2020 年 4 月 第 4 期
Vol. 41 No. 4 2020

基于强化学习的超参数优化方法

陈森朋,吴 佳,陈修云

(电子科技大学 信息与软件工程学院,成都 610054)
E-mail: jiawu@ uestc. edu. cn

摘 要: 近年来,机器学习算法广泛应用于多个领域. 超参数的选择直接影响了算法模型的性能,然而超参数优化过程往往依赖于专业知识和长期经验的积累. 为了解决上述问题,本文提出了一种基于强化学习的自动超参数优化方法. 该方法将超参数优化问题作为序列决策问题并建模为马尔科夫决策过程,通过使用一个强化学习智能体(agent),自动为机器学习算法选择超参数. 该智能体以最大化待优化模型在验证数据集上的准确率为目标,将模型在验证数据集上的准确率作为奖赏值(reward),通过策略梯度算法训练智能体. 为了减小训练过程中的方差,我们设计了数据引导池模块. 实验将随机森林和 XGBoost 算法作为优化对象,在五个数据集上与随机搜索、贝叶斯优化、TPE、CM-AES 和 SMAC 五种优化方法进行了对比. 实验结果显示,本文所提出的方法在 90% 的优化任务上表现出更优的性能. 同时,我们通过执行一系列消融实验验证了 agent 结构和数据引导池的有效性.

关 键 词: 机器学习; 超参数优化; 长短时记忆神经网络; 强化学习
中图分类号: TP393 **文献标识码:** A **文 章 编 号:** 1000-1220(2020)04-0679-06

Hyperparameter Optimization Method Based on Reinforcement Learning

CHEN Sen-peng, WU Jia, CHEN Xiu-yun
(School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China)

Abstract: Recently, machine learning algorithms have been widely used in many fields. Hyperparameter directly affects the performance of the machine learning algorithms. However, hyperparameter tuning depends on the professional knowledge and the expert experience. In order to solve the above problem, we propose an automatic hyperparameter optimization method based on reinforcement learning. This method considers the hyperparameter optimization problem as a sequence decision problem and models it as a Markov decision process (MDP). An reinforcement learning agent automatically selects hyperparameters for a machine learning algorithm. The accuracy of the model on the validation data set is used as a reward. To reduce the variance during the training, a data boot pool technique is designed. We have conducted a series of experiments to tune hyperparameters for the Random forest and XGBoost. We have compared our method with five optimization methods: random search, Bayesian optimization, TPE, CM-AES and SMAC on five datasets. The experimental results show that the proposed method achieves the best performance on 90% of the tasks. . In addition, we have verified the effectiveness of the agent structure and the data boot pool by performing the ablation experiments.
Key words: machine learning; hyper-parameter optimization; long short-term memory neural network; reinforcement learning

1 引 言

近年来,机器学习算法已成功应用于众多领域,但同时也面临着巨大挑战. 诸如随机森林(Random Forest)[1]、XGBoost[2]和支持向量机(Support Vector Machines)[3]等机器学习算法在实际应用的过程中存在繁琐的超参数优化过程.

超参数优化对机器学习算法的性能起着至关重要的作用,然而机器学习算法的性能和超参数之间的函数关系尚不明确. 在实际应用中,往往通过不断调整超参数的值来提高机器学习算法的实践性能. 当机器学习算法的超参数空间较大时,优化过程将非常耗时和低效. 因此,超参数优化成为了机器学习算法应用中的难点之一.

针对上述问题,本文提出了一种基于强化学习的超参数优化方法(图 1). 该方法将超参数优化问题抽象为序列决策过程,即分步选择待优化算法的超参数,这样超参数选择过程可建模为马尔科夫决策过程(Markov Decision Process-MDP),进而采用强化学习来求解. 具体的,该方法利用长短时记忆神经网络(Long Short-Term Memory Neural Network, LSTM)[4]构建一个智能体(agent)来代替算法使用者设置超参数的值; 然后,agent 在训练集上训练算法模型,并在验证数据集上得到该算法模型的验证集性能,并以此为奖赏信号,利用策略梯度算法(Policy Gradient)[5]优化 agent 的决策.

收稿日期: 2019-07-05 收修改稿日期: 2019-11-01 基金项目: 国家自然科学基金项目(61503059)资助; CCF-腾讯犀牛鸟创意基金项目(CCF-Tencent TAGR20170201)资助. 作者简介: 陈森朋,男,1997 年生,硕士研究生,研究方向为深度学习、强化学习和数据分析; 吴 佳,女,1980 年生,博士,副教授,CCF 会员,研究方向为深度学习、强化学习和数据分析; 陈修云,男,1993 年生,硕士研究生,研究方向为深度学习、深度强化学习和大数据分析.

本文结构如下: 第 2 节介绍了超参数优化问题的定义及相关工作; 第 3 节详细描述了本文所提出的超参数优化方法以及如何减小训练方差; 第 4 节针对两个具有代表性的机器学习算法,将本文所提出的方法与五种常用超参数优化方法进行对比,并且讨论了 agent 结构和数据引导池的有效性; 第 5 节总结全文并展望未来工作.

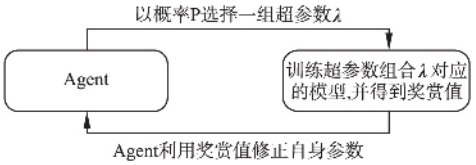

图 1 基于强化学习的超参数优化方法

Fig. 1 Hyperparameter optimization method based on deep reinforcement learning

2 背景及相关工作

超参数优化问题(HPO)的通常定义为: 假设一个机器学习算法 M 有 N 个超参数,第 n 个超参数空间为 Λ_n,那么算法的超参数搜索空间为 $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \cdots \Lambda_N$. M_{λ} 表示超参数为 $\boldsymbol{\lambda}$ 的算法,其中向量 $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ 为算法 M 的一个超参数组合. 当给定数据集 $\mathbf{D}$,HPO 问题的优化目标为最优的超参数组合 $\boldsymbol{\lambda}^*$:

$$\boldsymbol{\lambda}^* = \operatorname{argmin} E_{(\mathbf{D}_{train}, \mathbf{D}_{valid}) \sim D} L(M_{\lambda}, \mathbf{D}_{train}, \mathbf{D}_{valid}) \quad (1)$$

其中,$\mathbf{D}_{train}$ 和 $\mathbf{D}_{valid}$ 分别表示训练集和验证集; $L(M_{\lambda}, \mathbf{D}_{train}, \mathbf{D}_{valid})$ 表示算法 M_{λ} 在数据集 $\mathbf{D}$ 上的交叉验证误差,以此作为损失函数值.

近年来,具有代表性的超参数优化方法有随机搜索(Random Search)、贝叶斯优化(Bayesian Optimization),TPE(Tree-structured Parzen Estimator)以及自适应协方差矩阵进化策略(CMA-ES)算法. 随机搜索方法[6]在超参数搜索空间中随机采样,执行效率高且操作简单,经过多次搜索可以获得性能较好的超参数组合. 然而,随机搜索方法稳定性较差,且只有在达到或接近最优值的超参数组合的比重超过 5% 时,搜索效率较高. 自适应协方差矩阵进化策略(CMA-ES)算法[7]是一种基于进化算法的改进算法,主要用来解决非线性、非凸的优化问题,但算法运行具有一定的随机性,优化性能不稳定. 贝叶斯优化[8,9]方法使用高斯过程对代理函数进行建模,以一组超参数 $\boldsymbol{\lambda}$ 为条件对优化目标 y 进行建模,形成先验模型 $P(y|\boldsymbol{\lambda})$. 虽然该方法能够达到很好优化结果,但是随着迭代次数增加,优化过程耗费大量时间. 文献[10]实验证明了基于高斯过程的贝叶斯优化方法在一些标准任务上优于随机搜索方法. 另一种贝叶斯优化的变体是基于序列模型的优化方法(SMAC)[11],该方法使用随机森林对代理函数进行建模. 与基于高斯过程的贝叶斯优化方法类似,TPE[12]是一种基于树状结构 Parzen 密度估计的非标准贝叶斯优化算法,也能达到很好的优化性能.

相比于上述工作,本文的创新点主要有以下几点:

1) 将超参数优化问题抽象为序列决策问题并建模为 MDP,分步选择超参数,提高优化效率;

2) 采用强化学习智能体(agent),并使用策略梯度算法进行训练以避免直接求解超参数优化的黑盒目标函数,从而搜索到最优超参数组合;

3) 提出数据引导池技术,降低训练方差,提高方法稳定性.

3 基于强化学习的超参数优化方法

3.1 整体结构

针对超参数优化问题(HPO),本文提出了一种基于强化学习的优化方法. 该方法将超参数优化问题抽象为序列决策问题(即每次决策只选择一个超参数)是基于以下原因:

1) 一个复杂问题通常通过分解成多个易于求解的子问题来解决. 由于一个复杂机器学习算法具有巨大的超参数空间,同时进行所有超参数的选择极具困难.

2) 相反的,如果 agent 分步进行超参数选择,整个搜索空间可大大缩小,从而提高搜索效率.

我们将上述的序列决策过程建模为 MDP,即 $M = (\mathbf{S}, \mathbf{A}, \mathbf{P}, R)$:

· $\mathbf{S}$ 表示状态集合,$s_t \in S$,s_t 表示 t 时刻环境的状态,即 agent 的输入;

· $\mathbf{A}$ 表示动作集合,$a_t \in A$,a_t 表示 t 时刻的 agent 选择的动作,即超参数选择;

· $\mathbf{P}$ 表示在当前状态 s 下,执行动作 a 后,环境转移到下一状态的概率. 在 HPO 问题中它是未知的;

· R 表示 reward 函数,$R: \mathbf{S} \times \mathbf{A} \rightarrow R$,R 表示在当前状态 s 下执行动作 a 的奖励值,即为超参数配置的验证集准确度.

Agent 的目标是找到一个策略 $\pi: \mathbf{S} \rightarrow \mathbf{A}$ 使得累积收益最大化. Agent 工作流程如下: 对每一次迭代,agent 以概率 P 为算法模型选择一组超参数 $\boldsymbol{\lambda}$; 然后在训练数据集 $\mathbf{D}_{train}$ 上训练算法模型 M_{λ}; 最后将 M_{λ} 在验证数据集 $\mathbf{D}_{valid}$ 上的准确率作为奖赏值,并利用策略梯度算法[5]来更新策略. 经过多次训练,agent 会以更高的概率选择准确率高的超参数配置. 为了确保该方法具有更好的稳定性,提出了数据引导池以减小训练方差.

3.2 详细设计

3.2.1 Agent 结构设计

根据 3.1 节,我们将超参数优化问题看作一个序列决策问题,即每个时刻针对某个超参数进行选择,因此不同时刻优化了不同的超参数,这样可以大大减少每次决策的搜索空间. 为了更加清晰的说明序列选择超参数的优势,我们将进一步分析超参数优化的搜索空间. 假设一个算法具有 N 个待优化的超参数. 一种简单的方法是将超参数优化问题看作一个多臂机问题(multi-armed bandit problem),直接在整个超参数搜索空间中选择整个超参数配置,则决策的搜索空间为: $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \cdots \Lambda_N$($\times$ 表示笛卡尔乘积). 相反,如果我们将超参数优化问题作为序列决策问题,基于前一次决策顺序的选择每一个超参数,则决策的搜索空间为: $\boldsymbol{\Lambda} = \Lambda_1 \cup \Lambda_2 \cup \cdots \Lambda_N$. 显然,后者能够大大缩减超参数优化问题的搜索空间,从而提高优化效率.

为了适应顺序选择超参数的方法,我们将 agent 设计为自循环的结构. 每次循环时,我们将 agent 上一次的输出作为 agent 下一次的输入,以保持超参数优化的整体性. 同时,由于超参数之间可能存在相关性,也就是每个时刻的选择可能是

相互关联的. 若只将超参数优化问题分步进行,而不考虑超参数之间的内部关系,超参数的优化顺序则会成为一个影响因素. 基于上述特点,我们利用 LSTM 构造了一个强化学习 agent(图 2). 使用 LSTM 网络作为 agent 的核心结构的主要原因在于: LSTM 网络独特的内部设计能够使 agent 保留或遗忘超参数之间的内在联系,从而有利于超参数选择,也避免了由于超参数优化顺序而造成的影响. 尽管 LSTM 网络的训练比较困难,但是 LSTM 网络被认为是解决时序问题的最好结构.

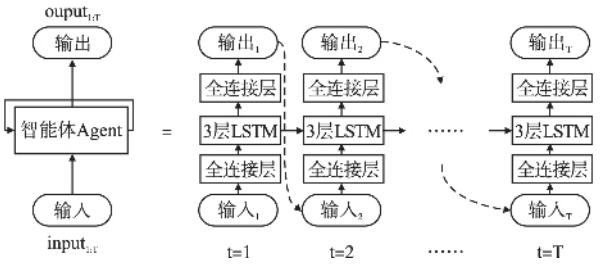

图 2　Agent 结构图

Fig. 2　Structure of agent

图 2 展示了 agent 内部结构,图中左边部分表示 agent 整体结构,右边部分(“=”右)表示按时间步展开的 agent 结构. Agent 的核心结构由 3 层 LSTM 网络构成,且输入、输出与 LSTM 网络之间各有一个全连接层,该全连接层用来调整前后输入和输出的维度. 在每一时刻 t(t∈[1,T],T 为待优化模型的超参数个数),agent 选择一个超参数 a_t,并将 a_t 的 one-hot 编码作为下一时刻 agent 的输入,也就是 t+1 时刻状态 s_{t+1} 为 a_t. 在 t=1 时刻,agent 输入状态 s_1 为全 1 向量.

通过这样的设计,agent 在不同时刻只需选择对应的超参数,减小了超参数的搜索空间. 同时,由于将前一时刻的输出作为下一时刻的输入,使得采用 LSTM 网络作为核心结构的 agent 能够学习超参数之间的潜在关系.

3.2.2　Agent 训练

策略梯度方法[13] 使用逼近器(函数)来近似表示策略,通过不断计算策略期望的总奖赏并基于梯度来更新策略参数,最终收敛于最优策略. 它的优点非常明显: 能够直接优化策略的期望总奖赏,并以端对端的方式直接在策略空间中搜索最优策略,省去了繁琐的中间环节. 因此,本文采用策略梯度方法训练 agent.

假设 θ 表示 agent 的模型参数; R 表示 agent 在每次选择超参数组合 $a_{1:T}$ 后,与所选择的超参数组合结合的待优化模型在验证数据集上的准确率. 定义期望的总奖赏值为:

$$J(\theta)=E_{P(a_{1:T};\theta)}[R] \tag{2}$$

其中,$P(a_{1:T};\theta)$表示表示参数为 θ 的 agent 输出超参数组合 $a_{1:T}$ 的概率.

Agent 的训练目标是找到一个合理的参数 θ 使得期望奖赏值 $J(\theta)$最大化:

$$\max_\theta J(\theta)=\max_\theta E_{P(a_{1:T};\theta)}[R] \tag{3}$$

根据优化算法,通过求解 $J(\theta)$ 的梯度 $\nabla_\theta(J(\theta))$,进而更新参数 θ,最终可求得函数 $J(\theta)$ 的局部最优解. 目标函数的梯度 $\nabla_\theta(J(\theta))$ 为:

$$\nabla_\theta(J(\theta))=E_{P(a_{1:T};\theta)}\left[\sum_{t=1}^{T}\nabla_\theta\log P(a_t|a_{(t-1):1};\theta)R\right] \tag{4}$$

从上式可以看出,目标函数的梯度 $\nabla_\theta(J(\theta))$ 为函数 $\sum_{t=1}^{T}[R\nabla_\theta\log P(a_t|a_{(t-1):1};\theta)]$ 的期望. 该期望通常由模型参数为 θ 的 agent 进行 m 次采样得到,即生成 m 个超参数组合 $a_{1:T}$. 利用 m 次采样的均值作为梯度值 $\nabla_\theta(J(\theta))$ 的无偏估计:

$$\nabla_\theta(J(\theta))\approx\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_\theta\log P(a_t|a_{(t-1):1};\theta)(R_i-b) \tag{5}$$

其中,T 为待优化算法的超参数个数; R_i 为在第 i 个超参数组合下模型的 k-折交叉验证结果; b 是基准值,即模型交叉验证结果的指数移动平均值.

3.2.3　数据引导池(Boot Pool)模块

在使用本文所提出的方法进行超参数优化时,虽然添加了基线函数 b 减小训练误差,但是仍存在训练方差较大的问题,造成其优化结果稳定性较差. 为此,我们提出了数据引导池模块.

数据引导池是一个固定大小的存储区域,用来保存最优的 K 条(top-K)超参数组合及对应奖励值. 在 agent 训练过程中,引导池中的数据会根据新的采样数据进行实时更新,并定期提供给 agent 进行学习. 若 K 过大,则使得引导过强,陷入前期较差的局部最优值; 若 K 过小,则引导力度变弱,策略更多的进行探索,从而导致训练不稳定. 事实上,通过对参数 K 的调整来平衡策略的利用和探索.

4　实验结果及分析

在实验中,我们将随机森林和 XGBoost 两种算法作为超参数优化对象,使用 UCI 数据库中的五个标准数据集作为实验数据集(表 1). 为了验证本文提出方法的性能,我们将本文所提出的方法与随机搜索优化方法、基于贝叶斯的优化方法、TPE 优化方法、CM-AES 优化方法和 SMAC 优化方法进行了对比. 此外,通过一系列消融实验来验证 agent 结构和数据引导池的有效性.

4.1　实验细节

数据集: 实验中,我们选择五个大小各异的 UCI 数据集作为优化任务(详细信息见表 1). UCI 数据集是常用的、种类丰富的数据集. 在实验中,每个数据集按照 8:2 的比例分成训练集和测试集两部分. 实验在训练集下采用 5-折交叉验证的方法训练待优化模型; 训练完成后,使用测试集测试超参数优化方法的最终性能.

参数设置: 在实验中,所有参数均是选择多个随机种子中的最优参数. 针对不同的优化任务,我们设置了不同的学习率 α 和数据引导池大小 K(详细信息见表 1). 基准函数的折扣系数 γ 设置为 0.8. 以 -0.2 与 0.2 之间的随机值对网络中的权重进行初始化.

搜索空间: 实验中我们选择对随机森林(6 个超参数)和 XGBoost(10 个超参数)两种分类算法进行超参数优化(详细信息见表 2),随机森林和 XGBoost 算法的具体实现基于 scikit-learn[15]. 选择上述两种算法进行优化主要是由于:

1) 文献[14]中评估了 179 种机器学习分类算法在 UCI 数据集上的表现,实验结果表明随机森林分类算法是最优的

分类器; XGBoost 算法具有更多的待优化超参数,并且解决分类任务具有很大的潜力;

表 1 数据集信息及对应参数设置表

Table 1 Data sets information and parameter settings

编号	数据集	样本量	特征数	K	学习率
UCI-1	Breast Cancer	569	2	8	0.0007
UCI-2	Optdigits	5,620	64	8	0.0008
UCI-3	Crowdsourced Mapping	10,846	28	8	0.001
UCI-4	Letter Recognition	20,000	16	16	0.001
UCI-5	HTRU_2	17,898	9	8	0.001

2) 两种算法均属于先进的分类算法,广泛应用在数据科学竞赛和工业界.

表 2 随机森林算法和 XGBoost 算法的超参数搜索空间

Table 2 Hyperparameters search spaces of the random forest and the XGBoost

算法	超参数	范围	间隔	类型
Random Forest	n_estimators	[100,1200]	100	int
	max_depth	[3,30]	3	int
	min_samples_split	[0,100]	5	int
	min_samples_leaf	[0,100]	5	int
	max_features	[0.1,0.9]	0.1	float
	bootstrap	True,False	–	bool
XGBoost	max_depth	[3,25]	2	int
	learning_rate	[0.01,0.1]	0.01	float
	n_estimators	[100,1200]	100	int
	gamma	[0.05,1.0]	0.01	float
	min_child_weight	[1,9]	2	int
	subsample	[0.5,1.0]	0.1	float
	colsample_bytree	[0.5,1.0]	0.1	float
	colsample_bylevel	[0.5,1.0]	0.1	float
	reg_alpha	[0.1,1.0]	0.1	float
	reg_lambda	[0.01,1.0]	0.01	float

4.2 Agent 结构的有效性

本小节中,我们将验证 agent 结构的有效性,即验证将超参数优化问题作为序列决策问题的正确性. 实验中,我们所提出的方法简称为 BP-Agent,同时也设计了对比方法 BP-FC: 该方法使用全连接网络(FC)作为 agent 的核心结构,并且直接使用全连接网络一次输出所有超参数的选择,而不是逐步选择超参数. 为了满足对比实验的公平性,我们确保 BP-FC 方法中的全连接网络的可训练参数的数量与本文提出的方法的可训练参数量大致相等. 另外,该方法也采用了引导池技术(BP)来减小训练过程的方差. 为充分利用计算资源,我们在 UCI-(1-4)数据集上进行对比实验,每组对比实验独立执行 3 次,每种优化方法每次独立运行 300 分钟. 实验结果如图 3 和图 4 所示. 图中,分别展示了本文所提出的方法(BP-Agent)和对比方法(BP-FC)在验证集上的训练过程. 我们可以看出: BP-FC 方法使用全连接网络直接输出所有超参数的选择,在部分任务上具有优化效果,但优化效果较差,并且优化效率低; 相比于 BP-Agent 方法,BP-Agent 方法具有更好的优化效果和稳定性,也具有更高的优化效率. 因此,上述实验证明将超参数优化问题序列化. 并逐步选择超参数的 agent 设计有利于提高优化性能.

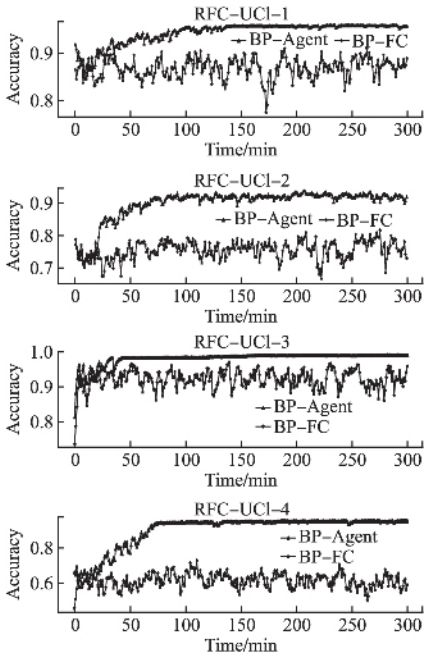

图 3 不同 agent 结构在四个 UCI 数据集上优化随机森林的性能比较图

Fig. 3 Performance comparison of agents with different structures for optimizing Random forests on four UCI datasets

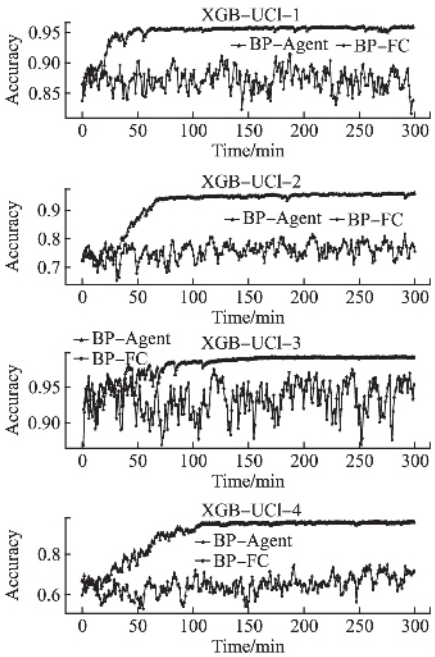

图 4 不同 agent 结构在四个 UCI 数据集上优化 XGBoost 的性能比较图

Fig. 4 Performance comparison of agents with different structures for optimizing XGBoost on four UCI datasets

4.3 数据引导池模块对优化结果的影响

为了验证数据引导池的有效性,我们设计了 BP-Agent 方

法(含有 BP 模块)与 Agent 方法(不含有 BP 模块)的对比实验. 我们在 UCI-(1-4)数据集下对随机森林和 XGBoost 算法的超参数进行优化,每种优化方法在每个优化任务上独立运行 5 次,对比 5 次优化的平均性能.

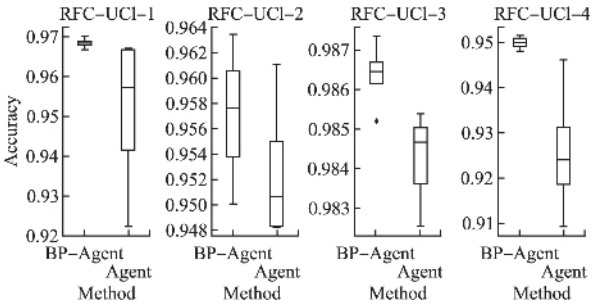

图 5　BP-Agent 和 Agent 方法在四个 UCI 数据集上优化随机森林的性能比较图

Fig. 5　Performance comparison of the BP-Agent and the Agent for optimizing Random forests on four UCI datasets

实验结果以箱型图的形式展示,如图 5 和图 6 所示. 通过观察可以发现: Agent 方法能够达到很好优化效果(即箱型图的中位数),但是其稳定性较差(即箱型图的触须);相比于 Agent 方法,BP-Agent 方法具有更好的优化结果,并且其稳定性较好. 因此,可以得出以下结论: 添加方向引导池能够把握优化方向,增强方法的稳定性.

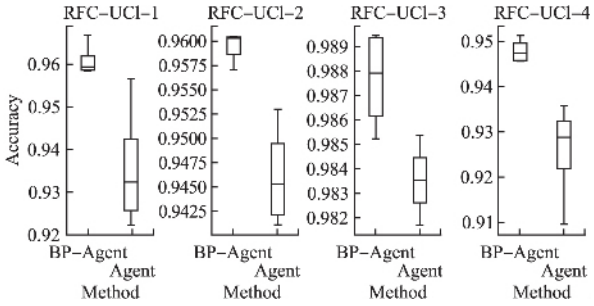

图 6　BP-Agent 和 Agent 方法在四个 UCI 数据集上优化 XGBoost 的性能比较图

Fig. 6　Performance comparison of the BP-Agent and the Agent for optimizing XGBoost on four UCI datasets

4.4　对比 BP-Agent 方法与其他优化方法

为了进一步验证本文所提出的方法,我们将其与常用的且具有代表性的五种优化方法(随机搜索,TPE,贝叶斯优化,CM-AES,SMAC)进行对比. 除此之外,我们也将对比随机森林和 XGBoost 两个算法默认超参数配置的性能,默认的超参数配置基于 scikit-learn[13]. 实验在 UCI-(1-5)数据集上分别优化随机森林和 XGBoost 两个分类算法的超参数,因此共包含 10 个优化任务. 同样的,为充分利用计算资源,每组对比实验独立执行 3 次,每种优化方法每次独立运行 300 分钟. 随机搜索、TPE 和贝叶斯优化三种方法的具体实现基于 Hyperopt[1],CM-AES 方法的具体实现基于 Chocolate[2],SMAC 方法的具体实现基于 SMAC3[3].

对比指标选取的是待优化模型在测试集上的错误率(如表"Err"所示). 实验结果以 3 次对比实验的 Err 平均值和方差进行展示(详细实验结果见表3),不仅能够表示待优化模型在测试集上的准确度,还能够反映优化方法的稳定性. 通过观察表中实验数据,可以看出: 所有的优化方法在大部分优化任务上都能得到优于默认参数性能的超参数配置. 具体的,在 10 个优化任务中,贝叶斯优化、CM-AES 和 SMAC 三种优化方法都达到了很好优化结果,且具有很好的稳定性,而随机搜

表 3　六种超参数优化方法的性能对比表

Table 3　Performance comparison of five HPO optimization methods

数据集	优化算法	随机森林	XGBoost
		Err	Err
UCI-1	随机搜索	0.0774 ± 0.0212	0.0862 ± 0.0198
	TPE	0.0594 ± 0.0149	0.0563 ± 0.0101
	贝叶斯优化	0.0507 ± 0.0061	0.0477 ± 0.0096
	CM-AES	0.0521 ± 0.005	0.0473 ± 0.0083
	SMAC	0.0479 ± 0.016	0.0561 ± 0.037
	BP-Agent	**0.0472 ± 0.0021**	**0.0452 ± 0.0019**
	默认参数	0.0548	0.0523
UCI-2	随机搜索	0.0725 ± 0.0180	0.0443 ± 0.0098
	TPE	0.0562 ± 0.0187	0.0403 ± 0.0078
	贝叶斯优化	0.0553 ± 0.0022	0.0419 ± 0.0028
	CM-AES	0.0561 ± 0.0069	0.0547 ± 0.0041
	SMAC	0.0566 ± 0.0031	0.0434 ± 0.0047
	BP-Agent	**0.0544 ± 0.0015**	**0.0393 ± 0.0016**
	默认参数	0.0811	0.0593
UCI-3	随机搜索	0.0187 ± 0.0191	0.0169 ± 0.0098
	TPE	0.0186 ± 0.0165	0.0179 ± 0.0078
	贝叶斯优化	0.0169 ± 0.0039	0.0160 ± 0.0028
	CM-AES	0.0165 ± 0.0057	0.0167 ± 0.0017
	SMAC	0.0171 ± 0.0103	0.0154 ± 0.0035
	BP-Agent	**0.0160 ± 0.0047**	**0.0151 ± 0.0016**
	默认参数	0.0370	0.0214
UCI-4	随机搜索	0.0520 ± 0.0728	0.0619 ± 0.0250
	TPE	0.1239 ± 0.0702	0.0570 ± 0.0111
	贝叶斯优化	0.0530 ± 0.0294	0.0596 ± 0.0045
	CM-AES	0.0473 ± 0.0069	0.0588 ± 0.0057
	SMAC	**0.0471** ± 0.0061	0.0603 ± 0.0039
	BP-Agent	0.0499 ± **0.0055**	**0.0564 ± 0.0028**
	默认参数	0.1011	0.1293
UCI-5	随机搜索	0.0191 ± 0.0102	0.0204 ± 0.0157
	TPE	0.0196 ± 0.0079	0.0174 ± 0.0103
	贝叶斯优化	0.0153 ± 0.0081	0.0162 ± 0.0076
	CM-AES	0.0141 ± 0.0041	0.0159 ± 0.0044
	SMAC	0.0157 ± 0.0049	0.0160 ± 0.0053
	BP-Agent	**0.0131 ± 0.0036**	**0.0128 ± 0.0039**
	默认参数	0.0219	0.0202

[1] https://github.com/hyperopt/hyperopt-sklearn
[2] https://github.com/AIworx-Labs/chocolate
[3] https://github.com/mlindauer/SMAC3

索和 TPE 两种优化方法的优化性能相对较差; 相比之下,BP-Agent 方法在 8 个优化任务中分别达到了最好的优化结果和稳定性.

另外,我们对实验结果进行统计检验. 假设显著性水平 $\alpha = 0.05$,检验结果显示: 在具有优势的 8 个优化任务中,BP-Agent 的性能提升均具有显著性差异($P < 0.05$) .

上述实验表明本文所提的 BP-Agent 方法能够得到更好优化结果,且具有最好的稳定性.

4.5 讨论与分析

对于超参数优化问题,当前工作主要分类三类: 基础搜索方法[10]、基于采样的方法[15,16]和基于梯度的方法[17-19]. 虽然当前新方法层出不穷,超参数优化问题仍面临以下难点:

1) 优化目标属于黑盒函数. 对于给定任务,超参数选择与性能表现之间的函数无法显式表达.

2) 搜索空间巨大. 由于每种待优化算法都有相应的超参数空间,选择的可能性是指数级的.

3) 耗费巨大的资源. 当评估所选择的超参数配置时,需要进行完整的训练过程并在测试集上测试最终性能,整个优化过程耗费大量计算资源和时间.

通过实验可以看出,本文所提出的方法能够在大部分任务达到最好的优化结果,并具有很好的稳定性. 我们认为主要原因在于: 在超参数选择过程中,由于逐个选择超参数,因此每次选择只需针对当前超参数的搜索空间进行探索,而不需要搜索整个超参数空间,这样可以极大地提高搜索效率; 同时,我们选择 LSTM 网络作为 agent 的核心结构,使 agent 能够在分步决策过程中学习超参数选择的内在联系; 另外,训练过程中添加了数据引导池(BP) 模块,在一定程度上平衡了策略的探索和利用,使得优化方法性能更加稳定.

5 结束语

随着机器学习的广泛应用,快速高效的解决超参数优化问题(HPO) 越来越重要. 针对超参数优化问题(HPO) ,本文提出了一种基于强化学习的超参数优化方法. 该方法将超参数优化问题看作序列决策问题,即将复杂问题分解为多个易于求解的子问题来解决. 进一步将该问题抽象为 MDP,利用强化学习算法来求解该问题. 具体的,以 LSTM 网络为核心构造 agent,逐步为待优化的机器学习算法选择超参数. Agent 的动作(action) 为超参数的选择; agent 的输入,即状态(state) 为上一时刻的动作选择; 待优化算法在验证数据集上的准确率作为奖赏值(reward) .

为了验证所提出方法的有效性,我们选择了五个 UCI 数据集,分别对随机森林和 XGBoost 这两种算法的超参数进行优化. 通过对比随机搜索、TPE、贝叶斯优化、CM-AES 和 SMAC 五种具有代表性的超参数优化方法,我们发现本文提出的方法在优化结果和稳定性上均优于对比方法. 同时,一系列消融实验验证了 agent 结构和数据引导池的有效性.

References:

[1] Breiman L. Random forests [J]. Machine Learning,2001,45(1): 5-32.

[2] Chen T, Guestrin C. XGBoost: a scalable tree boosting system [C] //Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,2016: 785-794.

[3] Cortes C, Vapnik V. Support-vector networks [J]. Machine Learning,1995,20(3): 273-297.

[4] Sepp Hochreiter, Jürgen Schmidhuber. Long short-term memory [J]. Neural Computation,1997,9(8): 1735-1780.

[5] Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning [J]. Machine Learning,1992,8(3-4): 229-256.

[6] Bergstra J, Bengio Y. Random search for hyper-parameter optimization [J]. Journal of Machine Learning Research,2012,13(1): 281-305.

[7] Hansen N. The CMA evolution strategy: a comparing review [M]. Berlin: Springer Press,2006.

[8] Snoek J, Larochelle H, Adams R P. Practical bayesian optimization of machine learning algorithms [C] //Proceedings of Neural Information Processing Systems,2012: 2951-2959.

[9] Klein A, Falkner S, Mansur N, et al. RoBO: a flexible and robust bayesian optimization framework in python [C] //Proceedings of Neural Information Processing Systems Workshop on Bayesian Optimization,2017: 1-5.

[10] Bergstra J, Bengio Y. Random search for hyper-parameter optimization [J]. Journal of Machine Learning Research,2012,13(5): 281-305.

[11] Frank Hutter, Holger H Hoos, Kevin Leyton Brown. Sequential model-based optimization for general algorithm configuration [J]. Learning and Intelligent Optimization,2011,5(8): 507-523.

[12] Bergstra J, Bardenet R, Bengio Y, et al. Algorithms for hyper-parameter optimization [C] //Proceedings of Neural Information Processing Systems,2011: 2546-2554.

[13] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, et al. Scikit-learn: machine learning in python [J]. Journal of Machine Learning Research,2011,12(2): 2825-2830.

[14] Cernadas E, Amorim D. Do we need hundreds of classifiers to solve real world classification problems? [J]. Journal of Machine Learning Research,2014,15(1): 3133-3181.

[15] Marius Lindauer, Frank Hutter. Warmstarting of model-based algorithm configuration [C] //Proceedings of Association for Advancement of Artificial Intelligence,2018: 1355-1362.

[16] Hu Yi Qi, Qian Hong, Yu Yang. Sequential classification-based optimization for direct policy searc [C] //Proceedings of Association for Advancement of Artificial Intelligence,2017: 2029-2035.

[17] Kurutach T, Clavera I, Duan Y, et al. Model-ensemble trust-region policy optimization [C] //Proceedings of International Conference on Learning Representations,2018: 269-278.

[18] Dougal Maclaurin, David Duvenaud, Ryan P Adams. Gradient-based hyperparameter optimization through reversible learning [C] //Proceedings of International Conference on Machine Learning,2015: 2113-2122.

[19] Fabian Pedregosa. Hyperparameter optimization with approximate gradient [C] //Proceedings of International Conference on Machine Learning,2017: 737-746.