DOI:10.7544/issn1000-1239.2020.20190670
57(12):2596-2609,2020

AccSMBO:一种基于超参梯度和元学习的SMBO加速算法

程大宁[1,2] 张汉平[3,5] 夏 粉[3] 李士刚[4] 袁 良[2] 张云泉[2]
[1](中国科学院大学 北京 100190)
[2](中国科学院计算技术研究所 北京 100190)
[3](智铀科技有限公司 北京 100190)
[4](苏黎世理工大学 瑞士苏黎世 8914)
[5](纽约州立大学布法罗分校 纽约 14260)
(chengdaning@ict.ac.cn)

AccSMBO: Using Hyperparameters Gradient and Meta-Learning to Accelerate SMBO

Cheng Daning[1,2], Zhang Hanping[3,5], Xia Fen[3], Li Shigang[4], Yuan Liang[2], and Zhang Yunquan[2]
[1](*University of Chinese Academy of Sciences*, *Beijing* 100190)
[2](*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[3](*Wisdom Uranium Technology Co. Ltd*, *Beijing* 100190)
[4](*Swiss Federal Institute of Technology Zurich*, *Zurich*, *Switzerland* 8914)
[5](*University at Buffalo*, *The State University of New York*, *New York* 14260)

Abstract Current machine learning models require numbers of hyperparameters. Adjusting those hyperparameters is an exhausting job. Thus, hyperparameters optimization algorithms play important roles in machine learning application. In hyperparameters optimization algorithms, sequential model-based optimization algorithms (SMBO) and parallel SMBO algorithms are state-of-the-art hyperparameter optimization methods. However, (parallel) SMBO algorithms do not take the best hyperparameters high possibility range and gradients into considerasion. It is obvious that best hyperparameters high possibility range and hyperparameter gradients can accelerate traditional hyperparameters optimization algorithms. In this paper, we accelerate the traditional SMBO method and name our method as AccSMBO. In AccSMBO, we build a novel gradient-based multikernel Gaussian process. Our multikernel Gaussian process has a good generalization ability which reduces the gradient noise influence on SMBO algorithm. And we also design meta-acquisition function and parallel resource allocation plan which encourage that (parallel) SMBO puts more attention on the best hyperparameters high possibility range. In theory, our method ensures that all hyperparameter gradient information and the best hyperparameters high possibility range information are fully used. In L2 norm regularised logistic loss function experiments, on different scales datasets: small-scale dataset Pc4, middle-scale dataset Rcv1, large-scale dataset Real-sim, compared with state-of-the-art gradient based algorithm: HOAG and state-of-the-art SMBO algorithm: SMAC, our method exhibits the best performance.

收稿日期:2019-09-12;**修回日期**:2020-04-03
基金项目:国家自然科学基金项目(61432018,61521092,61272136,61521092,61502450);国家重点研发计划项目(2016YFB0200803);北京自然科学基金项目(L1802053)
This work was supported by the National Natural Science Foundation of China (61432018, 61521092, 61272136, 61521092, 61502450), the National Key Research and Development Program of China (2016YFB0200803), and the Beijing Natural Science Foundation (L1802053).
通信作者:袁良(yuanliang@ict.ac.cn)

Key words AutoML; SMBO; black box optimization; hypergradient; metalearning; parallel resource allocation

摘 要 为了利用最佳超参高概率范围和超参梯度,提出了加速的序列模型优化算法(sequential model-based optimization algorithms, SMBO)——AccSMBO 算法.AccSMBO 使用了具有良好抗噪能力的基于梯度的多核高斯过程回归方法,利用元学习数据集的 meta-acquisition 函数.AccSMBO 自然对应的并行算法则使用了基于元学习数据集的并行算法资源调度方案.基于梯度的多核高斯过程回归可以避免超参梯度噪音对拟合高斯过程的影响,加快构建较好超参-效果模型的速度.meta-acquisition 函数通过读取元学习数据集,总结最佳超参高概率范围,加快最优超参搜索.在 AccSMBO 自然对应的并行算法中,并行资源调度方法使更多的并行计算资源用于计算最佳超参高概率范围中的超参,更快探索最佳超参高概率范围.上述 3 个技术充分利用超参梯度和最佳超参高概率范围加速 SMBO 算法.在实验中,相比于基于传统的 SMBO 算法实现的 SMAC(sequential model-based algorithm configuration)算法、基于梯度下降的 HOAG(hyperparameter optimization with approximate gradient)算法和常用的随机搜索算法,AccSMBO 使用最少的资源找到了效果最好的超参.

关键词 AutoML 技术;SMBO 算法;黑箱调优算法;超参梯度;元学习;并行资源调度

中图法分类号 TP181; TP302.7

随着机器学习技术的发展,自动机器学习技术成为了机器学习的关键技术之一[1],即 AutoML 技术.当前的机器学习模型的训练需要大量的超参,超参优化是 AutoML 的重要部分.

为了解决上述问题,超参优化成为了一个非常热门的研究方向.近年来研究者提出了不同的超参优化方法,如 Hyperband 算法 SMAC 算法、TPE 算法等.这些算法将历史调参信息或者梯度信息作为输入,使用不同模型对最优超参进行预测,从而迭代求解出最优超参.

基于超参梯度的调优是超参调优的重要手段之一,现有的很多工作已经形式化定义了超参梯度并开发出了相关的调参算法.除了超参梯度,元学习也是现代超参调优的重要研究方向.传统元学习方法依照任务目标.机器学习模型和数据集的不同对最优调参任务进行分类,并记录其最优超参值.每次遇到新的机器学习任务时,都从原有的记录中寻找历史记录中这类情况的最优超参.此外,基于经验的人工调参是现有主流调参手段.相比于调参算法,人工调参可以在机器学习训练中的不同阶段灵活依照经验调参.

序列模型优化算法(sequential model-based optimization algorithms, SMBO)是现在最好的超参优化算法框架[2]之一.然而,作为黑箱优化算法的延伸,传统的 SMBO 算法不考虑已有的超参信息.非常显然的是,合理使用已知的超参信息可以加快黑箱调优算法的收敛速度,提升搜索到的超参的效果.

我们总结出机器学习使用的超参常常满足 2 个超参规律,这 2 个规律往往是在超参调优前可以得到的已知的超参信息:1)超参-效果之间的关系往往是充满“噪音”的单峰或者单调的函数.这种趋势上简单的函数意味着使用传统 SMBO 算法求解超参优化问题时,往往使用较少的采样点即可拟合出总体趋势.但是超参-效果之间的关系在局部上往往充满了“噪音”,这意味着过分依赖局部信息,如梯度,过分依赖局部信息会导致拟合函数对总体趋势的表述变形.2)最优超参的分布情况往往是不均匀的,即有些超参容易取特定的几个值.

基于上述相关的技术和我们总结的规律,针对 SMBO 算法,我们考虑 2 个问题:1)如何使用超参梯度加速调参速度,其中的难点是合理使用超参梯度信息的同时避免局部“噪音”带来的影响;2)如何使用元学习,替代人工调参中经验的成分进而加速调参.

本文提出了 AccSBMO 算法.AccSMBO 使用 3 种方法来优化传统 SMBO 算法.

1) 基于第 1 个超参规律,我们设计了基于梯度的多核高斯过程回归模型来拟合观测值和观测梯度值.基于此,AccSMBO 可以更快地建立完整有效的超参-效果模型(也称作响应面[3]).基于梯度的多核高斯过程回归模型具有良好的抗噪能力和较少的计算负载.

2) 基于第 2 个超参规律,AccSMBO 使用 meta-acquisition 函数替代原本的获取函数.基于元学习

数据集,AccSMBO 寻找出现最佳超参概率较高的范围(称为最佳超参高概率范围),并使用 EPDF (empirical probability density function)表述这一范围.在每次 AccSMBO 迭代中,AccSMBO 算法基于 EPDF 提供的信息,在最佳超参高概率范围内选择预测最优值.

3) 同样基于第 2 个超参规律,我们在并行方法上设计的策略:在 SMBO 自然对应的并行算法中,为了充分使用并行资源,将更多并行计算资源用于探索最佳超参高概率范围,AccSMBO 算法使用了基于元学习数据集的并行资源调配方案.这使得在计算超参效果的过程中充分利用计算资源,减少计算过程中同步操作的消耗,同时将计算资源倾斜在最佳超参高概率范围中.

上述 3 种方法加快了 SMBO 的收敛速度:在实验中,在不同数据集上,从迭代次数上,串行 AccSMBO 算法的收敛速度是 SMAC 算法速度的 175%~330%,是 HOAG 算法的 100%~150%;从绝对时间上,串行 AccSMBO 算法的收敛速度是 SMAC 算法的 167%~329%,是 HOAG 算法的 109%~150%.从绝对时间上,使用 4 个 *worker* 的并行 AccSMBO 算法的收敛速度是 SMAC 算法的 322%~350%,是 HOAG 算法的 145%~316%.从调优结果上,AccSMBO 算法和 SMAC 算法在所有数据集上都找到了最优效果超参,HOAG 算法在小规模数据集上未能收敛.

1 相关工作

近年来,自动机器学习工具相继被开发出来,如谷歌的 AutoML,Auto-Weka,AutoSklearn 等[4].这些工具使用了不同的超参优化算法.

在上述的应用中,研究人员通常在所有参数空间使用网格搜索,但随着超参数量的增加,这种搜索很快变得令人望而却步.

贝叶斯优化算法[2-5]是最常用的用于超参数调优的黑箱优化算法.贝叶斯优化算法通过假设损失函数的先验分布,然后根据新的观测值不断更新该先验分布.每一个新的观测值都是根据一个获取函数计算得来,这个函数负责维持整个探索过程的开发和探索平衡(exploitation and exploration trade off),使得新的观测值能带来更好的结果,或者有助于获得更多关于超参-效果函数的信息.

随机优化方法是现在最常用的超参调优方法,例如网格搜索、随机搜索、启发式算法和神经网络算法[6],但是这类算法效率低下.

傅里叶分析法是近几年发力的超参调优算法,如 Harmonica[7]算法.傅里叶分析法也叫作低阶布尔函数多项式拟合法,这类方法首先随机选取抽样点,再使用正交函数组拟合这些采样点[8-10],进而使用拟合函数预测最优超参值.

决策理论方法考虑的是在现有超参候选集中使用最少的资源选出最佳超参,这类算法在机器学习模型的训练中逐步寻找该模型可能的效果下限,早停效果不好的机器学习模型的训练[11].这种算法最常见的是 Hyperband 算法[12].

在用于超参调优的贝叶斯优化算法[2,4,13]中,SMBO 算法[2]框架使用最为广泛.SMAC[2](sequential model-based algorithm configuration),TPE[14](tree-structured parzen estimator)和基于高斯过程的 SMBO 算法[15]是当前效果最好、求解速度最快的 SMBO 算法实现.

随着计算平台的发展,并行计算环境逐渐成为了机器学习计算环境的主流.并行贝叶斯调优算法也逐渐成为了研究热点[16-19].

如何使用元学习(metalearning)[20]和超参梯度[21-23]加速自动调参优化过程是 AutoML 领域中讨论最多的话题之一.

2 研究背景

2.1 问题定义

本文使用形式化定义超参优化[4]问题:

给定一个由参数 $\boldsymbol{\lambda}$ 所影响的学习算法 $A(\boldsymbol{\lambda})$,和数量有限的数据集 $D=\{(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)\}$,超参优化是寻找最优超参 $\boldsymbol{\lambda}$ 使超参-效果函数 f 取到最小值的任务.f 需要由算法 $A(\boldsymbol{\lambda})$ 和由数据集 D 产生的训练集 D_{train}、验证集 D_{vail} 来计算得出.对超参优化问题的表述可表达为

$$\boldsymbol{\lambda}^* \in \arg\min f(\boldsymbol{\lambda}) \triangleq g(A(\boldsymbol{\lambda}),\boldsymbol{\lambda}) \triangleq \mathcal{L}(A(\boldsymbol{\lambda}),D_{\text{train}},D_{\text{vail}})$$
$$\text{s.t. } A(\boldsymbol{\lambda}) \in \arg\min_{\boldsymbol{model}\in\mathbf{R}^n} h(\boldsymbol{\lambda},D_{\text{train}},\boldsymbol{model}), \quad (1)$$

其中,$g(\cdot)$和 $\mathcal{L}(\cdot)$是真实的机器学习任务的目标,如机器学习模型在验证集上的 AUC(area under curve)或者分类误差率.$h(\boldsymbol{\lambda},D_{\text{train}},\boldsymbol{model})$是训练模型所使用的损失函数,如用于训练逻辑回归模型使用的 log loss 或者用于训练支持向量机的 hingle loss.

2.2 算法背景

2.2.1 SMBO 算法及其最优实现

SMBO 算法是用于超参调优的黑盒优化算法框架.

SMBO 算法在算法 1 中给出.由于 SMBO 算法框架是基于传统的黑箱调优算法,因此它继承了黑箱调优算法的调优条件,即算法不包含有关 $f(\cdot)$ 的任何信息.黑箱调优算法只能够使用历史信息进行最优点推测.每次迭代,SMBO 算法都将上一轮迭代得到的 $(\boldsymbol{\lambda}, f(\boldsymbol{\lambda}))$ 加入到历史信息中,依照历史信息来构建 $f(\boldsymbol{\lambda})$ 的替代函数 $\mathcal{M}_L$(在有些研究中,$\mathcal{M}_L$ 也被称为响应面).基于替代函数 $\mathcal{M}_L$ 寻找到预测最优值进入下一轮迭代[3,5].

算法 1. SMBO 算法.

输入:超参历史 $\mathcal{H}$、初始超参 $\boldsymbol{\lambda}_0$;

输出:超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

① 计算 $f(\boldsymbol{\lambda}_0)$ 并将 $(\boldsymbol{\lambda}_0, f(\boldsymbol{\lambda}_0))$ 加入到 $\mathcal{H}$ 中;

Repeat

② 使用 $\mathcal{H}$ 更新 $\mathcal{M}_L$ 并计算 AC 函数;

③ 根据 $\mathcal{M}_L$ 和 AC 函数计算出候选超参集;

④ $\boldsymbol{\lambda} \leftarrow$ 从③中的超参集中选出最优超参;

⑤ 计算 $f(\boldsymbol{\lambda})$;

⑥ $\mathcal{H} \leftarrow \{\boldsymbol{\lambda}, f(\boldsymbol{\lambda})\}$;

Until 消耗完给定计算资源;

Return 超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

SMBO 算法的核心是通过构建替代模型 $\mathcal{M}_L$ 来捕获各种不同超参对 $f(\cdot)$ 的影响,并使用获取函数(acquisition function, AC)来选择候选超参.

许多研究者提出了不同的算法来填充 SMBO 算法框架.表 1 总结了现有效果较好的算法实现.

Table 1 State of the Art Algorithm for SMBO Algorithm Frame

表 1 SMBO 算法中不同步骤可选择的算法

Algorithm	Step in SMBO
Meta Learning[20,24] Random Selection	Step①, the choice of $\boldsymbol{\lambda}_0$
Random Forest & Gaussian Process(SMAC)[2] TPE[14] Neural Network[6] Fourier Analysis[7] Gaussian Process[15]	Step②, the choice of $\mathcal{M}_L$
Probability of Improvement[25] Expected Improvement[3] d_KG[23] GP Upper Confidence Bound[26]	Step②, the choice of AC function
Hyperband[12] Intensify Process[2]	Step④, the choice of $\boldsymbol{\lambda}$

元学习是提升 SMBO 算法效果的重要手段.基于元学习方法,SMBO 可以在第 1 轮迭代到第 2 轮迭代中取到最优或者次最优的超参值.但是,为了通过元学习取得较好的超参初始值,经常需要花费较多成本构建足够大的元学习数据库.

为了较好地反映出原函数 $f(\boldsymbol{\lambda})$ 的趋势,不同研究提出了不同的响应面模型,但是这些模型并没有考虑如何合理使用梯度信息.在这些模型中,最常使用且效果最好的响应面模型是随机森林和高斯过程[13].使用随机森林和高斯过程的 SMBO 算法实现的是 SMAC 算法.因此本文实验的主要 benchmark 是 SMAC 算法.

现有效果最好的 AC 函数是 EI(expected improvement)函数[3],EI 函数使用期望下降值作为输出,EI 函数是最为常用的 AC 函数的选择.因为我们的加速方法并没有涉及 AC 函数的选择,所以在实验中使用 EI 函数作为 AC 函数.

算法 1 的第④步的主要目的是从有限个超参中使用最少的资源选择出效果最好的超参.这一步中最好的算法是 Hyperband 算法和 intensify process.因为加速算法不涉及如何在这一步选择最优超参,因此在实验中,AccSMBO 算法在这一步使用 intensify process 算法.

2.2.2 代

在 SMBO 算法中,本文称一轮循环为一代(epoch).在每一代中,SMBO 算法都需要做相同的工作:构建模型—选择超参—计算超参效果—写入历史,从而 SMBO 算法在每一代花费的时间基本相同,因此 SMBO 算法中的一代是测量 SMBO 算法性能的最小单位.

在本文的实验中,除了以时间作为衡量指标的实验外,还会增加以代作为实验衡量指标的实验,基于 2 点原因:

1) 因为算法在代码实现和运行平台上的不同,每代所花费的时间在不同情况下也不尽相同,而当初始值和参数设置几乎一致时,调优过程几乎是稳定的,所以在读者复现本实验时,横坐标为代的实验结果可以用较少代价得到复现.

2) 更重要的是,绝大多数算法,如 d_KG 算法[23]和 Hyperband 算法[12]的理论结果、理论分析所用的指标均为迭代次数,即代数.所以使用代作为横坐标可以更好地对应理论结果.

2.2.3 高斯过程回归($\mathcal{GP}$)

高斯过程是现有最好的响应面的选择之一[27].

基于高斯过程的 SMBO 算法是最常用的 SMBO 算法实现之一[15].

在基于高斯过程的 SMBO 算法中,$\mathcal{M}_L$ 是使用 $\mathcal{H}$ 中数据回归得到的高斯过程.在算法 1 中,$\mathcal{H}$ 的形式:

$$\mathcal{H}=\{(\boldsymbol{\lambda}_1,f(\boldsymbol{\lambda}_1)),(\boldsymbol{\lambda}_2,f(\boldsymbol{\lambda}_2)),\cdots,(\boldsymbol{\lambda}_n,f(\boldsymbol{\lambda}_n))\}.$$

本文中,设置 $\boldsymbol{\lambda}$ 包含了 d 个不同的超参.$k(\cdot,\cdot)$是内核函数(协方差函数).

本文中定义向量

$$\boldsymbol{f}=(f(\boldsymbol{\lambda}_1),f(\boldsymbol{\lambda}_2),\cdots,f(\boldsymbol{\lambda}_n))^{\mathrm{T}},$$

其中,$\boldsymbol{f}$ 维度为 n.

定义矩阵 $\boldsymbol{M}_{\boldsymbol{\lambda}}=(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\cdots,\boldsymbol{\lambda}_n)^{\mathrm{T}}$,其维度是 $d\times n$.

定义维度为 $n\times n$ 的内核矩阵 $K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})$:

$$K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})=\begin{pmatrix}k(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_1) & k(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2) & \cdots & k(\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_n)\\ k(\boldsymbol{\lambda}_2,\boldsymbol{\lambda}_1) & k(\boldsymbol{\lambda}_2,\boldsymbol{\lambda}_2) & \cdots & k(\boldsymbol{\lambda}_2,\boldsymbol{\lambda}_n)\\ \vdots & \vdots & & \vdots\\ k(\boldsymbol{\lambda}_n,\boldsymbol{\lambda}_1) & k(\boldsymbol{\lambda}_n,\boldsymbol{\lambda}_2) & \cdots & k(\boldsymbol{\lambda}_n,\boldsymbol{\lambda}_n)\end{pmatrix}.$$

定义 $K(\boldsymbol{\lambda}^*,\boldsymbol{M}_{\boldsymbol{\lambda}})$形式为

$$K(\boldsymbol{\lambda}^*,\boldsymbol{M}_{\boldsymbol{\lambda}})=(k(\boldsymbol{\lambda}^*,\boldsymbol{\lambda}_1),k(\boldsymbol{\lambda}^*,\boldsymbol{\lambda}_2),\cdots,k(\boldsymbol{\lambda}^*,\boldsymbol{\lambda}_n)).$$

在传统无噪音观测值高斯回归中,其期望函数为 $m(\boldsymbol{\lambda}^*)=\sum_{i=1}^{n}\gamma_i k(\boldsymbol{\lambda}_i,\boldsymbol{\lambda}^*)$,即:

$$m(\boldsymbol{\lambda}^*)=\gamma K(\boldsymbol{\lambda}^*,\boldsymbol{M}_{\boldsymbol{\lambda}}),$$

$$\gamma=K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})^{-1}\boldsymbol{f}.$$

协方差函数形式为

$$var(\boldsymbol{\lambda}^*)=k(\boldsymbol{\lambda}^*,\boldsymbol{\lambda}^*)-K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{\lambda}^*)K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})^{-1}K(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{\lambda}^*).$$

基于上述表达,在基于高斯过程的 SMBO 算法中 $\mathcal{M}_L$ 可以表达为 $N(m(\boldsymbol{\lambda}^*),var(\boldsymbol{\lambda}^*))$.

2.2.4 超参梯度

很多工作都提供了基于超参梯度的优化方法[21,22].超参的梯度可以定义和计算[22]:

$$\nabla f=\nabla_2 g+(\nabla A)^{\mathrm{T}}\ \nabla_1 g=\nabla_2 g-(\nabla^2_{1,2}h)^{\mathrm{T}}(\nabla^2_1 h)^{-1}\nabla_1 g.$$

现有很多基于超参梯度的优化方法都是基于梯度下降.

2.2.5 元学习数据库

元学习(metalearning)是加速 SMBO 算法的重要手段.依照分类或者回归任务的不同,数据集特征的不同和目标函数的不同,元学习可以提供针对不同情况的最优或者次优的参考超参.元学习数据集记录了在不同情况下的这些最优超参.SMBO 算法可以将元学习提供的参考超参作为初始值,进一步搜索得到更好的超参,从而加快超参搜索速度.

尽管元学习在 AutoML 中取得了巨大的成功,对于大多数情况,元学习只有在元学习数据集庞大的基础上才能取得足够好的效果.当元学习数据集较小时,元学习对 AutoML 的提升有限.

2.2.6 并行 SMBO 算法

作为贝叶斯优化算法的一种,SMBO 算法也可以先天在并行环境中得到加速.现在最优的贝叶斯优化算法是 2018 年提出的异步并行贝叶斯优化算法[16].这种优化算法使用异步并行的方法避免了集群间同步的开销,在工程上受益.这种异步并行的 SMBO 算法是可以实现在当前主流机器学习框架-参数服务器上的.异步并行的 SMBO 算法在算法 2 中给出.

算法 2. 异步并行 SMBO 算法.

输入:超参历史 $\mathcal{H}$、初始超参 $\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\cdots,\boldsymbol{\lambda}_k$、*worker* 数量 k;

输出:超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

Worker 端:

Repeat

① 计算从 *server* 接收预测最佳超参 $\boldsymbol{\lambda}$;

② 计算 $\boldsymbol{\lambda}$ 对应的 $f(\boldsymbol{\lambda})$;

③ 将$(\boldsymbol{\lambda},f(\boldsymbol{\lambda}))$推回 *server*;

Until 计算资源耗尽.

Server 端:

① 将 $\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\cdots,\boldsymbol{\lambda}_k$ 给 $worker_1,worker_2,\cdots,worker_k$,使得 $worker_1,worker_2,\cdots,worker_k$ 开始计算 $f(\boldsymbol{\lambda}_1),f(\boldsymbol{\lambda}_2),\cdots,f(\boldsymbol{\lambda}_k)$;

Repeat

② 从 $worker_i$ 中接受一个$(\boldsymbol{\lambda}_{\mathrm{pre}},f(\boldsymbol{\lambda}_{\mathrm{pre}}))$组;

③ $\mathcal{H}\leftarrow\{\boldsymbol{\lambda}_{\mathrm{pre}},f(\boldsymbol{\lambda}_{\mathrm{pre}})\}$;

④ 使用 $\mathcal{H}$ 更新 $\mathcal{M}_L$ 并计算 AC 函数;

⑤ 根据 $\mathcal{M}_L$ 和 AC 函数计算出候选超参集;

⑥ $\boldsymbol{\lambda}_{\mathrm{next}}\leftarrow$从⑤中的超参集中选最优超参;

⑦ 将 $\boldsymbol{\lambda}_{\mathrm{next}}$ 发给 $worker_i$ 计算 $f(\boldsymbol{\lambda}_{\mathrm{next}})$;

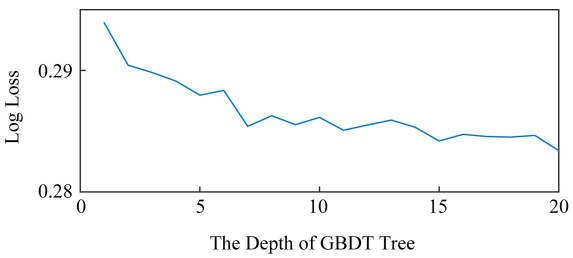
Until 计算资源耗尽;

Return 超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

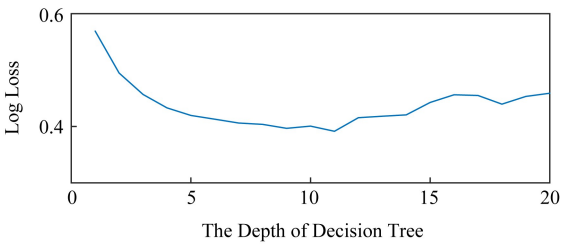
3 出发点和主要思想

对于大多数超参,超参—性能函数 $f(\boldsymbol{\lambda})$经常呈现 2 种模式:1)$f(\boldsymbol{\lambda})$的全局变化趋势简单,$f(\boldsymbol{\lambda})$在全局上经常呈现为单调函数或单峰函数.但是,超参的效果局部变化趋势是不稳定的,即充满了噪音,如图 1 所示.图 1 中展示了在决策树和 GBDT 模型中

不同超参对模型效果(log loss)的 $f(\boldsymbol{\lambda})$.图 1 具体呈现了模式 1).2)最佳超参的分布不是均匀分布.实际调参时,调参人员经常会提前知道一些先验的、合理的范围,即最佳超参高概率存在的范围.很明显,元学习数据集可以反映这些信息.通过总结 Auto-Sklearn 中元学习模块使用的数据集,图 2 展示了在 F1 范式多分类任务中一些参数的频度统计情况与其拟合的概率密度分布函数.图 2 展示了最佳超参分布不均的特点,同时展示了如何通过函数拟合统计值得到 EPDF 函数.



(a) The relationship between the decision tree depth and the log loss on the Rcv1 dataset



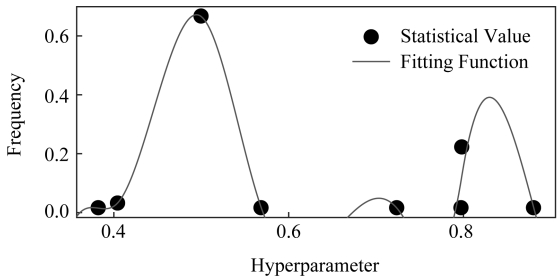
(b) The relationship between the GBDT tree depth and the log loss on the Rcv1 dataset

Fig. 1 Some of those performance functions are close to unimodal functions or monotonic functions

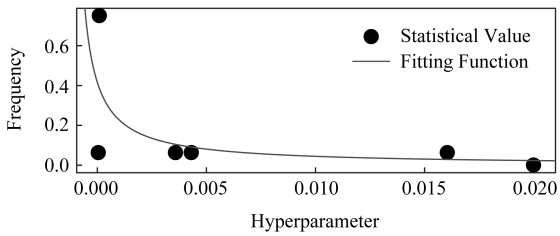
图 1 $f(\boldsymbol{\lambda})$曲线在全局上经常呈现为单峰函数或者单调函数

为了在 SMBO 算法中充分利用模式 1)、模式 2),我们提出了 AccSMBO 算法,串行 AccSMBO 在算法 3 中给出.异步并行 AccSMBO 算法在算法 4 中给出.

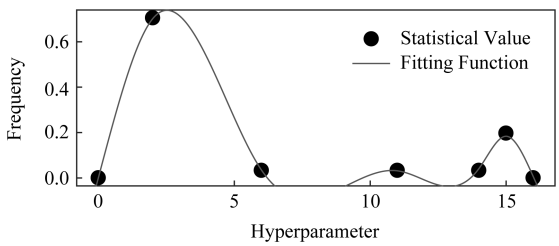
AccSMBO 使用了 2 种方法提升了传统串行 SMBO 算法的效果和一种资源调度方法提升并行 SMBO 算法的效果.1) AccSMBO 使用了基于梯度的多核高斯过程,AccSMBO 使用的高斯过程可以最大程度上利用梯度信息,同时可以避免精细拟合梯度信息而导致噪音对拟合过程的负面作用;2)在迭代过程中,AccSMBO 使用元学习数据集调整 AC 函数,即 metaAC 函数(meta-acquisition functions).metaAC 函数可以在最佳超参高概率范围中有较高



(a) The distribution of "*max_features*" in random_forest and its EPDF



(b) The distribution of "*tol*" in liblinear_svc and its EPDF



(c) The distribution of "*min_samples*" in random_forest and its EPDF

Fig. 2 The frequency and its EPDF of hyperparameter value for F1 norm multi-class task on sparse dataset

图 2 稀疏数据集下 F1 范式多分类任务中超参分布情况及其 EPDF

输出,使得 AccSMBO 算法将更多的搜索资源放在最佳超参高概率范围中,加快搜索到最佳超参的效率;3)根据元学习数据集,我们优化了并行 AccSMBO 算法.使在最佳超参高概率范围中的超参使用更多的计算资源,加快搜索到最佳超参的效率.

算法 3. 串行 AccSMBO 算法.

输入:超参历史 $\mathcal{H}$、初始超参 $\boldsymbol{\lambda}_0$、元学习数据集;

输出:超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

① 计算 $f(\boldsymbol{\lambda}_0)$并将$(\boldsymbol{\lambda}_0, f(\boldsymbol{\lambda}_0))$加入到 $\mathcal{H}$ 中;

Repeat

② 使用 $\mathcal{H}$ 更新基于梯度的多核高斯过程回归 $\mathcal{M}_L$ 并计算 AC 函数;

③ 使用元学习数据集和 AC 函数计算 metaAC 函数;

④ 根据 $\mathcal{M}_L$ 和 metaAC 函数计算出候选超参集;

⑤ $\boldsymbol{\lambda}$←从④中的超参集中选出最优超参;

⑥ 计算 $f(\boldsymbol{\lambda})$;

⑦ $\mathcal{H}\leftarrow\{\boldsymbol{\lambda}, f(\boldsymbol{\lambda})\}$;

Until 消耗完给定计算资源;

Return 超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

算法 4. 异步并行 AccSMBO 算法.

输入:超参历史 $\mathcal{H}$、初始超参 $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_{k+m}$、快栈 $stack_{\text{fast}}$、慢栈 $stack_{\text{slow}}$、对应于 $stack_{\text{fast}}$ 的 $worker_1, worker_2, \cdots, worker_k$、对应于 $stack_{\text{slow}}$ 的 $worker_{k+1}, worker_{k+2}, \cdots, worker_{k+m}$、元学习数据集;

输出:超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

Worker 端:

Repeat

① 计算从对应的栈中接收弹出的预测最佳超参 $\boldsymbol{\lambda}$;

② 计算 $\boldsymbol{\lambda}$ 对应的 $f(\boldsymbol{\lambda})$;

③ 将 $(\boldsymbol{\lambda}, f(\boldsymbol{\lambda}))$ 推回 $server$;

Until 计算资源耗尽.

Server 端:

① 将 $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_{k+m}$ 给 $worker_{k+1}, worker_{k+2}, \cdots, worker_{k+m}$,使得 $worker_1, worker_2, \cdots, worker_{k+m}$ 开始计算 $f(\boldsymbol{\lambda}_1), f(\boldsymbol{\lambda}_2), \cdots, f(\boldsymbol{\lambda}_{k+m})$;

Repeat

② 从 $worker_i$ 中接受一个 $(\boldsymbol{\lambda}_{\text{pre}}, f(\boldsymbol{\lambda}_{\text{pre}}))$ 组;

③ $\mathcal{H}\leftarrow\{\boldsymbol{\lambda}_{\text{pre}}, f(\boldsymbol{\lambda}_{\text{pre}})\}$

④ 使用 $\mathcal{H}$ 更新基于梯度的多核高斯过程回归 $\mathcal{M}_{\text{L}}$ 并计算 AC 函数;

⑤ 使用元学习数据集和 AC 函数计算 metaAC 函数;

⑥ 根据 $\mathcal{M}_{\text{L}}$ 和 metaAC 函数计算候选超参集;

⑦ $\boldsymbol{\lambda}_{\text{next}_1}, \boldsymbol{\lambda}_{\text{next}_2}, \cdots, \boldsymbol{\lambda}_{\text{next}_q}\leftarrow$ 从⑤中的超参集中选 $q(q>2)$ 个最优超参;

⑧ 分别判断 $\boldsymbol{\lambda}_{\text{next}_i}$ 的优先级并压入 $stack_{\text{fast}}$ 或者 $stack_{\text{slow}}$ 中;

Until 计算资源耗尽;

Return 超参历史 $\mathcal{H}$ 中最优超参 $\boldsymbol{\lambda}^*$.

4 AccSMBO 算法

4.1 基于梯度的多核高斯过程

很多研究表明高斯过程是 $\mathcal{M}_{\text{L}}$ 最好的选择之一[23,28].但是传统的基于梯度的高斯过程对梯度信息的噪音非常敏感.带有噪音的梯度信息会导致 SMBO 算法收敛速度变慢.同时,传统基于梯度的高斯过程的计算负载较高.为了减少局部噪音带来的负作用并减少计算负载,我们设计了基于梯度的多核高斯过程.AccSMBO 中使用的基于梯度的多核高斯过程和现有的多核高斯过程在算法和出发点上都有非常明显的不同[23,28].

4.1.1 数学符号

本节主要介绍后文中使用的额外的数学符号.

AccSMBO 算法中使用的 $\mathcal{H}$ 形式为

$$\mathcal{H}=\{(\boldsymbol{\lambda}_1, f(\boldsymbol{\lambda}_1), \nabla f(\boldsymbol{\lambda}_1)), (\boldsymbol{\lambda}_2, f(\boldsymbol{\lambda}_2), \nabla f(\boldsymbol{\lambda}_2)), \cdots, (\boldsymbol{\lambda}_n, f(\boldsymbol{\lambda}_n), \nabla f(\boldsymbol{\lambda}_n))\}.$$

定义向量 $\nabla \boldsymbol{f}$:

$$\nabla \boldsymbol{f}=(\nabla f(\boldsymbol{\lambda}_1)^{\text{T}}, \nabla f(\boldsymbol{\lambda}_2)^{\text{T}}, \cdots, \nabla f(\boldsymbol{\lambda}_n)^{\text{T}})^{\text{T}}.$$

$\nabla \boldsymbol{f}$ 的维度是 $d\times n$.

定义规模为 $n\times n$ 的内核矩阵 $K(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})$ 和内核梯度矩阵 $\nabla K(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})$.

其中,$\nabla K(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})$ 定义:

$$\nabla K(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})=\begin{pmatrix} \nabla k(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_1) & \nabla k(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) & \cdots & \nabla k(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_n) \\ \nabla k(\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_1) & \nabla k(\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_2) & \cdots & \nabla k(\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_n) \\ \vdots & \vdots & & \vdots \\ \nabla k(\boldsymbol{\lambda}_n, \boldsymbol{\lambda}_1) & \nabla k(\boldsymbol{\lambda}_n, \boldsymbol{\lambda}_2) & \cdots & \nabla k(\boldsymbol{\lambda}_n, \boldsymbol{\lambda}_n) \end{pmatrix}.$$

本文中,将矩阵的组合简写为

$$\boldsymbol{K}_{m:n}=(\boldsymbol{K}_m; \boldsymbol{K}_{m+1}; \cdots; \boldsymbol{K}_n).$$

4.1.2 基于梯度的多核高斯过程回归

本文设计了一种新的基于梯度的高斯过程回归方法,这种算法在算法 5 中给出.

算法 5. 基于多核高斯过程回归算法.

输入:包含 $\boldsymbol{\lambda}$ 和 $\boldsymbol{f}$ 的超参历史 $\mathcal{H}$、内核矩阵 $\boldsymbol{K}_1, \boldsymbol{K}_2, \cdots, \boldsymbol{K}_n$;

输出:均值函数为 $m(\boldsymbol{\lambda}^*)$、方差函数为 $var_i(\boldsymbol{\lambda}^*)$ 的高斯过程 $\mathcal{N}\left(\sum_{i=1}^{d+1} m_i(\boldsymbol{\lambda}^*), \sum_{i=1}^{d+1} var_i(\boldsymbol{\lambda}^*)\right)$.

① 使用最小二乘法求解式(5)中的 $\boldsymbol{\alpha}$ 和 $\boldsymbol{\beta}$:

$$(\nabla K_{2:n}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})- K_{2:n}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})K_1^{-1}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})\nabla K_1(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}}))\boldsymbol{\alpha}= \nabla \boldsymbol{f}-K_{2:n}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})K_1^{-1}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})\nabla K_1(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{f}$$

$$K_1(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\beta}=\boldsymbol{f}-K_{2:n}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}; \quad (2)$$

② $m(\boldsymbol{\lambda}^*)=K_{2:n}(\boldsymbol{\lambda}^*, \boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}+K_1(\boldsymbol{\lambda}^*, \boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\beta}$;

③ $var(\boldsymbol{\lambda}^*)=\sum_{i=1}^{n} k_i(\boldsymbol{\lambda}^*, \boldsymbol{\lambda}^*)- K_i(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}^*)K_i^{-1}(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{M}_{\boldsymbol{\lambda}})K_i(\boldsymbol{M}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}^*)$;

Return $\mathcal{N}\left(\sum_{i=1}^{d+1} m_i(\boldsymbol{\lambda}^*), \sum_{i=1}^{d+1} var_i(\boldsymbol{\lambda}^*)\right)$.

在算法 5 中,AccSMBO 从 $\mathcal{H}$ 中抽取出 $(\boldsymbol{\lambda},$

$f(\boldsymbol{\lambda}),\nabla f(\boldsymbol{\lambda}))$.AccSMBO 希望构建出一个高斯过程回归,使得其均值的梯度尽可能逼近观测值.

算法 5 的本质是多个高斯过程的线性组合,即 $\mathcal{M}_{\mathrm{L}}=\mathcal{GP}_{\mathrm{combine}}=\mathcal{GP}k_1+\mathcal{GP}k_2+\cdots+\mathcal{GP}k_{d+1}$,其中 $\mathcal{GP}k_i$ 是指使用 $k_i(\cdot,\cdot)$作为内核回归得到的高斯过程.相比于传统高斯过程回归,算法 5 通过增加内核的数量进而增多了高斯过程回归的自由度,即 $\boldsymbol{\alpha}$ 是 $d+1$ 维向量.这些额外的维度帮助高斯过程回归能够拟合梯度的观测值.

4.1.3 均值函数

在传统的高斯过程回归中,均值函数 $m(\boldsymbol{\lambda})$是使用内核函数作为基底对观测点的拟合函数.

在算法 5 中,AccSMBO 线性组合了线性无关的内核,即式(3)、(4).当内核函数的数量等于 $\nabla\boldsymbol{f}$ 的维度时,均值函数就能准确地反映观测值和观测的梯度值.

$$f=K_1(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_1+K_2(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_2+\cdots+K_{d+1}(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_{d+1},\tag{3}$$

$$\nabla f=\nabla K_1(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_1+\nabla K_2(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_2+\cdots+\nabla K_{d+1}(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_{d+1},\tag{4}$$

通过式(3)、(4)可解得 $\boldsymbol{\alpha}_1,\boldsymbol{\alpha}_2,\cdots,\boldsymbol{\alpha}_{d+1}$,则

$$m(\boldsymbol{\lambda}^*)=\sum_{i=1}^{d+1}K_i(\boldsymbol{\lambda}^*,\boldsymbol{M}_{\boldsymbol{\lambda}})\boldsymbol{\alpha}_i.$$

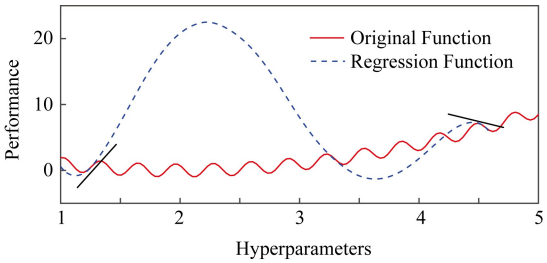
4.1.4 方差函数

多核高斯过程的方差反映了预测点 $\boldsymbol{\lambda}^*$ 与观测点 $\boldsymbol{\lambda}_i$ 之间的距离.对于使用 $k_i(\cdot,\cdot)$作为内核的高斯过程,其方差函数的计算方法为 $var_i(\boldsymbol{\lambda}^*)=k_i(\boldsymbol{\lambda}^*,\boldsymbol{\lambda}^*)-K_i(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{\lambda}^*)K_i(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{M}_{\boldsymbol{\lambda}})^{-1}K_i(\boldsymbol{M}_{\boldsymbol{\lambda}},\boldsymbol{\lambda}^*)$.因为高斯过程的和依旧为高斯过程,和的高斯过程的方差是不同高斯过程方差的和,所以多核高斯过程回归的数学形式为

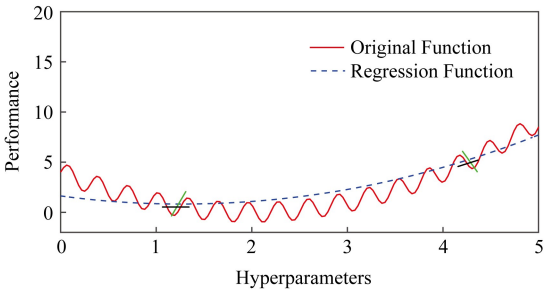
$$\mathcal{N}\Big(\sum_{i=1}^{d+1}m_i(\boldsymbol{\lambda}^*),\sum_{i=1}^{d+1}var_i(\boldsymbol{\lambda}^*)\Big).$$

4.1.5 抗噪、约近和计算负载的减少

虽然径向基核函数(radial basis function, RBF)满足了拟合梯度的要求:提供了足够多线性无关的内核函数,但计算负载的减少和抗噪能力的增加也是考量的重点之一.正如第 3 节中提到的,从整体上,$f(\boldsymbol{\lambda})$相对简单,但曲线的局部充满了噪音.梯度反映的是曲线的局部信息,而不是 $f(\boldsymbol{\lambda})$曲线的总体趋势.对梯度信息的精确拟合会对高斯过程回归产生负面影响,并且增加计算量:如当历史超参的梯度因为噪音与总趋势相反时,梯度信息的精确引入会对高斯过程回归产生非常显著的负面影响,如图 3 所示.



(a) Regression function with accurate gradient fitting and original function



(b) Regression function with approximate gradient fitting and original function

Fig. 3 Regression function with different accurate gradient and original function

图 3 使用不同精准度的梯度的拟合函数与原函数

AccSMBO 算法可以通过减少核函数的数量来减少梯度噪音带来的影响.在实践中,噪音越多,使用的内核数量就越少.在求解方程时,式(3)(4)的地位是不同的.在大多数情况下,噪音对梯度方程的影响会更大,如图 3 所示.因此,我们期望 $\boldsymbol{\alpha}$ 和 $\boldsymbol{\beta}$ 可以满足式(3),然后使用最小二乘法来求解式(4)除此之外,减少内核可以减少计算的负载.

因此,考虑到 $f(\boldsymbol{\lambda})$曲线的性质和计算负载,上述基于梯度的多核高斯过程回归更适合于解决超参优化的问题.

4.2 metaAC 函数

现有 SMBO 算法都是在使用者对目标函数毫无所知的情况下设计的.但是在真实情况下,算法使用者对超参都有一些了解.元学习数据集可以反映出这些信息.合理使用这些信息可以提升超参调优的速度.基于这些信息,本文提出了 metaAC 函数对 SMBO 算法进行加速.

4.2.1 metaAC 函数及其收敛性

使用 metaAC 函数需要 2 个步骤:1)为了使得 metaAC 函数在最佳超参高概率范围的输出较高,需要根据元学习数据集总结出最佳超参高概率范围.

2)依照最佳超参高概率范围,在保证收敛性的前提下调整 AC 函数.

最佳超参的概率密度分布函数是最佳超参高概率范围的表达方式之一.AccSMBO 算法使用最佳超参的概率密度分布函数对 AC 函数进行调整.AccSMBO 根据元学习数据集中的信息构建一个经验概率密度分布函数:首先,根据元学习数据集中的信息统计出不同超参的频率;然后,使用函数依照频率分布的信息拟合得到概率密度分布函数 $p(\boldsymbol{\lambda})$(简写为 EPDF).

构建最佳超参的概率密度分布函数并不要求元学习数据集包含不同情况下所有的最优超参值,只需要元学习数据集能够反映出最优超参分布的趋势即可.此时 $p(\boldsymbol{\lambda})$就可以表达出最佳超参高概率范围信息.$p(\boldsymbol{\lambda})$的输出区域越高,则这一超参越有可能成为最优超参.

在设计 metaAC 函数时,metaAC 应具有 2 个性质:1)在迭代优化的过程中,$p(\boldsymbol{\lambda})$输出高的区域应具有更高的优先权,这样有更大概率更快地找到最优超参.2)随着算法的进行,$p(\boldsymbol{\lambda})$的影响应该递减.过度依赖 $p(\boldsymbol{\lambda})$会严重破坏 AC 函数的开发和探索平衡(exploitation and exploration trade off),从而导致随着迭代地进行 AccSMBO 算法无法描绘出目标函数的全局形式.

基于上述的性质,本文设计了 metaAC 函数:

$$\text{metaAC}(\boldsymbol{\lambda},epoch,p(\boldsymbol{\lambda}),AC(\boldsymbol{\lambda}))=AC(\boldsymbol{\lambda})\times(rate\times p(\boldsymbol{\lambda})\times e^{-epoch}+1-rate\times e^{-epoch})$$

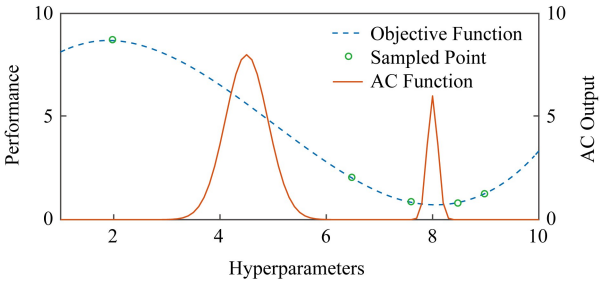
其中,$rate\in[0,1]$.$rate$ 主要用于描述元学习数据集是否足够完整和算法使用者想多大程度依赖于元学习数据集进行加速.元学习数据集越大越完整时,$rate$ 应越大.

metaAC 函数实际上是传统 AC 函数的调整,而且可以看到随着迭代地进行,metaAC 函数将会收敛于传统的 AC 函数.换句话说当在资源充足的情况下,metaAC 函数的收敛性证明等同于未被调整过的 AC 函数的证明,即 metaAC 是可以保证算法收敛的.

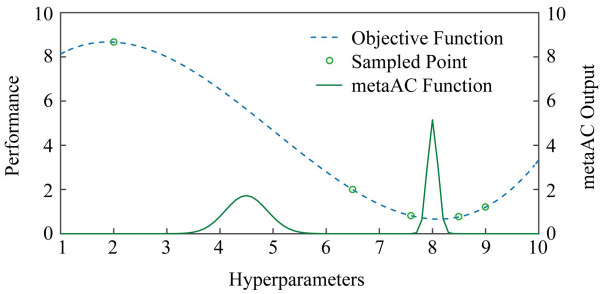
在实际的生产中,遇到的最优超参在最优超参高概率范围内的情况更多一些.从图 4(a)和图 4(b)中可以看到,metaAC 函数比起 AC 函数更侧重于搜索最优超参高概率范围,从而起到加速算法的效果.

4.2.2 元学习的缺陷和 metaAC 的优势

元学习是当前提升超参调优的重要手段.但是



(a) Original objective function and AC function



(b) Original objective function and metaAC function

Fig. 4 The change of AC function to metaAC function

图 4 从 AC 函数调整到 metaAC 函数

元学习技术存在 2 个问题:1)当前元学习技术只能给超参调优提供最优/次优的初始值.元学习技术不能在调参过程中对调优算法产生影响.2)元学习数据集一定要非常大.如果元学习数据集较小,对于元学习数据集中没有被记录的任务,元学习就难以提供较好的参考超参.同时,因为 AC 函数会进行开发探索平衡,在提供较好的初始值之后,AC 函数会选择在未探索区域中进行探索,而往往未探索的区域存在最优超参的可能性比较低,如图 4(a)所示,对于没有被采样过的区域[3,6],因为没有被探索过,AC 函数的输出更高.

metaAC 函数可以克服上述的 2 个缺点:1)metaAC 函数参与到了自动调优的过程中;2)metaAC 函数的构造并不需要非常完整的元学习数据集.metaAC 函数只需要元学习数据集反映出整体的趋势即可.

4.3 异步并行 AccSMBO 算法

并行算法的高效优化一直是并行计算领域关注的重要问题.传统对贝叶斯优化算法的并行加速是一个成熟的领域.但是基于第 3 节中固有模式和元学习数据集,可以进一步加速异步并行 AccSMBO 算法.

因为最佳超参在最佳超参高概率范围中出现的概率较高,因此,将更多资源投入到最佳超参高概率范围中,基于 EPDF 提供的最佳超参高概率范围,

异步并行 AccSMBO 算法可以将更多资源用于探索最佳超参高概率范围中的参数,将 $f(\boldsymbol{\lambda})$在最佳超参高概率性范围中的情况尽快描绘出来.

AccSMBO 使用了图 5 所示的算法.在设计并行资源调度算法时,AccSMBO 设置 2 个栈 $stack_{\text{fast}}$ 和 $stack_{\text{slow}}$.新产生的预测点包含了更多关于 $f(\boldsymbol{\lambda})$的信息.因此总是优先计算最新更新的预测点,即使用栈数据结构.每次 *worker* 计算 $\boldsymbol{\lambda}$ 时,都从对应的栈中弹出的最新的 $\boldsymbol{\lambda}$ 用于计算效果.不同的 *worker* 使用异步并行的方法进行并行计算.异步并行极大地减少了同步操作带来的巨大计算开销,是现在最好的并行模式之一.

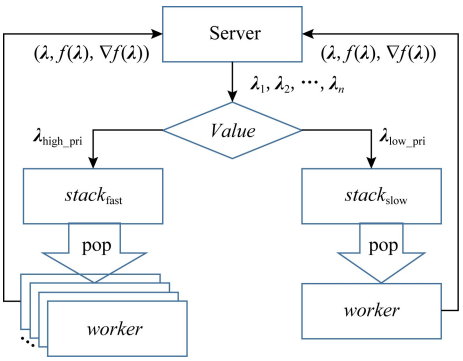


Fig. 5 The parallel workflow of AccSMBO

图 5 AccSMBO 的并行工作模式

为了并行加速 SMBO,AccSMBO 将更多的计算资源放在最佳超参高概率范围中的预测超参,即将最佳超参高概率范围中的预测超参放在高优先级队列中.但是考虑到算法收敛性,即保证开发和探索平衡(exploitation and exploration trade off),这种资源分配的不平衡应该随着迭代的进行而逐渐减弱.基于此,使用和 metaAC 函数同样的设计思路,AccSMBO 使用 *Value* 函数完成上述目的.

首先 AccSMBO 需要依照传统挑选参数的算法对候选参数集中超参的挑选顺序付给初始优先级,如 intensify process 或者 Hyperband 算法.其中优先级越高,输出越大.

例如,$\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \cdots, \boldsymbol{\lambda}_k$ 按照 $1, 2, \cdots, k$ 的顺序被 Hyperband 选出,则 $pri(\boldsymbol{\lambda}_i)=i$,$pri(\boldsymbol{\lambda})$越高,优先级越高.即:

$pri(\boldsymbol{\lambda})$=选择算法选出 $\boldsymbol{\lambda}$ 的顺序.

基于 $pri(\boldsymbol{\lambda})$,这里使用 metaAC 函数的设计思路设计 *Value* 函数:

$$Value(\boldsymbol{\lambda}, epoch, p(\boldsymbol{\lambda}), pri(\boldsymbol{\lambda}))=$$
$$pri(\boldsymbol{\lambda})\times(rate\times p(\boldsymbol{\lambda})\times e^{-epoch}+1-rate\times e^{-epoch}).$$

基于 *Value* 函数进行降序排序之后得到的序列作为候选超参的优先级.选取 10%的低优先级超参进入 $stack_{\text{slow}}$ 栈,其余超参放入 $stack_{\text{fast}}$ 栈中.

异步并行 AccSMBO 算法的优化本质是对传统异步并行 SMBO 算法的调整.随着算法的逐步进行,算法 5 中的算法会退化为传统异步并行贝叶斯优化算法,即能保证算法收敛.

与 metaAC 函数的加速原理一样,实际问题中,调参人员遇到的最优超参在最优超参高概率范围内的情况更多一些.AccSMBO 侧重于搜索最优超参高概率范围,从而起到加速算法的效果.

5 实 验

在本文中,实验使用 HOAG 算法的实验框架[22]进行实验.

5.1 问题设置

5.1.1 数据集

首先,本实验选择不同规模的数据集进行训练,包含了小规模数据集(OpenML 数据库中的 Pc4 数据集[29])、中型规模数据集(LibSVM 数据库中的 Rcv1 数据集[30])、大规模数据(LivSVM 数据库中的 Real-sim 数据集[22])进行实验.这些数据集的具体情况在表 2 中给出.

Table 2 Dataset Information

表 2 数据集信息

Dataset	# Feature	# Size	Feature Range	Sparsity
Real-sim	20 958	72 309	(0,1)	Sparse
Rcv1	47 236	20 242	(0,1)	Sparse
Pc4	38	1 458	(0,10 000)	Dense

在实验中,本实验抽取 30%的数据作为验证集,剩下的数据作为训练集.

5.1.2 目标问题

在实验中使用超参调优算法搜索逻辑回归模型中 2 阶正则参数.本实验选择这个参数作为实验参数是因为这个参数较容易求其超参梯度[21].在这个实验中,外层优化的目标为 log loss,这里选择 log loss 是因为 log loss 克服了传统 0-1 损失带来的非凸问题和不连续问题[21].内层优化目标 $h(\boldsymbol{\lambda})$是 2 阶正则化的 logistic 损失函数.目标函数的形式化表达为

$$\arg\min_{\boldsymbol{\lambda}\in\Lambda} f(\boldsymbol{\lambda})=\sum_{i=1}^{m}\Phi((\boldsymbol{x}_i y_i)^{\mathrm{T}}A(\boldsymbol{\lambda})),$$
$$\text{s.t. } A(\boldsymbol{\lambda})\in\arg\min_{\boldsymbol{model}\in\mathbf{R}^n} h(\boldsymbol{model})=$$
$$\sum_{i=1}^{m}\Phi((\boldsymbol{x}_i y_i)^{\mathrm{T}}\boldsymbol{model})+\boldsymbol{\lambda}\|\boldsymbol{model}\|^2, \quad (5)$$

其中,$\Phi(t)=\ln(1+e^{-t})$.在求解内层优化问题时,本实验使用了一种拟牛顿法:L-BFGS(limited-memory BFGS)算法,作为优化算法[31].其中 L-BFGS 算法代码经过了工业级代码优化,使用在相关广告 CTR 预测产品中.

5.1.3 内核函数

在实验中,超参的维度是 1;因此本实验选择 2 个内核:高斯径向基函数和 3 阶径向基函数,即:

$$k_2(\boldsymbol{x}_1,\boldsymbol{x}_2)=e^{-\|\boldsymbol{x}_1-\boldsymbol{x}_2\|^2}$$

和

$$k_1(\boldsymbol{x}_1,\boldsymbol{x}_2)=\|\boldsymbol{x}_1-\boldsymbol{x}_2\|^3.$$

在本实验中,需要对矩阵 $\boldsymbol{K}_2$ 求逆矩阵,基于对精准度的考虑,本实验选择高斯径向基函数对应的矩阵进行求逆矩阵操作.

5.1.4 元学习数据集

现有唯一开源的元学习数据集是 AutoSklearn 中的元学习数据集.此元学习数据集由 OpenML 数据集库构建.为了使实验更有说服力,实验中随机删除了这个数据集中 20%的内容,并将这份数据集命名为半元学习数据集.半元学习数据集不能用于提供最佳参考超参.但是,因为半元学习数据集保留了整体超参分布的趋势,所以,半元数据集仍然可以用于构建 metaAC 函数.

5.1.5 算 法

本实验比较了 5 种现有的超参优化方法.为了使收敛过程更清晰,所有算法在选择初始值时关闭了元学习功能,并将初始值都置为 1.

1) SMAC 算法.SMAC 算法是现有最好的 SMBO 算法实现之一.SMAC 算法被使用在了 AutoSklearn 和 Auto-Weka 等自动调优工具中.在每一步中,AC 函数选出 4 个超参作为备选超参.再使用 intensify process 从 4 个超参中选择效果最好的超参.本实验选择 *EI* 函数作为 *AC* 函数.

2) AccSMBO 算法.AccSMBO 算法代码是使用 AutoSklearn 代码修改而来.在实验中,设置 $rate=1$.为了使得 $p(\boldsymbol{\lambda})$更准确,本实验中,AccSMBO 算法先将任务按照目标函数(log loss)、任务(二分类)和数据集特性(稀疏/稠密)分类,统计分类后超参的分布情况.与 SMAC 算法一样,每代 *AC* 函数选择 4 个超参作为备选超参.在使用 intensify process 从 4 个超参中选择效果最好的超参.本实验选择 *EI* 函数作为 *AC* 函数.

3) 网格搜索.网格搜索是现在最常用的超参搜索方法之一.在实验中,网格搜索算法从区间[0,1]中均匀取出 20 个超参,每一轮从 1 开始,逐个计算一个超参的效果.

4) 随机搜索.随机搜索是现在最常用的超参搜索方法之一.在实验中,随机搜索算法从区间[0,1]中随机取出 20 个超参,每一轮计算一个超参的效果.

5) HOAG(hyperparameter optimization with approximate gradient)算法.HOAG 算法是现在最优的使用梯度求解最优超参的算法.每一代,超参都会朝着梯度下降的方向进行调整.本实验直接使用开源的 HOAG 算法代码作为实验代码.在实验过程中为了保证算法收敛,实验设置 $\varepsilon_k=10^{-12}$.为了保证算法的收敛速度并且保证算法能够收敛到最小值,本实验预先采样了 $f(\boldsymbol{\lambda})$曲线中的点,基于采样点的梯度估测在实验中步长最大为10^{-3}.

6) 并行 AccSMBO 算法 PaccSMBO.多线程 AccSMBO 算法.在实验中,并行 AccSMBO 算法使用 4 个 *worker* 组成的系统.在算法设置上,多线程 AccSMBO 算法与串行 AccSMBO 算法一致.在并行设置上,3 个 *worker* 分配给 $stack_{\text{fast}}$,1 个 *worker* 分配给 $stack_{\text{slow}}$.为了使不同 *worker* 具有不同的初始值,不同 *worker* 从[0,1]间网格地抽取使用初始值,即初始值为 0.25,0.5,0.75,1.实验中,这种算法被称为 PaccSMBO.

上面的 5 个算法分别代表了常用的搜索算法(随机搜索、网格搜索),基于梯度的算法(HOAG 算法)和其他 SMBO 算法实现(SMAC 算法).

5.2 实验环境和实验信息

本实验的实验服务器硬件环为:

服务器使用 2 核 Intel® Xeon® CPU E5-2680 2.88 GHz,共有 24 个内核.同时服务器使用最大内存为 60 GB.

服务器使用网卡为 Intel Corporation I350 Gigabit Network Connection.

本实验的实验服务器软件环境为:

操作系统为 CentOS 7.5.1804 系统,C/C++编译器版本为 gcc 7.3.

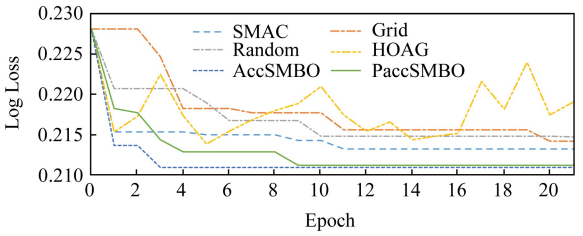
Python 版本为 Python 3.7.5,SMAC 的版本为 SMAC-0.8.0,参数服务器使用 MXNet(版本为 1.1.0)中的参数服务器模块.

5.3 实验结果和分析

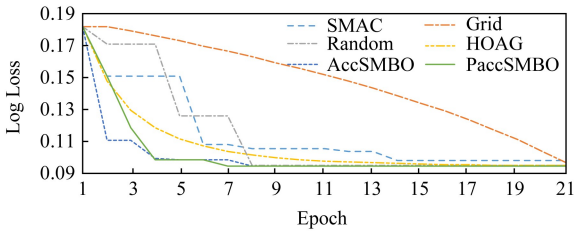
在串行算法的比较中,从代数上看,串行 AccSMBO 算法的收敛速度最快,同时搜索到了最好的超参,而 SMAC 找到了次优的超参.串行 AccSMBO

算法的收敛速度是 SMAC 的 330%.由于当数据集规模较小时,$f(\boldsymbol{\lambda})$曲线不稳定,$f(\boldsymbol{\lambda})$曲线的梯度信息充满了噪音,不能够指示曲线的走向.因此会导致 HOAG 算法的收敛速度减慢.在本实验中,HOAG 并没有达到收敛.对于并行的 AccSMBO 算法(PaccSMBO 算法),尽管从代数上 PaccSMBO 算法没有占有优势,但因为是 4 个 worker 同时运行,所以绝对时间上少于串行 AccSMBO 算法:串行 AccSMBO 算法使用 3 代达到收敛,PaccSMBO 算法使用 9 代达到收敛,平均每个 worker 运行时间为 2 代.从绝对时间上看,串行 AccSMBO 的收敛速度是 SMAC 的 329%,由于通信的损耗,并行 AccSMBO 算法(PaccSMBO 算法)几乎同时与串行 AccSMBO 算法收敛.

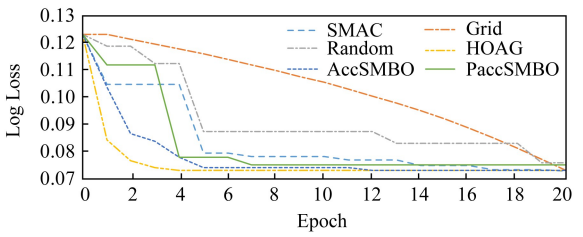
在 Pc4 实验中,由于 HOAG 算法使用精确的梯度作为优化方向,在这种情况下无法找到最佳的超参.黑箱优化算法在这里显示了它的优势,因为它减少了曲线中的"噪声"的影响,更快地逼近梯度.图 6(b)和图 7(b)显示了 Rcv1 数据集的实验结果.同样地,在串行算法的比较中,从代数上看,串行 AccSMBO 算法的收敛速度最快,并找到了最佳的超参.在 Rcv1 数据集中,$f(\boldsymbol{\lambda})$曲线特性相对稳定,接近单峰函数,所以 HOAG 算法找到的超参和收敛速度是次优的.AccSMBO 算法的收敛速度是 HOAG 算法的 150%,是 SMAC 算法的收敛速度的 175%.SMAC 算法得到的结果和收敛速度几乎与随机搜索相同,因为 SMAC 需要更多的采样点来构建整个 $f(\boldsymbol{\lambda})$的信息.并行 AccSMBO 算法(PaccSMBO 算法)的收敛过程虽然在绝对的代数上比串行 AccSMBO 算法多,但是平均到每个 worker,其消耗的代数会比串行 AccSMBO 更少,绝对时间会更快.从绝对时间上看,串行 AccSMBO 算法是 HOAG 收敛速度的 150%,是 SMAC 算法的 167%.PaccSMBO 算法的收敛速度是 HOAG 算法的 316%,是 SMAC 算法的 350%.



(a) Algorithm performances on the Pc4 dataset



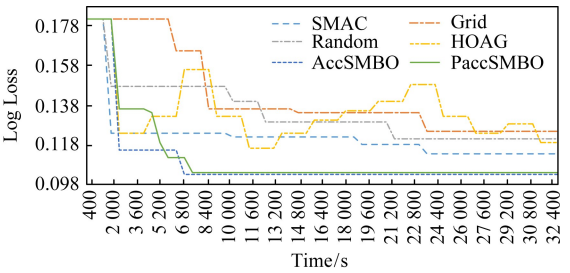
(b) Algorithm performances on the Rcv1 dataset



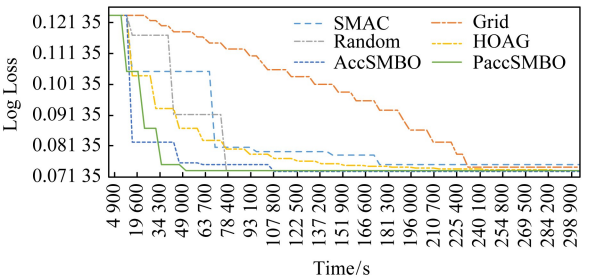
(c) Algorithm performances on the Real-sim dataset

Fig. 6 Algorithm iteration performances on the different datasets

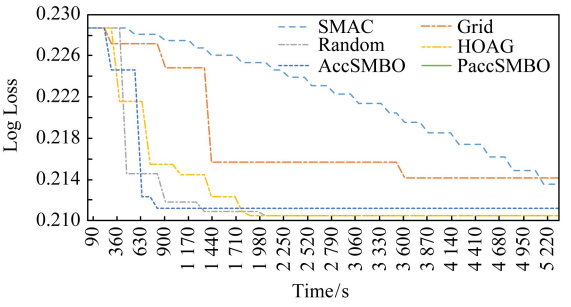
图 6 不同算法在不同数据集上的迭代性能



(a) Algorithm performances on the Pc4 dataset



(b) Algorithm performances on the Rcv1 dataset



(c) Algorithm performances on the Real-sim dataset

Fig. 7 Algorithm time performances on the different dataset

图 7 不同算法在不同数据集上的时间性能

图 6(c)和图 7(c)展示了不同算法在 Real-sim

数据集上的实验结果.在串行算法的比较中,从代数上看,HOAG 算法和串行 AccSMBO 算法都得到了最快的收敛速度并找到最佳超参.串行 AccSMBO 算法的收敛速度是 SMAC 算法的 240%.并行 AccSMBO 算法(PaccSMBO 算法)在收敛的代数上虽然在绝对的代数上比串行 AccSMBO 算法多,但是平均到每个 worker,其消耗的代数会比串行 AccSMBO 更少,绝对时间会更快.从绝对时间上看,串行 AccSMBO 算法的收敛速度是 HOAG 算法收敛速度的 109%,是 SMAC 算法收敛速度的 260%.PaccSMBO 算法的收敛速度是 HOAG 算法的 145%,是 SMAC 算法的 347%.

6 结 论

本文提出了 AccSMBO 算法.AccSMBO 算法通过 3 种方法取得了当前最优的效果:1)使用基于梯度的多核高斯过程回归来构建 $\mathcal{M}_L$;2)使用了 metaAC 函数;3)并行 AccSMBO 算法使用了基于元学习的资源调度方法.在实验中,AccSMBO 算法达到了最好的性能:在 Pc4,Rcv1,Real-sim 数据集上的收敛速度和超参搜索结果都优于 SMAC 和 HOAG 等算法.

参 考 文 献

[1] Yu Kai, Jia Lei, Chen Yuqiang, et al. Deep learning: Yesterday, today, and tomorrow [J]. Journal of Computer Research and Development, 2013, 50(9): 1799-1804 (in Chinese)
(余凯, 贾磊, 陈雨强, 等. 深度学习的昨天、今天和明天[J]. 计算机研究与发展, 2013, 50(9): 1799-1804)

[2] Hutter F, Hoos H H, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration [C] // Proc of LION's 11. Berlin: Springer, 2011: 507-523

[3] Jones D R, Schonlau M, Welch W J. Efficient global optimization of expensive black-box functions [J]. Journal of Global Optimization, 1998, 13(4): 455-492

[4] Thornton C, Hutter F, Hoos H H, et al. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms [C] //Proc of KDD's 13. New York: ACM, 2013: 847-855

[5] Regis R G, Shoemaker C A. Constrained global optimization of expensive black box functions using radial basis functions [J]. Journal of Global Optimization, 2005, 31(1): 153-171

[6] Mendoza H, Klein A, Feurer M, et al. Towards automatically-tuned neural networks [C] //Proc of ICML's 16. New York: ACM, 2016: 58-65

[7] Hazan E, Klivans A R, Yuan Y, et al. Hyperparameter optimization: A spectral approach [EB/OL]. (2018-01-20) [2019-11-26]. https://arxiv.org/abs/1706.00764

[8] Kocaoglu M, Shanmugam K, Dimakis A G, et al. Sparse polynomial learning and graph sketching [C] //Proc of NIPS'14. Cambridge, MA: MIT Press, 2014: 3122-3130

[9] Negahban S, Shah D. Learning sparse boolean polynomials [C] //Proc of Communication, Control, and Computing's 12. Piscataway, NJ: IEEE, 2012: 2032-2036

[10] Linial N, Mansour Y, Nisan N. Constant depth circuits, Fourier transform, and learnability [J]. Journal of the ACM, 1993, 40(3): 607-620

[11] Jamieson K, Talwalkar A. Non-stochastic best arm identification and hyperparameter optimization [C] //Proc of AISTATS'16. Cambridge, MA: MIT Press, 2016: 240-248

[12] Li Lisha, Jamieson K, Desalvo G, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization [J]. Journal of Machine Learning Research, 2016, 18(1): 1-52

[13] Feurer M, Klein A, Eggensperger K, et al. Efficient and robust automated machine learning [C] //Proc of NIPS'15. Cambridge, MA: MIT Press, 2015: 2755-2763

[14] Bergstra J, Bengio Y. Algorithms for hyper-parameter optimization [C] //Proc of NIPS'11. Cambridge, MA: MIT Press, 2011: 2546-2554

[15] Snoek J, Larochelle H, Adams R P. Practical Bayesian optimization of machine learning algorithms [C] //Proc of NIPS'12. Cambridge, MA: MIT Press, 2012: 2951-2959

[16] Kandasamy K, Krishnamurthy A, Schneider J, et al. Parallel Bayesian optimization via thompson sampling [C] // Proc of AISTATS'18. Cambridge, MA: MIT Press, 2018

[17] Contal E, Buffoni D, Robicquet A, et al. Parallel Gaussian process optimization with upper confidence bound and pure exploration [C] //Proc of ECML's 13. Berlin: Springer, 2013: 225-240

[18] Desautels T, Krause A, Burdick J W. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization [J]. Journal of Machine Learning Research, 2012, 15(1): 827-834

[19] Kathuria T, Deshpande A, Kohli P. Batched Gaussian process bandit optimization via determinantal point processes [C] //Proc of NIPS'16. Cambridge, MA: MIT Press, 2016: 377-384

[20] Brazdil P, Carrier C G G, Soares C, et al. Metalearning-Applications to Data Mining [M]. New York: Cognitive Technologies, 2009

[21] Foo C, Do C B, Ng A Y, et al. Efficient multiple hyperparameter learning for log-linear models [C] //Proc of NIPS'07. Cambridge, MA: MIT Press, 2007: 377-384

[22] Pedregosa F. Hyperparameter optimization with approximate gradient [C] //Proc of ICML's 16. New York: ACM, 2016: 737-746

[23] Wu Jian, Poloczek M, Wilson A G, et al. Bayesian optimization with gradients [C] //Proc of NIPS'17. Cambridge, MA: MIT Press, 2017: 5267-5278

[24] Lemke C, Budka M, Gabrys B. Metalearning: A survey of trends and technologies [J]. Artificial Intelligence Review, 2015, 44(1): 117-130

[25] Kushner H J. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise [J]. Journal of Basic Engineering, 1964, 86(1): 97-106

[26] Grunewalder S, Audibert J, Opper M, et al. Regret bounds for Gaussian process bandit problems [C] //Proc of AISTATS'10. Cambridge, MA: MIT Press, 2010: 273-280

[27] Rasmussen C E, Williams C K I. Gaussian Processes for Machine Learning [M]. Cambridge, MA: MIT Press, 2005

[28] Jie Yu, Chen Kuilin, Rashid M M. A Bayesian model averaging based multi-kernel Gaussian process regression framework for nonlinear state estimation and quality prediction of multiphase batch processes with transient dynamics and uncertainty [J]. Chemical Engineering Science, 2013, 93(4): 96-109

[29] Shirabad, Jelber S, Tim M. Predictor models in software engineering (PROMISE) [C] //Proc of ICSE's 05. Piscataway, NJ: IEEE, 2005: 692-692

[30] Lewis D D, Yang Yiming, Rose T G, et al. Rcv1: A new benchmark collection for text categorization research [J]. Journal of Machine Learning Research, 2004, 5(2): 361-397

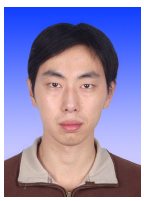
[31] Boyd S, Vandenberghe L. Convex optimization [J]. IEEE Transactions on Automatic Control, 2006, 51(11): 1859-1859

Cheng Daning, born in 1990. PhD. His main research interests include parallel stochastic optimization algorithm, parallel black-box optimization algorithm and machine learning.

Zhang Hanping, born in 1994. PhD candidate. His main research interests include automl, deep learning, and optimization algorithm. (zhanghanping@ebrain.ai)

Xia Fen, born in 1982. PhD. His main research interests include automl machine learning, rangking, large scale machine learning algorithms, regularization methods, and information retrieval. (xiafen@ebrain.ai)

Li Shigang, born in 1986. PhD. Member of CCF. His main research interests include parallel mahine learning algorithm and parallel software. (shigangli.cs@gmail.com)



Yuan Liang, born in 1984. PhD and assistant professor. Member of CCF. His main research interests include parallel algorithm and parallel software.

Zhang Yunquan, born in 1973. PhD and professor. Fellow member of CCF. His main research interests include parallel algorithm andparallel software, large scala parallel numerical software, parallel computation model, performance benchmark and optimization. (zyq@ict.ac.cn)