软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(7):2499−2524 [doi: 10.13328/j.cnki.jos.006583]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

采用多目标优化的深度学习测试优化方法*

沐燕舟[1], 王 赞[1,3], 陈 翔[2], 陈俊洁[1], 赵静珂[3], 王建敏[4]

[1](天津大学 智能与计算学部, 天津 300350)

[2](南通大学 信息科学技术学院, 江苏 南通 226019)

[3](天津大学 新媒体与传播学院, 天津 300072)

[4](中国科学院 空间应用工程与技术中心, 北京 100094)

通信作者: 王赞, E-mail: wangzan@tju.edu.cn

摘 要: 随着深度学习技术的快速发展, 对其质量保障的研究也逐步增多. 传感器等技术的迅速发展, 使得收集测试数据变得不再困难, 但对收集到的数据进行标记却需要花费高昂的代价. 已有工作尝试从原始测试集中筛选出一个测试子集以降低标记成本, 这些测试子集保证了与原始测试集具有相近的整体准确率(即待测深度学习模型在测试集全体测试输入上的准确率), 但却不能保证在其他测试性质上与原始测试集相近. 例如, 不能充分覆盖原始测试集中各个类别的测试输入. 提出了一种基于多目标优化的深度学习测试输入选择方法 DMOS (deep multi-objective selection), 其首先基于 HDBSCAN (hierarchical density-based spatial clustering of applications with noise)聚类方法初步分析原始测试集的数据分布, 然后基于聚类结果的特征设计多个优化目标, 接着利用多目标优化求解出合适的选择方案. 在8组经典的深度学习测试集和模型上进行了大量实验, 结果表明, DMOS 方法选出的最佳测试子集(性能最好的 Pareto 最优解对应的测试子集)不仅能够覆盖原始测试集中更多的测试输入类别, 而且对各个类别测试输入的准确率估计非常接近原始测试集. 同时, 它还能保证在整体准确率以及测试充分性上的估计也接近于原始测试集: 对整体准确率估计的平均误差仅为 1.081%, 比最新方法 PACE (practical accuracy estimation)减小了 0.845%的误差, 提升幅度为 43.87%; 对各个类别测试输入的准确率估计的平均误差仅为 5.547%, 比最新方法 PACE 减小了 2.926%的误差, 提升幅度为 34.53%; 对 5 种测试充分性度量的平均估计误差仅为 8.739%, 比最新方法 PACE 减小了 7.328%的误差, 提升幅度为 45.61%.

关键词: 深度学习; 软件测试; 测试输入选择; 多目标优化; 遗传进化

中图法分类号: TP311

中文引用格式: 沐燕舟, 王赞, 陈翔, 陈俊洁, 赵静珂, 王建敏. 采用多目标优化的深度学习测试优化方法. 软件学报, 2022, 33(7): 2499–2524. http://www.jos.org.cn/1000-9825/6583.htm

英文引用格式: Mu YZ, Wang Z, Chen X, Chen JJ, Zhao JK, Wang JM. Deep Learning Test Optimization Method Using Multi-objective Optimization. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2499–2524 (in Chinese). http://www.jos.org.cn/1000-9825/6583.htm

Deep Learning Test Optimization Method Using Multi-objective Optimization

MU Yan-Zhou[1], WANG Zan[1,3], CHEN Xiang[2], CHEN Jun-Jie[1], ZHAO Jing-Ke[3], WANG Jian-Min[4]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[2](School of Information Science and Technology, Nantong University, Nantong 226019, China)

[3](School of New Media and Communication, Tianjin University, Tianjin 300072, China)

* 基金项目:国家自然科学基金(61872263); 基础加强计划技术领域基金(2020-JCJQ-JJ-490); 2020 年天津市智能制造专项资金(20201180)

本文由“智能系统的分析和验证”专题特约编辑明仲教授、张立军教授和秦胜潮教授推荐.

收稿时间: 2021-09-05; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

[4](Technology and Engineering Center for Space Utilization, Chinese Academy of Sciences, Beijing 100094, China)

Abstract: With the rapid development of deep learning technology, the research on its quality assurance is raising more attention. Meanwhile, it is no longer difficult to collect test data owing to the mature sensor technology, but it costs a lot to label the collected data. In order to reduce the cost of labeling, the existing work attempts to select a test subset from the original test set. They only ensure that the overall accuracy (the accuracy of the target deep learning model on all test inputs of the test set) of the test subset is similar to that of the original test set. However, existing work only focuses on estimating overall accuracy, ignoring other properties of the original test set. For example, it can not fully cover all kinds of test input in the original test set. This study proposes a method based on multi-objective optimization called DMOS (deep multi-objective selection). It firstly analyzes the data distribution of the original test set based on HDBSCAN (hierarchical density-based spatial clustering of applications with noise) clustering method. Then, it designs the optimization objective based on the characteristics of the clustering results and then carries out multi-objective optimization to find out the appropriate selection solution. A large number of experiments are carried out on 8 pairs of classic deep learning test sets and models. The results show that the best test subset selected by DMOS method (corresponding to the Pareto optimal solution with the best performance) can not only cover more test input categories in the original test set, but also estimate the accuracy of each test input category extremely close to the original test set. Meanwhile, it can also ensure that the overall accuracy and test adequacy are close to the original test set: The average error of the overall accuracy estimation is only 1.081%, which is 0.845% less than the PACE (practical accuracy estimation), with the improvement of 43.87%. The average error of the accuracy estimation of each category of test input is only 5.547%, which is 2.926% less than PACE, with the improvement of 34.53%. The average estimation error of the five test adequacy measures is only 8.739%, which is 7.328% lower than PACE, with the increase improvement of 45.61%.

Key words: deep learning; software testing; test input selection; multi-objective optimization; genetic evolution

随着深度学习(deep learing, DL)技术的迅猛发展, 基于大量数据训练好的深度神经网络(deep neural network, DNN)模型在越来越多的应用领域中(例如自动驾驶[1]、人脸识别[2]、语音识别[3,4]、医学诊断[5]、飞机防撞系统[6]、软件工程[7−10]等)取得优异的性能[11]. 但是, DNN 模型仍然像传统软件系统那样存在缺陷. 由于其内部结构复杂, 数据中任何微小的扰动都有可能导致 DNN 模型作出难以理解的错误预测, 这在安全攸关的领域内将会导致严重的后果. 例如, 2018 年, 一名行人在亚利桑那州坦佩被一辆 Uber 自动驾驶汽车撞击身亡(https://www.vice.com/en/article/9kga85/uber-is-giving-up-on-self-driving-cars-in-california-after-deadly-crash). 此外, 一辆处于自动驾驶模式的特斯拉 S 型汽车于 2018 年在加州高速公路上撞上了一辆闪着灯的消防车(https://www.newsweek.com/autonomous-tesla-crashes-parked-fire-truck-california-freeway-789177). 因此, 保证 DNN 模型的质量至关重要. 软件测试作为一种传统的质量保障方法, 可以有效检测出待测系统中的内在缺陷, 因此, 针对 DNN 模型设计有效的测试方法是近些年来深度学习和软件测试这两个领域的研究热点.

与传统软件测试任务一致, 深度学习测试的目标同样是快速且充分地暴露出待测 DNN 模型内的缺陷. 但由于 DNN 模型基于数据驱动方式进行建模, 其预测的最终结果由训练过程获得的各个神经元权重以及测试输入所决定, 因此, 对深度学习测试而言, 维护一个高质量的测试集是实现上述目标的重要保障. 随着传感器等技术的高速发展, 较短时间内收集到大量数据变得不再困难, 但这同时也大幅度提高了 DNN 模型的测试代价. 例如, Facebook 的人脸识别系统 deepface[12]使用了大约 22 万张人脸图像进行测试. DeepTest[13]使用了 254 221 张图像, 对基于 CNN 的自动驾驶模型进行了测试. 流行的 ImageNet 数据集[14]包含 100 000 个测试图像(即共 1 000 个分类, 每一类包含 100 个图像), 用于测试各种图像分类模型. 大量的测试数据可以保证 DNN 模型得到充分的测试, 但也会消耗大量的测试时间. 此外, 收集到的大量原始数据需要正确标记才能投入测试, 当前, 手工标记是主要的方法. 为了保证标记的正确性, 通常需要多个用户协同完成标记任务, 这造成了大量人力物力的消耗. 此外, 一些涉及专业领域的测试数据(例如来自医学领域的影像图片), 则需要领域专家进行标记, 其标记成本又会大幅度增加. 因此, 我们不难发现: 冗余的测试数据不仅会浪费大量的标记成本, 还将大幅度降低测试的效率. 在与企业的合作中, 我们发现, 测试集的标记问题目前已成为困扰他们测试 DNN 模型的严峻挑战之一. 因此, 维护一个小规模且具备良好测试能力的测试集, 已经成为当前深度学习测试领域的重要研究问题. 深度学习测试输入选择方法的目标就是在原始无标签的测试集中挑选出有代表性的(即能够保持原始测试集性质)的测试输入数据, 随后对构建出的测试子集进行标记, 以降低整体标记及测试

执行的成本.

目前, 针对该问题的研究还处于起步阶段, Li 等人[15]首先提出了 CSS 方法(confidence-based stratified sampling), 其利用待测 DNN 模型对各个测试输入预测的置信度进行区间划分, 按照一定比例对各区间的样本进行采样, 直到达到用户指定的标记数量为止. 然而, CSS 方法仅适用于准确率较高的模型. 随后, Li 等人提出了基于交叉熵的采样方法 CES (cross entropy-based sampling)[15]. 其首先对原始测试集进行随机采样, 然后不断迭代, 以缩小测试子集与原始测试集之间的交叉熵为优化目标, 直到达到指定的终止阈值为止. 与 CSS 方法相比, CES 方法的泛化性能更强, 但该方法受到随机性的影响, 其稳定性较差, 因此难以应用于实际场景. Zhou 等人[16]提出了一种基于双目标的两阶段选择方法 DeepReduce, 其首先基于神经元覆盖指标, 通过贪心策略选出一个测试子集, 并抽取待测 DNN 模型的最后一层神经元输出作为每个输入的特征表示; 然后, 以最小化测试子集与原始测试集之间的相对熵作为优化目标, 从原始测试集中启发式地选择样本加入测试子集中, 直到达到指定阈值为止. Chen 等人[17]也提出一种实用准确度评估方法 PACE (practical accuracy estimation), 其首先通过 HDBSCAN (hierarchical density-based spatial clustering of applications with noise)聚类方法[18]对原始测试集进行聚类, 然后分别利用基于 MMD-critic 的原型采样方法[19]和自适应随机选择方法[20], 按照相应比例对聚类结果采样, 并最终合并生成测试子集. 实验结果表明, PACE 能够有效约减原始测试集的规模, 并准确估计待测 DNN 模型的整体准确率. 即 PACE 只需选择原始测试集中 2%左右的测试输入样本, 即可达到与原始测试集非常接近的整体准确率, 平均误差仅为 1.181%−2.302%. 然而, PACE 并不能保证选出的测试子集能够覆盖原始测试集中不同类别的测试输入, 并且对各个类别测试输入的准确率估计存在较大误差. 例如, 实验结果显示, 在含有 100 个类别的测试集 CIFAR100 上, 当采样率设置为 1.5%时, PACE 选出的测试子集将会遗漏原始测试集中 28%的测试输入类别, 且对于待测 DNN 模型 ResNet20, PACE 选出的测试子集在各个类别测试输入上, 准确率估计的平均误差高达 43%.

选出的新测试子集如果仅仅只是从整体准确率上与原始测试集接近, 极有可能导致测试子集不能充分覆盖到原始测试集中不同类别的测试输入, 丧失了在原始测试集上对待测 DNN 模型的其他测试特性. 评估一个测试子集的质量, 需要考察其是否能够全面地具备原始测试集多方面的属性和特征, 因此, 我们希望选出的测试子集不仅应该要充分覆盖原始测试集中的各个类别的测试输入, 并且能精准估计对应类别下测试输入的准确率(即达到原始测试集中接近的准确率水平). 为了达成此目标, 本文提出了一种基于多目标优化的深度学习测试输入选择方法 DMOS (deep multi-objective selection). 具体来说, DMOS 方法首先利用待测 DNN 模型抽取原始测试集中各个测试输入的特征, 并利用 HDBSCAN 聚类方法[18]进行分层分簇, 以尽可能保证采样时的质量; 然后, 将原始测试集与测试子集在聚类生成的每个簇上的类别分布差异作为优化目标, 结合 NSGA-II[21]这一经典的多目标优化算法进行求解. 在用户指定的选择数量下, 如果选出的测试输入样本偏向于某一个簇, 那必将导致其他簇中样本的减少, 进而使得选出的测试子集中其他簇所占比例与原始测试集中的所占比例间的差异变大, 因此可以看出这些优化目标之间存在着明显的冲突. 在 DMOS 方法中, 最终会返回非支配解集, 其中每个解代表一种具体的选择方案, 其构成元素代表该位置的测试输入样本被选中的权重. 用户可以根据自己对测试需求的偏好选择合适的解, 以形成最终的子集并加以标记, 节省标记及测试成本. 在保证整体准确率的基础上, 也使得选择出的测试子集不仅能够更充分地覆盖原始测试集中不同类别的测试输入, 而且对各个类别测试输入的准确率估计也非常精准.

我们在 8 组经典的深度学习测试集和 DNN 模型组合构建出的实验对象上展开了实证研究. 这 8 组实验对象中的 DNN 模型均用于分类任务, 从整体准确率角度看, 这些模型可细分为高精度模型(即整体准确率超过 0.8)和低精度模型(即整体准确率不超过 0.8); 从网络的层次结构角度来看, 这些模型可细分为 CNN 模型(convolutional neural network)和 RNN 模型(recurrent neural network). 此外, 实验对象中的数据集按样本类型分, 可分为语音数据和图片数据. 实验结果表明, DMOS 方法选出的测试子集, 无论从整体准确率估计还是从各个类别测试输入的准确率估计来看, 都要显著优于最新的选择方法 PACE[17]; 此外, DMOS 选出的测试子集具备与原始测试集更为相似的测试充分性. 具体来说, DMOS 方法对整体准确率估计的平均误差仅为 1.081%,

比最新方法 PACE 减小了 0.845%的误差, 提升幅度为 43.87%; 对各个类别测试输入的准确率估计的平均误差仅为 5.547%, 比最新方法 PACE 减小了 2.926%的误差, 提升幅度为 34.53%; 对 5 种测试充分性度量的平均估计误差仅为 8.739%, 比最新方法 PACE 减小了 7.328%的误差, 提升幅度为 45.61%.

综上所述, 本文的主要贡献可总结如下:

(1) 我们首次将深度学习测试输入选择问题建模成多目标优化问题, 提出了 DMOS 方法加以解决, 其利用 HDBSCAN 聚类方法[18]和经典的多目标优化算法 NSGA-II[21]进行求解. 实验结果表明, DMOS 方法最终返回的非支配解集可以确保选出的测试子集能够尽可能多地覆盖原始测试集中的测试输入类别, 并对各个类别测试输入的准确率进行精准估计. 该方法可以有效降低针对 DNN 模型的测试成本, 并提高测试效率.

(2) 我们在 8 组经典的 DNN 模型和深度学习测试集上进行了深入的研究, 验证了 DMOS 方法在不同类型的数据集和模型上的有效性.

(3) 我们基于 Keras2.3.1 和 TensorFlow1.15.0 等框架实现了 DMOS 方法, 并将其共享到了网址 https://github.com/EzioQR/DMOS2021, 以方便其他研究者开展后续研究.

1 背景知识

1.1 深度学习测试及优化

DNN 模型由多层组成, 每层包含大量神经元[22]. 层与层之间的神经元通过带有权值的链接进行连接, 而这些链接的权重值基于训练数据和训练过程计算得出. DNN 模型基于这些计算出的权重, 将输入映射到输出. 目前, DNN 模型主要分为卷积神经网络(CNN)和循环神经网络(RNN). CNN 涉及卷积计算, 通常用于处理具有网格状拓扑结构的数据(如图像)[23], 而 RNN 模型通常会基于已有数据计算得出的信息进行处理决策, 因此更适合处理变化的时序信息, 例如语音、自然语言等包含序列信息的数据[24]. 软件测试是保证 DNN 模型质量的一种常用方法[25,26], 与传统软件系统一样, DNN 模型测试也涉及测试输入和测试预言. DNN 模型测试中的测试输入是指收集到的各类语音、图像和文本等测试模型性能的数据. DNN 模型测试中的测试预言主要是由人工标记而得的类别标签. 也就是说, 每个测试输入都需要人工标记其基本事实. 通过比较标记的实际类型和预测类型, 研究人员可以判定 DNN 模型是否正确地预测了测试输入.

然而, 对收集到的大量测试输入进行标记的代价是极其高昂的. 为了提高 DNN 模型测试的效率, 研究人员提出了两类优化方法.

- 第 1 类方法从测试输入选择入手.

这类方法的目的是选择一个小规模测试子集, 以精确估计出测试集整体准确率. 用户可以只标记这些测试输入来完成测试工作, 有效节省标记成本. 较早的研究工作是 Li 等人[15]提出的 CSS 方法和 CES 方法: CSS 方法通过待测 DNN 模型对原始测试集中各个测试输入的预测置信度排序然后划分区间, 根据指定的选择数量分层采样合并成最终的测试子集; CES 方法则基于待测 DNN 模型的最后一个隐藏层的输出, 通过不断缩小测试子集和原始测试集之间的交叉熵来选择测试输入. Zhou 等人[16]提出了一种两阶段顺序采样的选择方法 DeepReduce, 该方法首先利用神经元覆盖率作为指导规则, 并基于贪心策略从原始测试集中选出一个测试子集; 然后基于待测 DNN 模型最后一层的输出不断缩小测试子集与原始测试集之间的相对熵, 直到达到指定的选择数量为止. Chen 等人[17]最近提出了一种基于聚类的实用精度估计方法 PACE, 该方法通过待测 DNN 模型抽取的测试输入特征进行聚类, 针对聚类完成后的正常点簇和异常点簇, 分别利用基于 MMD-critic 的原型采样方法[19]和自适应随机选择方法[20]对两种簇分别进行采样, 将各个簇中取出的测试输入合并成最终的测试子集.

- 第 2 类方法从测试输入排序入手.

这类方法的目的是根据待测 DNN 模型的预测错误概率, 对所有测试输入进行排序. 与第 1 类方法相比, 这类方法不需要丢弃任何测试输入, 而且用户可以更早地发现会使模型产生错误预测行为的测试输入. Feng

等人[27]提出了一种测试输入排序方法 Deepgini, 其基于待测 DNN 模型对测试输入预测的置信度来计算出基尼纯度[28]并完成排序. Zhang 等人[29]提出通过计算测试输入的噪声敏感性来对测试输入进行排序, 他们发现: 通过在测试输入中加入相同的噪声, 噪声灵敏度高的测试输入比噪声灵敏度低的测试输入更容易欺骗 DNN 模型. 此外, Ma 等人[30]提出了一组基于特定测试输入的模型置信度的指标, 指导选择更可能被错误分类的测试输入, 这与上述测试排序的目标类似.

本文研究工作隶属于第 1 类方法, 即尝试选出一个新的测试子集以代替原始测试集, 完成对 DNN 模型的测试任务.

1.2 多目标优化

多目标优化问题(multi-objective optimization problems, MOP)出现在日常生活中的很多场景, 例如职位计划安排[31]、金融投资组合管理[32]. 该问题涉及多个需要同时优化的目标. 由于部分优化目标之间存在冲突, 一个 MOP 问题可能不存在单一的最优解, 而是一组折中解(即 Pareto 最优解集). 其形式化定义如下[33]:

$$\min_{\text{s.t. } x\in S\subset R} / \max f(x) = \min / \max(f_1(x), f_2(x), ..., f_M(x)) \tag{1}$$

其中, x 是搜索空间中的 S 维决策向量; $f_M(x)$是第 m 个待优化的目标(m=1,…,M), 其优化目标可能是最大化(max), 也可能是最小化(min); M 是优化目标的总数. 除此之外, 在多目标优化问题中, 一些相关术语[34]定义如下.

定义 1(Pareto 优势). 对于两个不同的解决策略 x_1 和 x_2, 如果采用 x_1 达到的目标没有一个输于 x_2, 并且至少存在一个目标优于 x_2, 则称 x_1 支配 x_2, 或称 x_1 对 x_2 具备 Pareto 优势.

定义 2(Pareto 最优解集). 所有的全局 Pareto 最优解所在的集合被称为 Pareto 最优解集(Pareto optimal solution set, PS).

定义 3(Pareto 最优前沿). Pareto 最优解集中, 每个解在目标空间中对应的图像称为 Pareto 最优前沿(Pareto optimal front, PF).

基于 Pareto 的多目标进化算法(multi-objective evolutionary algorithms, MOEA)是求解 MOP 问题的常用算法, 这些算法首先利用 Pareto 优势原理选择收敛性较好的非支配解, 然后基于密度对非支配解进一步选择, 保证最终解集的质量. 在过去的 20 年中, 研究人员提出了大量的 MOEA 来解决 MOP 问题, 如非支配排序遗传算法 NSGA-II[21]、高强度 Pareto 进化算法 SPEA2[35]、基于分解的 MOEA(MOEA/D)算法[36]等. 本文使用 MOEA 方法进行求解, 因为 MOEA 的目标不仅尝试寻找接近 Pareto 最优前沿(即收敛性能)的解, 而且尝试寻找均匀和广泛分布的解.

2 DMOS 方法

2.1 研究动机

在衡量选出的测试子集与原始测试集之间测试能力的差异时, 已有方法通常仅仅只考虑测试集整体准确率, 这会影响到选出的测试子集丧失了原始测试集的其他测试特性. 实验结果显示, 已有方法选出的测试子集虽然能够在不同选择数量下对原始测试集的整体准确率进行精准估计, 但极有可能漏掉原始测试集中部分类别的测试输入样本, 并且对覆盖到的各个类别测试输入的准确率估计误差非常大. 例如: 当最新的选择方法 PACE[17]在 CIFAR100 测试集以及 ResNet20 模型构建的实验对象上运行, 当采样量为原始测试集的 1.5%时, 其整体准确率误差仅为 3.9%, 但此时, 选出的测试子集遗漏了原始测试集中 28%的测试输入类别, 且对各个类别测试输入的准确率平均估计误差高达 43%. 因此不难看出, 单从整体准确率出发, 极有可能导致最终选择出的测试子集不能有效代表原始测试集. 一个测试集能够代替另一个测试集, 意味着被选择出的子集应该具备原始测试集多方面的属性和特征, 包括具有相似的样本多样性. 近年来, 传统软件测试领域中涌现出大量度量测试集多样性的技术[37], 这些研究表明, 测试集的有效性很大程度上受益于样本的多样性. 因此, 从保证样本多样性及模型测试效果的角度出发, 我们认为: 选出的新测试子集不仅应该尽可能多地覆盖到原始

测试集中包含的测试输入类别, 而且应该保证在每一个类别上都能达到与原始测试集接近的准确率.

基于上述分析, 我们希望 DMOS 选出的测试子集能够尽可能多地覆盖原始测试集中的测试输入类别, 同时能对各个类别测试输入的准确率精准估计. 具体来说, DMOS 方法首先利用聚类算法对原始测试集各个类别测试输入的分布情况做出预估; 然后以各个簇之间各类别数据的分布情况差异作为优化目标, 并利用多目标优化算法进行求解, 最终得出接近预期目标的近似最优选择方案.

2.2 问题建模

目前已有方法一般基于测试集整体准确率来评估原始测试集与测试子集间的测试能力差异, Zhou 等人提出的 DeepReduce 方法[16]尝试使用多个优化目标作为选出子集的质量评估依据. 下面对研究问题进行建模.

定义 4(基于多目标优化的深度学习测试输入选择问题建模). 对于待测 DNN 模型 M 和原始无标签的深度学习测试集 T, 记 $f_1(T,M),f_2(T,M),\ldots,f_s(T,M)$是 s 个评价 M 在测试集 T 上性能表现的目标函数, T'是选出的测试子集, 深度学习测试输入选择问题就是要使选出的子集 $T'(||T'||<<||T||)$, 满足:

$$f_1(T,M)\approx f_1(T',M),\ f_2(T,M)\approx f_2(T',M),\ldots,\ f_s(T,M)\approx f_s(T',M).$$

在 DMOS 方法具体的执行过程中, 由于原始的测试数据都是待标记的无标签数据, 我们以待测 DNN 模型对其的预测标签作为区分测试输入类别的依据, 在对原始测试集完成聚类后, 以原始测试集和测试子集的对应簇中不同类别数据占比的误差之和作为优化目标, 进行迭代求解, 以期降低这两个集合中相同簇的分布差异, 进而保证前后两个集合能够具有一致的分布. 如果不采用聚类方法直接根据模型预测的标签类别设计优化目标, 将会使得方法性能受到待测 DNN 模型的准确率影响(低精度模型对原始测试集中测试输入估计的类别会存在较大偏差). 而聚类方法可以在没有标签的情况下对原始测试集中各个类别的样本进行初步的估计, 因此采用聚类方法的目的就是为了校正不同精度 DNN 模型带来的偏差, 以使得随后多目标优化的过程能够在一个相对可信的数据基础上进行. DMOS 方法进行多目标优化的过程可以通过公式(2)进行表示.

$$\begin{cases}\min y_k = Ratio_{diff}(x,cluster_k), & k\in 1,2,...,m\\ \text{s.t. } x=(x_1,x_2,...,x_n),\ x_i\in[0,100],\ i\in 1,2,...,n\end{cases} \tag{2}$$

其中, x 是一个 n 维向量(n 是原始测试集的大小), 代表针对原始测试集的一种具体选择方案, 其中任一元素的取值代表了对应位置的测试输入被选中的权重, 范围是 1–100 之间的任意实数, 根据用户指定的选择数量, 排序选择出权重最大的若干个位置的元素组成新的测试子集. $cluster_k$ 代表原始测试集聚类完成后的第 k 个簇:

$$Ratio_{diff}(x,cluster_k)=\sum_{i=1}^{L}(|l_i-l_i'|) \tag{3}$$

假设聚类完成后共形成了 m 个簇, 则 y_k 表示依据选择方案 x 筛选出的测试子集与原始测试集在簇 k 上各个类别测试输入占比的差异之和, 其计算形式如公式(3)所示, L 为待测 DNN 模型对原始测试集预测出的不同的类别总数, l_i 代表在原始测试集簇 k 中类别为 l 的样本占簇 k 的比例, l_i' 代表在测试子集的簇 k 中类别为 l 的样本占簇 k 的比例. 具体来说, 假设待测 DNN 模型对原始测试集预测的类别只有 10 种, 聚类完成后假设原始测试集的簇 1 中共有 100 个测试输入, 待测 DNN 模型预测的 10 个类别在原始测试集的簇 1 中的个数分别是 65, 5, 3, 1, 5, 1, 5, 5, 5, 5, 其对应所占比例为 65%, 5%, 3%, 1%, 5%, 1%, 5%, 5%, 5%, 5%; 按照种群个体对应形成的测试子集中簇 1 只含 10 个测试输入, 模型预测的 10 个类别, 在测试子集的簇 1 中的个数分别是 0, 0, 1, 9, 0, 0, 0, 0, 0, 0(也就是说, 只选出了 2 个类别的测试输入), 其对应所占比例 0%, 0%, 1%, 9%, 0%, 0%, 0%, 0%, 0%, 0%; 则簇 1 上的优化目标最终计算结果为 1.06, 即前后两个集合在各个类别数据占比的差值的绝对值之和, 其他簇上的优化目标的计算以此类推. 在优化求解的过程中, 我们希望多目标优化算法能够有效协调各个簇对应优化目标之间的关系, 使得最终求解出的选择方案能够在各个簇上都能使得生成的测试子集的数据分布最大程度接近原始测试集. DMOS 方法的潜在直觉是, 通过聚类对原始测试集的数据分布进行预估, 然后利用多目标优化方法不断缩小选出的测试子集与原始测试集间的分布差异, 进而保证测试子集具备与原始测试集相似的性质.

2.3 算法流程

0. 整体框架: 输入输出参数及整体流程简介.

DMOS 方法的输入包括: 待测 DNN 模型 D, 原始测试集 T, 共包含 t 个测试输入, 以及用户指定的测试输入选择数量 n. DMOS 方法首先利用 D 抽取 T 中测试输入的特征作为聚类的依据, 并根据 D 的预测结果作为测试输入的类别标签, 利用测试子集与原始测试集之间各个簇中各类别数据占比差异作为优化目标, 迭代求解出 Pareto 最优的选择方案, 根据用户的测试需求返回一个从 T 中筛选出的测试子集 X. DMOS 方法的具体流程如算法 1 所示, 工作流程如图 1 所示.

算法 1. DMOS 方法流程.

输入: 待测 DNN 模型 D;

待筛选的原始测试集 T, 其尺寸大小记为 st;

用户指定从 T 中筛选的样本个数 n.

输出: 筛选生成的新样本集合 X, 其尺寸大小为 n.

1. $X\leftarrow\varnothing$
2. **foreach** t_i in T **do**
3. $f_i\leftarrow extractFeatures(t_i,D)$ //抽取测试输入的特征向量表示
4. **end**
5. $\{f_1', f_2',..., f_{st}'\}\leftarrow Preprocess(f_1, f_2,..., f_{st})$ //对测试输入的特征向量进行预处理操作
6. $predictLabels\leftarrow D.predict(T)$ //得到 T 中所有测试输入被预测的标签
7. $G\leftarrow\{g_1,g_2,...,g_m\}\leftarrow cluster(f_1', f_2',..., f_{st}')$ //将所有测试输入根据其特征向量聚类分成 m 个簇
8. $totalLabelsProportion\leftarrow\varnothing$ //存储所有簇中每个类别样本占当前簇比
9. **foreach** g_k in G **do**
10. $singleClusterProportion\leftarrow\varnothing$ //存储单个簇中每个类别样本占当前簇比例
11. **foreach** l in $Set(predictLabels)$ **do**
12. $singleClusterNums\leftarrow Where(g_k==l).size(\cdot)$
13. $singleClusterProportion.append\ (singleClusterNums/|g_k|)$
14. **end**
15. $totalLabelsProportion.append\ (singleClusterProportion)$
16. **end**
17. $Iters\leftarrow 50$, $NIND\leftarrow 50$, $M\leftarrow m$ //设置进化迭代次数、种群规模以及优化目标的个数
18. $Population\leftarrow Random.round(NIND,st)$ //随机初始化种群矩阵
19. $NDSet\leftarrow$NSGA-II ($Population,Iters,M,@Fitness$) //执行多目标优化, 将求出的 Pareto 解集存入 $NDSet$
20. $X.add\ (Convert(NDSet,n))$ //用户根据需求确定选择方案, 将确定选中的测试输入添加入 X
21. **Return** X

1. 数据预处理: 包括特征抽取、特征降维、特征取值标准化等操作.

DMOS 方法首先抽取出每个测试输入的特征表示(算法 1 第 2 行–第 4 行), 随后执行特征降维和特征取值标准化(即 Min-Max 标准化)等预处理操作(算法 1 第 5 行). 由于 T 中的数据没有标签, 为了在选择过程中区分不同测试输入之间的类别, DMOS 方法采用 D 的预测结果给 T 中的测试输入作为其类别标签(算法 1 第 6 行).

2. HDBSCAN 聚类: 对数据进行分层聚类以便采样.

DMOS 方法采用 HDBSCAN 算法[18], 通过对原始测试集进行聚类, 以对其数据分布进行初步评估, 为之后采样做准备. 选择 HDBSCAN 的原因可总结如下.

(1) 因为难以预知测试集中测试输入的真实类别, 因此需要预先知道分簇数量的聚类算法无法适用, 例如 K-means 算法[38]. 而 HDBSCAN 则不需要预先设定分簇的数量, 也不需要根据密度进行聚类.

(2) HDBSCAN 的有效性已经得到了验证[18], 因此使用 HDBSCAN 可以取得较好的聚类结果.

(3) HDBSCAN 需要设置的参数较少[18].

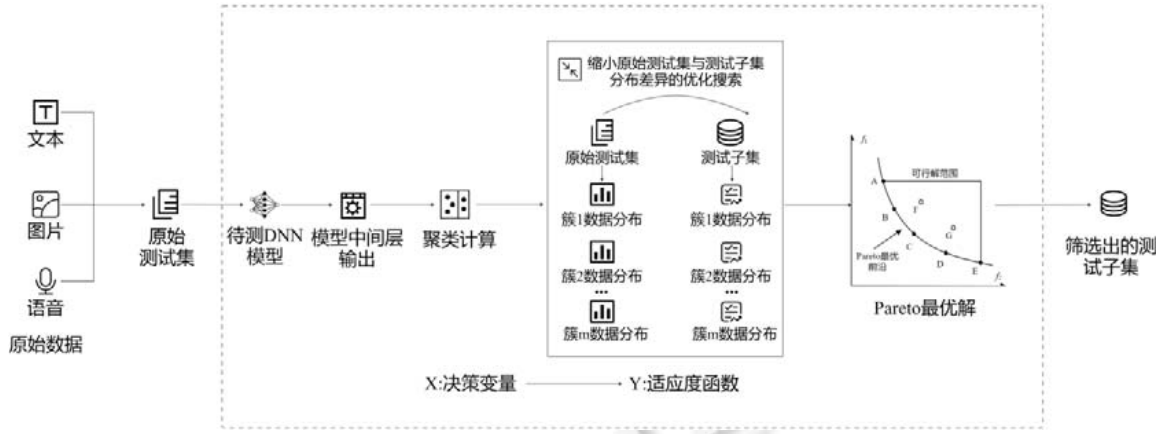

图 1 DMOS 方法流程图

基于之前预处理过的测试输入特征向量, DMOS 方法利用 HDBSCAN 方法进行软聚类, 将 T 划分成 m 个大小不等的簇(算法 1 第 7 行), 然后计算原始测试集聚类形成的簇上各类数据的占比情况, 为之后计算优化目标作好准备(算法 1 第 9 行–第 16 行). 具体操作为: 遍历簇集合 G, 每遍历一个新簇 g_k 时, 计算 g_k 中不同类别数据的占比情况(算法 1 第 10 行–第 14 行), 将结果存储入 *totalLabelsProportion* 中(算法 1 第 15 行).

3. 多目标优化: 运行参数初始化及求解结果处理.

原始测试集的参数计算完毕之后, 设置多目标优化的基本参数(算法 1 第 17 行): 进化迭代次数 *Iters*、种群规模 *NIND*、优化目标的个数 *M*(在本方法中 HDBSCAN 聚类完成形成了 m 个簇, 就有 m 个优化目标); 然后随机初始化种群 *Population*(算法 1 第 18 行), 在深度学习测试输入选择场景中, 种群 *Population* 中任意个体为一个 st 维向量, 代表针对原始测试集的一种具体选择方案, 其中的任一元素的取值为 1–100 的实数, 代表该位置的测试输入被选中的权重. 确定最终选择方案时, 将个体向量元素进行降序排序, 取权重最大的前 n 个位置的测试输入构成新的测试子集. 我们使用多目标优化领域内的经典算法 NSGA-II[21], 对上述问题进行求解. 与其他多目标遗传进化算法类似, NSGA-II 首先随机生成一定规模的初始种群, 然后对种群中的个体进行非支配排序后, 通过选择、交叉和变异这 3 个基本操作算子, 得到第 1 代子代种群; 然后从第 2 代开始, 将父代种群与子代种群合并, 在这个新生成的种群中进行快速非支配排序, 同时对每个非支配层中的个体进行拥挤度计算, 根据非支配关系以及个体的拥挤度选取合适的个体组成新的父代种群; 最后, 还是通过基本操作算子产生新的子代; 当迭代到指定轮次或达到其他指定终止条件后, 迭代过程结束并返回 Pareto 最优解集. DMOS 方法选择 NSGA-II 的理由总结如下: (1) 在处理种群个体上引入了精英策略, 让父代种群和子代种群共同竞争产生新的个体, 扩大了采样空间, 保证了个体的质量; (2) 提出了快速非支配排序, 降低了算法运行的时间复杂度; (3) 引入了拥挤度及其比较算子, 弥补了需要人为指定共享参数的缺陷, 并使得种群个体均匀地收敛, 有力地保证了种群的多样性. 运行完成后, 将得到的 Pareto 最优解存储到 *NDSet* 中(算法 1 第 19 行). 最终, 用户可根据自己的测试需求确定最终的选择方案. 将对应 Pareto 最优解对应选出的测试子集存入 X(算法 1 第 20 行)并返回(算法 1 第 21 行).

算法 2. Fitness(适应度函数的计算).

输入: 待优化种群总体 *Population*;

原始测试集 T 聚类后每个簇中每个类别占当前簇比例 *totalLabelsProportion*;

用户指定从原始测试集中筛选的样本个数 n;

待测模型对原始测试集 T 中每个样本预测的类别 *predictLabels*.

输出: 种群中所有个体对应的优化目标取值 *Objvalues*, 其形状为 *NIND*×*M*.

```
1.  clusterIndex←0
2.  labelNums←predictLabels.size(·)
3.  Objvalues←∅
4.  foreach Individual in Population do
5.    T′←argsort(Individual)[:n]   //排序选出权重最大的前 n 个位置的样本作为新生成的测试子集, 记为 T′
6.    totalProportionErrors←∅   //存储每一个体的优化目标值
7.    G′←{g1,g2,…,gt}←GetNew{T′,G}   //得到筛选的子集 T′对应的簇, 其中, t≤m
8.    foreach i in range(0,|G|) do
9.      if G[i] not in G′ then   //如果 G′中没有选中簇 G[i]中的样本
10.       totalProportionErrors.append (sum(totalLabelsProportion[clusterIndex]))
11.       clusterIndex+=1
12.     else
13.       s←∅
14.       foreach l in Set(predictLabels) do   //计算 G′[i]中每个类别的样本占 G′[i]的比例
15.         singleClusterNums←where(G′[i]==l).size(·)
16.         s.append(singleClusterNums/|G′[i]|)
17.       end   //计算同一类簇中前后两个集合每个类别占比的误差
18.       e←[abs(totalLabelsProportion[clusterIndex][j]−s[j]) for j in range(0,labeNums)]
19.       clusterIndex+=1
20.       totalProportionErrors.append(sum(e))   //计算当前簇中前后两个集合所有类别的误差之和
21.     end
22.   end
23.   Objvalues.append (totalProportionErrors)
24. end
25. Return Objvalues
```

4. 适应度函数设计: 优化目标与种群构建函数关系.

设计适应度函数是多目标优化求解的过程中的关键步骤, 即计算种群中个体对应的优化目标取值. 在深度学习测试输入选择的场景中, DMOS 方法对应的适应度函数如算法 2 所示. 它接收待优化的种群总体 *Population*、原始测试集 *T* 聚类完成后, 每个簇中各个类别数据占比情况 *totalLabelsProportion*、用户指定选择数量 *n* 以及待测 DNN 模型 *D* 预测的类别标签集合 *predictLabels* 作为输入参数, 最终返回种群中每个个体的优化目标取值集合 *Objvalues*. 具体求解过程为: 遍历种群(算法 2 第 4 行), 并根据其元素取值排序, 选取权重较大的前 *n* 个位置的测试输入组成新的测试子集记为 *T*′(算法 2 第 5 行), 其中, *T*′中测试输入数据的聚类信息记为 $G'=\{g'_1,g'_2,...,g'_t\}$, $t\leqslant m$ (算法 2 第 7 行). 遍历 *G* 中每个簇(算法 2 第 8 行), 如果 *G*′中不包含 *G*[*i*]的样本, 则将在簇 *G*[*i*]优化目标取值直接记为原始测试集中簇 *G*[*i*]中各个类别占比之和(算法 2 第 9 行–第 11 行); 否则, 依次计算 *G*′[*i*]簇中各个类别样本占比(算法 2 第 14 行–第 17 行), 并和 *G*[*i*]簇中各类别占比依次作差取其绝对值记入 *e* 中(算法 2 第 18 行、第 19 行), 最终将 *e* 中所有元素之和作为簇 *G*[*i*]对应的优化目标取值记录下来(算法 2 第 20 行). 每次求解完个体的各个优化目标值存入 *Objvalues*(算法 2 第 23 行), 待所有个体计算完成后返回(算法 2 第 25 行).

3 实验设计与结果分析

3.1 实验对象介绍

本文研究的实验对象包括深度学习测试集以及待测 DNN 模型，我们总共使用了 8 组分类任务的深度学习测试集-DNN 模型组合构成的实验对象，作为实验及评估分析的最小单位. 表 1 给出了所用模型及测试集的详细信息. 在该表中，最后 4 列分别表示待测 DNN 模型的大小、深度学习测试集规模(包含的测试输入的数量)、模型在测试集上所达到的整体准确率以及测试集中包含的不同的测试输入类别数量. 由于深度学习的实际应用情况非常复杂，因此我们在研究中试图构建出一个全面的测试基准.

- 深度学习测试集

实验所采用的 DNN 模型分别基于 5 个流行数据集(即 MNIST, CIFAR10, CIFAR100, ImageNet 和 Speech-Commands)进行训练，这些数据集在已有研究中得到了广泛应用[11,27,39]，在一些研究中也被用来生成对抗[40]数据测试 DNN 模型. 具体来说，MNIST (http://yann.lecun.com/exdb/mnist/)是一个关于识别手写体数字(数字 0–9，共 10 个分类)的数据集，CIFAR10 (http://www.cs.toronto.edu/~kriz/CIFAR.html)是一个包含了 10 类真实世界中普适对象(飞机、汽车、鸟类等)的数据集; CIFAR100 则与 CIFAR10 类似，不过包含的真实对象比 CIFAR10 更为丰富，共包含了 100 类真实对象; ImageNet (http://www.image-net.org)是按照 WordNet 层次结构组织的图像数据集(包含 1 000 个不同类别的图像); 而 Speech-Commands[39]是语音识别数据集，它包含一组使用众包方法收集的长度约为 1 s 的 wav 音频文件，每个文件仅包含一个英语口语单词.

- 待测 DNN 模型

对于待测 DNN 模型的类型，本文首先考虑了两种不同精度的 DNN 模型，即高精度模型和低精度模型. 这里，我们将整体准确率大于 80%的模型称为高精度模型; 否则，该模型称为低精度模型. 我们使用了两个真实世界的低精度模型，即基于 CIFAR100 的 ResNet20 模型和基于 ImageNet 的 VGG19 模型. 另外，本文在研究中主要考虑了基于分类任务的 DNN 模型. 例如，DeepSpeech 是一个多标签分类模型(其预测结果不是一个确定的类别标签而是一串音素序列)，而其他分类模型是单标签分类模型(预测结果为确定的类别). 最后，实验使用的 DNN 模型中大部分都是 CNN 模型，只有 DeepSpeech (https://github.com/bjtommychen/Keras_DeepSpeech2_SpeechRecognition)是 RNN 模型.

表 1 DNN 模型和相关测试集

ID	测试集名称	模型	模型大小(KB)	测试输入数量	整体准确率(%)	类别数量
1	MNIST	LeNet1	113	10 000	94.86	10
2		LeNet4	947	10 000	96.79	10
3		LeNet5	1 093	10 000	98.68	10
4	CIFAR10	VGG16	21 814	10 000	78.71	10
5		ResNet20	3 507	10 000	91.45	10
6	CIFAR100	ResNet20	10 615	10 000	71.42	100
7	ImageNet	VGG19	562 176	50 000	64.73	1 000
8	Speech-Commands	DeepSpeech	6 734	6 471	94.53	30

注：我们研究中的 ImageNet 模型的精确度比参考文献[41]中的要小一些，因为我们研究中使用的预处理方法和参考文献[41]略有不同. 更具体地说，为了从 ImageNet 获得固定大小的 224×224 的输入图像, Simonyan 等人[41]从重新缩放的图像中随机裁剪图像，而我们按照官方 Keras 示例提供的方法调整图像大小却不对图像进行裁剪

3.2 评测指标与对比方法介绍

3.2.1 评测指标

- 准确率估计误差(整体准确率和各个类别测试输入准确率)

准确率是待测模型在测试集上分类正确的样本数占总数的比例. 在之前各类深度学习测试输入选择的方法中，研究者通常考虑整体准确率是否接近，以此来衡量测试子集与原始测试集测试能力是否相近. 本文中，我们还将考虑类别信息，即考虑使用各个类别测试输入的准确率. 我们将原始测试集与测试子集两者间整体

准确率之差的绝对值以及各个类别测试输入准确率之差的绝对值, 作为衡量两者测试性质是否接近的标准.

- 覆盖率估计误差

近几年, 研究人员提出了许多测试覆盖指标来度量 DNN 模型的测试充分性[25,26,42]. 本文中, 我们共采用了 5 种神经元覆盖率指标来衡量测试充分性. 第 1 个指标是由 Pei 等人[26]提出的 DNC (DeepXplore's neuron coverage, DNC), 其计算公式为待测 DNN 模型中激活神经元(在执行测试输入后, 神经元的输出值若大于事先设定的阈值, 则认为该神经元被激活)占总神经元的比例. 剩下的 4 个指标是由 Ma 等人[25]提出的多粒度级别的神经元覆盖度量, 包括 TKNC (top-K neuron coverage), KMNC (K-multisection neuron coverage), NBC (neuron boundary coverage)和 SNAC (strong neuron activation coverage). 具体来说, TKNC 是一种层级神经元覆盖度量, 计算公式为执行测试输入后按输出值降序排列的每层中前 k 个神经元占总神经元的比例. KMNC 首先从训练数据中将每个神经元的输出范围划分为 k 个区段, 如果一个神经元的输出值(记神经元 n 的输出值区间为$[a,b]$)在执行下一个测试输入后落入某区段, 则其被视为覆盖区间, 因此, 其计算的是所有神经元覆盖区间的占比情况. 不同于 KMNC, NBC 考虑在执行测试输入后, 神经元输出值的外部区域(即$(-\infty,a)$和$(b,+\infty)$)是否被覆盖. 而 SNAC 只考虑在执行测试输入之后是否覆盖了上界的外部区域(即$(b,+\infty)$). 本文中, 我们将考虑原始测试集与选择生成的测试子集在这 5 个测试覆盖指标上差值的绝对值, 作为衡量它们测试性质是否接近的标准.

- 反世代距离

准确率和覆盖率主要从深度学习测试输入选择场景下来评估性能. 为了对 DMOS 返回的 Pareto 解进行有效评估, 本文采用反世代距离(inverted generational distance, IGD)[36]作为评测指标. 它计算理论最优的 Pareto 前沿中所有解与待评估多目标优化算法求得的 Pareto 前沿中所有解的平均欧式距离(即衡量待评估方法与理论最优解的差距). IGD 值越小, 表明待评估算法求得的 Pareto 最优解集越逼近理论最优解, 并且分布更均匀, 该算法的收敛性和多样性也更好.

- 统计检验方法

为了验证各个选择方法的实验结果是否存在统计学显著性差异, 我们采用 Wilcoxon signed-rank 检验方法[43]对实验结果进行分析. 这是一种非参数假设检验方法, 用于检验成对数据的中位差是否满足为零的假设. 本文中, 我们将置信度设为 0.05, 即当计算出的 p 值小于 0.05 时, 拒绝零假设, 即两组实验结果存在统计学显著性差异; 如果该值大于 0.05, 则接受零假设, 这意味着两组实验结果的差异可以忽略不计. 基于 Wilcoxon signed-rank 检验的结果, 我们还使用了 Win/Tie/Loss 检验方法对不同选择方法的性能进行比较, 这个方法在许多传统研究中被广泛应用[44−46]. Win 意味着我们的方法在 95%的置信水平下, 显著优于 PACE; Tie 则意味着与 PACE 之间没有统计学显著差异; 其他情况被归为 Loss. 此外, 我们还采用了 Scott-Knott ESD 测试[47], 其被广泛用于分析某些方法是否优于其他方法, 并且可以生成这些方法的全局排名. 它保证了同组方法之间没有统计学显著的性能差异, 不同组方法之间存在统计学显著的性能差异. 我们将 Scott-Knott ESD 测试的结果绘制成了箱线图(如图 2–图 7 所示), 其中, 存在统计学显著性差异的方法组之间使用虚线进行分隔, 位于虚线同一侧的方法组内的方法不存在统计学显著性差异.

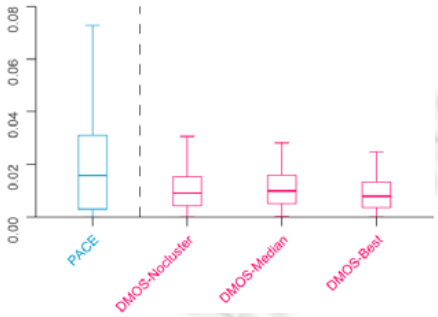

图 2 整体准确率估计误差的 Scott-Knott ESD 检验结果

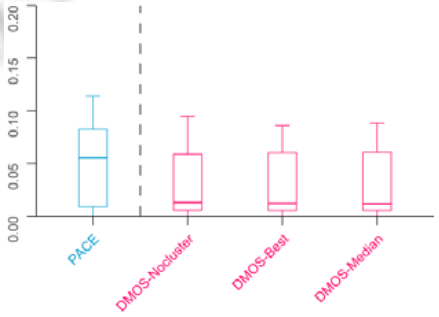

图 3 NC 估计误差的 Scott-Knott ESD 检验结果

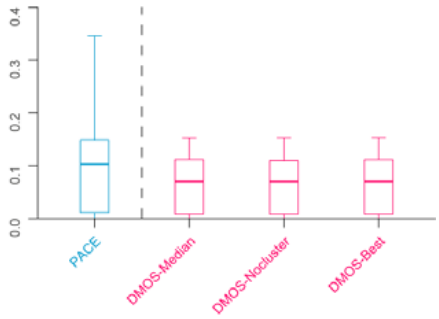

图 4 NBC 估计误差的 Scott-Knott ESD 检验结果

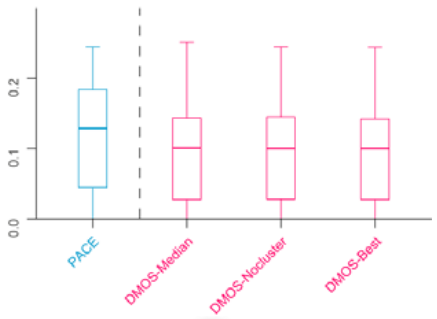

图 5 SNAC 估计误差的 Scott-Knott ESD 检验结果

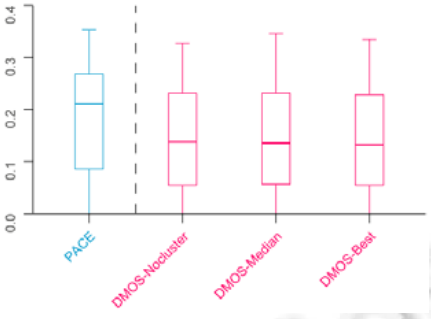

图 6 TKNC 估计误差的 Scott-Knott ESD 检验结果

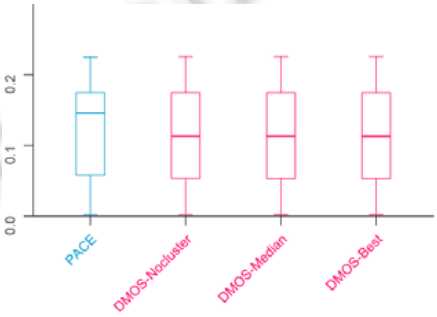

图 7 KMNC 估计误差的 Scott-Knott ESD 检验结果

3.2.2 对比方法

PACE[17]是最新提出的一种深度学习测试输入选择方法，并在之前的研究中已经与现有方法(CES 等)做过全面比较. 因此在本文的研究中，我们采用 PACE 作为对比方法. PACE 的输入包括一个待测 DNN 模型、待标记的测试集以及用户指定的筛选数量. 其背后潜在的直觉是: 对于整个测试集，一些测试输入具有相似的测试能力，而另一些则具备不同的测试能力，因此，选出的测试子集应该覆盖原始测试集的各种类型的测试能力，以便可以更好地代表原始测试集. PACE 首先通过提取特征，将每个测试输入转换为特征向量，对其进行预处理，包括特征取值归一化或特征降维；然后，通过聚类将它们分为若干组正常点簇和异常点簇；接着，PACE 通过定义阈值来确定从正常点簇和异常点簇采样的比例，进而确定采样数量；然后，基于 MMD-Critic 的原型选择方法[19]和自适应随机选择方法[20]分别对正常点簇和异常点空间进行采样;最后构造出一组新的测试子集，即 PACE 的输出. 开发人员可以仅标记该组数据，就可以完成对模型的测试，从而大幅度节省测试成本.

为了从多个角度综合评估 DMOS 方法的性能，我们还设计了 DMOS 的几种变体方法进行研究.

(1) DMOS-Nocluster 方法. 该方法跳过了聚类的步骤，直接以原始测试集与选出的测试子集在不同类别测试输入的占比差异作为优化目标进行求解，以求出的 Pareto 解集中各个类别测试输入的准确率估计平均误差最小的解作为最终选择方案. 类似地，我们同样通过整体准确率、测试覆盖等指标评估最终求解出的选择方案的性能，进而分析聚类的步骤对 DMOS 方法的贡献.

(2) DMOS-Median 方法. 该方法选取了 DMOS 方法求解出的 Pareto 解集中各个类别测试输入的准确率估计误差的中位数作为最终结果.

(3) DMOS-Best 方法. 该方法选取了 DMOS 方法求解出的 Pareto 解集中各个类别测试输入的准确率估计平均误差最小的解作为最终结果.

(4) RandomSearch 方法. 该方法在评价函数和选择函数上采取随机搜索策略进行决策(这意味着它的优化搜索方向是随机的)，在经过指定次数的进化轮次后，得到最终选择方案. 设计该方法的目的主要

是为了从多目标优化本身的角度评估 DMOS 方法的性能, 因此主要采用 IGD 指标来评估该方法与 DMOS 方法的优化效果及解集质量, 进而分析多目标优化的步骤对 DMOS 的贡献.

(5) EA-Best 方法. 该方法直接以原始测试集与选出的测试子集在各个真实类别上测试输入的准确率估计误差作为优化目标, 尝试多组参数, 使用 NSGA-II 进行求解. EA-Best 方法设计的初衷是为了探究采用多目标优化的形式对各个类别测试输入准确覆盖的极限情况, 所以每次选取各个类别测试输入的准确率估计平均误差最小的解作为最终结果. 虽然 EA-Best 方法求出的结果并不是理论最优的, 但是结果显示, 不同实验参数设置下的性能差异并不大, 因此可以认为, EA-Best 方法求出的结果是接近理论最优的情况, 可以作为基准与 DMOS 方法进行比较.

3.3 实验设置

DMOS 方法的实现采用 Python3.7.4 编码完成, 并基于 Keras2.3.1[48]和 TensorFlow1.15.0[49]对实验对象中的深度学习测试集提取特征. 此外, 在算法实现过程中, 我们还分别采用了特征降维算法、聚类算法以及多目标优化算法, 这些都由 Python 的成熟框架 scikit-learn0.23.1[50], hdbscan0.8.26[51], geatpy2.6.0[52]提供. 对于 DMOS 方法所涉及的参数, 主要包括聚类参数、降维参数、特征类型参数以及多目标优化参数. 其中, 前 3 组参数共同影响聚类效果, 而后一组参数则直接影响最终求解出的 Pareto 解的质量. 具体来说, 我们希望聚类的结果能够有效区分出不同类别的测试输入, 每个簇的不同类别应该越少越好. 针对于不同类型的实验对象, 聚类参数和降维参数的设置均遵循 PACE 中的推荐进行取值; 而特征类型参数的最佳设置取–1, 即模型最后的输出层参数. 而多目标优化参数主要是决定进化迭代的次数以及种群总体个数, 前者决定了收敛的时机, 后者决定了搜索空间的范围, 我们将这两个参数都设置为 50. 其他选择方法的实现以及参数设置, 我们都遵循已有工作的推荐[17].

为了使得在不同选择数量下不同实验对象中各个类别测试输入的准确率估计误差的效果明显, 针对所有的 10 分类实验对象, 选择数量均从 55 开始一直变化到 205, 步长为 10 进行迭代; 针对 100 分类的实验对象, 选择数量从 350 开始一直变化到 2 050, 步长为 100 进行迭代; 针对 1000 分类的实验对象, 选择数量从 500 开始一直到 20 500 为止, 其中步长为 1 000 进行迭代; 针对 Speech-Commnds 测试集, 选择数量从 22 开始, 一直到 1 590 结束, 步长为 64 进行迭代.

所有实验均在 Intel Xeon E5-2640 服务器进行, 其操作系统为 Ubuntu 18.04.2, 内存为 128 GB.

3.4 研究问题

我们从以下研究问题切入, 并执行相应的分析实验, 以验证 DMOS 方法的有效性.

- RQ1: DMOS 方法能否保证选出的测试子集在各个类别测试输入的准确率估计与原始测试集类似?

筛选形成的测试子集不仅要尽可能多地覆盖原始测试集中的类别, 而且要保证覆盖到的各个类别测试输入的准确率应与原始测试集接近, 这对其保持原始测试集的测试性质有着重要意义. 现有的方法在选择时优先考虑测试子集与原始测试集在整体准确率上的差异, 这极有可能使得现有方法忽略了筛选出的测试子集在各个类别测试输入上的准确率. 因此, 本研究问题希望分析: 与现有方法相比, DMOS 方法能否有效保证选出的测试子集在各个类别测试输入上的准确率估计更为精准.

- RQ2: DMOS 方法选出的测试子集在整体准确率、测试覆盖标准上能否具备与原始测试集相近的测试能力?

除了对各个类别测试输入上的准确率估计之外, 在之前的研究中, 整体准确率以及一些测试覆盖指标[16]常被用来衡量选出的测试子集与原始测试集测试性质的差异. 因此, 一种性能良好的选择方法除了保证选出的测试子集在各个类别测试输入上的准确率估计与原始测试集接近以外, 对于整体准确率以及测试覆盖指标的估计误差也不应该过大. 在本研究问题中, 我们希望从其他评测指标来分析: DMOS 方法与其他方法相比, 能否更有效地保持与原始测试集更为接近的测试能力.

- RQ3: 聚类以及多目标优化对 DMOS 方法的贡献如何?

DMOS 方法最关键的两个步骤就是聚类和多目标优化: 聚类的目的是为了在没有真实标签的情况下, 对原始测试集的各类数据的分布占比进行初步的估计, 为之后的优化求解采样打好基础; 而多目标优化的目的则是为了在聚类完成的基础上, 不断地使选出的测试子集的数据分布更加接近原始测试集. 因此, 在此研究问题下, 我们希望分析这两个关键部分对 DMOS 方法最终性能的贡献程度.

- RQ4: DMOS 方法的运行时间开销如何?

现有的深度学习测试输入选择方法的本质都是为了约减无标签的原始数据集以降低研究人员的标注成本和测试执行成本, 与这些工作所需的时间开销相比, 现有选择方法的执行开销是微不足道的. 但是, 为了全面研究 DMOS 方法的性能, 在本研究问题中, 我们还是从运行时间开销的角度切入进行实验, 并与现有方法进行比较.

3.5 结果分析

- RQ1: DMOS 方法能否保证选出的测试子集在各个类别测试输入的准确率估计与原始测试集类似?

(1) 设计

为了评估选择生成的测试子集能否尽可能多地包含原始测试集中的类别, 且在各个类别上对待测 DNN 模型保持与原始测试集相近的准确率, 我们主要从以下两个角度对 DMOS-Best, DMOS-Median, EA-Best 和 PACE 方法的实验结果进行分析: 一方面, 通过观察随着选择数量的增大, 不同选择方法对实验对象各个类别测试输入的准确率估计平均误差的变化情况, 去分析各个方法能否更早地将平均误差收敛至一个更小的变化范围, 以及从标准差的角度对其性能稳定性进行分析; 另一方面, 我们使用 Wilcoxon signed-rank 检验以及 Win/Tie/Loss 分析去评估: 对于相同实验对象下的任一类别, 各个选择方法在不同的选择数量下能否显著降低对当前类别的准确率估计误差, 并将各个方法的性能表现互相比较.

(2) 结果

8 个实验对象下各个类别测试输入的准确率估计平均误差与选择数量的曲线变化情况见表 2–表 9. 表 2–表 9 中, 第 1 列是选择数量, 第 2 列–第 4 列括号中每个元素, 是选择方法在不同选择数量下各个类别测试输入的准确率估计误差的平均值和标准差. 表格中深灰色阴影的元素表示在同一选择数量下, 该方法的性能最好; 而浅灰色阴影的元素代表该方法性能表现处于第二. 从各个类别测试输入准确率估计误差的平均值角度看, 对于 10 分类实验对象, 见表 3、表 4、表 6, 由于测试集中测试输入类别的数量过少, DMOS-Best 和 DMOS-Median 方法在采样量为原始测试集的 0.7%时, 选出的测试子集的平均估计误差已经开始收敛至较小范围, 而 PACE 虽然也已开始收敛, 但平均误差估计水平要明显高于其他方法; 而如表 2 和表 5 所示, DMOS-Median 方法在 3 号和 5 号实验对象上虽然比 PACE 更早地收敛到较小误差范围, 但随着选择数量的增加, 其优势变得不再明显. 对于 100 分类实验对象(6 号实验对象), 见表 7, DMOS-Best 和 DMOS-Median 方法在收敛之前选出的测试子集始终能够保持比 PACE 更低的平均误差估计, 且更早地达到收敛范围(当采样量约为原始测试集的 8%时). DMOS-Median 方法虽然在收敛之后的误差估计范围与 PACE 相差不大, 但采样量达到 16%之后仍可以优于 PACE, 且收敛之后的平均估计误差的范围(6.9%–12.4%)仍优于 PACE (7.5%–13%); 而 DMOS-Best 方法则始终可以保持对 PACE 的优势, 误差估计的收敛区间(5.9%–10%)远远低于 PACE (7.5%–13%). 而在 8 号实验对象上, 见表 8, 虽然 DMOS-Best 方法的性能远远优于 PACE, 但是 DMOS-Median 方法选出的测试子集随着选择数量的变化, 性能却始终落后于 PACE. 这是由于 DMOS 方法进行实验时采用聚类的特征是待测 DNN 模型最后一层的输出, 而 8 号实验对象中所采用的模型最后一层的输出是单个字母的预测概率, 仍需要进一步转换才能得到最终的预测类别, 即最大概率字母组合而成的单词. 也就是说, 8 号实验对象中最终的测试输入类别与聚类采用的特征之间并不像其他实验对象那样是直接的映射关系, 这使得聚类方法并不能有效估计其测试输入分布情况, 最终使得 DMOS 方法求解出的各个测试子集在 8 号实验对象上的表现并不稳定. 在之后的研究中, 我们将更加重视对语音数据集特征的分析, 探究语音数据集中类别与特征的关系, 提升 DMOS 方法的性能. 此外, 值得注意的是, 表 2–表 9 中, DMOS-Best 方法的估计误差平均值变化几乎与 EA-Best 方法重合. 这说明 DMOS 方法求解出的 Pareto 解集中存在着与理论最优极其接近的解, 而同时, DMOS-

Median 方法的性能在绝大部分的实验对象上(除在 3 号和 5 号实验对象上对 PACE 没有显著优势以及 8 号实验对象上弱于 PACE)都能超过最新选择方法 PACE, 充分证明了 DMOS 方法能够有效保证选出的测试子集对各个类别测试输入的准确率估计都非常精准. 而对于各个选择方法来说, 在没有真实标签信息的情况下进行选择, 类别数量越多, 准确估计每个类别的难度越大, 这就导致了虽然选择数量远远多于 10 分类实验对象. 但所有选择方法在 100 分类(6 号实验对象)、1000 分类实验对象(7 号实验对象)上对于各个类别测试输入的准确率估计误差要显著高于 10 分类测试集(即表 7 和表 9 中各个方法的平均估计误差曲线变化范围要高于表 2–表 6). 此外, 各个选择方法在看不到真实标签的情况下, 实际上覆盖的类别以及数量都是极少的. 如此不充分的选择, 使得各个方法对各个类别测试输入的准确率估计其实都处于较高的误差范围(在表 7 和表 9 中, 即使是最接近理论最优情况的 EA-Best 方法在选择数量较少的情况下性能仍然很差), 且性能相差不大, 但表 9 显示, DMOS-Best 和 DMOS-Median 方法仍然保持着对 PACE 微弱的优势, 在相同选择数量下, 它们选出的测试子集仍然要比 PACE 能够达到更低的各个测试输入类别准确率估计的平均误差. 最后, 从各个类别测试输入准确率估计误差的标准差角度看, 随着采样数量的增加, 各个选择方法的估计误差都在趋向平缓, 标准差都在逐渐减小, 但无论是 DMOS-Best 方法还是 DMOS-Median 方法, 其标准差整体取值范围及波动变化均比 PACE 更小. 这充分说明 DMOS 方法选出的测试子集能够比 PACE 选出的测试子集更均匀、更全面地对各个类别测试输入的准确率进行精准估计, 且性能更为稳定.

表 10 展示了各个选择方法进行 Wilcoxon signed-rank 检验结合 Win/Tie/Loss 分析的结果. 具体来说, 表 10 展示的是 DMOS-Nocluster, DMOS-Best, EA-Best 及 DMOS-Median 方法在 8 组实验对象上各个类别测试输入的准确率估计与 PACE 的比较结果(即在当前实验对象上, 4 个方法能在多少类别上的准确率估计显著优于 PACE). 同表 2–表 9, 表 10 中深灰色阴影的结果表示在当前实验对象上性能表现最好的方法, 浅灰色阴影的结果性能处于第二. 表 10 中显示, DMOS-Best 方法在全部 10 分类实验对象中, 大约 50%的类别上的准确率估计要显著优于 PACE; 而 DMOS-Median 方法准确估计的类别数量也多于 PACE, 并且除了实验对象 1 号和 5 号实验对象, DMOS-Best 和 DMOS-Median 方法在其他 10 分类实验对象上的准确率估计都没有显著弱于 PACE 的情况. 对于采用语音数据的 8 号实验对象, DMOS-Median 方法的性能表现要弱于 PACE: 仅在 7 个类别上的估计超过 PACE, 而在其他 10 个类上弱于 PACE. 而 DMOS-Best 方法的性能仍然保持对 PACE 的优势, 能够比 PACE 在更多的类上准确估计. 这同样说明了在之后的研究中, 我们应该对语音数据集进行更多的研究, 提升整体最终求解出的 Pareto 解集的质量. 而对于 100 分类、1000 分类实验对象, 正如上节提到的, 类别数量的增加, 使得各个选择方法在 6 号和 7 号实验对象上的性能表现差异减小, 而 DMOS-Median 方法取的是求出的 Pareto 解集中所有解的中位数, 因此不同选择数量下的性能表现比 DMOS-Best 方法更为稳定. 因此, DMOS- Median 方法能够在更多的类上比 DMOS-Best 方法显著优于 PACE. 最后, 从 EA-Best 方法的角度来看, DMOS- Best 方法与其性能表现相差不多. 这也说明了 DMOS 方法的有效性.

表 2 CIFAR10-ResNet20 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
55	(0.058,0.029)	(0.191,0.272)	(0.056,0.024)	(0.054,0.022)	(0.096,0.049)
65	(0.050,0.022)	(0.131,0.126)	(0.040,0.025)	(0.048,0.021)	(0.084,0.036)
75	(0.051,0.028)	(0.121,0.116)	(0.049,0.032)	(0.044,0.020)	(0.083,0.023)
85	(0.047,0.017)	(0.110,0.121)	(0.044,0.026)	(0.040,0.023)	(0.078,0.021)
95	(0.048,0.023)	(0.116,0.117)	(0.046,0.034)	(0.041,0.020)	(0.076,0.022)
105	(0.038,0.027)	(0.108,0.068)	(0.045,0.036)	(0.028,0.021)	(0.078,0.033)
115	(0.039,0.034)	(0.120,0.115)	(0.034,0.021)	(0.033,0.023)	(0.063,0.014)
125	(0.039,0.031)	(0.107,0.115)	(0.039,0.022)	(0.026,0.024)	(0.063,0.017)
135	(0.032,0.026)	(0.085,0.066)	(0.027,0.024)	(0.032,0.027)	(0.060,0.021)
145	(0.042,0.031)	(0.076,0.054)	(0.037,0.032)	(0.029,0.024)	(0.062,0.014)
155	(0.043,0.036)	(0.072,0.057)	(0.037,0.037)	(0.024,0.021)	(0.061,0.017)
165	(0.036,0.025)	(0.065,0.049)	(0.025,0.019)	(0.021,0.019)	(0.052,0.019)
175	(0.030,0.022)	(0.051,0.044)	(0.028,0.033)	(0.023,0.022)	(0.053,0.015)
185	(0.027,0.030)	(0.047,0.046)	(0.027,0.020)	(0.021,0.018)	(0.045,0.019)
195	(0.037,0.043)	(0.047,0.041)	(0.031,0.019)	(0.023,0.017)	(0.055,0.017)
205	(0.037,0.027)	(0.044,0.042)	(0.019,0.022)	(0.017,0.016)	(0.046,0.010)

表 3 MNIST-LENET1 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
55	(0.039,0.024)	(0.078,0.058)	(0.037,0.018)	(0.035,0.018)	(0.076,0.033)
65	(0.038,0.029)	(0.071,0.054)	(0.038,0.023)	(0.037,0.021)	(0.057,0.033)
75	(0.041,0.021)	(0.076,0.082)	(0.037,0.018)	(0.033,0.021)	(0.061,0.033)
85	(0.029,0.020)	(0.075,0.087)	(0.035,0.017)	(0.036,0.018)	(0.064,0.033)
95	(0.036,0.019)	(0.073,0.088)	(0.038,0.025)	(0.030,0.023)	(0.053,0.024)
105	(0.033,0.023)	(0.070,0.089)	(0.031,0.017)	(0.029,0.019)	(0.061,0.033)
115	(0.035,0.027)	(0.067,0.090)	(0.030,0.018)	(0.030,0.013)	(0.050,0.020)
125	(0.034,0.022)	(0.074,0.089)	(0.031,0.024)	(0.022,0.016)	(0.049,0.021)
135	(0.031,0.024)	(0.076,0.088)	(0.027,0.021)	(0.022,0.015)	(0.046,0.020)
145	(0.030,0.024)	(0.082,0.105)	(0.028,0.026)	(0.022,0.027)	(0.041,0.020)
155	(0.028,0.023)	(0.079,0.106)	(0.025,0.017)	(0.019,0.013)	(0.045,0.021)
165	(0.023,0.026)	(0.069,0.085)	(0.029,0.025)	(0.016,0.013)	(0.045,0.022)
175	(0.027,0.022)	(0.065,0.076)	(0.024,0.019)	(0.012,0.008)	(0.039,0.016)
185	(0.021,0.017)	(0.064,0.061)	(0.014,0.010)	(0.014,0.012)	(0.039,0.023)
195	(0.020,0.018)	(0.064,0.060)	(0.019,0.012)	(0.016,0.016)	(0.043,0.015)
205	(0.020,0.023)	(0.063,0.072)	(0.025,0.024)	(0.013,0.016)	(0.041,0.019)

表 4 MNIST-LENET4 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
55	(0.032,0.028)	(0.052,0.048)	(0.025,0.012)	(0.026,0.015)	(0.046,0.028)
65	(0.023,0.015)	(0.050,0.045)	(0.022,0.016)	(0.046,0.055)	(0.041,0.028)
75	(0.022,0.016)	(0.042,0.038)	(0.022,0.016)	(0.023,0.016)	(0.049,0.028)
85	(0.023,0.014)	(0.048,0.053)	(0.019 0.011)	(0.022,0.014)	(0.040,0.028)
95	(0.022,0.016)	(0.043,0.043)	(0.022,0.016)	(0.024,0.014)	(0.042,0.023)
105	(0.025,0.014)	(0.041,0.042)	(0.024,0.013)	(0.020,0.009)	(0.037,0.027)
115	(0.027,0.015)	(0.038,0.038)	(0.022,0.016)	(0.018,0.011)	(0.037,0.015)
125	(0.023,0.018)	(0.044,0.034)	(0.018,0.008)	(0.020,0.007)	(0.034,0.016)
135	(0.019,0.008)	(0.041,0.031)	(0.022,0.010)	(0.017,0.007)	(0.035,0.017)
145	(0.024,0.013)	(0.045,0.035)	(0.019,0.010)	(0.018,0.011)	(0.035,0.015)
155	(0.017,0.009)	(0.043,0.032)	(0.020,0.009)	(0.018,0.013)	(0.034,0.017)
165	(0.020,0.012)	(0.045,0.040)	(0.018,0.009)	(0.018,0.008)	(0.027,0.015)
175	(0.021,0.011)	(0.048,0.051)	(0.020,0.011)	(0.015,0.009)	(0.032,0.017)
185	(0.021,0.011)	(0.045,0.047)	(0.016,0.009)	(0.013,0.009)	(0.027,0.016)
195	(0.018,0.013)	(0.045,0.046)	(0.014,0.007)	(0.014,0.008)	(0.031,0.016)
205	(0.019,0.012)	(0.050,0.046)	(0.017,0.012)	(0.016,0.008)	(0.030,0.015)

表 5 MNIST-LENET5 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
55	(0.013,0.007)	(0.025,0.038)	(0.013,0.007)	(0.013,0.007)	(0.020,0.007)
65	(0.013,0.007)	(0.023,0.032)	(0.013,0.007)	(0.013,0.007)	(0.013,0.007)
75	(0.013,0.007)	(0.023,0.032)	(0.013,0.007)	(0.013,0.007)	(0.016,0.007)
85	(0.013,0.007)	(0.022,0.028)	(0.013,0.007)	(0.013,0.007)	(0.020,0.007)
95	(0.013,0.007)	(0.021,0.025)	(0.013,0.007)	(0.013,0.007)	(0.025,0.007)
105	(0.013,0.007)	(0.021,0.025)	(0.013,0.007)	(0.013,0.007)	(0.018,0.007)
115	(0.013,0.006)	(0.019,0.020)	(0.013,0.007)	(0.013,0.007)	(0.020,0.007)
125	(0.013,0.007)	(0.019,0.018)	(0.013,0.007)	(0.013,0.007)	(0.019,0.007)
135	(0.013,0.007)	(0.018,0.015)	(0.013,0.007)	(0.020,0.016)	(0.015,0.007)
145	(0.013,0.007)	(0.017,0.014)	(0.013,0.007)	(0.013,0.007)	(0.017,0.007)
155	(0.012,0.005)	(0.019,0.016)	(0.013,0.007)	(0.014,0.008)	(0.018,0.007)
165	(0.013,0.007)	(0.018,0.014)	(0.013,0.006)	(0.013,0.006)	(0.019,0.007)
175	(0.012,0.005)	(0.018,0.013)	(0.013,0.006)	(0.012,0.005)	(0.022,0.008)
185	(0.016,0.011)	(0.017,0.012)	(0.012,0.005)	(0.012,0.005)	(0.019,0.007)
195	(0.013,0.006)	(0.016,0.011)	(0.012,0.005)	(0.012,0.005)	(0.015,0.007)
205	(0.012,0.005)	(0.016,0.010)	(0.011,0.004)	(0.012,0.005)	(0.019,0.007)

表 6 CIFAR10-VGG16 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
55	(0.075,0.081)	(0.159,0.105)	(0.079,0.055)	(0.075,0.076)	(0.151,0.046)
65	(0.070,0.041)	(0.160,0.123)	(0.060,0.042)	(0.052,0.051)	(0.131,0.040)
75	(0.079,0.056)	(0.145,0.112)	(0.061,0.062)	(0.046,0.034)	(0.121,0.025)
85	(0.066,0.064)	(0.155,0.101)	(0.066,0.057)	(0.051,0.035)	(0.123,0.037)
95	(0.061,0.031)	(0.140,0.085)	(0.066,0.041)	(0.051,0.048)	(0.109,0.027)
105	(0.056,0.069)	(0.147,0.090)	(0.060,0.055)	(0.042,0.026)	(0.097,0.027)
115	(0.052,0.039)	(0.152,0.109)	(0.053,0.071)	(0.048,0.039)	(0.100,0.026)
125	(0.059,0.045)	(0.130,0.100)	(0.042,0.020)	(0.041,0.041)	(0.086,0.022)
135	(0.067,0.057)	(0.118,0.092)	(0.056,0.047)	(0.043,0.039)	(0.096,0.036)
145	(0.053,0.044)	(0.103,0.090)	(0.031,0.012)	(0.035,0.025)	(0.086,0.018)
155	(0.047,0.045)	(0.108,0.097)	(0.045,0.029)	(0.037,0.026)	(0.085,0.029)
165	(0.047,0.028)	(0.091,0.093)	(0.048,0.041)	(0.030,0.032)	(0.080,0.018)
175	(0.057,0.061)	(0.093,0.083)	(0.042,0.044)	(0.032,0.028)	(0.081,0.021)
185	(0.039,0.035)	(0.095,0.069)	(0.028,0.024)	(0.041,0.029)	(0.070,0.018)
195	(0.053,0.055)	(0.085,0.065)	(0.032,0.021)	(0.027,0.021)	(0.071,0.020)
205	(0.043,0.033)	(0.080,0.050)	(0.029,0.018)	(0.028,0.023)	(0.072,0.015)

表 7 CIFAR100-ResNet20 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
350	(0.208,0.163)	(0.256,0.202)	(0.190,0.162)	(0.184,0.145)	(0.220,0.063)
450	(0.165,0.137)	(0.214,0.158)	(0.154,0.131)	(0.157,0.135)	(0.185,0.051)
550	(0.143,0.122)	(0.182,0.147)	(0.131,0.110)	(0.130,0.097)	(0.164,0.041)
650	(0.127,0.099)	(0.160,0.100)	(0.117,0.090)	(0.116,0.096)	(0.146,0.037)
750	(0.121,0.088)	(0.140,0.098)	(0.106,0.074)	(0.104,0.083)	(0.129,0.034)
850	(0.114,0.089)	(0.130,0.095)	(0.099,0.094)	(0.092,0.076)	(0.124,0.030)
950	(0.095,0.073)	(0.111,0.090)	(0.097,0.079)	(0.088,0.073)	(0.111,0.027)
1 050	(0.097,0.084)	(0.105,0.082)	(0.087,0.076)	(0.086,0.080)	(0.109,0.026)
1 150	(0.086,0.070)	(0.097,0.081)	(0.079,0.067)	(0.081,0.063)	(0.101,0.023)
1 250	(0.090,0.068)	(0.091,0.074)	(0.083,0.064)	(0.074,0.064)	(0.094,0.022)
1 350	(0.084,0.070)	(0.084,0.076)	(0.079,0.065)	(0.071,0.062)	(0.091,0.019)
1 450	(0.075,0.063)	(0.084,0.068)	(0.075,0.059)	(0.070,0.064)	(0.087,0.021)
1 550	(0.073,0.057)	(0.081,0.065)	(0.072,0.058)	(0.067,0.062)	(0.083,0.021)
1 650	(0.077,0.062)	(0.085,0.063)	(0.069,0.066)	(0.061,0.053)	(0.079,0.016)
1 750	(0.070,0.059)	(0.083,0.061)	(0.062,0.052)	(0.058,0.048)	(0.079,0.022)
1 850	(0.064,0.053)	(0.077,0.060)	(0.065,0.045)	(0.062,0.049)	(0.076,0.018)
1 950	(0.061,0.052)	(0.076,0.060)	(0.059,0.048)	(0.058,0.052)	(0.071,0.014)
2 050	(0.058,0.046)	(0.075,0.059)	(0.059,0.047)	(0.058,0.047)	(0.069,0.019)

表 8 Speech-Commands-DeepSpeech 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
22	(0.420,0.448)	(0.472,0.454)	(0.324,0.418)	(0.361,0.433)	(0.535,0.403)
86	(0.097,0.175)	(0.077,0.084)	(0.053,0.026)	(0.058,0.041)	(0.150,0.035)
150	(0.058,0.043)	(0.061,0.048)	(0.056,0.028)	(0.050,0.025)	(0.086,0.031)
214	(0.060,0.032)	(0.050,0.031)	(0.050,0.031)	(0.045,0.030)	(0.065,0.026)
278	(0.046,0.029)	(0.047,0.030)	(0.044,0.026)	(0.041,0.025)	(0.061,0.027)
342	(0.040,0.028)	(0.045,0.030)	(0.037,0.027)	(0.036,0.030)	(0.057,0.025)
406	(0.037,0.029)	(0.039,0.025)	(0.035,0.032)	(0.027,0.020)	(0.052,0.022)
470	(0.031,0.028)	(0.039,0.024)	(0.034,0.026)	(0.030,0.026)	(0.044,0.021)
534	(0.029,0.023)	(0.034,0.021)	(0.025,0.021)	(0.027,0.020)	(0.040,0.013)
598	(0.029,0.022)	(0.035,0.028)	(0.028,0.022)	(0.024,0.018)	(0.037,0.015)
662	(0.024,0.020)	(0.032,0.024)	(0.023,0.014)	(0.023,0.016)	(0.035,0.011)
726	(0.027,0.020)	(0.031,0.025)	(0.024,0.020)	(0.021,0.013)	(0.034,0.010)
790	(0.023,0.014)	(0.032,0.025)	(0.022,0.018)	(0.020,0.014)	(0.031,0.012)
854	(0.027,0.022)	(0.030,0.026)	(0.023,0.018)	(0.020,0.016)	(0.031,0.012)
918	(0.022,0.019)	(0.029,0.025)	(0.019,0.021)	(0.019,0.015)	(0.028,0.008)
982	(0.022,0.016)	(0.028,0.026)	(0.022,0.016)	(0.019,0.015)	(0.027,0.010)
1 046	(0.022,0.019)	(0.027,0.024)	(0.022,0.016)	(0.018,0.012)	(0.026,0.008)
1 110	(0.018,0.013)	(0.026,0.023)	(0.020,0.017)	(0.016,0.013)	(0.026,0.009)
1 174	(0.017,0.013)	(0.027,0.027)	(0.019,0.018)	(0.018,0.012)	(0.027,0.009)
1 238	(0.019,0.017)	(0.027,0.027)	(0.018,0.012)	(0.014,0.014)	(0.026,0.008)
1 302	(0.016,0.013)	(0.023,0.023)	(0.019,0.014)	(0.016,0.012)	(0.024,0.008)
1 366	(0.017,0.010)	(0.023,0.022)	(0.018,0.015)	(0.017,0.014)	(0.022,0.006)
1 430	(0.017,0.012)	(0.022,0.021)	(0.018,0.015)	(0.011,0.011)	(0.022,0.007)
1 494	(0.016,0.015)	(0.021,0.019)	(0.016,0.010)	(0.013,0.009)	(0.020,0.006)
1 558	(0.016,0.013)	(0.020,0.019)	(0.018,0.013)	(0.014,0.011)	(0.021,0.005)

表 9 ImageNet-VGG19 实验对象上各个类别测试输入的准确率估计平均误差变化

选择数量	DMOS-Nocluster	PACE	DMOS-Best	EA-Best	DMOS-Median
500	(0.525,0.241)	(0.526,0.241)	(0.517,0.241)	(0.515,0.242)	(0.531,0.216)
1 050	(0.360,0.237)	(0.376,0.248)	(0.356,0.245)	(0.361,0.235)	(0.372,0.177)
2 050	(0.272,0.204)	(0.283,0.215)	(0.266,0.202)	(0.267,0.207)	(0.278,0.124)
3 050	(0.212,0.171)	(0.228,0.183)	(0.209,0.173)	(0.208,0.167)	(0.220,0.077)
4 050	(0.177,0.148)	(0.183,0.156)	(0.173,0.141)	(0.170,0.136)	(0.182,0.063)
5 050	(0.150,0.122)	(0.159,0.133)	(0.151,0.124)	(0.149,0.127)	(0.157,0.057)
6 050	(0.137,0.111)	(0.138,0.114)	(0.133,0.111)	(0.135,0.113)	(0.143,0.044)
7 050	(0.119,0.096)	(0.126,0.109)	(0.117,0.095)	(0.121,0.099)	(0.127,0.037)
8 050	(0.112,0.092)	(0.118,0.098)	(0.108,0.088)	(0.111,0.091)	(0.116,0.034)
9 050	(0.103,0.085)	(0.111,0.092)	(0.101,0.082)	(0.101,0.083)	(0.108,0.030)
10 500	(0.098,0.079)	(0.104,0.086)	(0.095,0.078)	(0.095,0.080)	(0.100,0.030)
11 500	(0.092,0.075)	(0.097,0.078)	(0.090,0.074)	(0.090,0.073)	(0.094,0.030)
12 500	(0.083,0.069)	(0.091,0.075)	(0.084,0.068)	(0.086,0.069)	(0.089,0.025)
13 500	(0.081,0.066)	(0.087,0.070)	(0.081,0.066)	(0.080,0.064)	(0.085,0.022)
14 500	(0.077,0.063)	(0.083,0.067)	(0.076,0.060)	(0.075,0.061)	(0.080,0.021)
15 500	(0.074,0.060)	(0.079,0.064)	(0.072,0.058)	(0.070,0.057)	(0.076,0.021)
16 500	(0.067,0.055)	(0.075,0.062)	(0.070,0.057)	(0.070,0.056)	(0.073,0.020)
17 500	(0.066,0.053)	(0.072,0.060)	(0.066,0.053)	(0.066,0.054)	(0.069,0.020)
18 500	(0.064,0.052)	(0.069,0.058)	(0.062,0.049)	(0.062,0.052)	(0.066,0.019)
19 500	(0.061,0.050)	(0.068,0.057)	(0.060,0.048)	(0.059,0.048)	(0.063,0.018)
20 500	(0.059,0.048)	(0.065,0.056)	(0.056,0.044)	(0.060,0.048)	(0.061,0.018)

表 10 4 种多目标优化方法与 PACE 在 8 组实验对象上不同类别的 Win/Tie/Loss 分析

ID	DMOS-Nocluster VS PACE	DMOS-Best VS PACE	EA-Best VS PACE	DMOS-Median VS PACE
1	6/3/1	8/1/1	8/1/1	4/5/1
2	3/7/0	5/5/0	6/4/0	4/6/0
3	4/6/0	4/6/0	4/6/0	3/7/0
4	7/3/0	7/3/0	7/3/0	3/7/0
5	5/4/1	4/6/0	6/4/0	2/6/2
6	23/65/12	23/64/13	23/67/10	31/42/27
7	195/655/150	208/642/150	199/653/148	277/493/230
8	7/16/7	11/12/7	11/14/5	7/13/10

表 11 展示了各个方法在不同评测指标上的平均误差结果, 从第 1 行各个类别测试输入的平均准确率估计误差来看, DMOS-Nocluster, DMOS-Best, DMOE-Median, PACE 方法在 8 组实验对象共计 144 组实验上各个类别测试输入准确率的平均误差分别 5.954%, 5.547%, 7.589%和 8.473%, 与 PACE 相比, 这 3 种方法平均降低了 2.519%, 2.926%, 0.884%的误差, 平均提升幅度(计算公式为(PACE-DMOS-X)/PACE)分别为 29.73%, 34.53%, 10.43%. 这同样也说明了 DMOS 方法对于目前领域内经典方法 PACE 的性能优越性.

表 11 各个方法在 7 个评测指标上的平均误差结果

Metric	DMOS-Nocluster (%)	DMOS-Best (%)	DMOS-Median (%)	PACE (%)
AvgAcc	5.954	**5.547**	7.589	8.473
TotalAcc	1.238	**1.081**	1.211	1.926
KMNC	10.520	**10.516**	10.517	11.721
NBC	**6.412**	**6.412**	6.435	11.527
NC	3.157	3.149	**3.143**	11.471
SNAC	9.258	**9.223**	9.285	18.938
TKNC	14.526	**14.394**	14.438	26.676

小结: (1) DMOS 方法求解出的 Pareto 最优解集中最好的解接近理论最优的情况, 且在语音、图像测试集上均表现优异, 超过现有方法 PACE; (2) DMOS 方法求解出的 Pareto 最优解集中, 性能一般的解仍然在图像测试集上超过现有方法 PACE, 但在语音测试数据集上表现不佳; (3) 原始测试集中包含的类别越多, 对选择方法有效估计各个类别测试输入的准确率带来的难度越大, 但在这些测试集上, DMOS 方法求解出的 Pareto 最优解集中, 性能一般的解仍然能够保持对现有方法 PACE 的优势.

- RQ2: DMOS 方法选出的测试子集在整体准确率、测试覆盖标准上能否具备与原始测试集相近的测试能力?

(1) 设计

在本实验中, 我们将 DMOS-Best, DMOS-Median, DMOS-Nocluster 和 PACE 在 8 组实验对象上的实验结果(共计 144 组数据, 在 8 号实验对象上共 25 次实验, 1–5 号实验对象上均是 16 次实验, 6 号实验对象上 18 次实验, 7 号实验对象上 21 次实验)从整体准确率的估计误差、NC、NBC、SNAC、TKNC、KMNC 这 5 个测试覆盖指标的估计误差共 6 个角度进行 Scott-Knott ESD 检验分析, 以探究 DMOS 方法从其他评测指标来看能否比其他选择方法具备与原始测试集更相似的测试性质.

(2) 结果

如图 2–图 7 所示, 各类选择方法在整体准确率和测试覆盖度量上的估计误差越小越好, 因此, 图中方法的排名应是越靠右表示方法的性能越好. 结果显示, DMOS-Best 和 DMOS-Median 方法在 4 种测试覆盖指标上的估计误差显著低于 PACE (在 KMNC 上, DMOS-Best 和 DMOS-Median 方法虽然没有显著超过 PACE, 但仍然保持微弱优势), 且整体准确率的估计误差也显著超过 PACE. 从箱线图的各个方法的上界看, 虽然 DMOS-Best, DMOS-Median 方法与 PACE 持平且有时超过 PACE, 但从箱线图中的中位线及下界来看, DMOS 方法在 6 个指标上能够达到更低的估计误差且性能更稳定. 从表 11 中第 3 行–第 7 行来看, DMOS-Nocluster, DMOS-Best, DMOE-Median, PACE 方法在 8 组实验对象共计 144 组实验上平均整体准确率误差分别 1.238%, 1.081%, 1.211%和 1.926%, 与 PACE 相比, 这 3 种方法分别平均降低了 0.688%, 0.845%, 0.715%的误差, 对 PACE 的平均提升幅度分别为 35.72%, 43.87%, 37.12%; DMOS-Nocluster, DMOS-Best, DMOE-Median, PACE 方法在 8 组实验对象共计 144 组实验上, 5 种测试覆盖标准的平均误差分别 8.775%, 8.739%, 8.763%和 16.067%, 与 PACE 相比, 这 3 种方法平均降低了 7.292%, 7.328%, 7.304%的误差, 对 PACE 的平均提升幅度分别为 45.39%, 45.61%, 45.46%. 这同样充分说明了 DMOS 方法性能的优越性.

小结: (1) DMOS 方法选出的测试子集能够在其他测试评测指标上比 PACE 选出的测试子集更为接近原始测试集; (2) 确保测试子集对于各个类别测试输入的准确率的精准估计, 对于保证测试子集的其他性质与原始测试集相似具有重要的意义.

- RQ3: 聚类以及多目标优化对 DMOS 方法的贡献如何?

(1) 设计

为了调研聚类以及多目标优化对 DMOS 方法的贡献, 我们首先将 DMOS-Nocluster 方法与采取聚类的 DMOS-Best 方法(因 DMOS-Nocluster 方法取的也是 Pareto 解集中各个类别测试输入的准确率估计平均误差最小的解作为最终结果, 因此为了探究聚类对 DMOS 方法的贡献, 所以在此 RQ 中只与 DMOS-Best 方法进行比较)从各个类别测试输入的准确率、整体准确率以及测试覆盖率等几个方面评估它们的性能差异; 然后, 我们将 DMOS 方法求解的 Pareto 解和 RandomSearch 方法求解出的解集在 IGD 指标上进行比较, 从多目标优化的角度评估求出的解集的收敛性和多样性.

(2) 结果

与 DMOS-Nocluster 的比较. 我们选取了 DMOS-Nocluster 求得的解集中在各个类别测试输入的准确率估计平均误差最小的解作为代表 DMOS-Nocluster 性能的最终解. 从表 2–表 6 来看, DMOS-Nocluster 方法的性能同样比 PACE 要好很多, 能够在更少的选择数量下更早地达到误差更低的估计水平, 但还是略差于 DMOS-Best 方法. 而从整体准确率和测试覆盖指标来看, DMOS-Nocluster 方法与 DMOS-Best 方法并没有显著性的性能差异, 但 DMOS-Best 方法仍然保持微弱的优势. 聚类背后的潜在直觉是: DMOS 方法并不盲信于待测 DNN 模型最终预测的标签信息, 而是通过抽取其中间层输出作为原始测试集数据的特征表示, 以保留更多性质. 因此, 在此基础上, 利用聚类方法对原始测试集进行数据分布的评估时(不同类别测试输入的占比情况), 会使得多目标优化采样时参考的信息更为准确. 因此我们建议, 在实际使用过程中, 先应用聚类对原始测试集各个类别的分布进行预估, 然后采样.

与 RandomSearch 的比较. 如图 8 所示, RandomSearch 方法在 IGD 指标上的表现非常不稳定, DMOS 方法显著优于 RandomSearch. 这说明在优化求解的过程中, RandomSearch 方法的搜索方向波动幅度更大, 且整体

走势没有明显下降趋势. 这说明其未能找到合适的优化方向. 与 RandomSearch 的比较结果不仅说明 DMOS 方法的多目标优化求解过程具备更好的收敛性, 求得的解集具有更好的多样性, 同时也意味着在深度学习测试输入选择问题中, 设计合适的优化目标进行求解是必要的, 随机的搜索方向不能找到合适的解.

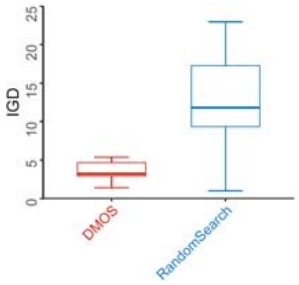

图 8 IGD 指标评估结果

小结: (1) 聚类方法能够有效矫正 DMOS 使用待测 DNN 模型预测的类别标签带来的偏差, 进而更好地估计原始测试集的数据分布, 为多目标优化打好基础, 保证 DMOS 方法的性能; (2) 设计合适的优化目标, 对于求解出期望的 Pareto 最优选择方案有着极大的帮助, 深度学习测试输入选择问题, 不是一个可以通过随机搜索决策即可达到期望性能的简单问题.

- RQ4: DMOS 方法的运行时间开销如何?

(1) 设计

DMOS 方法的时间运行开销主要集中在抽取测试输入特征表示、降维、聚类以及多目标优化上. 在 8 个实验对象上, 指定 DMOS 和 PACE 都从原始测试集上筛选出包含 1 000 个测试输入的测试子集, 计时从读取模型及数据开始, 直到选出测试子集为止. 最终, 我们将 DMOS 方法的时间开销结果与 PACE 进行了比较.

(2) 结果

实验结果见表 12. 第 1 列中实验对象的 ID 与表 1 中的信息对应, 第 2 列中括号中左边的数字代表 DMOS 方法在当前实验对象下选出 1 000 个测试输入的测试子集花费的时间开销, 右边的数字则是 PACE 的时间开销, 加粗的元素表示该方法时间开销更小, 方法性能更好. 值得注意的是, 表中结果显示, 现有测试输入选择方法进行筛选的时间成本开销(以秒为量级单位)远远小于人工标注的成本. 而从表中的数据看, DMOS 方法在 6 个实验对象上的时间开销要小于最新的选择方法 PACE, 充分说明了 DMOS 方法的高效性; 同时, DMOS 仅在 5 号实验对象上略高于 PACE, 在 7 号实验对象上由于类别数量过多(1000 分类)加重了多目标优化中进化迭代的每一步时间开销, 最终使其性能表现输于 PACE. 在 8 号实验对象上, PACE 由于根据最佳参数推荐抽取的模型中间层表示过于复杂, 大幅度增加了 MMD-critic 的原型采样方法的时间开销, 最终导致其在此实验对象上的性能表现远远弱于 DMOS 方法. DMOS 方法和 PACE 方法在 8 个实验对象上分别选择 1 000 个测试输入的时间开销分析可见表 12.

表 12 DMOS 和 PACE 方法的时间开销分析

ID	DMOS VS PACE(单位: s)
1	(**85**,203)
2	(**86**,113)
3	(86,**48**)
4	(**93**,202)
5	(**106**,120)
6	(**146**,233)
7	(4683,**3591**)
8	(**132**,1967)

小结: (1) 现有深度学习测试输入选择方法的时间开销远远小于人工标注的成本开销, 因此, 设计行之有效的深度学习测试输入选择方法具有较高的实际意义; (2) 采用多目标优化的 DMOS 方法的运行时间开销要

显著优于目前最新的选择方法 PACE.

4 讨 论

4.1 DMOS求解的Pareto解集的质量研究

上一节实验结果充分说明了 DMOS 方法求解出的选择方案, 能够有效构建出接近原始测试集测试能力的测试子集. 但我们同时注意到, 在同一选择数量下求出的选择方案最终的质量参差不齐(从各个类别测试输入的准确率平均估计误差来看), 但从中位数的角度来看, 目前 DMOS 方法求解出的解仍然优于现有的选择方法. 为了在之后的研究中进一步提升 DMOS 求解出的解集质量, 我们对实验结果进行了进一步分析, 并得出了以下的结论.

(1) 选择数量的增加, 会使解的性能波动变小. 在选择数量极其稀少时, 各个解的性能都表现不佳差异不大; 而随着选择数量的增多, 各个解之间的性能开始分化, 性能差异随之增加. 而当选择数量增加到一定程度时, 由于各个解对应的选择方案之间选中的重复样本开始增加, 各个解之间性能的差异又开始逐渐变小. 例如, 实验结果显示, 原始测试集为 10 000 个样本的 10 分类或百分类数据集, 在选择数量为 800 左右时, 求解出的 Pareto 最优解集之间的性能开始趋于稳定.

(2) 原始测试集中包含的类别越多, 解的性能波动越小. 原始测试集中包含的类别越多, 对所有类别准确估计的难度也就越大, 各个解的性能差异也就越小. 因此, 在测试输入类别更丰富的测试集中, 虽然在不同选择数量下解的性能变化符合上一点中描述的规律, 但是波动变化的区间要远远小于类别数量少的测试集.

(3) DMOS 方法在没有进行任何变异的测试集上, 求解出的 Pareto 解集的质量更高. 实验结果显示, DMOS 方法对于那些混入了对特征进行微小扰动使模型致错的对抗测试输入的测试集表现不佳. 原因是这些对抗测试输入的特征与原始测试输入的特征相差极小, 但模型预测的结果却相差极大, 这使得聚类步骤无法对测试集的数据分布进行准确估计, 进而影响了多目标优化的性能.

4.2 有效性威胁

- 内部有效性威胁

内部有效性威胁主要来自于对 DMOS 方法的实现、对比实验中采用的选择方法的实现以及对所有实验结果进行分析评估的脚本实现. 为了有效地减少这些威胁, 对于 DMOS 方法, 我们依赖于 Python 中一些现有的成熟框架中的封装算法进行实现; 对于其他对比方法, 我们采用了这些方法共享出的开源链接上最新的版本, 并遵循了原文中给出的最佳参数推荐进行实验; 对于分析评估实验结果的所有脚本, 我们编码实现了多种方式的脚本相互验算校对, 确保结果无误. 此外, 在实现过程中, 我们对所有涉及的代码都进行了仔细检查.

- 外部有效性威胁

外部有效性威胁主要来源于研究的实验对象, 即所采用的深度学习测试集以及待测 DNN 模型. 对于使用的 DNN 模型, 我们采用了基于流行数据集(包括 MNIST, CIFAR-10, CIFAR-100, ImageNet 和 Speech-Commands)训练而成的模型. 由于优化目标的原因, 我们只考虑了分类模型, 但是为了减少其带来的威胁, 我们从高精度模型和低精度模型、CNN 模型和 RNN 模型(即 DeepSpeech)这几个方面考虑了待测 DNN 模型的不同类型. 对于所使用的深度学习测试集, 我们从不同类型的测试输入, 包括图片类型测试输入、语音类型测试输入出发, 丰富类型以减少外部有效性威胁. 目前没有在文本数据及相关模型上进行实验的原因是, 现有常见的一些文本分类任务的数据中, 其数据类别数量(通常是二分类, 例如垃圾邮件识别等)远远少于图像数据(可以达到百分类甚至千分类), 不能充分展示各个选择方法的选择效果. 此外, 简单分类的文本任务本质上与图像数据的分类任务极为相似: 以待测 DNN 模型对文本数据学习到的中间层输出作为特征进行聚类, 并打上类别标签, DMOS 就可以在此基础上进行多目标优化, 同样能够以聚类形成的各个簇中不同类别测试输入的数据占比情况设计优化目标迭代求解, 因此我们暂时没有考虑这类文本数据及相关模型. 在之后的研究中,

我们将继续收集任务更加复杂的实验对象, 并进行更广泛地研究.

- 结构有效性威胁

结构有效性的威胁主要在于实验中方法运行时指定的参数. 在本项研究中, 主要包括聚类参数、降维参数、特征类型参数(选择待测 DNN 模型的哪一层输出作为聚类时的特征)、多目标优化参数(种群规模及进化迭代次数). 对于聚类参数和降维参数, 我们遵循了 Chen 等人[17]工作中最佳参数的推荐; 对于特征类型参数, 我们尝试了 4 组参数(–1, –2, 0, 1), 最终选出了一组鲁棒性较高的结果作为最佳参数推荐在第 3.3 节中进行了介绍. 此外, 对于多目标优化参数, 其主要决定了搜索空间的范围以及进化迭代的时间. 根据其他相关工作中的参数推荐, 我们尝试了 4 组组合(20–50, 50–100, 100–200, 200–400), 最终选出了在时间成本和最终结果上综合性能最好的一组参数作为最终结果. 在未来的工作中, 我们将继续研究这些参数对方法性能的影响.

5 相关工作

5.1 深度学习测试

DNN 模型中的可信性问题促使研究人员开发出各种能够有效、完备地进行测试的技术[53]. 除了保证深度学习测试的高效性以外, 对待测 DNN 模型测试的充分性也是目前深度学习测试领域中的一个热点. 大量的研究提出了丰富的测试充分性度量指标来评估测试方法, 测试输入扩增成为提升测试充分性的主要方法.

近几年研究最深入的深度学习测试充分性度量指标是神经元覆盖率标准[25,26,54], 例如, DeepXplore[26]引入了神经元覆盖率来测量激活值高于预定义阈值的神经元的比率. 类似地, DeepGauge[25]引入了一系列基于神经元激活值的测试充分性标准. 最近的研究还提出了由符号执行[55]、覆盖引导模糊[56,57]和变形变换[13]驱动的测试标准和技术. Gerasimou 等人[58]提出了一种基于重要神经元覆盖的 DNN 模型测试充分性准则(importance-driven coverage, IDC). 重要神经元指对决策过程有核心贡献的神经元, 即当测试输入经过神经网络时, 对结果影响最大的神经元. 相关实验已经证明, 其能够从神经元激活层面对测试集充分性进行有效的综合评估.

测试输入扩增的目的在于在原有测试集的基础上产生新的测试数据, 以揭露原来无法暴露的错误, 进而提升测试充分性. 目前, 针对深度学习系统的测试输入扩增研究大致可以分为以下两类.

- 第 1 类方法是以测试覆盖率为指导准则, 通过变换随机种子, 使用特定的变异算子有指导地扩增测试集, 使得扩增后的测试集具有更高的测试覆盖率. 例如, Guo 等人[59]提出了 DLFuzz 方法使用神经元覆盖率[26]作为最终目标, 指导变异生成新的测试输入. 在迭代过程中, DLFuzz 在测试输入上引入微小的扰动, 并保留能够增加神经元覆盖范围的测试输入作为下一轮迭代中变异的测试输入.
- 第 2 类方法则聚焦深度神经网络的特点, 产生对抗样本, 即在原有测试输入上对其特征进行微小扰动、变换, 使得新的测试输入与原始的相近却能够触发原始测试输入不能触发的缺陷. 例如, Szegedy 等人[60]使用对抗样本与检测结果的损失函数作为目标函数来指导对抗样本的生成. Xiao 等人[61]提出了一种在高维空间对像素点变换的方法, 并证明了该方法产生的对抗性样本更加平滑, 变化也比较微小, 真实性也更高.

与之不同的是, 我们的工作旨在通过筛选出一个能够保持原始测试集性质的小规模测试子集, 帮助开发人员降低标注成本. Li 等人[15]首先提出了 CSS 方法, 基于模型预测的置信度划分区间比例采样形成最终的测试子集, 然后提出了 CES 方法, 使用交叉熵衡量测试子集与原始测试集之间的差异, 经过多轮迭代采样, 缩小两者间的差异形成最终的测试子集. Zhou 等人[16]提出了一种两阶段的采样方法 DeepReduce, 其首先根据神经元覆盖率挑选出一个小规模的测试子集; 然后利用启发式规则不断往其中添加入更多的测试数据, 直到达到用户指定的采样数量. Chen 等人[17]提出了一种实用精度估计方法 PACE 进行选择采样, 其首先通过聚类对原始测试集的数据进行分层分簇, 然后针对正常簇和异常簇分别利用 MMD-critic 的采样方法[19]和自适应随机选择方法[20]进行采样, 最终合并形成测试子集. 现有方法生成的测试子集在整体准确率上已经达到了原始测试集极为接近的水平, 但在选择数量稀少时, 不能充分覆盖原始测试集中各个类别的测试输入, 且对其准

确率估计误差也非常大. 本文创新性地从多目标优化的思路出发, 通过聚类, 对原始测试集的数据分布进行预估并设计优化目标, 不断缩小测试子集与原始测试集之间的分布差异, 以此保证测试子集能够充分覆盖并且准确估计原始测试集中不同类别的测试输入.

5.2 测试输入选择

传统的软件测试面临着由于冗余测试输入导致测试成本提升的问题, 因此也需要对原始测试集进行筛选, 以提升测试效率, 即在数量最小化的情况下满足于原始测试集相同的测试功能. 传统的软件测试领域中, 常见的测试输入约减算法有贪心算法、启发式算法等[62−67], 这些方法均以移除原始测试集中与某些测试能力度量相关的冗余测试输入为主要目标. Harrold 等人[68]提出了一种贪心策略与启发式搜索相结合的方法(harrold-gupta-sof, HGS), 它以设计好的测试需求作为依据, 反复迭代筛选原始测试集中的测试输入. 华丽等人[69]首先利用遗传算法对原始测试集进行进化迭代, 求解得出测试输入最优解集合; 然后利用蚁群算法对结果进一步约减, 得到最小化之后的最优测试子集. 聂长海等人[70]首先分析测试能力度量间的关系, 并对原始测试集进行划分, 然后再基于启发式算法、贪心算法分别对划分之后的小集合进行采样, 最终得到筛选之后的小规模测试子集. 然而, 由于深度学习系统是由基于数据驱动的编程范式开发而成, 深度学习测试输入选择问题旨在通过选出一组小规模、能够代表原始测试集测试能力的测试子集对待测的深度学习系统进行测试, 这与传统软件测试中的测试约减问题有着很大区别, 因此无法直接将传统软件测试约减中的评价标准、算法等直接迁移解决深度学习测试输入选择的问题.

6 总 结

本文将深度学习测试输入选择问题建模为一个多目标优化问题, 并创新性地提出了 DMOS 方法, 从各个类别测试输入的准确率出发设计优化目标, 并结合多目标遗传进化算法优化求解, 得到一个高效的小规模测试子集. 我们在 8 组 DNN 模型和测试集构成的实验对象上对方法的性能进行了实证研究, 结果表明, DMOS 方法在保证对测试集各个类别测试输入的准确率估计精准的情况下, 同时保证了新生成的测试子集与原始测试集具有相近的其他测试性质(例如整体准确率、测试充分性等), 显著优于最新的选择方法 PACE. 在之后的研究工作中, 我们将尝试从更多的评价角度衡量测试子集的质量, 并尝试将 DMOS 方法拓展到回归任务的 DNN 模型及测试集上应用.

References:

[1] Chen CY, Seff A, Kornhauser AL, Xiao JX. Deepdriving: Learning affordance for direct perception in autonomous driving. In: Proc. of the 2015 IEEE Int'l Conf. on Computer Vision. 2015. 2722−2730.

[2] Sun Y, Chen YH, Wang XG, Tang XO. Deep learning face representation by joint identification-verification. In: Advances in Neural Information Processing Systems 27: Annual Conf. on Neural Information Processing Systems. 2014. 1988−1996.

[3] Goodfellow IJ, Bengio Y, Courville AC. Deep Learning. In: Adaptive Computation and Machine Learning. MIT Press, 2016.

[4] LeCun Y, Bengio Y, Hinton GE. Deep learning. Nature, 2015, 521(7553): 436−444.

[5] Obermeyer Z, Emanuel EJ. Predicting the future—Big data, machine learning, and clinical medicine. The New England Journal of Medicine, 2016, 375(13): 1216.

[6] Julian KD, Lopez J, Brush JS, Owen MP, Kochenderfer MJ. Policy compression for aircraft collision avoidance systems. In: Proc. of the 35th IEEE/AIAA Digital Avionics Systems Conf. 2016. 1−10.

[7] Chen JJ, He XT, Lin QW, Xu Y, Zhang HY, Hao D, Gao F, Xu ZW, Dang YN, Zhang DM. An empirical investigation of incident triage for online service systems. In: Proc. of the 41st Int'l Conf. on Software Engineering: Software Engineering in Practice. 2019. 111−120.

[8] Chen JJ, He XT, Lin QW, Zhang HY, Hao D, Gao F, Xu ZW, Dang YN, Zhang DM. Continuous incident triage for large-scale online service systems. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2019. 364−375.

[9] Li X, Li W, Zhang YQ, Zhang LM. DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 2019. 169−180.

[10] Zhang X, Xu Y, Lin QW, Qiao B, Zhang HY, Dang YN, Xie CY, Yang XS, Cheng Q, Li Z, Chen JJ, He XT, Yao R, Lou JG, Chintalapati M, Shen FR, Zhang DM. Robust log-based anomaly detection on unstable log data. In: Proc. of the ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. 2019. 807−817.

[11] Wang Z, You H, Chen J, Zhang Y, Dong X, Zhang W. Prioritizing test inputs for deep neural networks via mutation analysis. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. 2021. 397−409.

[12] Taigman Y, Yang M, Ranzato MA, Wolf L. DeepFace: Closing the gap to human-level performance in face verification. In: Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition. 2014. 1701−1708.

[13] Tian YC, Pei KX, Jana S, Ray B. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In: Proc. of the 40th Int'l Conf. on Software Engineering. 2018. 303−314.

[14] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang ZH, Karpathy A, Khosla A, Bernstein MS, Berg AC, Li FF. Imagenet large scale visual recognition challenge. Int'l Journal of Computer Vision, 2015, 115(3): 211−252.

[15] Li ZN, Ma X, Xu C, Cao C, Xu J, Lu J. Boosting operational DNN testing efficiency through conditioning. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. 2019. 499−509.

[16] Zhou JY, Li F, Dong JH, Zhang H, Hao D. Cost-effective testing of a deep learning model through input reduction. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering. 2020. 289−300.

[17] Chen JJ, Wu Z, Wang Z, You HM, Zhang L, Yan M. Practical accuracy estimation for efficient deep neural network testing. ACM Trans. on Software Engineering and Methodology, 2020, 29(4): 1−35.

[18] McInnes L, Healy J, Astels S. HDBScan: Hierarchical density based clustering. The Journal of Open Source Software, 2017, 2(11): Article No.205.

[19] Kim B, Khanna R, Koyejo O. Examples are not enough, learn to criticize! Criticism for interpretability. In: Proc. of the 30th Int'l Conf. on Neural Information Processing Systems. 2016. 2288−2296.

[20] Chen TY, Kuo FC, Merkel RG, Tse TH. Adaptive random testing: The ART of test case diversity. Journal of Systems and Software, 2010, 83(1): 60−66.

[21] Deb K, Agrawal S, Pratap A, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002, 6(2): 182−197.

[22] Liu WB, Wang ZD, Liu XH, Zeng NY, Liu YR, Alsaadi F. A survey of deep neural network architectures and their applications. Neurocomputing, 2017, 234: 11−26.

[23] Gatys LA, Ecker AS, Bethge M. Image style transfer using convolutional neural networks. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. 2016. 2414−2423.

[24] Lai SW, Xu L, Liu K, Zhao J. Recurrent convolutional neural networks for text classification. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. 2015. 2267−2273.

[25] Ma L, Juefei-Xu F, Zhang FY, Sun JY, Xue MH, Li B, Chen C, Su T, Li L, Liu Y, Zhao JJ, Wang YD. DeepGauge: Multi-granularity testing criteria for deep learning systems. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. 2018. 120−131.

[26] Pei KX, Cao YZ, Yang J, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. of the 26th Symp. on Operating Systems Principles. 2017. 1−18.

[27] Feng Y, Shi QK, Gao XY, Wan J, Fang CR, Chen ZY. DeepGini: Prioritizing massive tests to enhance the robustness of deep neural networks. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis, Virtual Event. 2020. 177−188.

[28] Quinlan JR. Induction of decision trees. Machine Learning, 2004, 1: 81−106.

[29] Zhang L, Sun XC, Li Y, Zhang Z. A noise-sensitivity-analysis-based test prioritization technique for deep neural networks. arXiv: 1901.00054, 2019.

[30] Ma W, Papadakis M, Tsakmalis A, Cordy M, Traon YL. Test selection for deep learning systems. ACM Trans. on Software Engineering and Methodology, 2021, 30(2): 1−22.

[31] Han Y, Gong DW, Jin Y, Pan QK. Evolutionary multiobjective blocking lot-streaming flow shop scheduling with machine breakdowns. IEEE Trans. on Cybernetics, 2019, 49(1): 184−197.

[32] Qi R, Yen G. Hybrid bi-objective portfolio optimization with pre-selection strategy. Information Sciences, 2017, 417: 401−419.

[33] Liu YP, Yen GG, Gong DW. A multimodal multiobjective evolutionary algorithm using two-archive and recombination strategies. IEEE Trans. on Evolutionary Computation, 2019, 23(4): 660−674.

[34] Deb K. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. Evolutionary Computation, 1999, 7(3): 205−230.

[35] Zitzler E, Laumanns M, Thiele L. Spea2: Improving the strength Pareto evolutionary algorithm. Technical Report, 103, Computer Engineering and Networks Laboratory (TIK), 2001.

[36] Zhang Q, Li H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Trans. on Evolutionary Computation, 2007, 11(6): 712−731.

[37] Shi QK, Chen Z, Fang CR, Feng Y, Xu B. Measuring the diversity of a test set with distance entropy. IEEE Trans. on Reliability, 2016, 65(1): 19−27.

[38] Hartigan JA, Wong MA. Algorithm as 136: A *k*-means clustering algorithm. Journal of the Royal Statistical Society: Series C (Applied Statistics), 1979, 28(1): 100−108.

[39] Warden P. Speech commands: A public dataset for single-word speech recognition. arXiv: 1804.03209, 2017.

[40] Kurakin A, Goodfellow IJ, Bengio S. Adversarial examples in the physical world. arXiv: 1607.02533, 2016.

[41] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv: 1409.1556, 2014.

[42] Kim JH, Feldt R, Yoo S. Guiding deep learning system testing using surprise adequacy. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. 2019. 1039−1049.

[43] Wilcoxon F. Individual comparisons by ranking methods. Breakthroughs in Statistics, 1992: 196−202.

[44] Kocaguneli E, Menzies T, Keung JW, Cok D, Madachy R. Active learning and effort estimation: Finding the essential content of software effort estimation data. IEEE Trans. on Software Engineering, 2013, 39(8): 1040−1053.

[45] Li M, Zhang H, Wu RX, Zhou Z. Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 2011, 19(2): 201−230.

[46] Valentini G, Dietterich TG. Low bias bagged support vector machines. In: Proc. of the 20th Int'l Conf. on Machine Learning. 2003. 752−759.

[47] Jelihovschi EG, Faria JC, Allaman I. ScottKnott: A package for performing the scott-knott clustering algorithm in R. Trends in Applied and Computational Mathematics, 2014, 15(3): 3−17.

[48] Keras. Accessed. 2021. https://keras.io

[49] Tensorflow. 2021. https://www.tensorflow.org/

[50] Fastica. 2021. https://scikit-learn.org/stable/modules/classes.html\#module-sklearn.decomposition

[51] Hdbscan. 2021. https://pypi.org/project/hdbscan/

[52] Geatpy. 2021. http://geatpy.com/

[53] Zhang J, Harman M, Ma L, Liu Y. Machine learning testing: Survey, landscapes and horizons. arXiv: 1906.10742, 2019.

[54] Sun Y, Huang X, Kroening D. Testing deep neural networks. arXiv: 1803.04792, 2018.

[55] Gopinath D, Wang KY, Zhang MS, Pasareanu C, Khurshid S. Symbolic execution for deep neural networks. arXiv: 1807.10439, 2018.

[56] Odena A, Goodfellow IJ. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In: Proc. of the 36th Int'l Conf. on Machine Learning. 2019. 4901−4911.

[57] Xie XF, Ma L, Juefei-Xu F, Xue MH, Chen HX, Liu Y, Zhao JJ, Li B, Yin JX, See S. DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 2019. 146−157.

[58] Gerasimou S, Eniser HF, Sen A, Cakan A. Importance-driven deep learning system testing. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering: Companion Proceeding. 2020. 322−323.

[59] Guo JM, Jiang Y, Zhao Y, Chen Q, Sun J. DLFuzz: Differential fuzzing testing of deep learning systems. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. 2018. 739−743.

[60] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv: 1312.6199, 2014.

[61] Xiao CW, Zhu JY, Li B, He W, Liu M, Song D. Spatially transformed adversarial examples. arXiv: 1801.02612, 2018.

[62] Zhang L, Marinov D, Khurshid S. An empirical study of JUnit test-suite reduction. In: Proc. of the 22nd IEEE Int'l Symp. on Software Reliability Engineering. 2011. 170−179.

[63] Shi A, Yung T, Gyori A, Marinov D. Comparing and combining test-suite reduction and regression test selection. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. 2015. 237−247.

[64] Rothermel G, Harrold MJ, von Ronne J, Hong C. Empirical studies of test-suite reduction. Software Testing, 2002, 12(4): 219−249.

[65] Chen J, Bai Y, Hao D, Zhang L, Xie B. How do assertions impact coverage-based test-suite reduction? In: Proc. of the 2017 IEEE Int'l Conf. on Software Testing. 2017. 418−423.

[66] Chen T, Lau M. A new heuristic for test suite reduction. Information and Software Technology, 1998, 40(5-6): 347−354.

[67] Cruciani E, Miranda B, Verdecchia R, Bertolino A. Scalable approaches for test suite reduction. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. 2019. 419−429.

[68] Harrold MJ, Gupta R, Soffa M. A methodology for controlling the size of a test suite. ACM Trans. on Software Engineering and Methodology, 1993, 2(3): 270−285.

[69] Hua L, Wang CY, Gu Q, Cheng H. Test case set reduction based on genetic ant colony algorithm. Journal of Engineering Mathematics, 2012, 29(4): 486−492 (in Chinese with English abstract).

[70] Nie CH, Xu BW. A method of generating minimum test case set. Journal of Computer Science, 2003, 26(12): 1690−1695 (in Chinese with English abstract).

附中文参考文献:

[69] 华丽, 王成勇, 谷琼, 程虹. 基于遗传蚁群算法的测试用例集约简. 工程数学学报, 2012, 29(4): 486−492.

[70] 聂长海, 徐宝文. 一种最小测试用例集生成方法. 计算机学报, 2003, 26(12): 1690−1695.

沐燕舟(1996－), 男, 硕士, CCF 学生会员, 主要研究领域为机器学习, 并发程序分析, 深度学习测试.

陈俊洁(1992－), 男, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为软件分析与测试.

王赞(1979－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件测试, 机器学习.

赵静珂(1997－), 男, 硕士生, 主要研究领域为深度学习安全质量保证.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件缺陷预测, 软件缺陷定位, 回归测试, 组合测试.

王建敏(1986－), 男, 博士, 助理研究员, 主要研究领域为智能软件测试, 系统仿真.