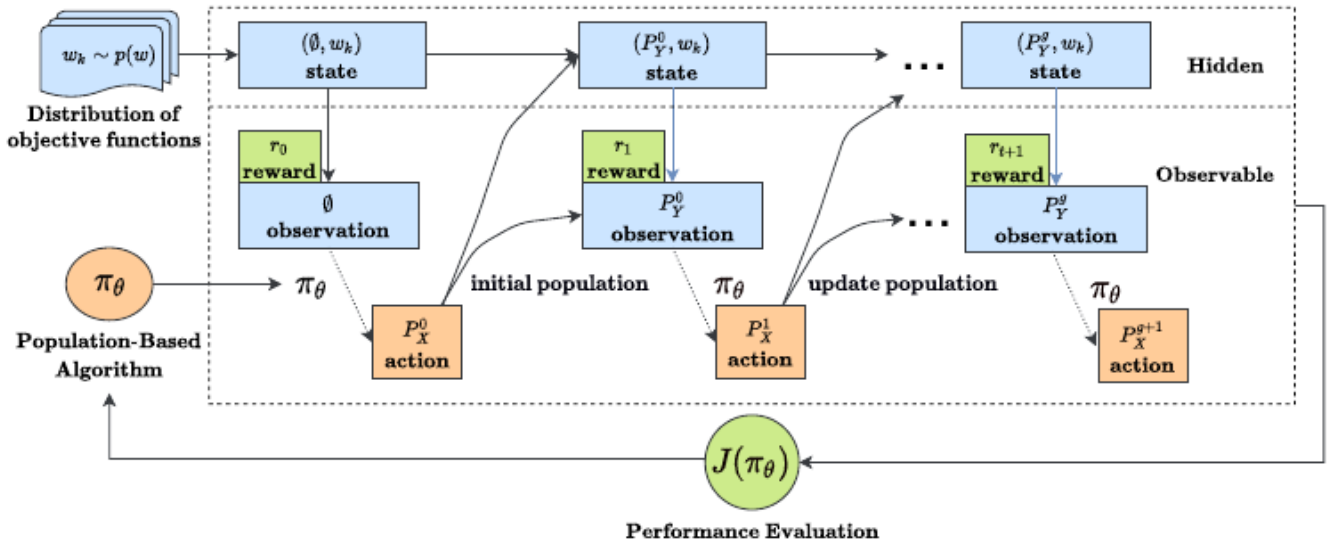


Figure 2: Overview of the learning-to-optimize POMDP (LTO-POMDP).

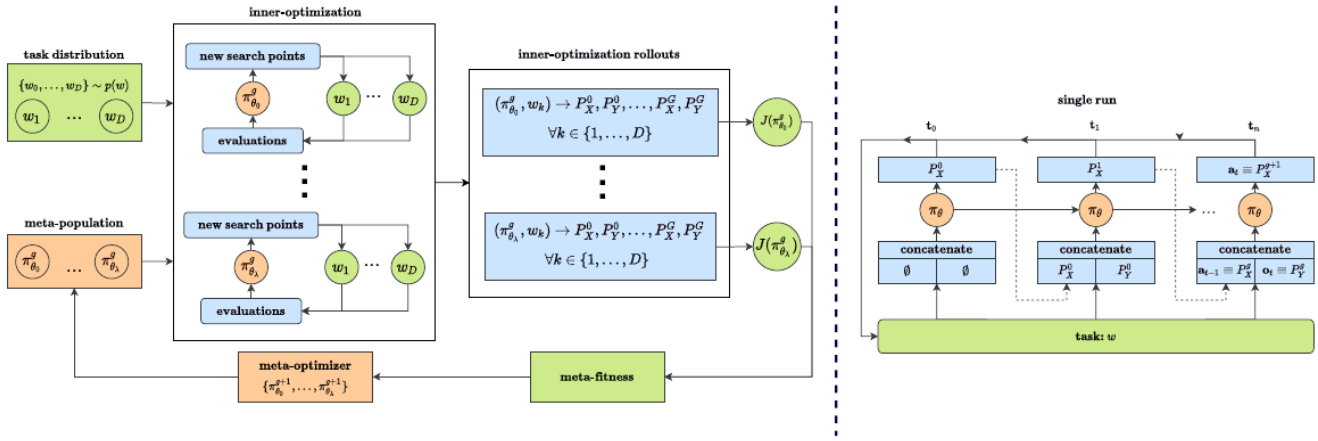


Figure 3: The two-level overview of the system. The outer-optimization loop (left) consists in improving a population of learned optimizers based on each performance $J(\pi_\theta)$ (green) through successive runs on a set of tasks. The inner-optimization loop (right) uses the learned optimizer π_θ (red) to propose search points through successive interactions with the task (green).

内循环（会执行一次具体的BBO task）

单次内循环的算法描述：

Algorithm 2 A single run in Learning-to-Optimize POMDP

At time $t = 0$:
  current task $w_k \sim p(w)$ is sampled
  initial state set to $s_0 = \{\emptyset, w_k\}$
  initial observation set to $o_0 = \{\emptyset\}$
for t = 0,1,2,...,T **do**
  policy takes an action a_t according to current observation o_t and available information $\mathcal{I}_t$:

$$a_t \sim \pi_\theta(a|o_t, \mathcal{I}_t) \in \mathcal{A}$$

$$\text{with } \mathcal{I}_t = \{\emptyset, a_0, r_0, \ldots, o_{t-1}, a_{t-1}, r_{t-1}\}$$

  policy receives the reward $r_t = r(s_t, a_t)$ for choosing a_t
  update state based on the current task w_k

$$s_{t+1} = \{w_k(a_t), w_k\}$$

  generate observation deterministically from the state

$$o_{t+1} = \{w_k(a_t)\}$$

  policy updates available information $\mathcal{I}_{t+1}$

$$\mathcal{I}_{t+1} = I_t \cup \{a_t, r_t, o_{t+1}\}$$

end for
Final policy reward $r_T = r(s_T, a_T)$ received

- a_t 是优化器实例的 t 代的种群(a population of configuations)

- r_t 是t代evaluation的rank，能处理不同scale的problems，泛化到新任务

外循环（迭代优化 θ ）

Python

```
populations = init_thetas() # theta是LSTM网络权重
for g in range(generation_num):
    for p in range(pop_size):
        ======
        policy.load_weight(populations[p])
        results = []
        for r in range(restarts_num):
            policy.reset(seed=r) # 实际固定种子，每次初始化的lstm_hidden一样?
            env.reset() # sample next w_k
            -----
            # 内循环
            obs, action = None, None
            while True:
                action = policy(rank(obs), action) # 一次优化器的suggest
                obs, done = env.step(action)
                if done:
                    break
            results.append(env.result)
            -----
         populations_fitness[p] = meta_loss(results)
         ======
     populations_fitness = rank(populations_fitness)
     selection(populations, populations_fitness)
     mutation(populations, populations_fitness)
```

Performance metric（元优化器的优化目标）

Meta-Loss function：

$$\min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[J^k \left(\pi_\theta \right) \right]$$

$J^k \left(\pi_\theta \right)$ 是策略 π_θ 在任务 w_k 上的performance（越小越好）

performance定义：达到某个目标值(success criterion)需要的function evaluations (FE)次数 的期望

- 具体表达式：

$$\min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[J\left(\pi_\theta, w_k\right) \right] = \min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[\mathrm{FE}_{\theta, w_k} \right]$$
$$= \min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[\left(\frac{1 - \hat{p}_s}{\hat{p}_s} \right) \mathrm{FE}_{\max} + \mathbb{E}\left[\mathrm{FE}^{\mathrm{succ}}_{\theta, w_k} \right] \right]$$

- p_s $\in (0, 1]$表示成功达到目标值的概率
- FE_{max} 表示设置的最大evaluation 数，即当 π_θ 没有达到目标值时的evaluation数
- $J\left(\pi_\theta, w_k\right) =$ $\mathrm{FE}\left(\pi_\theta, w_k\right) = \mathrm{FE}_{\theta, w_k}$ （一个随机变量，restart π_θ ）
- $\mathrm{FE}_{\theta, w_k} = \sum_{i=1}^{N} \mathrm{FE}_{\max} + \mathrm{FE}^{\mathrm{succ}}_{\theta, w_k}$ ，N是要达到目标值需要restart的次数

Policy 架构

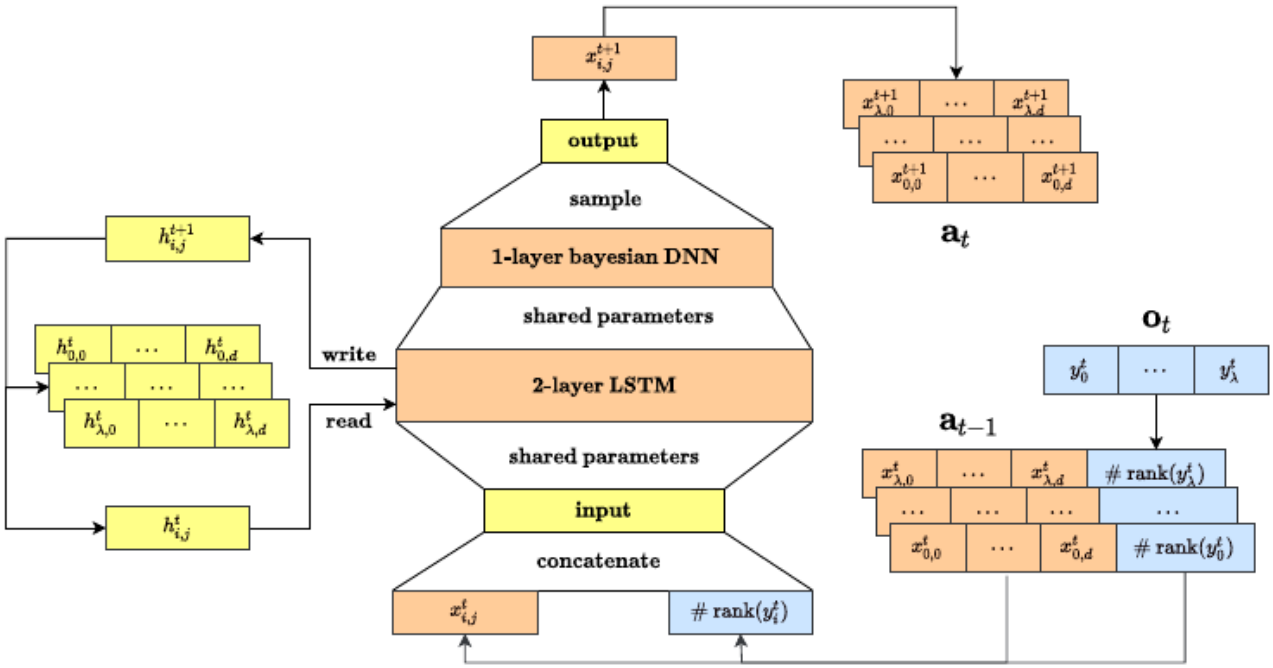


Figure 4: Policy Architecture. The population is updated based on the individual (value, fitness ranking) pair.

- 优化器是一个LSTM+MLP神经网络，output是policy π_θ
- 神经网络的权重参数 θ 使用deep GA优化（gradient-free）
- LSTM序列对应GA中的generation序列，hidden state在随generation t 变化
  - 一个LSTM cell的输入是一个第g-1代种群（x,rank_y）与上一个隐状态，输出是当前隐状态与next_x; 紧挨着下一个cell的输入是g代种群(next_x,next_rank_y)与上一个隐状态
- meta-learning体现在训练阶段结束得到 θ^* 后， π_θ 就是一个Learned**具体的优化器**（若不fine-tune）

- 实际上区别传统的遗传算法优化器只是在每次演化过程都是一个网络来做（传统的是使用人为的归纳偏置），策略是通过在train set里训练学到的

Task 分布

$w_k \sim p(C),\ \text{where } C = \mathcal{F} \times \mathcal{V}$

$\mathcal{F}$ 为目标函数类的空间， $\mathcal{V}$ 为 $\mathcal{F}$ 的configurations

---

实验

讨论三个问题：

1. Fit for purpose：能不能训练优化器在具体任务上表现的很好?
2. Generalization：learned optimizer能不能有能力根据不同任务来发现搜索策略?
3. Crossing the reality gap：能不能被用作现实世界优化问题?

实验一：训练、评估在同一分布任务上

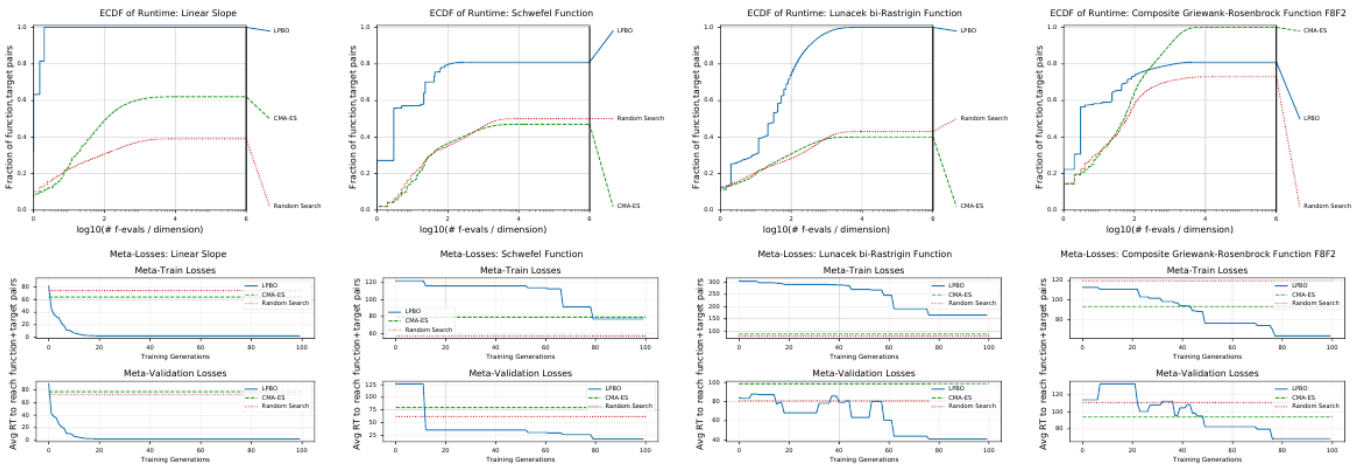
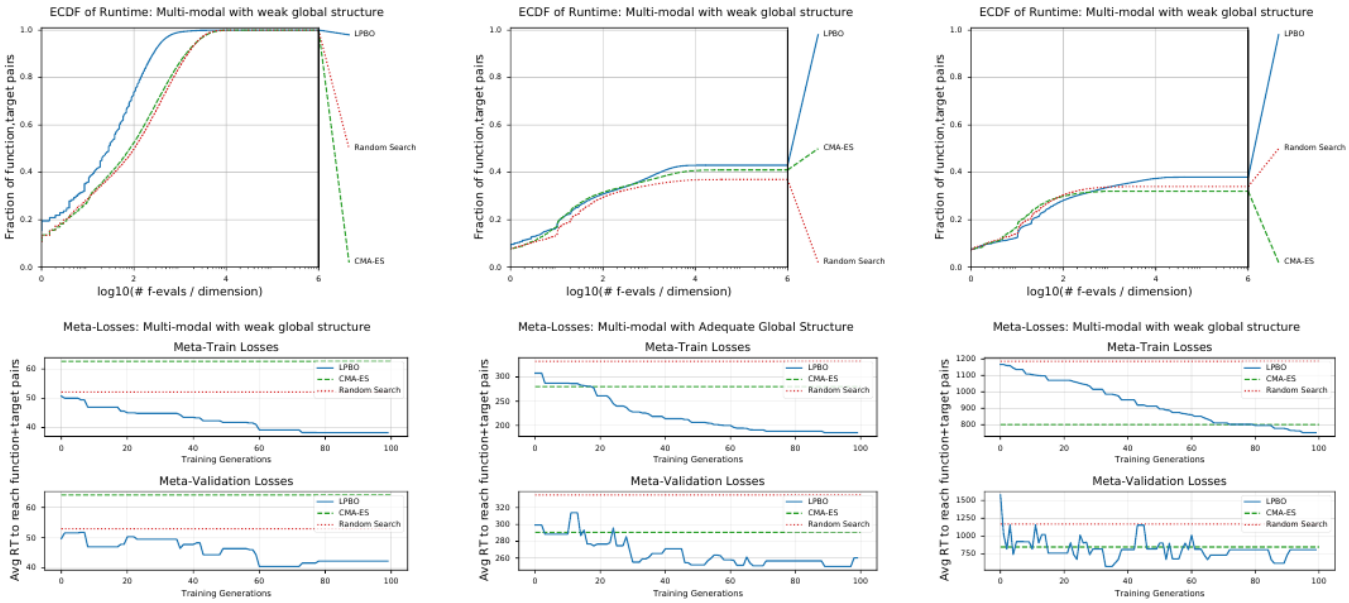


Figure 5: Results in meta-training multiple instances. ECDFs (top) and meta-losses (bottom) in *Linear-Slope*, *Schwefel*, *Lunacek bi-Rastrigin* and *Composite Griewank-Rosenbrock* functions in 2D (left to right).

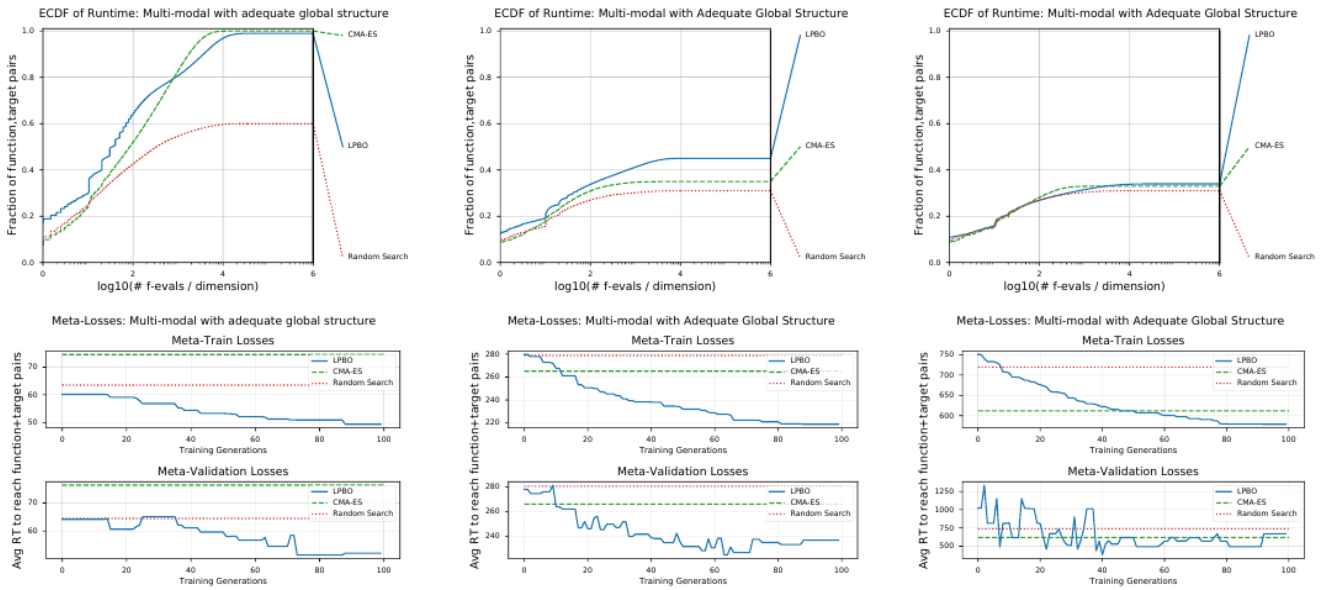
- 从三个任务上来看LPBO（learned population-based optimizer）都具有显著优势
- 最后一个任务（ a highly irregular multi-modal task）
  - CMA-ES能够最终找到所有targets
  - LPBO能更快的找到easy targets

实验二：不fine-tune直接迁移到新task分布

modal functions with weak global structure. We keep the same hyperparameters for the learned algorithm to show that zero (or minimal) fine-tuning is required for different black-box optimization tasks in real-world problems.



- 该任务具有deceptive and highly variable nature
- 需要算法能正确平衡局部搜索和全局感知



- 该任务需要算法具有“higher-level” search strategies（全局意识）
- 在2 dim的时候被CMA-ES略微超过， 但sample efficient to solve easier targets

实验三：HPO任务

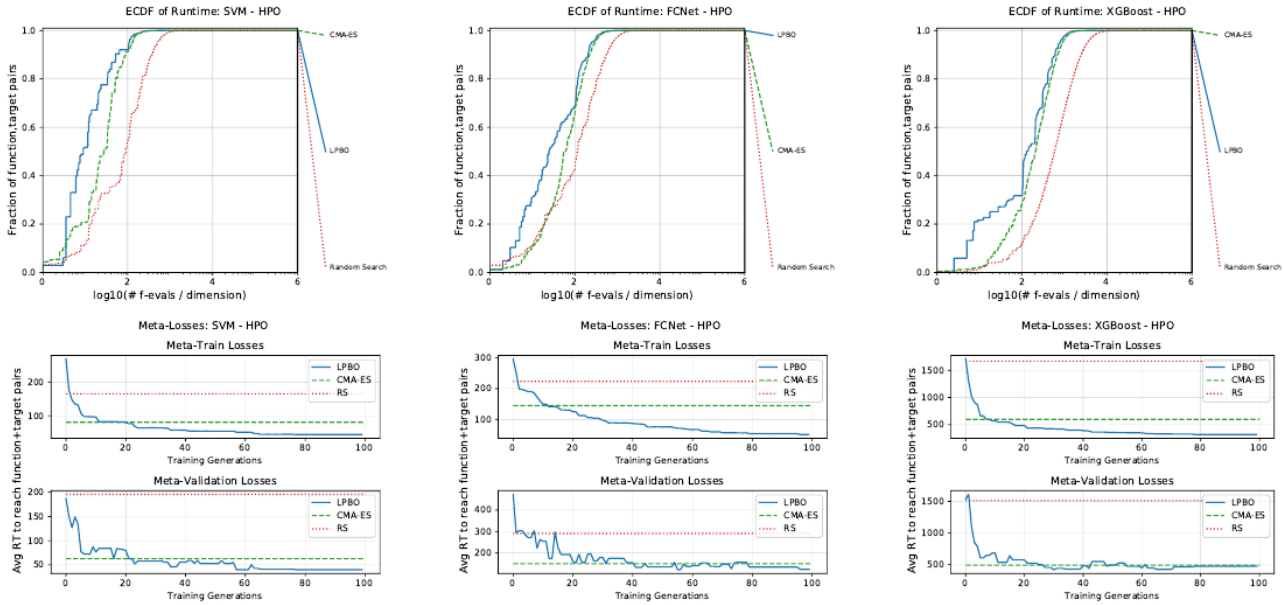


Figure 7: Results in meta-training HPO tasks. ECDFs (top) and meta-losses (bottom) for hyperparameter search of SVM (6D), a fully connected neural network (FC-Net, 6D) and XGBoost algorithm (8D).

- 结果说明能够在实际超参搜索中超过CMA-ES的表现

其他资料

Deep GA

《Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning》

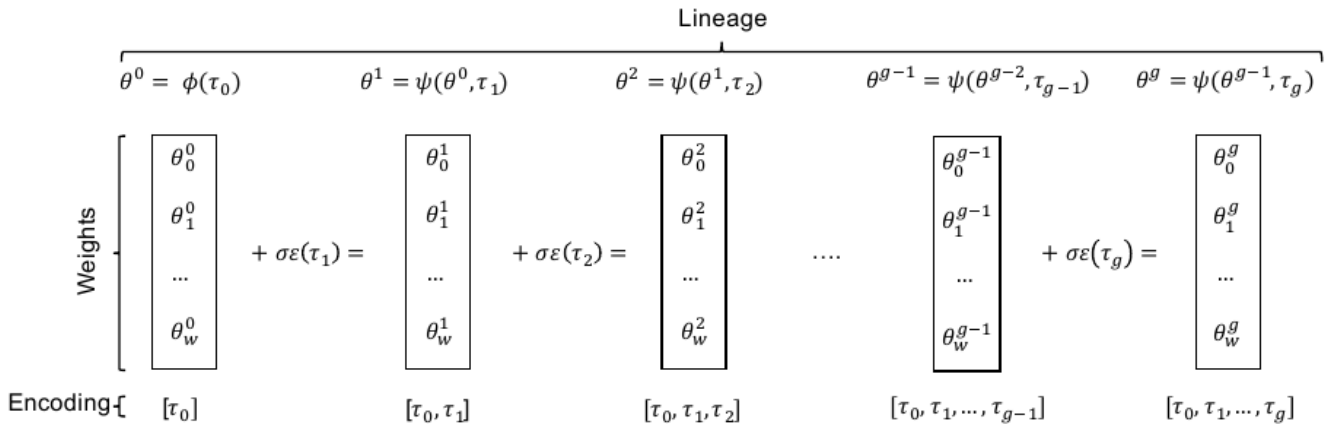


Figure 2. **Visual representation of the Deep GA encoding method.** From a randomly initialized parameter vector θ^0 (produced by an initialization function ϕ seeded by τ_0), the mutation function ψ (seeded by τ_1) applies a mutation that results in θ^1. The final parameter vector θ^g is the result of a series of such mutations. Recreating θ^g can be done by applying the mutation steps in the same order. Thus, knowing the series of seeds $\tau_0 \ldots \tau_g$ that produced this series of mutations is enough information to reconstruct θ^g (the initialization and mutation functions are deterministic). Since each τ is small (here, 28 bits long), and the number of generations is low (order hundreds or thousands), a large neural network parameter vector can be stored compactly.

维护随机种子

POMDP

Markov Models		Do we have control over the state transitons?	
		NO	YES
Are the states completely observable?	YES	**Markov Chain**	**MDP** Markov Decision Process
	NO	**HMM** Hidden Markov Model	**POMDP** Partially Observable Markov Decision Process

理解**部分可观测**：https://www.zhihu.com/question/27693760/answer/151976730