软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(11):4973−4992 [doi: 10.13328/j.cnki.jos.007005] [CSTR: 32375.14.jos.007005]

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

基于数据变异的神经网络测试用例选择方法*

曹雪洁[1], 陈俊洁[1], 闫 明[1], 尤翰墨[1], 吴 卓[2], 王 赞[1,2]

[1](天津大学 智能与计算学部, 天津 300350)

[2](天津大学 新媒体与传播学院, 天津 300350)

通信作者: 陈俊洁, E-mail: junjiechen@tju.edu.cn

摘 要: 深度神经网络目前已被广泛应用于自动驾驶、医疗诊断、语音识别、人脸识别等安全攸关领域, 因此深度神经网络测试对于保证其质量非常关键. 然而, 为判断 DNN 模型预测是否正确而对测试用例进行标注的成本很高. 因此, 筛选出能够揭示 DNN 模型错误行为的测试用例并优先对其进行标注, 能够尽快修复模型缺陷, 从而提升 DNN 测试的效率、保证 DNN 模型质量. 提出一种基于数据变异的测试用例选择方法 DMS. 该方法设计并实现数据变异算子生成变异模型, 以模拟模型缺陷并捕获测试用例揭错时的动态模式, 从而评估测试用例的揭错能力. 在 25 个深度学习测试集和模型的组合上进行实验, 结果表明, 无论是筛选出的样本中揭错用例的比例还是揭错方向的多样性, DMS 都要显著优于现有的测试用例选择方法. 具体来说, 以原始测试集作为候选集时, 在选择 10% 的测试用例时, DMS 能够筛选出候选集中 53.85%–99.22% 的揭错用例, 在选择 5% 的测试用例时, DMS 筛选出的测试用例已经几乎能覆盖所有的揭错方向. 相较于 8 种对比方法, DMS 平均多找出 12.38%–71.81% 的揭错用例, 证明了 DMS 在测试用例选择任务中的显著有效性.

关键词: 深度学习; 软件测试; 测试用例选择; 数据变异

中图法分类号: TP311

中文引用格式: 曹雪洁, 陈俊洁, 闫明, 尤翰墨, 吴卓, 王赞. 基于数据变异的神经网络测试用例选择方法. 软件学报, 2024, 35(11): 4973–4992. http://www.jos.org.cn/1000-9825/7005.htm

英文引用格式: Cao XJ, Chen JJ, Yan M, You HM, Wu Z, Wang Z. Test Case Selection for Neural Network via Data Mutation. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 4973–4992 (in Chinese). http://www.jos.org.cn/1000-9825/7005.htm

Test Case Selection for Neural Network via Data Mutation

CAO Xue-Jie[1], CHEN Jun-Jie[1], YAN Ming[1], YOU Han-Mo[1], WU Zhuo[2], WANG Zan[1,2]

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

[2](School of New Media and Communication, Tianjin University, Tianjin 300350, China)

Abstract: Nowadays, deep neural network (DNN) is widely used in autonomous driving, medical diagnosis, speech recognition, face recognition, and other safety-critical fields. Therefore, DNN testing is critical to ensure the quality of DNN. However, labeling test cases to judge whether the DNN model predictions are correct is costly. Therefore, selecting test cases that reveal incorrect behavior of DNN models and labeling them earlier can help developers debug DNN models as soon as possible, thus improving the efficiency of DNN testing and ensuring the quality of DNN models. This study proposes a test case selection method based on data mutation, namely DMS. In this method, a data mutation operator is designed and implemented to generate a mutation model to simulate model defects and capture the dynamic pattern of test case bug-revealing, so as to evaluate the ability of test case bug-revealing. Experiments are conducted on the combination of 25 deep learning test sets and models. The results show that DMS is significantly better than the existing test case selection methods in terms of both the proportion of bug-revealing and the diversity of bug-revealing directions in the selected samples.

* 基金项目: 国家自然科学基金 (62002256)
收稿时间: 2022-11-28; 修改时间: 2023-04-06; 采用时间: 2023-07-24; jos 在线出版时间: 2023-11-29
CNKI 网络首发时间: 2023-12-01

Specifically, taking the original test set as the candidate set, DMS can filter out 53.85%–99.22% of all bug-revealing test cases when selecting 10% of the test cases. Moreover, when 5% of the test cases are selected, the selected cases by DMS can cover almost all bug-revealing directions. Compared with the eight comparison methods, DMS finds 12.38%–71.81% more bug-revealing cases on average, which proves the significant effectiveness of DMS in the task of test case selection.

Key words: deep learning; software testing; test case selection; data mutation

1 引 言

近年来, 深度神经网络 (deep neural network, DNN) 在许多领域得到了应用, 并取得了巨大的成功, 例如自动驾驶汽车[1]、人脸识别[2]、语音识别[3]、医疗诊断[4]、飞机防撞系统[5]和软件工程[6]等. 然而, 和传统的软件系统一样, DNN 模型也存在缺陷. 由于深度神经网络具有内部结构复杂、可解释性低等特点, 数据中任何微小的扰动都有可能使得 DNN 做出难以理解的错误预测, 甚至导致严重后果[7]. 例如, 2016 年微软的聊天机器人 Tay 短时间内在社交网络上发表了大量的种族主义推文. 此外, 2021 年 11 月, 在线房地产巨头 Zillow 因购房算法失准而损失百万美元、大幅裁员. 因此, 保证 DNN 模型的质量具有重要的研究意义.

在实际应用中, DNN 测试是保证 DNN 质量最有效的方法之一[8]. 和传统软件测试任务一样, DNN 测试的目标同样是快速且充分地暴露出模型内的缺陷. 然而, 为判断 DNN 模型预测是否正确而对测试用例进行标注的成本很高[9], 主要原因如下: 1) 人工标注仍然是主流方法, 而且通常标注一个测试用例需要多人参与以确保标注的正确性; 2) 测试集的规模一般都是十分庞大的; 3) 在很多情况下, 样本的标注依赖于特定的领域知识, 因此雇用拥有特定领域知识的专家进行标注的成本会更高. 因此, 筛选出能够揭示 DNN 模型错误行为的测试用例优先进行标注, 可以降低标注成本、并帮助开发者尽快进行模型修复等任务, 从而提升 DNN 测试的效率、保证 DNN 模型质量.

现有的测试用例选择方法可以分为两类, 基于结构覆盖的选择方法和基于置信度的选择方法. 基于结构覆盖的选择方法利用 DNN 基本结构的覆盖信息指导测试用例选择. 具体而言, 这类方法借鉴传统软件测试中的代码覆盖思想, 将 DNN 拆解成基础结构神经元, 并利用测试用例对 DNN 的结构神经元覆盖 (neuron coverage, NC)[10]选择潜在的揭错测试用例[11,12]. 然而, 近期的研究表明[13−15], 测试集中的揭错测试用例数量与结构覆盖率之间的关联性较为有限, 即通过结构覆盖率难以有效区分普通测试用例和揭错测试用例, 导致基于结构覆盖的揭错测试用例选择方法的有效性较低.

此外, 基于置信度的测试用例选择方法根据 DNN 模型对测试用例预测得到的置信度向量选择揭错测试用例[16,17]. 具体而言, DeepGini[16]使用置信度向量计算每个测试用例的基尼不纯度 (Gini impurity) 以判断 DNN 模型对测试用例决策的不确定程度, 进而将更为不确定的测试用例赋予更高的优先级. ATS[17]将置信度向量划分为多个区间以表示测试用例的缺陷模式, 进而设计了一种适应度函数来自适应地选择不确定度较高且更多样化的测试用例. 虽然 DeepGini 和 ATS 可以在一定程度上有效地选择揭错测试用例, 但此类测试用例选择方法均使用 DNN 模型预测得到的静态结果来衡量测试用例的不确定度, 以完成揭错测试用例选择任务. 然而, 在实际中, 预测结果信息并不能保证高准确性, 同时预测得到的静态结果与揭错多样性之间关联较弱, 这也意味着上述方法的效果仍需进一步提升.

为了进一步提升揭错测试用例选择的准确性和多样性, 从而提升 DNN 测试的效率, 本文提出了一种基于数据变异的测试用例选择方法 DMS (data-mutation based selection). 相较于已有方法, 基于变异的方法通过模拟模型缺陷、捕获测试用例揭错时的动态模式 (即针对一个给定的测试用例, 从原始模型预测结果转变为变异模型预测结果的动态信息), 从而更为直接且准确地评估测试用例的揭错能力. 值得注意的是, 不同于结构变异对模型参数的直接扰动, DMS 通过对原始训练集进行数据变异, 使得在模型微调过程中, 更为真实地模拟模型的缺陷产生, 获取对缺陷敏感的变异模型. 此外, DMS 在对原始训练集数据进行变异时考虑了方向的差异, 即为训练集中每个被模型预测错误的数据注入不同方向的错误 (例如, 将被原模型预测为 1 的测试用例的训练目标定为 2、3 或者 4), 再进行模型微调, 以衡量测试用例对不同方向错误的敏感性, 从而明确测试用例揭示的变异缺陷类型. DMS 根据测试用例在变异模型上的动态模式对测试用例集合进行分组, 进而以组为单位完成揭错测试用例的选择. 这不仅

考虑组内测试用例的揭错能力, 也考虑不同组间测试用例的揭错多样性.

本文在 25 个实验组合上开展了实证研究, 其中使用的 DNN 模型均用于分类任务. 实验结果表明: 无论是筛选出的样本中揭错用例的比例, 还是揭错方向的多样性, DMS 都要显著优于现有的测试用例选择方法. 具体而言, 以原始测试集作为候选集时, 在选择 10% 的测试用例时, DMS 能够筛选出候选集中 53.85%–99.22% 的揭错用例, 在选择 5% 的测试用例时, DMS 筛选出的揭错用例已经几乎能覆盖所有的揭错方向. 相较于 8 种对比方法, DMS 平均多筛选出 12.38%–71.81% 的揭错用例, 证明了 DMS 在测试用例选择任务中的显著有效性. 综上所述, 本文的主要贡献总结如下.

(1) 本文提出了一种基于数据变异的 DNN 测试用例选择方法 DMS, 该方法设计并实现数据变异算子来生成 DNN 变异模型, 并以此筛选出能够揭示模型不同错误行为的测试用例, 从而提升 DNN 测试的效率.

(2) 本文在由 5 种 DNN 模型和深度学习测试集构造的 25 个实验组合上开展了系统的实证研究, 实验结果表明 DMS 方法相较于已有方法能够平均多找出 12.38%–71.81% 的揭错用例, 从而证明了 DMS 方法的有效性.

(3) 本文实现并开源了 DMS, 将相关代码开源在工具主页 https://github.com/MT010104/DMS, 以便其他研究者开展后续研究.

2 背景知识

2.1 深度神经网络测试及优化

DNN 模型是一种数据驱动的编程范式, 它由多种不同功能的层组成且每层包含大量的神经元[18]. 层与层之间的神经元通过不同的权值进行连接, 这些权重值通过在训练数据上不断训练计算得出. DNN 模型基于这些计算出的权重, 将输入映射到特定的输出域. DNN 测试是保证其质量最广泛使用的方法之一[2,10,19]. 与传统的软件系统一样, DNN 测试研究工作也重点关注测试用例[7]和测试预言[20]两方面. DNN 测试用例研究主要关注 DNN 模型输入的生成[7,10], 排序[9,16]和选择[12,17]. 其中 DNN 模型的输入在不同的具体任务中形式也有所不同, 如图像[1]、自然语言文本[21]或语音输入[22]等. DNN 测试预言则关注在给定测试输入时 DNN 模型的输出应该符合的预期. 在现阶段研究过程中, 诸多传统软件测试中的测试预言已被应用到了 DNN 测试中. 例如, Nejadgholi 等人[23]对深度学习库单元测试中的预言近似 (oracle approximations) 问题进行研究; Guo 等人[20]首次将差异测试 (differential testing) 应用到深度学习模型的模糊测试中. DNN 测试用例优化研究聚焦于减少 DNN 测试过程中诸如人工标注等高昂的测试成本, 以此来提高 DNN 测试的效率.

目前, DNN 测试用例优化方法主要分为两类: 1) 通过选择小规模测试用例子集来估计整个测试集的准确率, 以此减少开发人员标注测试用例集的成本同时保证对 DNN 模型质量的估算程度. Li 等人[24]提出了基于置信度的分层采样 (confidence-based stratified sampling, CSS) 和基于交叉熵的采样 (cross entropy-based sampling, CES) 两种方法对 DNN 模型测试集合进行约减. 与此同时, DNN 模型在被约减后的测试用例子集上的精度与在原始测试集上的精度差异最小. 其中, CSS 方法利用 DNN 模型对原始测试集中各个测试输入的预测置信度划分区间, 并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用例. Zhou 等人[25]提出了一种两阶段顺序采样的选择方法 DeepReduce, 该方法利用贪心选择策略, 以提升神经元覆盖率为指导, 迭代地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[26]提出了一种基于聚类的实用准确率估计方法 PACE, 该方法通过对待测 DNN 模型抽取的测试用例特征进行聚类, 并使用 MMD-critic 采样方法和自适应随机选择方法对正常样本簇和离群样本簇进行采样, 将从各个簇中取出的测试输入合并成最终的测试子集. 2) 筛选出能够揭示模型错误行为的测试用例并优先对揭错测试用例进行标注, 从而尽早发现 DNN 模型中的缺陷. Guerriero 等人[27]提出了 DeepEST, 该方法能够利用概率统计方法估算测试用例被错误预测的可能性. Feng 等人[16] 提出了一种测试输入排序方法 DeepGini, 其基于待测 DNN 模型对测试用例预测的置信度计算基尼不纯度并完成排序, 基尼不纯度高的测试用例将被优先选

择. Zhang 等人[28]提出根据测试用例的噪声敏感性对测试用例进行排序. 他们发现通过在测试用例中加入相同的噪声, 噪声灵敏度高的测试用例比噪声灵敏度低的测试用例更容易欺骗 DNN 模型. 本文的工作属于第 2 类揭错用例选择问题, 即筛选出揭错能力强的测试用例以揭示 DNN 模型的缺陷.

2.2 深度神经网络变异

变异测试是一种用于评估测试用例质量的重要方法[29−33]. 传统软件中的变异测试通过对代码进行变异生成大量存在潜在问题的代码即变异体[29,30], 而产生这些变异体的不同变异规则与操作称为变异算子. 研究人员通常以测试用例能够将多少变异体的错误暴露出来作为其质量的度量标准. 严格来说, 变异测试旨在评价测试用例的质量, 并没有生成新的测试用例, 但是评价测试用例质量对进一步地生成更高质量的测试用例具有重要的指导意义. 在 DNN 测试中, 如何衡量测试的充分性, 以及如何生成质量更高、更容易揭示模型缺陷的测试用例仍然是测试领域的重要研究问题.

Ma 等人[31]借鉴了传统软件变异测试的思想, 将其应用到针对 DNN 模型的测试中并提出了 DeepMutation. 它从源码级别和模型级别两个方面入手, 分别提出了 8 种变异算子对 DNN 系统的训练程序、训练数据以及模型文件进行变异. 后来他们又在此基础上提出了 DeepMutation++[32], 专门针对循环神经网络 (recurrent neuron network, RNN) 提出了 9 种模型级别的变异算子, 其中两种静态变异算子主要针对权重进行变异, 而剩余的 7 种动态变异算子主要针对隐藏状态和“门”进行变异. Humbatova 等人[33]进一步细化变异方法提出了 DeepCrime, 为模拟深度神经网络中的真实错误从训练数据、验证数据、正则化方法、激活函数、损失函数、优化函数等角度出发共设计了 35 种变异算子. 对于变异测试, 变异算子的变异有效性是测试效果的重要保证. 针对测试用例选择任务, 本文设计了一种基于数据变异的变异算子, 与 DeepCrime 和 DeepMutation 中提出的基于训练数据的变异算子相比, 两者的区别主要在于以下 3 个方面: 1) DeepMutation 和 DeepCrime 随机挑选一部分测试用例进行变异, 这种错误注入有可能影响模型对正确测试用例的认知从而导致识别揭错测试用例的有效性降低, 而 DMS 仅对被原模型预测错误的测试用例进行变异; 2) DeepCrime 统一将测试用例的标签更改至某一特定的标签, 而 DMS 使用的变异算子通过考虑多种不同错误注入方向对模型进行微调从而使测试用例能够揭示不同方向的变异缺陷; 3) DeepCrime 和 DeepMutation 中基于训练数据的变异算子均需要对模型进行完整的训练, 而 DMS 仅需要对原模型进行微调, 节省了大量的计算资源和时间, 进一步提升了模型变异过程的效率. 后续的实验中还将对比 DMS 和 DeepMutation 中提出的通用变异算子如 GF、WS、NAI 和 NEB 的变异有效性.

3 基于数据变异的测试用例选择方法

现有的测试用例选择方法主要分为两类: 1) 以神经元覆盖率为指导的测试用例选择方法. 此类方法能够在选择测试用例的同时, 最大化测试用例集合在 DNN 模型上的神经元覆盖率, 从而实现对 DNN 模型的充分测试. 2) 以 DNN 模型预测的置信度为指导的测试用例选择方法. 此类方法依据 DNN 模型预测过程中的静态信息衡量测试用例的不确定性, 一定程度上依赖于静态信息的准确性即模型本身的精度. 然而, 上述两类方法均无法捕获缺陷触发的动态模式 (如缺陷导致本应预测为 0 的测试用例被预测为 1, 具有特定的缺陷方向), 以更充分地揭示模型的不正确行为. 因此本文提出了一种基于数据变异的测试用例选择方法 DMS, 通过定义数据变异算子对 DNN 模型进行变异. 相较于通用的结构变异算子而言, 它能够将测试用例的揭错能力、揭示的变异缺陷和在变异前后模型上的不一致信息直接联系在一起. 具体来说, DMS 首先根据能够杀死变异模型的个数对测试用例的揭错能力进行度量; 而后根据原模型和变异模型的预测结果之间的改变对测试用例进行分组, 以区分测试用例对于不同方向缺陷的检测能力, 并综合考虑上述两方面因素对测试用例进行选择.

图 1 展示了 DMS 的主要工作流程, 其主要分为 4 个部分: 模型变异、特征信息提取、测试用例分组、采样. 本文在第 3.1 节中详细介绍了如何对训练数据进行变异以及变异模型的生成, 在第 3.2 节介绍了如何提取测试用例的特征信息及其形式化定义, 在第 3.3 节中介绍了如何基于特征信息对测试用例进行分组和采样, 最后在第 3.4 节中结合伪代码对 DMS 的流程进行了阐述.

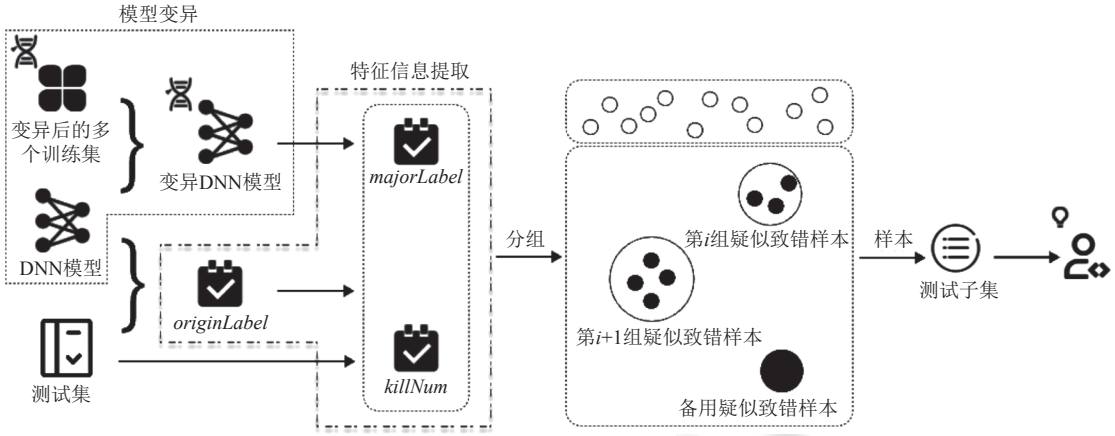

图 1 DMS 方法流程图

3.1 基于训练数据的模型变异

现阶段, DeepMutation 和 DeepCrime 等工作提出了基于训练数据的 DNN 模型变异算子, 例如删除部分训练数据、在数据中加入噪声等. 然而, 此类变异算子需要对原始模型进行完整的训练, 从而导致基于训练数据的模型变异过程效率过低. 对于一个深度学习任务而言, 相对于重新训练新的网络, 微调模型能节省大量的计算资源和计算时间, 进而提高计算效率[34]. 因此, 为了对原模型进行有效变异, DMS 设计了一种特定数据变异算子, 其通过随机更改错误样本的标签对训练数据进行变异. 并利用变异后的训练数据对已经训练好的模型进行微调, 从而提升模型变异过程的效率.

在单次变异过程中, 错误样本的标签可能被修改成不同于原模型预测结果的任一标签. 具体而言, 假设所给 DNN 模型为 M, 训练数据对 (x,y) 包括输入 x, 以及其真实标签 y. $y'=M(x)$ 为模型 M 对输入的预测类别标签. 我们对所有 $y'(x)\neq y(x)$ 的训练数据对 (x_i,y_i) 进行变异. 单次的变异规则为, 在类别标签集合 L 中随机选择 y_{i_mu} 且符合 $y_{i\_mu}\neq y'(x_i)$, 作为 x_i 的新标签. 因此, 本文针对训练数据对 (x_i,y_i), 构造了变异数据 (x_i,y_{i_mu}).

为了将错误以各种可能的方向动态地注入模型, DMS 对原始模型重复多轮变异, 以生成一定数量的具有不同缺陷的变异模型. 为了更有效地区分正确样本和错误样本, DMS 在变异过程中, 仅对训练集中的错误样本的标签进行修改, 通过这种方式, 微调后的模型相较于原模型在训练集的正确样本和错误样本上的表现会有较大的差异, 即基于数据的独立同分布特性[35], 大多数情况下对于正确样本来说, 变异模型和原模型的预测标签相同, 而对于错误样本则会产生区别.

3.2 特征信息提取

DMS 分析测试用例在变异模型和原模型上的动态行为差异, 综合考虑测试用例揭错能力和对于不同方向缺陷的检测能力两种特征信息, 对测试用例进行分组. 具体特征信息定义如下.

(1) 揭错能力: 在传统的变异测试中, 如果源程序的变体在同一测试用例上的执行情况与源程序不同, 就认为变体被这个测试用例杀死[36]. 同理, 在深度学习中, 如果变异模型在同一测试用例上的预测标签与原模型不同, 则此变异模型被杀死. 一般来说, 如果一个测试用例能够杀死更多的变异模型, 其可能具有更强的揭错能力. 此类测试用例更有可能会是被 DNN 模型预测错误的样本. 对于每一个测试用例, 它能够杀死的变异模型的总数 *killNum* 是衡量其揭错能力的重要度量标准.

(2) 变异方向: 对于一个可能揭错的测试用例而言, 它在大部分变异模型上的预测标签与原模型的预测标签是不同的, 且变异模型上的预测标签相互之间也并不完全相同. 为了进一步明确测试用例揭示的变异缺陷类型, DMS 依据多数投票算法[37]统计它在所有变异模型上出现频次最多的标签 *majorLabel* 作为变异模型集合上的预测标签, 并将原模型对其预测标签 *originLabel* 到 *majorLabel* 的变化方向定义为测试用例的变异方向, *originLabel*

和 *majorLabel* 不同的测试用例将其定义为疑似揭错样本; 对于 *originLabel* 和 *majorLabel* 相同的测试用例, 需要借助 *killNum* 再次进行区分.

以一个四分类任务为例, 假设 4 个测试用例 x_1, x_2, x_3, x_4 在 6 个变异模型 M_1, M_2, M_3, M_4, M_5 和 M_6 上的预测结果如表 1 所示. 对于测试用例 x_1 而言, 它在 M_2, M_4 和 M_5 上的预测标签都是 2, 在 M_1 上预测为 1, 在 M_3 和 M_6 上预测为 3, 根据多数投票算法 x_1 的 *majorLabel* 为 2, 因此 x_1 为疑似揭错样本. 同理可知, 在表 1 中 x_1 和 x_2 均为疑似揭错样本, 变异方向不同的测试用例更有可能揭示 DNN 模型中不同的缺陷. 而 x_3 和 x_4 的 *majorLabel* 和 *originLabel* 保持一致, 但 x_4 的 *killNum* 高于 x_3 的 *killNum*, 因此更有可能是一个揭错样本.

表 1 特征信息提取示例

ID	*originLabel*	*predictLabel*						*killNum*	*majorLabel*
		M_1	M_2	M_3	M_4	M_5	M_6		
x_1	0	1	2	3	2	2	3	6	2
x_2	1	3	1	1	3	2	3	4	3
x_3	2	2	2	2	2	2	2	0	2
x_4	3	3	3	3	1	0	3	2	3

3.3 测试用例分组及采样

为了能够尽可能地选择揭错行为更多样的测试用例子集, DMS 依据变异方向对测试用例进行过滤和分组. 其原因在于, 不同的测试用例具有不同的变异方向, 而不同的变异方向更有可能揭示 DNN 模型中不同的缺陷.

DMS 首先过滤掉没有杀死任何一个变异模型的样本, 因为 DMS 无法衡量此类样本的揭错能力和揭错行为, 被过滤后的样本会被放入过滤集合 A 中. 在剩余的样本中, 虽然有些样本能够杀死部分变异模型, 由于杀死变异模型数量不足, 其 *originLabel* 和 *majorLabel* 可能是相同的. DMS 将此类样本称之备用疑似揭错样本, 并将其统一放入备选集合 B 中. 此时, 过滤集合 A 和备用集合 B 之外的样本被称为疑似揭错样本. 而后, DMS 依据样本的变异方向对所有疑似揭错样本进行分组. 具体来说, 针对每一个疑似揭错样本, DMS 使用二元组 (*originLabel*, *majorLabel*) 表示其变异方向, 变异方向相同的样本会被划分到相同的组中.

在测试用例采样阶段, DMS 会按照分组规模和采样的用例总个数, 确定每个分组所要采样的测试用例个数. 在确定分组采样个数之后, DMS 会对每个分组内的样本, 按照其 *killNum* 进行排序并采样. 其原因在于 DMS 认为相同变异方向下, *killNum* 值越高的测试用例更有可能是揭错样本. 在某些情况下, 受模型变异等随机因素影响, 疑似揭错样本的总个数可能会小于要采样的总个数. 此时, DMS 会从对备选集合 B 中的测试用例按照 *killNum* 排序, 并从中采样部分样本, 以解决疑似揭错样本过少导致的采样不足的问题. 当备选集合亦不能满足采样需求时, DMS 会从过滤集合 A 中随机采样部分样本作为补充. 在本文中, 此极端情况并没有发生, 即 DMS 变异产生的变异模型在绝大多数情况下可以有效地区分测试用例的揭错行为. 为确保方法完整性, 本文对此特殊情况作出说明.

3.4 DMS 方法形式化流程

算法 1 展示了 DMS 的伪代码, 其输入包括: 待测 DNN 模型 m, 原始训练集 T, 候选集 C, 预先设定的测试用例选择数量 n. DMS 方法首先利用特定的数据变异算子对训练数据进行多次随机变异, 并对原模型进行重训练得到一定数量且多样的变异模型 (行 1, 2). 然后, DMS 获取原始模型对候选集合的预测结果 *originLabels* (行 3); 同时, DMS 获得所有变异模型对候选集 C 的预测标签 $L_{MutatedModels}$ (行 4–8). 对候选集 C 中的每一个测试用例, DMS 通过对比变异模型和原模型的预测结果的差异, 得到特征向量 f_1 和 f_2, 其中 f_1 表示所有变异模型被测试用例杀死的情况, 维度为变异模型个数, 而 f_2 表示测试用例在所有变异模型上的预测标签的统计信息, 维度为数据集的标签类别数, 继而根据两个特征向量提取重要信息即测试用例杀死的变异模型个数 *killNum* 和由变异模型预测结果“投票”得出的 *majorLabel* (行 9–14). 由于存在部分测试用例的 *killNum* 为 0, 没有杀死任何一个变异模型, 其为揭错用例的可能性较小, 因此 DMS 对候选集进行过滤, 将此类用例从候选集中移除, 并加入过滤集合 A 中, 其余的测试用例被加入到候选集子集 C' 中 (行 15). 然后, DMS 进一步从 C' 中分离出疑似揭错样本 S 和备用疑似揭错样本 B

(行 16). 接下来, DMS 判断疑似揭错样本 S 的数量是否足以完成采样目的. 如果无法完成, 则将所有疑似揭错样本加入到 X 中, 如果 B 中的样本能够满足采样需求, 对备选集合 B 按中的测试用例按照 *killNum* 进行排序并选择 *killNum* 靠前的样本, 加入 X 中 (行 17–21), 否则将所有备用疑似揭错样本加入 X 后还需从过滤集合 A 中随机采样部分样本作为补充; 如果疑似样本 S 的样本足以满足采样目标, DMS 会对按照每个样本的变异方向, 对疑似样本进行分组, 并获得分组后的结果 S' (行 23); 而后, 针对每个细分后的分组, DMS 首先根据该分组在 S 中所占比例确定在该组上的采样数量. 然后, DMS 按照 *killNum* 对该分组进行排序, 获取分组中 *killNum* 靠前的样本, 加入到 X 中, 直到完成采样任务.

算法 1. DMS 方法流程.

输入: 待测 DNN 模型 m; 原始训练集 T; 候选集 C; 预先设定的选择的样本个数 n;
输出: 挑选出的测试用例集合 X, 其大小为 n.

1. $T' \leftarrow Mutate(T)$
2. $MutatedModels \leftarrow Retrain(T', m)$
3. $originLabels \leftarrow predictLabels(C, m)$
4. $L_{MutatedModels} \leftarrow \{\}$
5. **foreach** m' in $MutatedModels$ **do**
6. $\quad L_{m'} \leftarrow predictLabels(C, m')$
7. $\quad L_{MutatedModels} \leftarrow L_{MutatedModels} \cup \{L_{m'}\}$
8. **end**
9. $killNum \leftarrow \{\}, majorLabels \leftarrow \{\}$
10. **foreach** c_i in C **do**
11. $\quad f_{i1}, f_{i2} \leftarrow extractFeatures(c_i, originLabels, L_{MutatedModels})$
12. $\quad killNum[c_i] \leftarrow sum(f_{i1})$
13. $\quad majorLabels[c_i] \leftarrow \arg\max(f_{i2})$
14. **end**
15. $A, C' \leftarrow filter(C, killNums)$
16. $B, S \leftarrow divideBackup(C', originLabels, majorLabels)$
17. **if** $|S| < n$ **then**
18. $\quad X \leftarrow S$
19. $\quad$ **if** $|B| > (n - |S|)$ **then**
20. $\quad\quad B \leftarrow prioritize(B, killNums)$
21. $\quad\quad g \leftarrow getTop(B, n - |S|)$
22. $\quad$ **else**
23. $\quad\quad X \leftarrow (X \cup B)$
24. $\quad\quad g \leftarrow randomChoice(A, n - |S| - |B|)$
25. $\quad\quad X \leftarrow X \cup g$
26. **else**
27. $\quad S' \leftarrow grouping(S, originLabels, majorLaebels)$
28. $\quad$ **foreach** s'_k in S'
29. $\quad\quad s'_k \leftarrow prioritize(s'_k, killNums)$
30. $\quad\quad t \leftarrow |s'_k| / |S'| \times n$
31. $\quad\quad g \leftarrow getTop(s'_k, t)$

32. $X \leftarrow X \cup g$
33. **end**
34. **end**
35. Return X

4 实验设计

4.1 实验研究问题

为了全面地探究 DMS 在揭错测试用例选择方面的整体有效性, 评估 DMS 各组件对整体有效性的贡献, 本文设计了如下 3 个研究问题.

研究问题 1: 与现有方法相比, 在给定所需要选择的测试用例规模下 DMS 能否选出更多的揭错用例?

研究问题 2: 与现有方法相比, 在给定所需要选择的测试用例规模下 DMS 选出的测试用例是否能覆盖更多的揭错方向?

研究问题 3: 相较于结构变异, DMS 中所使用的数据变异算子对于揭错测试用例的选择是否更有优势?

4.2 数据集和模型

本文使用了 5 组分类任务中主流的深度神经网络模型和数据集作为实验对象. 表 2 给出了 5 组实验对象中模型、训练集和测试集的具体信息. 其中, 第 5–8 列分别是训练集规模、模型训练集准确率、测试集规模以及模型测试集准确率, 最后一列是模型中可训练的参数个数, 即通过数据变异可改变的参数个数.

表 2 DNN 模型及其数据集

ID	数据集名称	模型	模型大小 (KB)	训练集规模	训练集准确率 (%)	测试集规模	测试集准确率 (%)	可训练参数规模
1	MNIST	LeNet1	115	60 000	94.50	10 000	94.86	7 206
2		LeNet4	969	60 000	96.48	10 000	96.79	77 998
3		LeNet5	1 330	60 000	99.08	10 000	98.72	107 786
4	CIFAR10	VGG16	20 109	50 000	95.17	10 000	87.41	1 671 114
5		ResNet20	3 590	50 000	98.56	10 000	91.22	273 066

数据集: 本文选用了现有研究中广泛使用的两个分类任务实验数据集 MNIST 和 CIFAR10. MNIST 是一个手写数字数据集[38], 其包含了 60 000 张训练数据和 10 000 张测试数据; CIFAR10 数据集包含了现实中常见对象 (飞机、汽车、鸟类等) 的低分辨率图片[39], 由 50 000 张训练图片和 10 000 张测试图片组成. 上述两个数据集均用于 10 分类任务.

深度神经网络模型: 本文使用了 3 种主流的卷积神经网络. LeNet 模型是图像分类任务中的经典神经网络结构, 包含了卷积层、池化层、全连接层等基本组件, 其常被用于识别手写数字集[38]. 相比于 LeNet, VGG 模型的网络规模更大, 结构更深, 其使用较小的卷积核和池化层, 使其能够在获得更多图像特征的同时控制参数的个数, 避免过多的计算量以及过于复杂的结构[40]. ResNet 模型在 VGG 模型的基础上引入了残差块, 通过这种方式在计算中保留部分原始信息, 同时能够防止反向传播产生梯度弥散[41]. 3 种神经网络在相应数据集上的分类准确率如表 2 中所示.

4.3 测试用例候选集构造

本文在构造测试用例候选集时综合考虑了自然样本 (数据集提供的原始样本) 和对抗样本. 对于后者, 本文针对原始测试集中的每个测试用例, 分别使用常见的 4 种对抗样本生成方法 C&W (Carlini&Wagner)[42]、BIM (basic iterative methods)[43]、JSMA (Jacobian-based saliency map attack)[44]和 FGSM (fast gradient sign method)[45]生成对应的对抗样本. 本文为每个数据集分别构造 5 个规模与原始测试集大小相同的测试用例候选集, 包括原始测试集中采样得到的自然候选集 C, 以及 4 个分别混合了不同对抗样本产生的对抗候选集 C_{CW}, C_{BIM}, C_{JSMA} 和 C_{FGSM} 每个对

抗候选集中自然样本和对抗样本各占 50%. 在每一个候选集上, 本文会应用每一个所探究的测试用例选择方法, 从而进行充分对比.

4.4 评测指标

为了衡量 DMS 和对比方法的有效性, 本文沿用了 ATS 中的两个评测指标.

(1) *fault_rate*. 对于一个测试用例选择方法而言, 它所选出的揭错测试用例数量更多, 则意味着它在该测试用例选择任务上效果更好. 对于测试候选集 C 和选定的测试用例子集 X, 错误样本率的计算公式如 (1) 所示:

$$fault\_rate = \frac{|X_{\text{wrong}}|}{|C_{\text{wrong}}|} \tag{1}$$

其中, $|X_{\text{wrong}}|$ 代表 X 中被待测 DNN 模型误分类的测试用例个数 (即揭错测试用例数量), 而 $|C_{\text{wrong}}|$ 代表候选集 C 中揭错测试用例的总数. ATS 中 *fault_rate* 表示 X 中误分类的测试用例比例, 区别在于计算公式中的分母代表 X 而非候选集中揭错用例的总数. 原因在于部分候选集中的揭错用例较少, 而如果所需筛选的测试用例数量超过揭错用例总数, 会导致 *fault_rate* 的理想最大值小于 1, 甚至出现选择更多测试用例而 *fault_rate* 下降的情况. 因此本文对此指标稍作修改.

(2) *fault_type*. 由于相似的错误可能反映的是 DNN 中的相似的缺陷, 为了更全面地分析模型, 测试用例选择方法不仅要能筛选出揭错能力较强的测试用例, 还要考虑测试用例所能够揭示的真实错误的多样性. 对于一个被误分类的测试用例 x, 其错误类型计算过程如下:

$$fault\_type(x) = label(x)^* \rightarrow label(x) \tag{2}$$

其中, $label(x)^*$ 代表它的真实标签, 而 $label(x)$ 代表 DNN 模型对其预测的标签. 例如, 手写数字“1”被模型预测为“7”, 那它的揭错方向就可以表示为: $fault\_type(x) = 1 \rightarrow 7$; 在 10 分类任务场景中, 错误类型的上限为 90 (9×10).

4.5 实验设计

4.5.1 研究问题 1

本文使用了目前最主流的 7 个测试用例选择方法作为基线方法, 以此评估 DMS 选择揭错测试用例方面的有效性. 本文所用基线方法具体介绍如下.

SRS (简单随机采样)[24]: 从整个候选集中随机选择一定数量的测试用例, 其中每个测试用例被选择的可能性相同.

NC[10]: 基于最为常见的神经元覆盖率 NC 指导测试用例选择. 此外, 针对神经元覆盖率而言本文考虑了以下两种测试用例的选择策略.

(1) CTM (coverage-total method): 不考虑已选择的测试用例集合, 它总是选择具有最高覆盖率的测试用例.

(2) CAM (coverage-additional method): 根据已选择的测试用例集合动态地调整选择策略, 它总是选择能够覆盖更多未覆盖神经元的测试用例.

LSA (likelihood-based surprise adequacy)、DSA (distance-based surprise adequacy): 分别基于核密度估计 (kernel density estimation, KDE) 和基于欧氏距离计算意外充分性 SA (surprise adequacy), 通过比对测试测用例与训练数据集的 SA 值差异来表示测试用例相对训练数据的意外值, 意外值越高的测试用例优先级越高.

DeepGini[16]: 基于 DNN 模型对测试用例预测的置信度来计算出基尼不纯度并完成排序, 基尼不纯度高的测试用例将被优先选择.

DeepEST[27]: 从自适应采样的角度出发, 每次选择测试用例时, 随机采样的概率为 r, 基于权重采样的概率为 $1–r$. 基于权重的采样方法更倾向于选择错误样本, 而随机采样能够避免 DeepEST 陷入局部最优. DeepEST 又可以被细分为基于预测置信度分配权重的方法 $\text{DeepEST}_{\text{CS}}$、基于 LSA 分配权重的方法 $\text{DeepEST}_{\text{LSA}}$、基于 DSA 分配权重的方法 $\text{DeepEST}_{\text{DSA}}$ 和同时基于预测置信度和 DSA 分配权重的方法 $\text{DeepEST}_{\text{C}}$. 先前的研究已经表明 $\text{DeepEST}_{\text{C}}$ 的效果总体上要优于另外 3 个方法, 所以这里仅将 $\text{DeepEST}_{\text{C}}$ 作为对比方法.

ATS[17]: 提出了一种映射关系将测试用例的输出转换为一组区间来描述选择的测试用例或测试用例集合的缺陷模式, 又进一步提出了一种适应度函数来衡量候选集和已选测试用例集合之间的差异, 最终根据缺陷模式和适应度函数自适应地选择测试用例.

为更加公平地比较 DMS 和各种基线方法的有效性, 本文参考了 ATS 中的实验设置, 首先运行每种选择方法筛选出候选集 5% 和 10% 的测试用例, 而后分别计算每种方法在两种测试用例数量下的 *fault_rate* 数值.

4.5.2 研究问题 2

本文使用研究问题一中提到的基线方法作为对比方法, 评估 DMS 选出的测试用例是否能覆盖更多的揭错方向. 除此之外, 为了证明测试用例分组这一步骤对 DMS 在揭错多样性方面的贡献, 本文还构建了 DMS 的变体方法 SimpleDMS, 其仅根据 *killNum* 对测试用例降序排序, 优先选择 *killNum* 较高的测试用例.

本文首先运行每种选择方法, 选出一组测试用例, 其中测试用例数量的大小范围为 [50, 500], 间隔大小为 50, 然后分别计算每种方法在各测试用例数量下的 *fault_type* 数值. 由于 *fault_type* 的上限数值相较于揭错用例的数量而言比较小, 例如 CIFAR10-ResNet20-origin 的揭错用例数量为 878, 而 *fault_type* 上限为 90, 在选择 5% 的测试用例时, 部分方法的 *fault_type* 已临近上限, 无法对比随着选择测试用例数量的增加各方法 *fault_type* 的上升速率. 为了更清晰地展示各方法选择出的测试用例在揭错多样性方面的差异, 因此本文设置了与研究问题一中不同的测试用例数量选择范围和间隔设置.

4.5.3 研究问题 3

为评估 DMS 所生成的变异模型相比于利用结构变异生成的变异模型在测试用例选择任务上是否更加有效, 本文构造了 DMS 的变体方法——SMS (structure-mutation based selection), 其利用通用结构变异算子对模型进行变异进而提取测试用例的特征信息. 其余方法细节与 DMS 保持一致, 以此来对比两种方法的有效性.

SMS 利用现有工作 [33,46]中使用到的 4 个通用结构变异算子 GF、WS、NAI 和 NEB 生成变异模型. 由于较大程度的变异往往会大幅降低 DNN 模型的预测能力, 从而生成对所有测试用例全部预测错的低质量无效模型变异体, 而轻微变异能够更好地模拟 DNN 的缺陷, 以对 DNN 模型进行轻微扰动, 从而找到揭错测试用例. 具体而言, 在每次扰动时, SMS 随机挑选神经网络中某一层中特定比例的神经元或权重进行变异, 而并非对全部的神经元或者权重进行变异. 为了进一步控制扰动的影响, 通过在小部分数据上不同参数的实证研究, 最终将算子扰动强度控制在 0.5 以内, 每次变异时生成 (0, 0.5] 间的随机数作为扰动强度, 具体的变异规则如下.

(1) WS (weights shuffling): 随机打乱部分链接的权重.

(2) GF (Gauss fuzzing): 向链接的权重添加数值满足高斯分布的噪声.

(3) NAI (neuron activation inverse): 改变神经元输出的符号以颠倒其激活状态.

(4) NEB (neuron effect block): 将神经元与下一层链接的权重置为 0 以阻止其对下一层的影响.

由于 SMS 应用时测试用例能够杀死的变异模型个数较少, 无法通过投票算法计算出 *majorLabel*, 所以其只能根据 *killNum* 对测试用例进行排序直接进行选择.

4.6 实现及环境配置

本文使用 Python 3.6.13 实现了 DMS, 并基于 Keras 2.2.4 和 TensorFlow 1.14.0 对原始模型进行变异并提取特征信息. 对于对比方法 DeepGini、DeepEST 和 ATS, 采用了作者提供的代码和参数. 相关所有的实验均在 Intel Xeon E5-2640 服务器上完成, 操作系统为 Ubuntu 18.04.2, 内存为 128 GB.

5 结果分析

研究问题 1: 与现有方法相比, 在给定所需要选择的测试用例规模下 DMS 能否选出更多的揭错用例?

表 3 中给出了 DMS 及各对比方法在选择揭错测试用例的整体有效性 (*fault_rate* 数值) 上的对比结果 (分别选择候选集 5% 和 10% 的测试用例), 即各方法选择出的测试用例占候选集中所有揭错用例的比例. 其中, 加粗数据表示每个实验场景下, 所有选择方法中最优方法对应的 *fault_rate* 数值.

表 3 各方法选择出的揭错用例占候选集中揭错用例的比例

数据集	模型	选择用例比例 (%)	No.	候选集	对比方法选择出的揭错用例占比 (%)							
					NC		LSA	DSA	DeepGini	ATS	DeepEST	SRS	DMS
					CTM	CAM							
MNIST	LeNet1	5	1	Origin	9.73	7.98	31.13	42.61	44.75	41.44	20.04	4.67	**84.05**
			2	+CW	4.58	6.02	24.07	27.79	25.14	24.5	20.49	4.87	**35.82**
			3	+BIM	6.55	5.98	24.48	28.22	28.29	28.65	21.53	4.9	**35.57**
			4	+FGSM	7.8	6.01	23.69	28.63	30.78	31.5	21.62	4.87	**35.79**
			5	+JSMA	4.94	6.01	24.41	28.7	26.27	25.7	21.55	4.87	**35.79**
		10	6	Origin	15.95	10.51	48.25	68.09	69.84	67.32	43.58	10.31	**96.11**
			7	+CW	10.6	9.89	41.12	52.44	54.08	45.34	40.19	10.17	**71.49**
			8	+BIM	12.46	9.94	40.82	52.48	55.22	50.4	41.47	10.15	**71.35**
			9	+FGSM	14.46	9.95	41.37	53.97	58.2	54.76	42.66	10.16	**71.51**
			10	+JSMA	10.59	9.95	41.23	52.97	54.69	46.96	42.45	10.16	**71.37**
	LeNet4	5	11	Origin	13.71	10.59	46.73	59.5	61.37	59.19	35.2	4.67	**91.28**
			12	+CW	10.52	6.28	28.96	32.87	31.81	29.28	20.8	4.4	**40.46**
			13	+BIM	10.8	6.3	28.81	33.39	33.39	32.73	19.89	4.42	**40.54**
			14	+FGSM	10.84	6.36	28.44	34.31	36.67	36.19	21.76	4.56	**40.59**
			15	+JSMA	11.98	6.44	28.69	32.76	33.25	31.3	24.37	4.4	**40.67**
		10	16	Origin	22.74	11.84	69.78	79.13	83.49	83.18	69.47	7.48	**98.75**
			17	+CW	18.11	10.11	49.51	61.5	69.9	47.15	40.62	9.38	**80.18**
			18	+BIM	18.09	10.15	49.02	61.78	70.05	53.6	42.23	9.41	**79.93**
			19	+FGSM	20.13	10.19	48.17	63.49	70.82	62.35	45.97	9.37	**80.2**
			20	+JSMA	19.8	10.27	48.08	61.04	70.01	53.06	49.39	9.37	**80.44**
	LeNet5	5	21	Origin	14.84	7.81	60.16	85.16	82.03	80.47	79.69	3.12	**94.53**
			22	+CW	8.21	5.44	36.96	43.36	46.13	31.52	37.15	4.68	**47.18**
			23	+BIM	8.02	5.54	37.44	43.84	46.32	36.68	37.82	5.16	**47.18**
			24	+FGSM	9.36	5.73	36.01	43.84	46.42	44.7	36.1	4.68	**47.28**
			25	+JSMA	12.61	5.54	36.77	43.55	46.32	41.07	36.29	5.16	**47.47**
		10	26	Origin	23.44	8.59	78.12	93.75	95.31	93.75	82.81	6.25	**99.22**
			27	+CW	15.09	9.55	64.09	78.22	**91.6**	61.41	72.21	9.93	90.35
			28	+BIM	13.75	9.65	63.99	79.56	**92.17**	63.99	74.5	9.65	89.76
			29	+FGSM	17.1	9.84	63.61	79.66	**92.17**	68.67	73.07	9.93	88.54
			30	+JSMA	19.1	9.65	64.37	78.22	**91.88**	66.09	73.64	9.65	90.45
CIFAR10	VGG16	5	31	Origin	5	4.92	10.64	21.37	23.75	22.4	15.25	4.45	**31.22**
			32	+CW	4.78	4.92	10.95	17.99	17.95	18.19	16.06	4.78	**23.3**
			33	+BIM	3.91	4.87	9.36	18.77	20.26	19.3	16.21	4.97	**23.2**
			34	+FGSM	3.67	4.82	8.15	18.38	21.37	20.74	17.12	4.78	**22.91**
			35	+JSMA	3.81	4.82	5.35	5.16	20.07	19.92	16.5	5.5	**23.35**
		10	36	Origin	8.5	10.09	18.03	38.68	41.7	40.67	29.23	10.01	**53.85**
			37	+CW	8.49	9.89	19.63	33.82	36.95	33.29	32.13	9.41	**45.34**
			38	+BIM	6.99	9.89	15.73	34.68	38.16	35.41	32.9	10.52	**45.1**
			39	+FGSM	6.32	9.84	13.94	34.15	40.47	38.06	33.29	9.41	**44.96**
			40	+JSMA	6.8	9.84	9.79	9.84	38.64	37.67	31.93	11.29	**45.3**
	ResNet20	5	41	Origin	14.01	5.01	14.35	23.23	33.03	32.12	9.68	5.24	**37.93**
			42	+CW	10.29	5.15	15.33	19.55	22.61	21.57	16.08	4.45	**28.4**
			43	+BIM	9.95	5.03	15.85	19.66	22.73	22.27	17.06	5.03	**28.34**
			44	+FGSM	11.16	5.38	15.62	20.19	24.18	24.29	18.74	4.45	**28.22**
			45	+JSMA	10.53	4.97	14.75	18.97	22.61	22.33	15.91	5.03	**28.05**
		10	46	Origin	24.94	10.14	25.4	41.34	56.38	53.76	20.96	10.59	**60.71**
			47	+CW	19.9	9.89	26.89	34.93	50.49	37.13	30.83	9.66	**53.61**
			48	+BIM	19.14	9.77	27.3	35.45	50.78	38.06	34.01	10.12	**53.79**
			49	+FGSM	21.98	10.12	27.01	36.9	50.14	42.68	39.39	9.66	**52.4**
			50	+JSMA	20.3	9.77	26.72	34.47	48.7	40.31	33.2	10.12	**52**

总体而言, 在 50 个实验场景 (25 个候选集×2 个选定的测试用例比例) 下, 46 种情况下 DMS 的 *fault_rate* 数值超过所有基线方法, 表明了 DMS 在选择揭错用例方面的有效性. 相较于其他仅使用模型预测时静态信息的对比方法, DMS 通过模拟模型缺陷、捕获测试用例揭错时的动态模式, 更为直接且准确地评估了测试用例的揭错能力. DMS 的整体 *fault_rate* 数值范围在 22.91%–99.22% 之间. 其中, 在 MNIST-LeNet5-Origin-10% 这组实验中, DMS 几乎能够找到所有揭错测试用例, 而对比方法只能找到 6.25%–95.31% 不等的揭错用例; 在 CIFAR10-VGG16-FGSM-5% 这组实验中, 虽然 DMS 仅能找到 22.91% 的揭错用例, 但是相比对比方法依然能够额外找到 1.54%–19.24% 的揭错用例. 本文还采用了 Wilcoxon signed-rank 检验来验证 DMS 与各方法的实验结果之间是否存在显著性差异, 分别对比两方法选择候选集 5% 和 10% 的测试用例时在每个实验场景下的 *fault_rate* 数值, 当置信度为 0.05 时. 选择候选集 5% 和 10% 测试用例的实验场景下的 *p*-value 均小于 1×10^{-5}, 例如, 选择 10% 的测试用例时 DMS 的 *fault_rate* 值相比于 ATS 和 DeepGini 的实验结果, *p*-value 分别为 8.17×10^{-6} 和 5.96×10^{-8}, 本实验说明了 DMS 在统计学上显著优于所有的对比方法. 进一步证明了 DMS 的有效性.

与此同时, 本文发现使用对抗候选集情况下的 *fault_rate* 数值普遍比使用原始候选集情况下的 *fault_rate* 低. 其原因在于, 在混入对抗样本后, 对样候选集中揭错用例的数量相比原始候选集显著增加, 在选择相同数量的测试用例时, 即使包含的揭错用例的数量更多, *fault_rate* 的数值也可能会下降. 此外, DMS 在以原始测试集作为候选集时相较于其他对比方法的优势更明显, 这表明 DMS 不仅能识别对抗样本, 还能更好地识别自然的揭错用例.

结论: DMS 能够很好地区分对抗样本和原始测试集中的自然揭错用例, 而且在以原始测试集作为候选集时的优势更加明显. 在 25 个实验组合下, 选择 5% 的测试用例时, DMS 的 *fault_rate* 范围为 31.22%–94.53%, 相较于对比方法能够额外找到 4.9%–39.3% 的揭错用例; 选择 10% 的测试用例时, DMS 的 *fault_rate* 范围为 53.85%–99.22%, 相较于对比方法能够额外找到 3.91%–26.27% 的揭错用例.

研究问题 2: 与现有方法相比, 在给定所需要选择的测试用例规模下 DMS 选出的测试用例是否能覆盖更多的揭错方向?

图 2 中展示了各方法在以原始测试集作为候选集时, 随着选择的测试用例数量的增加 *fault_type* 值的变化.

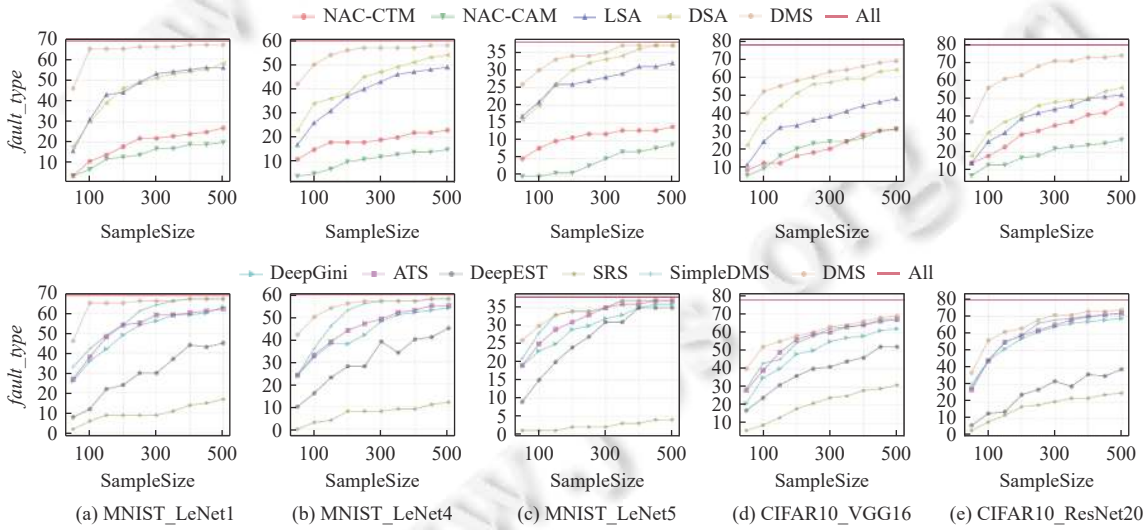

图 2 各方法在原始测试集上筛选出的测试用例 *fault_type* 对比

图 2 中红色横线代表候选集中所有揭错用例的 *fault_type* 值 (即所有方法所能达到的极限值). 由于基线方法的数量较多, 因此针对每一组模型和数据集, 各方法的 *fault_type* 的变化趋势由上下两个子图共同表示. 可以看出, DMS 的 *fault_type* 数值均明显优于其他基线方法, 其在数据变异时就通过多样化变异方向生成具有不同缺陷的变异模型以区分测试用例的揭错多样性, 而先对测试用例分组再采样也能够筛选出覆盖更多揭错方向的测试用例.

同时, 当选择测试数量越小时, DMS 的优势越明显. 其原因在于两方面: 1) DMS 相较于其他对比方法筛选出了更多的揭错用例; 2) DMS 在选择时就依据 *originLabel* 到 *majorLabel* 的变异方向对测试用例进行了分组. 对比 DMS 和直接按照 *killNum* 排序的 SimpleDMS, DMS 在选择的测试用例数量较少时优势会更明显. 此外, 不难发现除 DMS 外, ATS 方法的表现在绝大多数情况下优于其他基线方法. 虽然 ATS 利用的同样是模型预测的静态信息, 但在自适应随机选择的过程中也考虑到了测试用例的揭错方向, 其选择策略有助于选择具有不同揭错方向的测试用例, 所以 ATS 选择的测试用例集合在覆盖更多揭错方向方面略优于其他的对比方法.

结论: DMS 选出的测试用例能覆盖更多的揭错方向, 选择的测试用例数量较少时 DMS 的优势更明显.

研究问题 3: 相较于结构变异, DMS 中所使用的数据变异算子对于揭错测试用例的选择是否更有优势?

图 3 中展示了分别基于结构变异算子和数据变异算子生成的变异模型选择的测试用例, 对应的揭错能力 (*fault_rate*) 和揭错多样性 (*fault_type*) 的对比结果. 由于通用变异算子对于模型的扰动具有随机性, 较强的扰动容易生成无效模型, 导致揭错用例能够杀死的变异模型的个数有限, 从而无法利用投票算法算出 *majorLabel*, 无法判断测试用例的变异方向. 因此, 在变体 SMS 中, 本文只根据 *killNum* 对测试用例进行排序.

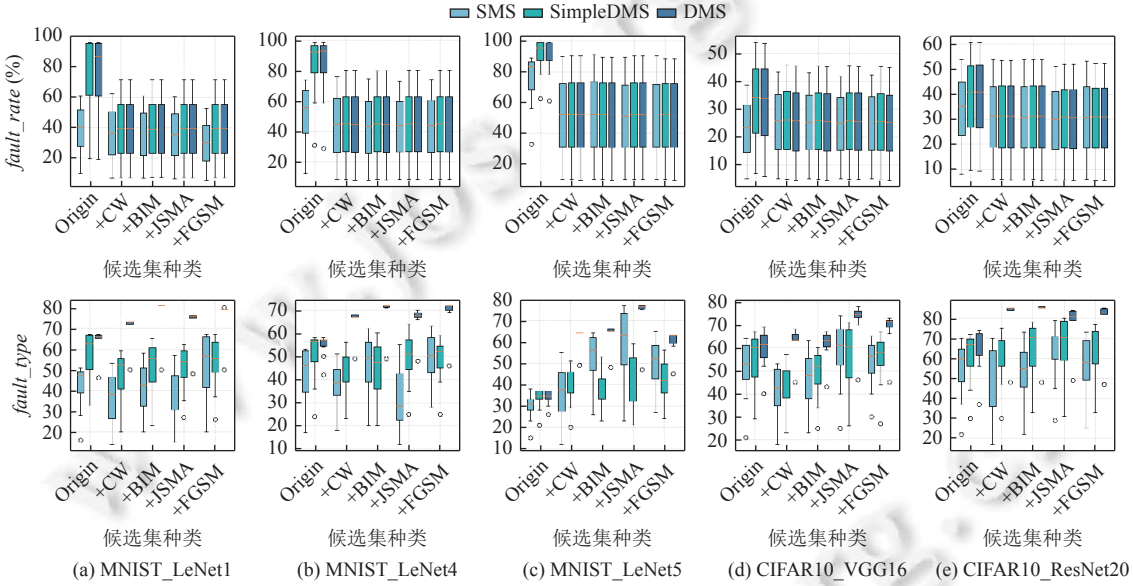

图 3 数据变异算子和结构变异算子的有效性对比

如图 3 所示, 以原始测试集作为候选集时, DMS 和 SimpleDMS 选出的测试用例的揭错能力 (*fault_rate*) 和揭错多样性 (*fault_type*) 都明显优于 SMS. 混入对抗样本后, 在揭错能力方面, SMS 与 DMS 和 SimpleDMS 的效果类似, 仅在少数情况下略逊于它们, 原因在于 SMS 所使用的变异算子 WS、GF 等通过对模型的参数随机扰动后生成的变异模型相较于自然样本, 对于对抗样本更为敏感. 但在揭错多样性方面, 3 种方法的性能差异较大, 由高到低依次为 DMS、SimpleDMS 和 SMS. SimpleDMS 并未对测试用例进行分组采样, 只是基于贪心策略选择能够杀死最多变异模型个数的测试用例. 对比每一列上下两个子图可以看出, SimpleDMS 虽然能够挑选出的揭错用例在数量上和 DMS 近似, 一些场景下甚至略优于 DMS, 但是几乎所有场景下 SimpleDMS 在多样性指标中的表现都明显不如 DMS, 这进一步说明了分组讨论的有效性, 其能在有效挑选揭错测试用例的同时提升测试用例选择的多样性. 由于数据变异时 SimpleDMS 也考虑了变异多样性, 因此同样仅根据 *killNum* 对测试用例进行排序的 SimpleDMS 的表现也优于 SMS.

结论: 综合来看, 在揭错能力方面,基于数据变异的 DMS 和 SimpleDMS 选出的测试用例要明显优于基于结构变异的 SMS; 在揭错多样性方面, DMS 考虑了变异方向的不同因此选出的测试用例能够覆盖更多的揭错方向.

6 讨 论

6.1 DMS 的效率

为了进一步研究 DMS 的效率, 本文在原始测试集上比较了各方法的效率, 如表 4 所示. DMS 花费的平均时间为 110.97 min, 虽然相对于重新训练新的网络, 微调模型能节省大量的计算资源和计算时间, 但模型训练涉及十分复杂的参数计算, 相较于单纯的模型预测而言所花费的时间会更多. SimpleDMS 和 DMS 在花费时间方面的差距可以忽略不计, 而其余的测试用例选择方法的平均耗时均在 10 min 以内. 尽管 DMS 的效率低于现有的方法, 但是和耗时且昂贵的人工标注过程相比, DMS 的成本仍然是可以接受的. 在未来工作中, 可以通过优化数据变异算子生成高质量的变异模型以减少模型微调的时间或采用并行策略加速 DMS 的执行过程, 进一步提升 DMS 的效率.

表 4 各测试用例选择方法的效率 (min)

用时	NC		LSA	DSA	ATS	DeepGini	DeepEST	SMS	DMS
	CTM	CAM							
平均值	1.26	7.03	1.28	4.06	0.58	0.06	4.25	35.59	110.97
最大值	4 023	9.00	2.48	6.47	0.51	0.12	7.32	100.36	308.49
最小值	0.14	5.19	0.53	2.97	0.08	0.03	2.98	16.43	55.68

模型的改进: 测试用例选择任务的最终目的之一是利用标注的揭错测试用例修复模型的缺陷, 提升模型的性能. 区别于传统软件系统, DNN 模型的缺陷不能直接被修复[17], 因此本文利用筛选出的测试用例对模型进行重训练, 验证模型的准确率是否能够提升. 本文选择 MNIST 数据集和 LeNet4 模型验证重训练的有效性, 由于 MNIST 在 LeNet4 模型的准确率可以达到 96.79%, 直接在该模型上进行训练难以展现重训练后的准确率提升, 本文将原测试集中 10% 的样本替换为使用 C&W 方法生成的对应的对抗样本. 此外, 为了更准确地展示模型的准确性提升, 避免出现用于筛选和验证的数据重叠, 本文按照相关工作的实验方法[17], 将候选集平均分为两份, 在其中一份上进行揭错测试用例选择, 在另一份上验证其对 DNN 模型的修复能力.

为了公平验证各种对比方法对模型的修复能力, 本文使用了 RQ1 中的 7 种对比方法, 以及在 RQ3 实验中基于模型结构变异的测试用例选择方法各在同样的候选集中选择 20% (1 000 个) 测试用例, 并将同时辅以随机选择的同样数量的训练数据对模型进行重训练, 并在验证集上验证重训练后模型准确率提升的效果. 请注意, 为了减小随机性带来的影响, 在确定需要选取的测试用例的数量后, 辅以重训练的训练样本相同, 且每次重训练过程重复 5 次取准确率提升数值的平均值. 具体实验结果如表 5 所示, 可以看出与随机方法相比, 所有的揭错测试用例选择方法均可以更有效地修复 DNN 模型的缺陷, 提升其准确率. 在其中, DMS 方法对 DNN 准确性提升最多, 达到 9.65%. 此外, 基于结构变异的测试用例选择方法 SMS 提升 DNN 模型 9.08% 的准确率, 位列第 3, 高于大多对比方法. 这说明基于变异的测试用例选择方法在模型修复任务中具有较高的有效性, 且基于数据变异的方法由于考虑到变异缺陷的多样性, 可以比其他对比方法更有效地修复模型缺陷, 提升模型性能.

表 5 MNIST-LeNet4-CW 重训练后准确率的提升 (%)

原始精度	NC		LSA	DSA	ATS	DeepGini	DeepEST	SRS	SMS	SimpleDMS	DMS
	CTM	CAM									
87.72	7.82	8.25	9.10	9.19	9.09	9.43	9.42	7.85	9.08	9.13	9.65

6.2 有效性威胁

内部有效性威胁. 内部有效性威胁主要来自对 DMS 方法的实现、对比方法的实现以及对所有实验结果进行分析的脚本实现. 为了有效减少这些威胁, 对于 DMS, 本文依赖于 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, 本文采用了这些方法的作者共享出的开源链接里的代码, 并遵循了原文中给出的参数

推荐进行实验; 对于分析实验结果的所有脚本, 本文的作者进行了多次校对, 确保结果无误.

● 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. 本文采用了基于流行数据集 MINST 和 CIFAR10 训练而成的不同精度不同复杂程度的模型进行实验. 在实际应用中, DMS 在获取到模型的训练集之后, 即可通过数据变异修改训练集标签生成大量变异模型, 捕获测试用例揭错的动态模式信息. 这意味着 DMS 不受训练集种类约束, 可以拓展到更多不同种类的数据集上 (如: 文本, 音频等). 此外, 在对变异算子进行调整之后, 基于数据变异的测试用例选择思想同样可以扩展至更多不同种类的深度学习任务中 (如回归任务、推荐任务). 例如在回归任务中, 我们可以度量回归任务输出值与真实标签的差异程度, 并向数据集中动态注入不同差异程度的错误. 在未来工作中, 我们会进一步根据任务不同设计出更多更精细化的数据变异算子, 以进一步提升 DMS 的有效性和可扩展性.

● 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, 主要包括选择的测试用例的比例、微调时的训练参数如学习率、优化器等和生成的变异模型数量. 对于选择的测试用例的比例, 本文参考了已有工作中的实验设置并基于不同 DNN 模型-候选集组合进行了实验. 结果表明, 相关设置稳定且 DMS 在此设置下表现良好, 后续研究中可以继续探索更多的实验设置以验证 DMS 的可靠性. 对于微调时的训练参数, DMS 参照了模型初始训练时的参数. 对于生成的变异模型数量, DMS 对训练集进行了 4 次随机变异, 每次利用变异后训练集对模型进行重训练并保留前 25 个周期训练得到的模型共 100 个模型. 为了验证 DMS 在生成其他数量的变异模型下的有效性, 此处以 VGG16 在 CIFAR10 原始测试集下的效果为例, 选择的测试用例数量范围为 100–1 000, 间隔为 100, 表 6 中展示了在使用不同数量的变异模型时筛选出的测试用例的 *fault_rate* 值和 *fault_type* 值, 以"/"隔开, 可以看出在使用较为多样的变异模型时效果会有一定的提升, 但与此同时, 时间和计算成本也会成倍增加. 因此, DMS 仅使用了 100 个变异模型, 后续的研究可以继续探究如何生成高质量的变异模型, 从而进一步提升 DMS 的效率和性能.

表 6 DMS 在使用不同数量变异模型时的有效性对比

变异模型数量	选择的测试用例数量									
	100	200	300	400	500	600	700	800	900	1 000
100	5.96/52	12.39/58	18.82/63	25.5/66	31.22/69	36.46/70	41.06/70	45.43/71	49.64/**72**	53.85/**72**
200	6.35/55	13.26/59	19.62/63	25.89/66	31.61/**71**	36.7/**71**	41.38/**71**	46.15/**72**	50.91/**72**	54.81/**72**
300	6.35/51	13.34/57	20.02/62	26.21/68	31.45/**71**	36.3/**71**	41.54/**71**	47.02/71	51.47/**72**	55.2/**72**
400	6.59/54	13.42/**62**	20.1/64	26.21/**69**	31.45/**71**	36.78/**71**	42.34/**71**	47.26/71	52.1/71	55.76/71
500	6.83/55	13.42/60	20.17/64	26.53/68	31.69/69	37.41/70	42.41/**71**	47.42/71	51.47/71	56.31/71
600	6.59/53	13.5/61	20.02/63	26.53/67	32.25/70	37.33/70	42.65/**71**	47.34/71	52.34/71	56.39/**72**
700	6.59/53	13.5/60	20.02/63	26.61/66	32.57/70	37.73/70	42.49/70	**48.05**/71	**52.42**/71	**56.95**/**72**
800	6.83/55	13.5/61	20.17/63	26.53/67	32.647 0	**37.89**/70	42.65/70	47.66/71	52.34/71	56.55/**72**
900	**6.91**/54	13.58/60	20.17/65	26.69/68	**32.88**/70	37.97/70	43.05/70	47.5/71	51.95/71	56.08/**72**
1 000	**6.91**/**56**	**13.58**/61	**20.25**/**66**	**26.77**/**69**	32.72/**71**	37.97/**71**	**43.13**/**71**	**48.05**/**72**	52.18/**72**	56/**72**

注: 符号"/"前为*fault_rate*值, 后为*fault_type*值; *fault_rate*的值为百分数; 加粗数值为最佳结果

7 相关工作

7.1 传统软件测试优化

在传统软件回归测试中[47], 测试优化也是提高测试效率的重要方向, 其中包括测试用例选择、测试用例排序和测试用例约减等. 在传统软件回归测试中, 测试用例选择旨在依照测试人员的需求选择部分重要的测试用例进行测试. 在回归测试场景下, 测试人员通常选择受软件更改影响的测试用例, 因为这些测试用例更有可能由于软件更改引入回归缺陷. 例如, Gligoric 等人[48]为 Java 项目提出了一种基于字节码类文件更改的文件级动态测试用例选择方法, Legunsen 等人[49]对多种不同的静态测试用例选择方法和它们的安全性进行了实证研究, 证明了类级静态测

试用例选择方法具有和动态测试用例选择方法相似的效益, 但方法级静态测试用例选择方法表现较差. 近年来, Zhang 等人[50]融合现有的动态测试选择方法的优点提出了第 1 个混合测试用例选择方法 HyRTS (hybrid regression test selection), 可应用于方法级、文件级等不同粒度的场景. 传统测试中的测试用例排序旨在优先执行更重要的测试用例以尽早揭示软件中的错误. 例如, Li 等人[51]根据代码覆盖信息基于搜索算法寻找测试用例执行的最佳顺序. Chen 等人[52]比较了各种测试用例排序方法并进一步提出了一种基于机器学习的方法, 根据测试用例分布相关信息为特定项目推荐最佳测试用例排序方法. 测试用例约减旨在移除与某些测试能力度量指标 (例如代码覆盖率) 相关的冗余测试用例. 其中, Harrold 等人[53]提出了一种将贪心策略与启发式搜索相结合的方法 HGS (Harrold Gupta SoF), 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用例.

本文的工作是对传统软件回归测试用例排序和选择任务在深度学习模型中的进一步拓展. DMS 旨在筛选出能够揭示 DNN 模型错误行为的测试用例并优先对其进行标注, 以尽快修复模型缺陷, 从而提升 DNN 测试的效率. 但与传统软件不同, 深度神经网络定义了一种新的数据驱动的编程范式, 传统测试中的代码覆盖等信息无法直接应用于深度神经网络, 本文利用数据变异模拟神经网络中的错误, 捕获变异模型在预测过程中产生的动态信息指导测试用例的选择.

7.2 深度神经网络测试

深度神经网络测试是保证 DNN 质量的重要方法. 其中, 测试用例生成[1,10,54–59]和测试充分性度量[7,10,59,60]是 DNN 测试的重要任务. 测试用例生成的目的是充分探索测试输入空间以寻找触发 DNN 错误行为的测试用例, 测试充分性度量的是检验现有的测试用例集合是否有优秀的揭错能力.

测试用例生成主要包括两种任务: 对抗样本生成和自然测试用例生成. 对抗样本由 Szegedy 等人[54]提出, 旨在向测试输入中加入微小的扰动生成新的测试用例, 以达到误导 DNN 判断的效果. 虽然生成的对抗样本可以在一定程度上暴露 DNN 的缺陷, 但是其不一定满足正常的数据分布, 在实际应用中难以出现. 相比而言, 自然测试用例更符合实际应用场景的数据分布. 常见的自然测试用例生成方法包括如下.

现有部分工作采用模糊测试的思想搜索揭错测试用例. Pei 等人[10]提出 DeepXplore, 利用白盒差分测试技术来生成 DNN 模型的测试用例. 受传统软件测试中测试覆盖率的启发, 作者提出了神经元覆盖率指导测试用例生成, 通过联合优化算法迭代搜索能够揭错的测试用例. 除 DeepXplore 之外, 以神经元覆盖率为首的指导准则在 DeepTest, DeepHunter 等框架得到了充分应用. Tian 等人[55]提出 DeepTest 框架, 以神经元覆盖为指导, 对图像进行一系列线性和仿射变化, 以模拟自动驾驶场景中图像捕捉设备设备出现故障的场景. DeepTest 使用贪心算法不断搜索揭错的测试用例, 并证明这些测试用例可以用来修复模型中的缺陷. Xie 等人[56]提出 DeepHunter 框架, 利用结构神经元覆盖以及一系列细粒度的结构覆盖规则指导测试用例生成, DeepHunter 集成了多种源于蜕变关系的变异算子, 利用模糊测试的思想高效生成揭错测试用例.

根据 DNN 模型任务的不同, 一些基于蜕变关系的测试用例生成方法被陆续提出. Guo 等人[57]对于自动驾驶中雷达任务进行分析, 提出雷达感知系统测试框架 LiRTest, 设计了一系列在极端天气和特定驾驶状况下雷达感知系统会出现的蜕变关系, 以测试其中的错误. Zhang 等人[1]聚焦自动驾驶任务, 提出基于 GAN 的测试用例生成方法 DeepRoad, DeepRoad 可以将自动驾驶图片转换为雨雪天气的图片, 以检测自动驾驶系统是否会受到天气影响做出错误决策. Yang 等人[58]对智能代码分类任务的 DNN 模型进行测试, 提出 ALERT 框架. ALERT 根据更改代码的变量名不应改变代码分类任务的结果这一蜕变关系, 不断迭代生成测试用例攻击代码模型. 基于蜕变关系的测试用例方法在不同的 DNN 任务中逐渐得到越来越广泛的应用.

测试充分性度量是软件工程中的重要任务, 它提出客观的指标评价现有的测试用例是否可以有效揭错. 在传统的软件测试中, 代码覆盖率用于衡量一个程序的源代码对源程序的探索程度. 测试套件的覆盖率越高, 越有可能揭露程序的缺陷. 与传统软件不同, DNN 模型的决策逻辑更多来源于训练数据而非代码, 代码覆盖率不是衡量 DNN 测试充分性的有效指标. 因此, 一些新的充分性度量指标被陆续提出.

正如代码行是传统软件程序的基本结构, Pei 等人[10]将神经网络的结构拆解成基本逻辑结构—神经元, 并提

出了神经元覆盖率作为指标. 神经元覆盖率需要统计测试用例在每个神经元的输出值, 一旦输出值达到一定阈值, 就意味着该神经元被激活. 神经元覆盖率表示神经元的测试用例激活的神经元数量与 DNN 模型中神经元总数的比率. Ma 等人[7]在后续研究中发现, 神经元覆盖很容易达到 100%, 因此提出了更细粒度的 4 种结构覆盖规则, 其中最广泛使用的是 KMNC (K-多区间神经元覆盖), 他根据每个神经元激活值的上下界将神经元分成 K 个区间, 更细粒度地度量测试用例的覆盖能力. 虽然以神经元结构作为度量指标在指导测试用例生成任务中取得了一定有效性, 但是近期的更多工作认为[15], 结构度量指标与测试用例的揭错能力相关性较为有限. 因此, 一些研究人员对神经网络的结构覆盖进行了进一步改进, Gerasimou 等人[60]认为并非所有神经元都对神经网络的决策具有重要意义, 因此提出了 DeepImportance, 优先选择更重要的神经元进行覆盖和分析. Kim 等人[59]则从神经网络对数据的认知角度提出了新的细粒度框架 SADL, 用于计算测试用例相对训练集的“意外值”, 并提出意外充分性以及意外覆盖 (surprise coverage, SC) 两种指标. 如果一个测试用例相较于训练集较为罕见, 这意味着它更有可能被模型预测出错, 它的“意外充分性”更高. 实验结果表明: 意外充分性和意外覆盖可以有效区分对抗样本和普通样本, 并对提升模型的质量有重要作用. 与上述工作中提出的度量标准不同, 本文使用了与变异分析相关的“变异模型杀死数”和“变异方向”衡量测试用例的揭错能力和揭错的多样性, 能够利用动态的信息更细粒度地筛选揭错的测试用例.

8 总 结

本文首次基于数据变异的思想提出了一种新的测试用例选择方法 DMS, 在由 5 种 DNN 模型和经典的深度学习测试集构造的 25 个实验组合上对方法的性能进行了实证研究. 结果表明, DMS 能够筛选出具有揭错能力且揭错方向不同的测试用例, 显著优于现有的测试用例选择方法如 ATS、DeepEST 等. 具体来说, 以原始测试集作为候选集时, 在选择 10% 的测试用例时, DMS 能够筛选出候选集中 53.85%–99.22% 的揭错用例, 在选择 5% 的测试用例时, DMS 筛选出的揭错用例已经几乎能覆盖所有的揭错方向. 相较于对比方法, DMS 平均多找出 12.38%–71.81% 的揭错用例, 证明了 DMS 在测试用例选择任务中的显著有效性. 在之后的研究工作中, 我们将尝试生成更高质量的变异模型以提升 DMS 的效率, 并尝试将 DMS 方法拓展到回归任务的 DNN 模型及测试集上应用.

References:

[1] Zhang MS, Zhang YQ, Zhang LM, Liu C, Khurshid S. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 132–142. [doi: 10.1145/3238147.3238187]

[2] Hu GS, Yang YX, Yi D, Kittler J, Christmas W, Li SZ, Hospedales T. When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition. In: Proc. of the 2015 IEEE Int'l Conf. on Computer Vision Workshops. Santiago: IEEE, 2015. 384–392. [doi: 10.1109/ICCVW.2015.58]

[3] Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY. Deep speech: Scaling up end-to-end speech recognition. arXiv:1412.5567, 2014.

[4] Chan HP, Samala RK, Hadjiiski LM, Zhou C. Deep learning in medical image analysis. In: Lee G, Fujita H, eds. Deep Learning in Medical Image Analysis: Challenges and Applications. Cham: Springer, 2020. 3–21. [doi: 10.1007/978-3-030-33128-3_1]

[5] Pan WJ, Duan YJ, Zhang Q, Tang JH, Zhou J. Deep learning for aircraft wake vortex identification. IOP Conf. Series: Materials Science and Engineering, 2019, 685: 012015. [doi: 10.1088/1757-899X/685/1/012015]

[6] Chen JJ, He XT, Lin QW, Zhang HY, Hao D, Gao F, Xu ZW, Dang YN, Zhang SM. Continuous incident triage for large-scale online service systems. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 364–375. [doi: 10.1109/ASE.2019.00042]

[7] Ma L, Juefei-Xu F, Zhang FY, Sun JY, Xue MH, Li B, Chen CY, Su T, Li L, Liu Y, Zhao JJ, Wang YD. DeepGauge: Multi-granularity testing criteria for deep learning systems. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 120–131. [doi: 10.1145/3238147.3238202]

[8] Zhang JM, Harman M, Ma L, Liu Y. Machine learning testing: Survey, landscapes and horizons. IEEE Trans. on Software Engineering, 2022, 48(1): 1–36. [doi: 10.1109/TSE.2019.2962027]

[9] Wang Z, You HM, Chen JJ, Zhang YY, Dong XY, Zhang WB. Prioritizing test inputs for deep neural networks via mutation analysis. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 397–409. [doi: 10.1109/ICSE43902.

2021.00046]

[10] Pei KX, Cao YZ, Yang JF, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. of the 26th Symp. on Operating Systems Principles. Shanghai: ACM, 2017. 1–18. [doi: 10.1145/3132747.3132785]

[11] Chen YS, Wang ZY, Wang D, Yao YM, Chen ZY. Behavior pattern-driven test case selection for deep neural networks. In: Proc. of the 2019 IEEE Int'l Conf. on Artificial Intelligence Testing (AITest). Newark: IEEE, 2019. 89–90. [doi: 10.1109/AITest.2019.000-2]

[12] Wang ZY, Xu SH, Cai XR, Ji H. Test input selection for deep neural networks. Journal of Physics: Conf. Series, 2020, 1693: 012017. [doi: 10.1088/1742-6596/1693/1/012017]

[13] Li ZN, Ma XX, Xu C, Cao C. Structural coverage criteria for neural networks could be misleading. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). Montreal: IEEE, 2019. 89–92. [doi: 10.1109/ICSE-NIER.2019.00031]

[14] Harel-Canada F, Wang LX, Gulzar MA, Gu QQ, Kim M. Is neuron coverage a meaningful measure for testing deep neural networks? In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 851–862. [doi: 10.1145/3368089.3409754]

[15] Dong YZ, Zhang PX, Wang JY, Liu S, Sun J, Hao JY, Wang XY, Wang L, Dong JS, Ting D. There is limited correlation between coverage and robustness for deep neural networks. arXiv:1911.05904, 2019.

[16] 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. ACM, 2020. 177–188. [doi: 10.1145/3395363.3397357]

[17] Gao XY, Feng Y, Yin YN, Liu ZX, Chen ZY, Xu BW. Adaptive test selection for deep neural networks. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 73–85. [doi: 10.1145/3510003.3510232]

[18] Liu WB, Wang ZD, Liu XH, Zeng NY, Liu YR, Alsaadi FE. A survey of deep neural network architectures and their applications. Neurocomputing, 2017, 234: 11–26. [doi: 10.1016/j.neucom.2016.12.038]

[19] Sun YC, Wu M, Ruan WJ, Huang XW, Kwiatkowska M, Kroening D. Concolic testing for deep neural networks. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 109–119. [doi: 10.1145/3238147.3238172]

[20] Guo JM, Jiang Y, Zhao Y, Chen Q, Sun JG. 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. Lake Buena Vista: ACM, 2018. 739–743. [doi: 10.1145/3236024.3264835]

[21] Maas AL, Daly RE, Pham PT, Huang D, Ng AY, Potts C. Learning word vectors for sentiment analysis. In: Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Portland: Association for Computational Linguistics, 2011. 142–150.

[22] Alzantot M, Balaji B, Srivastava M. Did you hear that? Adversarial examples against automatic speech recognition. arXiv:1801.00554, 2018.

[23] Nejadgholi M, Yang JQ. A study of oracle approximations in testing deep learning libraries. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 785–796. [doi: 10.1109/ASE.2019.00078]

[24] Li ZN, Ma XX, Xu C, Cao C, Xu JW, Lü 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. Tallinn: ACM, 2019. 499–509. [doi: 10.1145/3338906.3338930]

[25] Zhou JY, Li F, Dong JH, Zhang HY, 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 (ISSRE). Coimbra: IEEE, 2020. 289–300. [doi: 10.1109/ISSRE5003.2020.00035]

[26] Chen JJ, Wu Z, Wang Z, You HM, Zhang LM, Yan M. Practical accuracy estimation for efficient deep neural network testing. ACM Trans. on Software Engineering and Methodology, 2020, 29(4): 30. [doi: 10.1145/3394112]

[27] Guerriero A, Pietrantuono R, Russo S. Operation is the hardest teacher: Estimating DNN accuracy looking for mispredictions. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 348–358. [doi: 10.1109/ICSE43902.2021.00042]

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

[29] Lou YL, Hao D, Zhang L. Mutation-based test-case prioritization in software evolution. In: Proc. of the 26th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Gaithersbury: IEEE, 2015. 46–57. [doi: 10.1109/ISSRE.2015.7381798]

[30] Shin D, Yoo S, Papadakis M, Bae DH. Empirical evaluation of mutation-based test case prioritization techniques. Software Testing, Verification and Reliability, 2019, 29(1–2): E1695. [doi: 10.1002/stvr.1695]

[31] Ma L, Zhang FY, Sun JY, Xue MH, Li B, Juefei-Xu F, Xie C, Li L, Liu Y, Zhao JJ, Wang YD. DeepMutation: Mutation testing of deep learning systems. In: Proc. of the 29th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Memphis: IEEE, 2018. 100–111.

[doi: 10.1109/ISSRE.2018.00021]

[32] Hu Q, Ma L, Xie XF, Yu B, Liu Y, Zhao JJ. DeepMutation++: A mutation testing framework for deep learning systems. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1158–1161. [doi: 10.1109/ASE.2019.00126]

[33] Humbatova N, Jahangirova G, Tonella P. DeepCrime: Mutation testing of deep learning systems based on real faults. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 67–78. [doi: 10.1145/3460319.3464825]

[34] Käding C, Rodner E, Freytag A, Denzler J. Fine-tuning deep neural networks in continuous learning scenarios. In: Proc. of the 2017 Asian Conf. on Computer Vision. Taipei: Springer, 2017. 588–605. [doi: 10.1007/978-3-319-54526-4_43]

[35] Hendrycks D, Mu N, Cubuk ED, Zoph B, Gilmer J, Lakshminarayanan B. AugMix: A simple data processing method to improve robustness and uncertainty. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020.

[36] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]

[37] Meng LH, Li YH, Chen L, Wang Z, Wu D, Zhou YM, Xu BW. Measuring discrimination to boost comparative testing for multiple deep learning models. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 385–396. [doi: 10.1109/ICSE43902.2021.00045]

[38] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc. of the IEEE, 1998, 86(11): 2278–2324. [doi: 10.1109/5.726791]

[39] Krizhevsky A. Learning multiple layers of features from tiny images [MS. Thesis]. Toronto: University of Toronto, 2009.

[40] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proc. of the 3rd Int'l Conf. on Learning Representations. San Diego, 2015.

[41] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[42] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proc. of the 2017 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2017. 39–57. [doi: 10.1109/SP.2017.49]

[43] Wang J. Adversarial examples in physical world. In: Proc. of the 5th Int'l Conf. on Learning Representations (ICLR 2017). Toulon: IJCAI, 2021. 4925–4926.

[44] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. In: Proc. of the 2016 IEEE European Symp. on Security and Privacy (EuroS&P). Saarbruecken: IEEE, 2016. 372–387. [doi: 10.1109/EuroSP.2016.36]

[45] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. In: Proc. of the 3rd Int'l Conf. on Learning Representations. San Diego, 2015.

[46] Wang Z, Yan M, Chen JJ, Liu S, Zhang DD. Deep learning library testing via effective model generation. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 788–799. [doi: 10.1145/3368089.3409761]

[47] Chen JJ, Bai YW, Hao D, Zhang LM, Zhang L, Xie B, Mei H. Supporting oracle construction via static analysis. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 178–189.

[48] Gligoric M, Eloussi L, Marinov D. Practical regression test selection with dynamic file dependencies. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. Baltimore: ACM, 2015. 211–222. [doi: 10.1145/2771783.2771784]

[49] Legunsen O, Hariri F, Shi A, Lu YF, Zhang LM, Marinov D. An extensive study of static regression test selection in modern software evolution. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 583–594. [doi: 10.1145/2950290.2950361]

[50] Zhang LM. Hybrid regression test selection. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 199–209. [doi: 10.1145/3180155.3180198]

[51] Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering, 2007, 33(4): 225–237. [doi: 10.1109/TSE.2007.38]

[52] Chen JJ, Lou YL, Zhang LM, Zhou JY, Wang XL, Hao D, Zhang L. Optimizing test prioritization via test distribution analysis. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 656–667. [doi: 10.1145/3236024.3236053]

[53] Harrold MJ, Gupta R, Soffa ML. A methodology for controlling the size of a test suite. ACM Trans. on Software Engineering and Methodology, 1993, 2(3): 270–285. [doi: 10.1145/152388.152391]

[54] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow IJ, Fergus R. Intriguing properties of neural networks. In: Proc. of the

2nd Int'l Conf. on Learning Representations. Banff, 2014.

[55] 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. Gothenburg: IEEE, 2018. 303–314. [doi: 10.1145/3180155.3180220]

[56] 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. Beijing: ACM, 2019. 146–157. [doi: 10.1145/3293882.3330579]

[57] Guo A, Feng Y, Chen ZY. LiRTest: Augmenting LiDAR point clouds for automated testing of autonomous driving systems. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 480–492. [doi: 10.1145/3533767.3534397]

[58] Yang Z, Shi JK, He JD, Lo D. Natural attack for pre-trained models of code. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1482–1493. [doi: 10.1145/3510003.3510146]

[59] Kim J, 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 (ICSE). Montreal: IEEE, 2019. 1039–1049. [doi: 10.1109/ICSE.2019.00108]

[60] 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 Proc. (ICSE-Companion). Seoul: IEEE, 2020. 322–323.

曹雪洁(2001－), 女, 硕士, 主要研究领域为深度学习测试.

尤翰墨(1997－), 男, 博士生, CCF 学生会员, 主要研究领域为软件测试, 深度学习系统测试.

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

吴卓(1996－), 女, 博士生, CCF 学生会员, 主要研究领域为软件测试.

闫明(1996－), 男, 博士生, CCF 学生会员, 主要研究领域为深度学习系统测试, 芯片设计程序测试.

王赞(1979－), 男, 博士, CCF 专业会员, 主要研究领域为软件测试, 深度学习.