软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(5):2251−2267 [doi: 10.13328/j.cnki.jos.006422]

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

DeepRanger: 覆盖制导的深度森林测试方法*

崔展齐[1], 谢瑞麟[1], 陈 翔[2], 刘秀磊[1], 郑丽伟[1]

[1](北京信息科技大学 计算机学院, 北京 100101)

[2](南通大学 信息科学技术学院, 江苏 南通 226019)

通信作者: 崔展齐, E-mail: czq@bistu.edu.cn; 陈翔, E-mail: xchencs@ntu.edu.cn

摘 要: 深度学习软件的结构特征与传统软件存在明显差异, 因此即使展开了大量测试, 依然无法有效衡量测试数据对深度学习软件的覆盖情况和测试充分性, 并造成后续使用过程中依然可能存在大量未知错误. 深度森林是一种新型深度学习模型, 其克服了深度神经网络存在的一些缺点, 例如: 需要大量训练数据、需要高算力平台、需要大量超参数. 但目前还没有相关工作对深度森林的测试方法进行研究. 针对深度森林的结构特点, 制定了一组由随机森林结点覆盖率 RFNC、随机森林叶子覆盖率 RFLC、级联森林类型覆盖率 CFCC 和级联森林输出覆盖率 CFOC 组成的测试覆盖率评价指标. 在此基础上, 基于遗传算法设计了覆盖制导的测试数据自动生成方法 DeepRanger, 可自动生成能有效提高模型覆盖率的测试数据集. 为对所提出覆盖指标的有效性进行验证, 在深度森林开源项目 gcForest 和 MNIST 数据集上设计并进行了一组实验. 实验结果表明, 所提出的 4 种覆盖指标均能有效评价测试数据集对深度森林模型的测试充分性. 此外, 与基于随机选择的遗传算法相比, 使用覆盖信息制导的测试数据生成方法 DeepRanger 能达到更高的模型覆盖率.

关键词: 深度森林; 测试覆盖准则; 多粒度扫描覆盖; 级联森林覆盖

中图法分类号: TP311

中文引用格式: 崔展齐, 谢瑞麟, 陈翔, 刘秀磊, 郑丽伟. DeepRanger: 覆盖制导的深度森林测试方法. 软件学报, 2023, 34(5): 2251–2267. http://www.jos.org.cn/1000-9825/6422.htm

英文引用格式: Cui ZQ, Xie RL, Chen X, Liu XL, Zheng LW. DeepRanger: Coverage-guided Deep Forest Testing Approach. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2251–2267 (in Chinese). http://www.jos.org.cn/1000-9825/6422.htm

DeepRanger: Coverage-guided Deep Forest Testing Approach

CUI Zhan-Qi[1], XIE Rui-Lin[1], CHEN Xiang[2], LIU Xiu-Lei[1], ZHENG Li-Wei[1]

[1](School of Computer Science, Beijing Information Science and Technology University, Beijing 100101, China)

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

Abstract: Comparing with traditional software, the deep learning software has different structures. Even if a lot of test data is used for testing the deep learning software, the adequacy of testing still hard to be evaluted, and many unknown defects could be implied. The deep forest is an emerging deep learning model that overcomes many shortcomings of deep neural networks. For example, the deep neural network requires a lot of training data, high performance computing platform, and many hyperparameters. However, there is no research on testing deep forest. Based on the structural characteristics of deep forests, this study proposes a set of testing coverage criteria, including random forest node coverage (RFNC), random forest leaf coverage (RFLC), cascad forest class coverage (CFCC), and cascad forest output coverage (CFOC). DeepRanger, a coverage-oriented test data generation method based on genetic algorithm, is proposed to automatically generate new test data and effectively improve the model coverage of the test data. Experiments are carried out on the

* 基金项目: 江苏省前沿引领技术基础研究专项 (BK20202001); 国家自然科学基金 (61702041, 61601039); 北京信息科技大学"勤信人才"培育计划 (QXTCP C201906, QXTCP B201905)
收稿时间: 2020-09-16; 修改时间: 2021-01-15; 采用时间: 2021-07-28; jos 在线出版时间: 2022-09-16
CNKI 网络首发时间: 2022-11-15

MNIST data set and the gcForest, which is an open source deep forest project. The experimental results show that the four coverage criteria proposed can effectively evaluate the adequacy of the test data set for the deep forest model. In addition, comparing with the genetic algorithm based on random selection, DeepRanger, which is guided by coverage information, can improve the testing coverage of the deep forest model under testing.

Key words: deep forest; testing coverage criteria; multi-grained scanning coverage; cascade-forest coverage

深度学习软件建立类似人脑的分层结构模型, 通过非线性运算单元, 将输入数据逐级从底层转换为高层特征, 建立了从底层信号到高层语义的复杂映射关系[1]. 深度学习软件已经在计算机视觉、自然语言处理、自动驾驶等领域取得了显著进展和实际应用[2]. 但是, 深度学习软件与传统软件不同, 其结构复杂且内部复杂的逻辑关系基于训练过程所获取, 缺乏传统软件的可解释性, 即使进行了大量的测试, 也难以获取测试数据对整个深度学习系统的覆盖情况, 无法对测试的有效性和充分性进行评估[3]. 在实际应用中, 未经过充分测试的深度学习软件可能会因为未知的输入触发在测试阶段未被发现的错误, 从而引发严重事故. 例如, 近年来深度学习软件故障所导致的交通事故频发, 谷歌、特斯拉等公司的自动驾驶汽车都发生过严重的交通事故[4,5]. 因此如何有效测试深度学习软件已成为亟待解决的问题.

为此, Ma 等人提出了一种用于深度神经网络 (deep neural network, DNN) 的测试覆盖指标 DeepGauge[6], 为深度学习软件的测试提供了新的思路. DeepGauge 使用类似传统软件灰盒测试的思想, 针对神经网络的内部结构, 分析测试数据对神经网络结构不同区域的覆盖情况, 从而定义出一套覆盖指标以评价测试充分程度. DeepGauge 的提出使得对神经网络的测试充分性有了可量化的指标. 在其基础上, Xie 等人提出一种面向 DNN 的测试方法 DeepHunter[7], 其使用覆盖率作为指导, 基于模糊测试自动生成高覆盖率的测试数据集, 在一些经典的 DNN 模型中找到了大量错误.

深度森林 (deep forest, DF)[8,9]是 Zhou 等人提出的一种深度学习模型, 克服了 DNN 模型的部分缺点, 例如: 需要大量标记数据、高计算性能设备、大量超参数、对参数敏感以及低可解释性等. 深度森林已经在小规模数据集和低算力平台上获取了更好的预测性能, 而 DNN 往往需要更高的训练成本才能达到同样的效果, 成为深度学习领域的有力竞争者. 然而, 现有的面向深度学习软件的覆盖指标和测试方法主要关注 DNN, 对深度森林目前还缺少有效的测试覆盖指标和对应的测试数据生成方法.

针对上述问题, 本文针对深度森林以随机森林为最小单位、级联森林每层输出用于下一层部分输入等特点, 提出了一组有针对性的覆盖指标, 以用于评估深度森林的测试充分性, 并指导测试数据生成. 在 MNIST 数据集上的实验结果表明, 所提出的覆盖指标能有效评估测试数据集的充分性, 且可以对深度森林的不同区域进行不同粒度的测试充分性评价. 在所提出覆盖指标的基础上, 本文基于遗传算法设计并实现了使用深度森林覆盖信息制导的测试数据自动生成和覆盖率分析工具 DeepRanger (深度巡林者).

本文的主要贡献可总结如下:

(1) 提出了 4 种适用于深度森林的测试覆盖指标, 对深度森林模型不同区域的覆盖情况分别进行统计, 从而对深度森林模型的测试充分性提供了一种指导和评价方法.

(2) 设计并实现了覆盖信息制导的测试数据自动生成工具 DeepRanger, 使用面向覆盖信息的适应度函数, 基于遗传算法自动生成大量可有效提高模型覆盖率的测试数据.

本文第 1 节介绍了深度森林相关研究背景. 第 2 节详细介绍所提出的面向深度森林的测试覆盖评价指标. 第 3 节介绍了覆盖信息制导的测试数据自动生成方法 DeepRanger. 第 4 节介绍了实验设计, 并对实验结果进行了分析和讨论. 第 5 节介绍了本文的相关工作, 并强调了论文的创新处. 最后第 6 节总结全文并对未来工作进行展望.

1 研究背景

深度森林的结构如图 1 所示, 其主要包括两个部分: 多粒度扫描 (multi-grained scanning) 和级联森林 (cascade forest). 具体来说, 多粒度扫描可将单个高维度的样本数据通过多次不同采样生成多个低维度的样本. 级联森林中每层森林将多粒度扫描部分的预处理数据和上一层的输出作为输入进行处理.

多粒度扫描部分的过程如图 2 所示, 其中图的上半部分表示输入为序列类型, 图的下半部分表示输入为二维

图片类型. 以序列类型输入为例, 假设输入样本特征维度为 400、分类类型为 3 类, 若使用大小为 100 的滑动窗口进行扫描, 则每个样本特征维度为 100, 最终将得到 301 个滑动样本; 若使用大小为 200 的滑动窗口进行扫描, 则每个样本的特征维度为 200, 最终可以得到 201 个样本. 不难看出, 每个实例代表了一种局部结构或者亚采样样本.

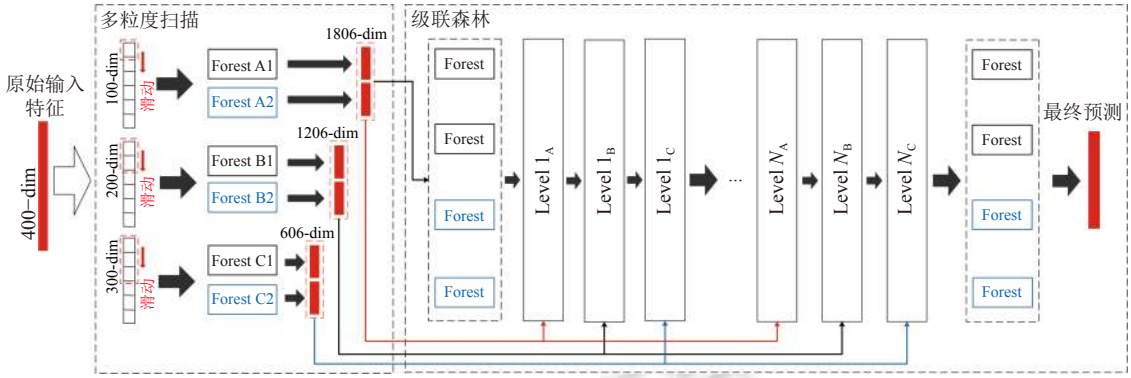

图 1 深度森林的结构[9]

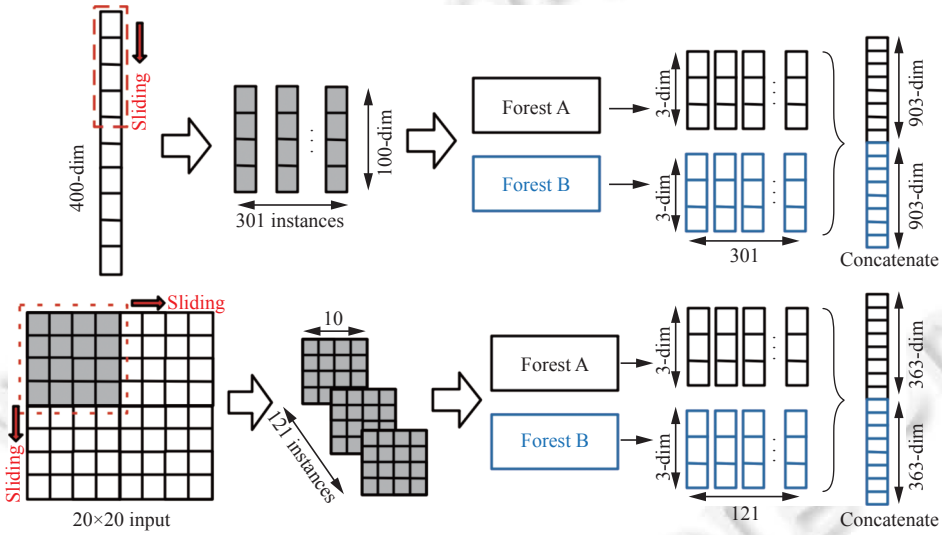

图 2 多粒度扫描示意图[9]

当滑动窗口大小为 100 时, 可得到 301 个滑动样本实例. 假如训练集内含有 m 个训练样本, 则一共可以得到 m×301 个滑动样本, 这 m×301 个 100 维特征的滑动样本将作为随机森林的输入. 每个滑动样本将得到一个 3 维的类向量表示该样本属于 3 个类的概率. 对 301 个实例 (局部结构), 则可以得到 301×3=903 维的类概率分布, 这些类概率分布连在一起作为特征向量, 输入到下一层级联森林.

级联森林如图 1 右半部分所示, 假设多粒度扫描窗口大小分别为 100、200 和 300, 因此维度为 400 的序列经扫描将分别得到 301、201 和 101 个滑动样本. 首先, 将两个多粒度扫描部分 (滑动窗口为 100) 的随机森林产生的维度为 1 806 (301×3×2) 的样本输入给 4 个随机森林, 得到维度为 12 (4×3) 的类向量, 与之前多粒度扫描产生的维度为 1 806 的样本合并, 构成维度为 1 818 的样本输入 level 1_A. 随后 level 1_A 输出维度为 12 (4×3) 的样本, 这个样本与维度为 1 206 (201×3×2) 的原始样本 (基于滑动窗口为 200 的多粒度扫描) 合并, 组成维度为 1 218 的样本输入 level 1_B; 接着, level 1_B 输出维度为 12 (4×3) 的样本, 和维度为 606 (101×3×2) 的原始样本 (基于滑动窗口为 300 的多粒度扫描) 合并, 得到维度为 618 的样本输入 level 1_C; 循环进行这个过程. 最后一层随机森林的输出类向量的平均值为级联森林的输出, 也是整个深度森林的最终预测结果.

2 针对深度森林的测试覆盖指标

测试覆盖充分性是指测试数据覆盖被测对象的程度[10], 一组测试数据会对被测软件的功能点、用例、代码进行不同程度的覆盖. 因此覆盖率是用来度量测试充分性的一种有效方式. 但在传统软件测试中, 块覆盖、语句覆盖、判定覆盖、条件覆盖等主要基于语句或语句块层面, 即从程序逻辑结构上来衡量测试的充分性. 然而, 在深度学习软件中, 由于结构的复杂性和难以解释性, 传统的覆盖指标无法有效评估深度学习软件的测试充分性. 深度学习软件一般将数据集随机分为两个部分 (一部分构成训练集用于构建模型, 剩余部分则构成测试集用于评估模型性能), 在这种情况下, 使用与训练集具有相似数据分布的测试集来测试深度学习软件, 将使测试数据只能覆盖到模型的小部分区域, 而那些模型中未被覆盖的区域可能隐藏着在训练阶段和测试阶段中未被发现的错误.

我们在第 1 节中介绍了 DF 的基本原理和结构, 分析 DF 的结构, 发现其有如下两个特点.

● 特点 1: 多粒度扫描和级联森林由数个随机森林组成, 每个森林的最小单位为决策树, 决策树的行为将影响 DF 的最终计算结果.

● 特点 2: 级联森林部分的每层森林都将上一层森林的输出作为本层的输入, 每层森林的输出都将影响级联森林部分的计算结果.

针对特点 1, 我们设计了随机森林节点覆盖率 (random forest node coverage, *RFNC*) 和随机森林叶子覆盖率 (random forest leaf coverage, *RFLC*); 针对特点 2, 我们设计了级联森林类型覆盖率 (cascade forest class coverage, *CFCC*) 和级联森林输出覆盖率 (cascade forest output coverage, *CFOC*). 如图 3 所示, 在测试中可使用随机森林内部结构的覆盖信息计算随机森林节点覆盖率和随机森林叶子覆盖率, 使用级联森林中每层输出的覆盖信息计算级联森林输出覆盖率和级联森林类型覆盖率.

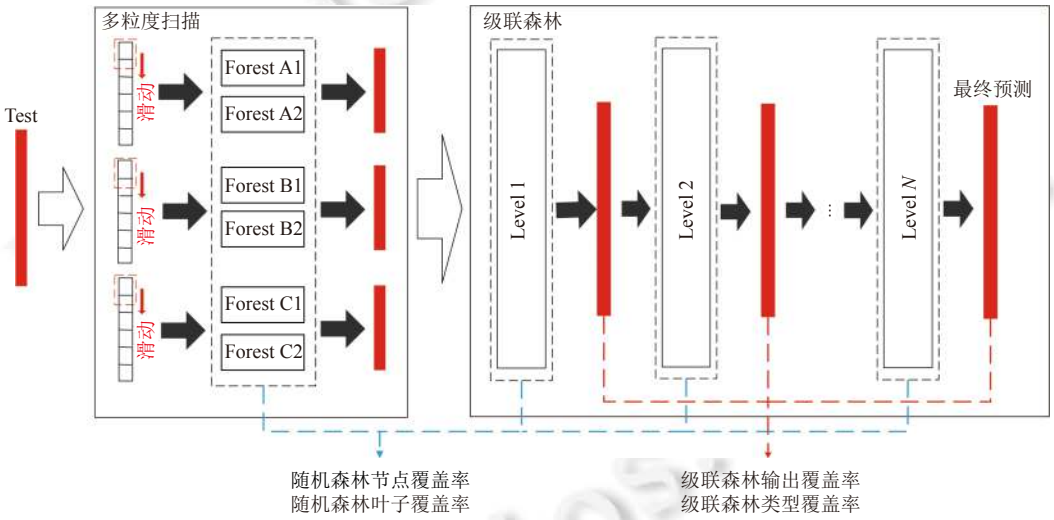

图 3 覆盖信息获取

4 种覆盖指标分别对深度森林的不同部分的测试覆盖情况进行评价, 其中随机森林叶子节点覆盖率侧重评价测试用例对决策树的决策路径的覆盖情况, 而随机森林节点覆盖率侧重评价决策树整体的节点覆盖情况. 级联森林类型覆盖率能分别显示测试用例对每个分类类型的覆盖情况, 级联森林输出覆盖率为级联森林类型覆盖率的平均值, 可综合评价测试用例对所有类型的总体输出覆盖情况.

2.1 随机森林覆盖信息

深度森林由多个随机森林组成, 每个随机森林是一种包含多个决策树的分类器, 其输出的分类结果由所有决策树输出的平均数或加权平均数决定. 本文选择将随机森林中决策树的内部节点作为测试覆盖率的最小覆盖粒度. 在一次预测中, 当覆盖了决策树的某个节点则称这个节点被覆盖, 在输入测试集后随机森林所达到的所有节点

数量与随机森林的总节点数量的比例即可为随机森林内部节点的覆盖率. 测试数据所到达的决策树叶子节点将直接决定决策树的输出, 进而影响最终分类结果, 所以计算叶子节点的测试覆盖非常重要, 因此本文定义了决策树的叶子节点覆盖率以评估测试数据对叶子节点的覆盖情况.

令 $D=\{d_1, d_2, d_3, \ldots\}$为一个随机森林中的所有决策树, $d_i \in D$ $(1<i<|D|)$ 为其中一个决策树, $T=\{t_1, t_2, t_3, \ldots\}$为测试集, $N(d_i)=\{n_{i,1}, n_{i,2}, n_{i,3}, \ldots\}$为决策树 d_i 的所有节点, $L(d_i)=\{l_{i,1}, l_{i,2}, l_{i,3}, \ldots\}$是决策树 d_i 的所有叶子节点, $\Psi(t_j, d_i)$ 是决策树 d_i 在测试 $t_j \in T$ 的分类过程中所有被覆盖的节点的集合, $\phi(t_j, d_i)$ 是决策树 d_i 在测试 $t_j \in T$ 的分类过程中所有被覆盖的叶子节点的集合. 也就是说, $\Psi(t_j, d_i)$ 和 $\phi(t_j, d_i)$ 分别是决策树 d_i 中被测试 $t_j \in T$ 覆盖的节点和叶子节点集.

公式 (1) 定义了决策树 d_i 中叶子节点的覆盖率 (leaf coverage, *LC*). 公式 (2) 定义了随机森林的叶子覆盖率 *RFLC*, 它是整个随机森林中所有决策树的平均叶子节点覆盖率.

$$LC(d_i, T) = \frac{\left|\left\{\phi\left(t_j, d_i\right) \middle| \exists t_j \in T\right\}\right|}{|L(d_i)|} \tag{1}$$

$$RFLC(D, T) = \frac{\sum_{d_i \in D}\left|\left\{\phi\left(t_j, d_i\right) \middle| \exists t_j \in T\right\}\right|}{\sum_{d_i \in D}|L(d_i)|} \tag{2}$$

需要注意的是, 不同路径的覆盖方式可能得到相同的叶子节点覆盖情况 (例如图 4 中左边和中间两棵决策树). 因此, 叶子节点的覆盖情况无法完整评估随机森林的覆盖范围. 为此, 本文引入节点覆盖率来评估决策树的内部结构覆盖率. 决策树的节点覆盖率 (node coverage, *NC*) 定义如公式 (3) 所示. 公式 (4) 定义了整个随机森林中所有决策树的平均节点覆盖率 *RFNC*.

$$NC(d_i, T) = \frac{\left|\left\{\Psi\left(t_j, d_i\right) \middle| \exists t_j \in T\right\}\right|}{|D(d_i)|} \tag{3}$$

$$RFNC(D, T) = \frac{\sum_{d_i \in D} NC(d_i, T)}{\sum_{d_i \in D}|D(d_i)|} \tag{4}$$

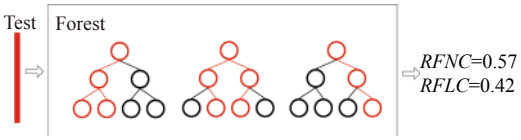

图 4 随机森林覆盖率计算

2.2 级联森林覆盖信息

级联森林使用多粒度扫描输出的类向量作为每层输入的一部分. 另外, 前一层级联森林的输出也会作为特征向量的一部分传递到下一层. 尽管来自前一层的类向量仅占所有特征向量的一小部分, 但每层的结果都会影响最终的预测. 本文使用级联森林内部结构的最小单位 (即随机森林) 来定义测试覆盖指标. 在一次预测中, 随机森林将输出一个类向量作为预测结果, 类向量中的每一维分别表示每个预测类型的概率, 取值为 0 到 1 之间, 将该区间平分为 K 个部分 (称为 K 分区), 每次预测结果将落入 K 个区间中的一个, 则称这个部分被覆盖. 在运行测试集后, 可通过随机森林输出所覆盖区间中不同部分的数量与 K 的比例计算覆盖率. K 值的取值越大则对测试的充分性越高, 对测试的要求也越严格.

设 $F=\{f_1, f_2, f_3, \ldots\}$是一个深度森林的级联森林部分中的所有随机森林, $f_l \in F$ 是其中一个随机森林, 集合 $C=\{c_1, c_2, c_3, \ldots\}$表示分类标签, $T=\{t_1, t_2, t_3, \ldots\}$为测试集, $\Omega(c_w, t_j, f_l)$ 表示森林 f_l 将 t_j 归类为 c_w 的概率. 以三分类深度森林为例, 输出是一个 3 维向量. 例如, 向量 (0.2, 0.5, 0.3) 表示将输入样本分为 3 个类型的可能性分别为 0.2、0.5 和 0.3. 分类可能性的值在 0 到 1 的范围内, 可以将其分为 K ($K \in N^*$) 个分段. $S_u^{f_l}$ ($1 \leqslant u \leqslant K$) 表示 f_l 的第 u 个输出范围. 如果 $\Omega(c_w, t_j, f_l)$ 落入 $S_u^{f_l}$, 则称测试 t_j 覆盖了 f_l 分类为 c_w 的第 u 个输出范围. 公式 (5) 描述了随机森林 f_l

的对类 c_w 的测试覆盖率. 公式 (6) 表示级联森林中所有随机森林对类 c_w 的测试覆盖率, 即级联森林类型覆盖率 (*CFCC*). 公式 (7) 表示级联森林对所有类型的测试覆盖率, 即级联森林输出覆盖率 (*CFOC*), 可以由 *CFCC* 的平均值获得.

$$FC(c_w, f_l, T) = \frac{\left|\left\{S_u^{f_l} \middle| \exists t_j \in T, \Omega\left(c_w, t_j, f_j\right) \in S_u^{f_l}\right\}\right|}{K} \tag{5}$$

$$CFCC(c_w, F, T) = \frac{\sum_{f_i \in F} FC(c_w, f_i, T)}{K \times |F|} \tag{6}$$

$$CFOC(C, F, T) = \frac{\sum_{c_w \in C} FCCC(c_w, F, T)}{|C|} \tag{7}$$

使用 *CFOC* 可综合计算整个级联森林的测试覆盖指标, 或使用 *CFCC* 单独计算对某个预测类型的测试指标. 例如有一个可用于识别手写汉字的深度森林软件, 预测的类型中有 2 000 个为常用汉字, 有 8 000 个为生僻汉字, 可以分别计算分析测试集对常用汉字和生僻字的测试覆盖率达到了什么水平.

2.3 示 例

如图 4 所示, 假设一个随机森林中有 3 棵决策树, 一组测试数据输入后, 每个测试数据都将被每个决策树分类到某个叶子节点. 图中红色部分为分类过程中经过的分支路径, 随机森林中一共有 21 个节点, 其中 12 个被覆盖; 一共有 12 个叶子节点, 其中 5 个被覆盖, 按照 *RFNC* 和 *RFLC* 的计算方法, 这个随机森林在当前测试集下所达到的 *RFNC* 覆盖率为 12/21=0.57, *RFLC* 覆盖率为 5/12=0.42. 可将多粒度扫描部分或级联森林部分的所有随机森林以相同方法计算覆盖率, 并求平均值即得到多粒度扫描部分或级联森林部分的 *RFNC* 和 *RFLC* 覆盖率.

假设级联森林部分的其中一层有 3 个随机森林, 如图 5 所示. 输入一条测试数据后, 每个随机森林都将对此测试数据进行预测, 并输出类向量作为预测结果 (假设为三分类问题). 假设 *K* 值为 5, 则输出域将被均分为 5 个 0.2 大小的区间, 3 个随机森林的输出将分别覆盖每个类型的一个区间.

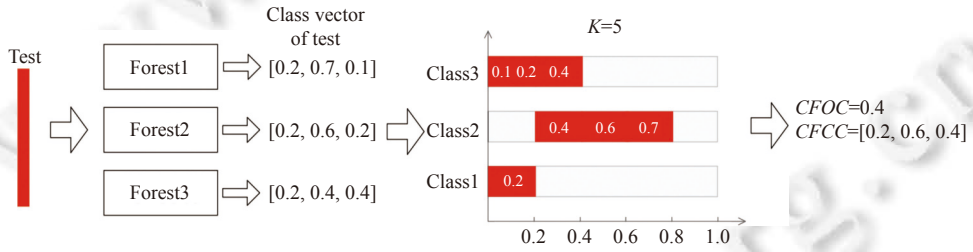

图 5 级联森林覆盖率计算

在图 5 中, Class1、Class2 和 Class3 代表 3 个分类类型, Class1 有一个区间被覆盖, Class2 和 Class3 分别有 3 个和 2 个区间被覆盖, 则本层 3 个随机森林的 15 个输出区间中有 6 个区间被覆盖. 按照 *CFCC* 和 *CFOC* 的计算方法, 本层在当前测试数据集下所达到的 *CFCC* 覆盖率为 [0.2, 0.6, 0.4], 这表示类型 Class1、Class2 和 Class3 的 *CFCC* 覆盖率分别为 0.2, 0.6 和 0.4. *CFOC* 覆盖率为 (0.2+0.6+0.4)/3=0.4. 将级联森林部分的所有层都用相同的方法计算覆盖率, 并求其平均值即可得到整个级联森林部分的 *CFCC* 和 *CFOC* 覆盖率.

3 覆盖制导的测试数据自动生成方法 DeepRanger

在上一节中, 介绍了针对 DF 的 4 种测试覆盖指标, 开发人员可使用上述覆盖指标对已有测试数据进行充分性度量. 但在实际应用中, 获得高覆盖率的有效测试数据集往往需要人工生成, 并手动标记测试数据集中的实例, 或使用训练-测试集的划分方式对模型进行训练和测试. 前者将提高测试成本, 而后者则将减小训练集规模并降低模型质量. 如何高效生成测试数据是深度学习领域的热点研究问题. 针对这一问题, 本文将面向深度森林的测试覆盖指标, 基于遗传算法使用覆盖率制导生成测试数据.

3.1 遗传算法

遗传算法 (genetic algorithm) 最早于 1975 年由 Holland 提出[11], 借鉴了达尔文进化论的自然选择和遗传学原理, 从生物种群的自然演化进程中提炼了自然选择、杂交遗传、变异等过程而发展起来. 遗传算法通常被用作搜索一个问题的最优解.

对于一个优化问题, 优化问题的解被称为个体, 可将其表示为一个变量序列 (在计算机领域通常为二进制串, 即基因序列), 这一过程称为编码. 算法开始时, 首先将输入或随机生成一定数量的个体, 这些个体被称为初始种群. 然后将通过适应度函数计算每个个体的适应度值, 按照适应度值进行排序, 再根据排序顺序使用不同的概率选择一部分个体 (适应度值越高, 被选择的概率越大) 进入下一代种群, 这个过程被称为自然选择. 然后被选择的子代个体将两两配对为父代, 每对父代有一定的概率会“生育”子代, 这个概率称为交叉概率, 将交叉的父代会交换其基因序列的一部分, 其中交换位置通常为随机选择, 这一过程称为交叉遗传. 交叉后产生的子代有一定概率会进行随机变异, 变异个体的基因序列将有极少量字节产生变异 (通常为字节取反), 变异位置通常为随机选择.

变异后的子代种群将重新进行适应度计算然后继续选择、交叉、变异过程, 不断重复以上操作, 直到满足终止条件. 遗传算法存在大量随机性行为, 只要相关参数设置得当, 在迭代过程中会产生适应度值相对较高的局部最优解, 而自然选择总是大概率选择高适应度的个体进入下一代, 所以多次迭代后的种群内个体的适应度将不断提高, 进而逐步接近全局最优解.

3.2 覆盖率制导的遗传算法

遗传算法使用适应度函数制导生成最优解. 当将其应用在测试数据生成问题时, 可使用指定的覆盖指标作为适应度函数, 以生成具有高覆盖率的测试数据. 本文选择使用遗传算法作为测试数据生成算法的基础, 并设计了针对基于深度森林覆盖率的适应度函数和相应的选择算子、交叉算子、变异算子, 形成了覆盖制导的深度森林测试数据生成方法 DeepRanger.

DeepRanger 的工作流程如图 6 所示. 首先, 将初始测试数据输入 DF 模型进行预测; 然后, 分别执行选择算子、交叉算子、变异算子. 其中选择算子和变异算子将输出父代种群和变异后的子代种群, 父代种群将被添加到优质个体种群, 然后与变异后的子代种群合并成为待选择种群输入给 DF 模型进行预测, 至此完成一次迭代; 最后, 不断重复以上过程直至满足终止条件. 终止条件可设置为预定的迭代次数或覆盖率值.

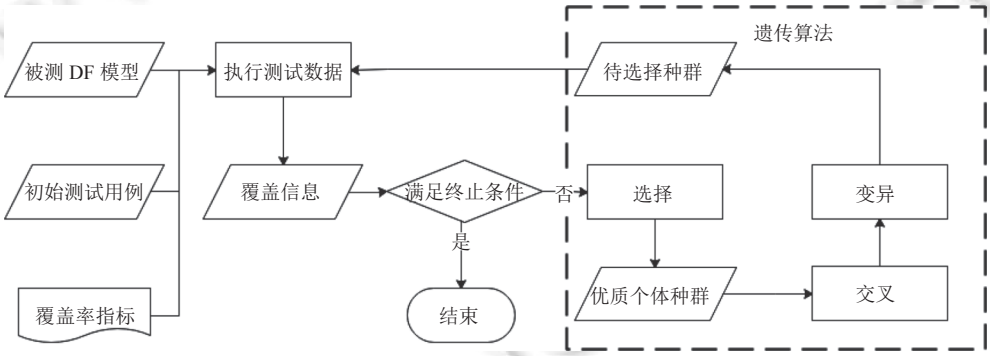

图 6 DeepRanger 流程图

3.2.1 选择算子

选择算子将分别计算输入种群个体的适应度值, 并以此作为自然选择的概率分配依据, 概率性选择其中的一部分作为父代种群. 选择算子的输入为作为测试目标的 DF 模型、待选择的当代种群、用于计算适应度的覆盖信息、选择的目标覆盖指标以及用于计算适应度的 K 分区数等参数. 其中, 使用覆盖信息计算适应度值, 而不是直接使用覆盖率作为适应度值. 这是因为不同个体可能会覆盖重复的决策树节点或随机森林输出区间, 简单选择高覆盖率的个体不利于提高种群的覆盖率水平, 无法制导选择高覆盖率的种群. 所以, 应使用覆盖信息作为计算适应度值的参数, 计算时可将单个个体输入 DF 模型以获取其覆盖率信息. 覆盖信息详细记录了所达到的每个节点和输出区间,

将个体的覆盖信息与当前待选择种群的覆盖信息相比较, 可获得个体所覆盖的节点或输出区在当前待选择种群中的覆盖频率, 频率越低则适应度值越高. 将所覆盖区域的频率倒数作为适应度值即可对个体进行适应度排序.

自然选择将选择种群中一定数量的个体进入下一代, 其余超出预定种群数量的未被选择个体将被抛弃, 选择的基本原则为适应度值越高的个体被选择的概率就越大. 具体做法为, 设全概率为 $x=100\%$、分配概率为 p, 在个体排序序列中从第 1 个开始分配选择概率, 使第 1 个个体的被选择概率为 $p_1=x\times p$, 使剩余未分配概率为 $x=x-p_1$; 然后使第 2 名个体的被选择概率为 $p_2=x\times p$, 使剩余未分配概率为 $x=x-p_2$. 以此类推, 直到所有个体的被选择概率都分配完成. 这样一来, 适应度越高的个体被选择的概率越高, 且适应度低的个体也有机会进入下一代种群, 符合遗传算法的基本原则.

3.2.2 交叉算子

交叉算子将选择算子选择出的父代种群按照不同的分类类型进行两两配对, 每对父代个体之间将有一定概率进行基因序列交叉产生子代. 交叉算子的输入为选择算子输出的待交叉种群、父代之间的交叉概率以及基因序列的交叉矩形窗口大小. 其中按照不同的分类类型进行两两配对是指, 配对组成父代的两个个体必须是分类结果相同的测试数据, 假如没有同类配对的限制将可能交叉产生无意义的子代个体. 例如将一张分类结果为数字 5 的图片和一张分类结果为数字 9 的图片进行交叉, 所产生的子代个体可能无法被分类为任何结果, 这样的个体即使拥有较高的适应度值, 但对 DF 模型的测试来说是没有意义的.

配对后每对父代都有一定概率 (通常为 0.5 到 0.9 之间) 进行交叉操作, 未进行交叉的父代将被抛弃. 即将交叉基因序列的父代将随机选择一个交叉位置, 然后交换位于交叉位置大小为交叉窗口大小区域中的编码 (例如, 对于图片类型数据为矩形区域的像素值). 交换后所产生的两个子代个体将进入子代种群.

3.2.3 变异算子

变异算子将随机选择交叉算子生成的子代种群中个体变异位置, 并进行变异操作, 产生变异后的子代种群. 变异算子的输入为交叉算子输出的待变异种群和单个个体的基因变异位数量. 其中单个个体的变异位置为随机选择, 每次选择与待变异位数相同的位置数, 然后将变异位置的基因进行变异 (例如, 对于二进制基因序列或黑白灰度图片可将变异位置取反, 对于 32 位 ARGB 图片或更高维的数据样本, 可对变异位置添加一个较小的扰动以防止数据标签被改变). 通常情况下, 遗传算法由概率决定是否对基因位进行变异, 不同基因位都有一定概率进行变异. 为提高效率, 我们不采用对每个基因位按照变异概率进行变异的方式, 而是对个体随机选择指定数量的基因位进行变异.

3.2.4 覆盖率制导的测试数据生成

DeepRanger 的具体过程如算法 1 所示. 算法 1 的输入包括: 被测深度森林模型 *gcf*、原始测试集 *orgTest*、原始测试集覆盖信息 *covInfo*、目标覆盖指标 *coverage*、K 分区取值 *K*、交叉概率 *CrossP*、交叉窗口大小 *crossN*、变异位置数 *mutationN* 以及迭代次数 *iterationN*. 算法 1 将返回迭代后获得的优质子代种群 *population*.

算法 1. 覆盖率制导的测试数据生成算法.

输入: *gcf*, *orgTest*, *covInfo*, *coverage*, *K*, *CrossP*, *crossN*, *mutationN*, *iterationN*;
输出: *population*.

1. *testSet* ←*orgTest*//将初始测试数据作为待选择种群
2. **for** *t* in range(*iteration*) **do** //指定迭代次数
3. *covInfo* ← getCov(*testSet*) //根据运行情况更新覆盖信息
4. *choose* ← select(*gcf, testSet, covInfo, coverage, K*) //执行选择算子
5. *population* ← *population*∪*choose*//合并
6. *children* ← cross(*choose, CrossP, crossN*) //执行交叉算子
7. *testSet* ← *mutations*(*children, mutationN*)∪*choose* //执行变异算子
8. **return** *population* //输出优质子代种群

在算法 1 中, 第 3 行获取测试数据集的覆盖信息, 用于适应度计算; 第 4 行为选择算子, 使用测试数据覆盖信息计算其适应度值, 并根据适应度值从待选择种群中选择父代种群; 第 5 行将父代种群与优质子代种群合并后成为新的优质子代种群; 第 6 行为交叉算子, 将输入的父代种群进行交叉操作, 产生子代种群; 第 7 行为变异算子, 将输入的子代种群进行变异操作, 产生变异子代种群, 变异子代种群与父代种群合并后成为待选择种群, 用于下一代选择. 整个过程将循环迭代 *iterationN* 次, 最后将返回优质子代种群 *population*.

4 实证研究

4.1 实验设计

实验部分使用了基于 Scikit-learn (https://scikit-learn.org/stable/) 中 RandomForestClassifier 实现的深度森林开源项目 gcForest (https://github.com/pylablanche/gcForest), 并使用 MNIST[12]手写数字数据集训练 DF 模型, 所有训练和测试都基于 32 GB RAM 和 AMD 3700X 8-Core Processor 计算机上完成.

本文通过研究以下研究问题来验证所提出覆盖指标和测试数据生成方法的有效性.

RQ1: 所提出的 4 种覆盖指标是否能评价测试集的充分性?

RQ2: 所提出的 4 种覆盖指标是否能评价训练集的质量?

RQ3: DeepRanger 是否能有效生成测试数据?

RQ4: 在不同迭代次数下, DeepRanger 是否能比基于随机选择的遗传算法更快地提高覆盖率?

有效的覆盖指标是制导生成高质量测试数据的前提, 设计了 RQ1 和 RQ2 用以讨论覆盖指标的有效性. 为验证 DeepRanger 的有效性, 设计了 RQ3 用以验证 DeepRanger 所生成的测试数据是否比随机方法生成测试数据的覆盖率更高. 为分析在不同测试开销下 DeepRanger 生成测试数据的性能, 设计了 RQ4 用以评估 DeepRanger 在不同迭代次数下的测试效果.

4.2 实验对象

本文的实验对象采用了 MNIST 数据集, 该数据集中每个样本是 28×28×1 的手写阿拉伯数字图片, 其包括 60 000 条训练集和 10 000 条测试集. 由于 DeepRanger 生成测试数时所添加的扰动很小, 通常不应该改变原始数据的标签, 所生成测试数据默认保持原标签不变. 另外, 当模型预测生成测试数据的标签与默认标签不一致时, 会进行人工核查, 以确保生成测试数据标签的正确性. 在训练 DF 模型时, 不同多粒度扫描窗口大小和数量将决定 DF 模型的复杂程度. 为此, 实验中我们使用 3 个不同复杂度的深度森林模型作为实验对象. 表 1 中列出了设置不同多粒度扫描窗口大小的超参数训练得到的 DF 模型信息.

表 1 用于实验的深度森林模型信息

模型	多粒度扫描 (窗口大小)	级联森林层数	测试精度 (%)
DF_0	14	1	98.96
DF_1	14, 18	2	98.68
DF_2	14, 18, 22	3	98.66

对于模型训练的超参数, 本文使用 Deep Forest 算法[8,9]的推荐值 (d/4=28×28/4=14×14) 作为多粒度扫描窗口训练 DF_0, 并在此基础上扩展为 (14, 18) 和 (14, 18, 22) 训练 DF_1 和 DF_2 模型. 即: DF_0 的多粒度扫描窗口为 14; 模型 DF_1 的多粒度扫描窗口为 14 和 18; 模型 DF_2 的多粒度扫描窗口为 14, 18 和 22. 这代表模型 DF_0 的多粒度扫描部分有 1 对随机森林, 而 DF_1 和 DF_2 的多粒度扫描部分分别为 2 对和 3 对随机森林. 级联森林部分的层数由训练自主决定, 有一定随机性. 3 个模型的测试精度都达到 98.6% 以上, 达到可实际应用的水平.

通过在 DF 模型内部插桩可获得测试数据对模型内部的覆盖情况, 具体方式为在随机森林使用决策树分类时, 调用 RandomForestClassifier 内部的 decision_path() 方法可获得测试数据所经过的所有决策树节点, 然后将节点覆盖信息和随机森林的分类结果一起作为覆盖信息输出.

4.3 针对 RQ1 的实验结果分析

有效的测试覆盖指标应该与测试数据集的充分性正相关[6]. 为验证论文所提覆盖指标的有效性, 本文使用原始测试数据中的前 5000 条和前 10000 条分别进行测试, 并使用不同的 K 分区取值计算其覆盖率. 通过执行不同测试数据集, 得到了相应的覆盖率信息, 其中表 2 显示的是随机森林节点覆盖率 *RFNC*; 表 3 显示的是随机森林叶子节点覆盖率 *RFLC*; 表 4 显示的是级联森林类型覆盖率 *CFCC*; 表 5 显示的是级联森林输出覆盖率 *CFOC*. 在表 2–表 5 中, “结构”表示计算的是深度森林中哪一部分的覆盖率, 其中, “MSG”代表多粒度扫描部分, “CF”代表级联森林部分; “测试集”中, “5000_{org}”和“10000_{org}”分别代表使用的测试集分别为 5000 条和 10000 条原始测试数据, 其中下标“org”表示所使用的测试数据来自原始测试集; “*K* 值”表示覆盖率计算时使用的 *K* 分区取值.

表 2 随机森林节点覆盖率 (*RFNC*)

模型	结构	测试集	
		5000_{org}	10000_{org}
DF_0	MGS	0.7895	0.8721
	CF	0.7716	0.8458
DF_1	MGS	0.7898	0.8722
	CF	0.7682	0.8401
DF_2	MGS	0.7890	0.8716
	CF	0.7643	0.8372

表 3 随机森林叶子节点覆盖率 (*RFLC*)

模型	结构	测试集	
		5000_{org}	10000_{org}
DF_0	MGS	0.5958	0.7443
	CF	0.5801	0.6998
DF_1	MGS	0.5969	0.7448
	CF	0.5737	0.6895
DF_2	MGS	0.5957	0.7436
	CF	0.5685	0.6847

表 4 级联森林类型覆盖率 (*CFCC*)

模型	*K*值	测试集	类覆盖率									
			0	1	2	3	4	5	6	7	8	9
DF_0	500	5000_{org}	0.3834	0.2833	0.4862	0.4623	0.4488	0.4782	0.4040	0.4935	0.4586	0.4912
		10000_{org}	0.4705	0.3562	0.5805	0.5609	0.5142	0.5723	0.4786	0.5644	0.5452	0.5663
	1000	5000_{org}	0.2714	0.1967	0.3572	0.3361	0.3213	0.3444	0.2857	0.3521	0.3294	0.3534
		10000_{org}	0.3412	0.2535	0.4414	0.4223	0.3895	0.4238	0.3599	0.4152	0.4101	0.4195
DF_1	500	5000_{org}	0.3442	0.2472	0.4304	0.4354	0.4144	0.4349	0.3544	0.4584	0.4532	0.4626
		10000_{org}	0.4182	0.3093	0.5157	0.5247	0.4746	0.5212	0.4172	0.5169	0.5433	0.5300
	1000	5000_{org}	0.2435	0.1743	0.3138	0.3085	0.2972	0.3122	0.2488	0.3190	0.3204	0.3312
		10000_{org}	0.3006	0.2181	0.3892	0.3853	0.3534	0.3843	0.3010	0.3753	0.3962	0.3924
DF_2	500	5000_{org}	0.3211	0.2450	0.4033	0.4314	0.4092	0.4161	0.3453	0.4434	0.4390	0.4585
		10000_{org}	0.3851	0.3082	0.4905	0.5200	0.4663	0.5044	0.4014	0.5010	0.5174	0.5289
	1000	5000_{org}	0.2250	0.1703	0.2891	0.3114	0.2914	0.2945	0.2413	0.3053	0.3084	0.3290
		10000_{org}	0.2763	0.2204	0.3621	0.3820	0.3496	0.3653	0.2881	0.3612	0.3755	0.3872

如表 2 所示, 在对 3 个模型 DF_0、DF_1 和 DF_2 的测试中, 5000 条原始测试集达到的 *RFNC* 均低于 10000 条原始测试集. 此外, 3 个模型中多粒度扫描部分的覆盖率均比级联森林部分高, 分析发现这是因为模型训练时设定的超参数使多粒度扫描部分随机森林的决策树数量比级联森林部分多, 从而使每个决策树所能使用的特征值少于级联森林部分的决策树, 所以多粒度扫描部分的决策树的平均节点数也比级联森林部分的决策树少, 从而更容易达到更高的覆盖率.

如表 3 所示, 在对 3 个模型 DF_0、DF_1 和 DF_2 的测试中, 5000 条原始测试集达到的 *RFLC* 均低于 10000 条原始测试集. 3 个模型中多粒度扫描部分的覆盖率均比级联森林部分高, 这个现象的原因与 *RFNC* 的原因一样, 是两个部分中决策树平均节点数量差异导致的. 另外, *RFLC* 的覆盖率均低于 *RFNC*, 分析发现这是因为单个测试数据只能覆盖一个叶子节点, 但能同时覆盖多个其他节点, 这使得 *RFLC* 的高覆盖率更难达到.

表 4 中表示的是 3 个模型 DF_0、DF_1 和 DF_2 使用不同数量的原始测试集时, 所达到的 *CFCC* 覆盖率. “类覆盖

率”中的“0–9”列分别表示测试集对 MNIST 手写识别分类 10 个类型 (即数字 0–9) 的不同覆盖率.

由表 4 可见, 在 3 个模型 DF_0、DF_1 和 DF_2 的测试中, 5 000 条原始测试集达到的 *CFCC* 均低于 10 000 条原始测试集, 且 *K* 值提高后各个类型的覆盖率均明显下降, 可以认为 *K* 值的提高使测试集更难达到高覆盖率. 其中, 个别类型 (如类 1 和类 6) 的覆盖率明显低于其他类型. 分析发现这是因为在原始测试集中, 这两个类型的测试数据数量远低于其他类型, 可以认为原始测试集中这两个类型的测试充分性远低于其他类型. 实验所得的覆盖率准确反映了这一情况. 在实际测试中, 若出现某类型的 *CFCC* 覆盖率明显低于其他类型时, 测试人员可针对性的增加该类型测试数据数量.

如表 5 所示, 在对 3 个模型 DF_0、DF_1 和 DF_2 的测试中, 5 000 条原始测试集达到的 *CFOC* 均低于 10 000 条原始测试集. 并且, 在相同模型和测试集下, 设置不同的 *K* 值能得到不同的覆盖率结果, 较低的 *K* 值设置均获得了更高的覆盖率.

上述实验结果表明, 对于测试数据更多的测试数据集, 覆盖率更高. 证实了所提出的 4 种覆盖指标所计算出的覆盖率值与测试的充分性有正相关关系, 测试使用的测试数据集越充分则计算出的覆盖率越高. 4 种覆盖率可分别计算深度森林不同部分以及针对不同分类类型的覆盖率, 在对深度森林的测试工作中具有有效的指导作用.

有效的覆盖指标应能对后续测试充分程度进行评价, 如应能告诉测试人员还需要添加哪些类型的测试数据. 为此, 本文设计了一组实验, 将 MNIST 数据集中的 10 000 条原始测试集做 10 次有放回随机抽样, 每次抽取 1 000 条测试数据, 使每次抽样都包含不同类型数量的样本 (第 1 次包含 1 种类型, 第 2 次包含 2 种, 以此类推至第 10 次包含全部 10 种类型, 所包含类型随机决定), 将 10 次的抽样数据输入 DF_0 计算覆盖率, 并统计了各种覆盖指标和测试集包含类数量之间的皮尔逊相关系数 (Pearson correlation coefficient).

皮尔逊相关系数是用于度量两个变量 *X* 和 *Y* 之间的线性相关程度的指标, 其值介于−1 与 1 之间. 通常认为, 皮尔逊相关系数在 0.8~1.0 为极强相关, 0.6~0.8 为强相关, 0.4~0.6 为中等程度相关, 0.2~0.4 为弱相关, 0.0~0.2 为极弱相关或无相关[13]. 在皮尔逊系数的假设检验中, 设原假设 H0: 没有线性相关关系, 备择假设 H1: 存在线性相关关系, 当 P-value 小于 0.01, 在 99% 的置信水平上拒绝原假设, 接受备择假设, 即样本有线性相关性; 当 P-value 小于 0.1 的时候, 在 90% 的置信水平上拒绝原假设, 接受备择假设, 即样本有线性相关性.

如表 6 所示, 10 次抽样结果的覆盖率与其所包含的类型数量呈正相关趋势, 在测试用例数量相同的情况下, 所包含的类型数量越多的抽样结果其覆盖率也越高. 其中, 除 CFOC 覆盖率的皮尔逊相关系数为 0.615, 呈现强相关以外, 其他 4 种覆盖率的皮尔逊相关系数均达到 0.9 左右, 呈极强相关. 其中 CFOC 的 P-value 小于 0.1, 在 90% 的置信水平上拒绝原假设, 其余覆盖率的 P-value 均小于 0.01, 在 99% 的置信水平上拒绝原假设, 表明覆盖率和测试集所包含类的数量线性正相关.

表 5 级联森林输出覆盖率 (*CFOC*)

模型	*K*值	测试集	
		5000_{org}	10000_{org}
DF_0	500	0.4390	0.5206
	1 000	0.3146	0.3856
DF_1	500	0.4031	0.4772
	1 000	0.2867	0.3495
DF_2	500	0.3908	0.4619
	1 000	0.2763	0.3356

表 6 相同规模不同类数量的测试集覆盖率对比

类数量	MGS_RFNC	MGS_RFLC	*CFOC*	CF_RFNC	CF_RFLC
1	0.3188	0.1595	0.0899	0.2506	0.1284
2	0.3620	0.1897	0.1019	0.3126	0.1529
3	0.3780	0.1874	0.0889	0.2901	0.1395
4	0.4436	0.2210	0.1196	0.4218	0.2156
5	0.4507	0.2258	0.1139	0.4246	0.2163
6	0.4684	0.2314	0.1095	0.4351	0.2146
7	0.4756	0.2349	0.1154	0.4650	0.2333
8	0.4973	0.2392	0.1103	0.4665	0.2324
9	0.5098	0.2430	0.1148	0.4894	0.2425
10	0.5174	0.2444	0.1092	0.4901	0.2381
皮尔逊相关系数	0.9613	0.9197	0.6150	0.9262	0.8961
P-value	9.41E–6	1.65E–4	5.84E–2	1.19E–4	4.49E–4

对级联森林类覆盖率 (*CFCC*) 进行进一步分析 10 次抽样的 *CFCC* 更加清晰地显示出了测试集之间的区别, 如表 7 所示. 表 7 中, 每次抽样所包含类型的覆盖率值都被加粗表示, 抽样所包含类型的类覆盖率明显高于没有包含的类型, 测试人员可通过参考类覆盖率得知在后续测试中需要重点添加哪些类型的测试数据以提高测试充分性.

表 7 相同规模不同类数量 *CFCC* 对比

类数量	包含类	*CFCC* (类覆盖率)									
		0	1	2	3	4	5	6	7	8	9
1	2	0.0669	0.0548	**0.2500**	0.1020	0.0565	0.0622	0.0581	0.1125	0.0768	0.0592
2	8 9	0.0705	0.0411	0.0780	0.0829	0.0919	0.0839	0.0620	0.0745	**0.2246**	**0.2096**
3	1 5 6	0.0705	**0.1091**	0.0630	0.0789	0.0582	**0.1721**	**0.1493**	0.0543	0.0780	0.0555
4	2 4 5 9	0.0700	0.0483	**0.1780**	0.1010	**0.1756**	**0.1735**	0.0702	0.0889	0.0980	**0.1926**
5	2 3 4 5 9	0.0688	0.0455	**0.1670**	**0.1664**	**0.1443**	**0.1578**	0.0609	0.0829	0.0816	**0.1637**
6	0 2 34 8 9	**0.1163**	0.0447	**0.1489**	**0.1430**	**0.1348**	0.0804	0.0569	0.0736	**0.1479**	**0.1481**
7	0 2 34 5 8 9	**0.1108**	0.0429	**0.1424**	**0.1481**	**0.1397**	**0.1458**	0.0636	**0.0772**	**0.1335**	**0.1500**
8	1 2 3 4 5 7 8 9	0.0501	**0.0828**	**0.1412**	**0.1443**	**0.1225**	**0.1279**	0.0503	**0.1280**	**0.1279**	**0.1275**
9	1 2 3 4 5 6 7 8 9	0.0647	**0.0809**	**0.1421**	**0.1321**	**0.1228**	**0.1213**	**0.1030**	**0.1228**	**0.1304**	**0.1284**
10	0 1 2 3 4 5 6 7 8 9	**0.1025**	**0.0788**	**0.1264**	**0.1057**	**0.1196**	**0.1088**	**0.1003**	**0.1096**	**0.1173**	**0.1232**

针对 RQ1 的结论: 在实验中, 所提出的 4 种覆盖指标与测试充分程度正相关, 可有效评价测试的充分性, 其中 CFCC 还可用于指导补充完善特定类型的测试数据.

4.4 针对 RQ2 的实验结果分析

为验证覆盖率与训练集质量的相关性, 本文对 MNIST 的 60000 条原始训练集做有放回随机抽样, 每次分别抽取 10000 至 50000 条数据作为训练集, 并分别训练了 5 个深度森林模型, 使用同样的 10000 条原始测试数据进行测试, 将相应的训练集输入 DF_0 以获取各训练集的覆盖率.

如表 8 所示, 5 个模型的训练集在 DF_0 中的覆盖率呈现出测试精度越高则覆盖率越高的趋势, 5 种覆盖率与测试精度的皮尔逊相关性系数达 0.9 以上, 呈极强相关, 且 P-value 均小于 0.01, 表明训练集的覆盖率与测试精度线性正相关.

表 8 不同规模训练集生成深度森林模型的测试精度及覆盖率

模型	$DF_{3.1}$	$DF_{3.2}$	$DF_{3.3}$	$DF_{3.4}$	$DF_{3.5}$	皮尔逊相关系数	P-value
测试精度	0.9796	0.9837	0.9862	0.9878	0.9879	—	—
MGS_RFNC	0.8859	0.9400	0.9657	0.9814	0.9921	0.9959	3.14E–4
MGS_RFLC	0.7719	0.8799	0.9314	0.9628	0.9841	0.9959	3.13E–4
CF_RFNC	0.2566	0.3461	0.4115	0.4562	0.4954	0.9838	2.46E–3
CF_RFLC	0.7907	0.8612	0.8944	0.9130	0.9258	0.9959	3.11E–4
CFOC	0.6067	0.7298	0.7921	0.8279	0.8528	0.9963	2.73E–4

针对 RQ2 的结论: 训练集的覆盖率与模型的测试精度呈正相关关系, 所提出的覆盖指标能有效评价训练集的质量.

4.5 针对 RQ3 的实验结果分析

为了验证 DeepRanger 生成测试数据的有效性, 实验中以 DF_0 模型为例使用 DeepRanger 进行测试. 实验使用规模为 50 的原始测试集, K 值为 1000, 交叉概率为 0.8, 交叉矩形边长为 5, 变异像素数量为 10, 迭代 20 次. 迭代完成后将运行所生成的新测试数据集并计算其对 DF_0 模型的覆盖率. 分别选择 4 种覆盖指标作为目标制导测试数据生成, 与基于随机选择的遗传算法生成测试数据进行对比, 每种方式运行 10 次后取其结果的平均值, 所得覆盖率值如表 9 所示. 基于随机选择的遗传算法在选择算子中使用随机选择的方式选择父代种群, 而 DeepRanger 使用覆盖率制导的选择方式, 除此之外其他参数设置和输入均相同.

表 9 中, “原始测试集”“随机选择”和“DeepRanger”分别代表测试所使用的测试集为: 50 条原始测试数据、基

于随机选择的遗传算法生成的测试集和 DeepRanger 生成的测试集. *CFCC* 中的数据为 10 类分类中, 每个类型分别达到的类覆盖率, 如原始测试集的 *CFCC* 覆盖率向量 [0.013, 0.013, 0.014, 0.013, 0.015, 0.012, 0.013, 0.015, 0.011, 0.016] 中的第一个元素为 0.013, 代表类型“0”的覆盖率为 0.013.

表 9 覆盖率对比

覆盖指标	结构	原始测试集	随机选择	DeepRanger
RFNC	MGS	0.0818	0.1507	0.1804
	CF	0.1307	0.2455	0.2945
RFLC	MGS	0.0163	0.0518	0.0707
	CF	0.0283	0.0988	0.1174
CFOC	—	0.0134	0.1278	0.1638
CFCC	—	[0.013, 0.013, 0.014, 0.013, 0.015, 0.012, 0.013, 0.015, 0.011, 0.016]	[0.158, 0.087, 0.153, 0.096, 0.182, 0.106, 0.098, 0.143, 0.122, 0.133]	[0.114, 0.193, 0.177, 0.173, 0.188, 0.149, 0.12, 0.165, 0.192, 0.168]

DeepRanger 所生成的新测试集的各个覆盖率相较原始测试集均有明显提升, 且相比随机选择的遗传算法, 多粒度扫描部分的 *RFNC* 和 *RFLC* 的覆盖率分别提高了 2.97% 和 1.89%, 级联森林部分的 *RFNC*、*RFLC* 和 *CFOC* 覆盖率分别提高了 4.9%、1.86% 和 3.6%, *CFCC* 除个别类型 (类型“0”) 的覆盖率有下降以外, 其他类型覆盖率均有不同程度提高. 这说明 DeepRanger 能生成更高覆盖率的测试数据.

针对 RQ3 的结论: DeepRanger 所生成的测试数据能提高原始测试集的覆盖率.

4.6 针对 RQ4 的实验结果分析

为分析在不同测试开销下 DeepRanger 生成测试数据的性能, 对不同迭代次数下 DeepRanger 和基于随机选择的遗传算法所生成测试数据的覆盖率情况进行对比.

如图 7 所示, 实线为 DeepRanger 在 20 次迭代中的覆盖率变化曲线, 虚线为基于随机选择的遗传算法的覆盖率变化曲线. 其中, 图 7(a) 为多粒度扫描部分的 *RFNC* 覆盖率 (蓝色曲线), 图 7(b) 为多粒度扫描部分的 *RFLC* 覆盖率 (绿色曲线), 图 7(c) 为级联森林部分的 *RFNC* 覆盖率 (红色曲线), 图 7(d) 为级联森林部分的 *RFLC* 覆盖率 (青色曲线), 图 7(e) 为 *CFOC* 覆盖率 (紫色曲线).

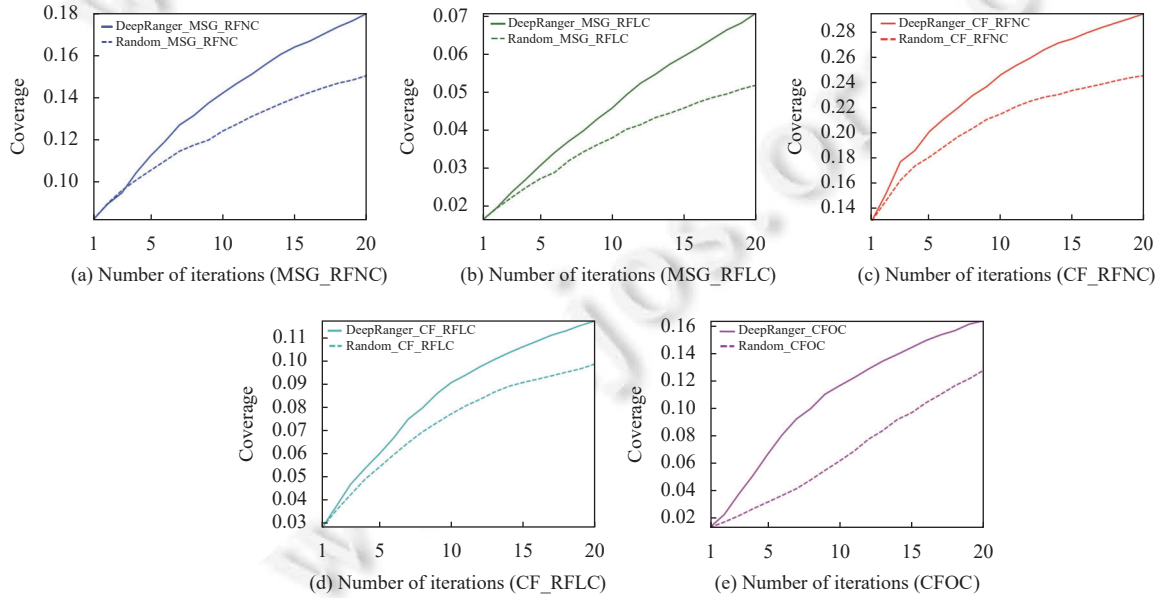

图 7 覆盖率增长速度对比

由图 7 可以看出, DeepRanger 所有覆盖指标的增长速度都比基于随机选择的遗传算法更快, DeepRanger 在不同迭代次数下均取得了比随机选择的遗传算法更高的覆盖率. 实验结果表明, 所设计实现的测试数据自动生成方法 DeepRanger 能比随机选择的遗传算法更快的提高测试数据集的覆盖率.

针对 RQ4 的结论: 与基于随机选择的遗传算法相比, DeepRanger 方法在不同迭代次数下所生成测试数据的覆盖率增长速度更快.

5 有效性分析

本节从内部有效性和外部有效性两个方面来分析本文所提出方法的有效性.

对覆盖指标和 DeepRanger 的内部有效性的影响主要在于 3 个方面. 首先是在覆盖指标的有效性验证实验中, 实验的测试集均取自 MNIST 数据集中的原始测试集, 且只有数量规模或包含类型数量的区别, 排除了其他因素对实验结果有影响的可能性. 其次是 DeepRanger 的有效性验证实验中, 本文所使用的覆盖率制导的遗传算法, 和用于对比的基于随机选择的遗传算法之间只有选择算子的选择策略存在区别, 且所有数据均为 10 次实验结果的平均值, 尽量排除了遗传算法的随机性造成的结果误差和其他因素对实验结果带来影响的可能性. 第三是代码实现的正确性, 为了减少重新实现可能会引入的人为因素影响, 本文使用了开源项目 gcForest 和开源框架 Sklearn, 并通过实例对实验代码的正确性进行了验证.

对覆盖指标和 DeepRanger 的外部有效性的影响主要在于两个方面. 首先是实验选取的实验对象. 由于目前相关领域内缺乏被广泛使用的通用 DF 模型, 所以本文使用的 3 个 DF 模型均为自行训练获得, 因此无法保证覆盖指标和 DeepRanger 在其他 DF 模型上的适用性. 本文使用被深度学习领域广泛使用的 MNIST 数据集用于模型的训练和测试, 在与随机生成测试数据对比的实验中使用只有一个扫描窗口 DF_0 以简化计算, 在覆盖率有效性评价的实验中使用了 3 个复杂程度不同的 DF 模型作为实验对象, 这在一定程度上弥补了实验的不足. 另外, DeepRanger 所生成的测试数据与真实的测试数据存在一定差异, 但可作为对抗样本, 以选择出面对攻击时性能更优的深度森林模型.

6 相关工作

本节将介绍与本文相关的其他研究工作, 包括基于覆盖的深度学习软件测试和使用覆盖制导生成测试数据的研究, 以及深度森林及其应用的相关研究工作.

(1) 深度学习软件测试

对于深度学习软件测试, 传统方法一般是使用一系列与训练集相似的数据集测试模型, 计算其精度以衡量模型质量[14]. 然而许多研究表明, 这种方式使模型中许多隐藏的错误无法在测试阶段被发现[15,16], 早前的 DeepTest[17]使用缩放、剪切、旋转等方法转换图片以测试自动驾驶系统, 并成功检测出自动驾驶系统中的潜在缺陷. DeepTest 和之后的一些研究[18,19]都表明, 深度学习系统对测试数据的覆盖深度要求比传统软件规模更大. 将传统软件的修正条件/判定覆盖 (modified condition/decision coverage, MC/DC)[20]指标的思想代入深度学习系统是一种可行的研究方向. 以 MC/DC 指标为灵感, Sun 等人[21]提出了一套改进的针对深度学习软件的 MC/DC 指标, 并在小规模的 DNN 模型中表现出了比随机测试更高的测试质量. Pei 等人提出的白盒差分测试算法[22], 引入了神经元覆盖的思想用于评价 DNN 内部的覆盖率. Ma 等人在 DeepCT[23]提出了一种以 DNN 不同层级之间互相作用的输入输出评价的覆盖指标, 并用其指导生成测试数据以实现缺陷检测.

Ma 等人提出了一种用于神经网络的测试覆盖指标 DeepGauge[6], 为智能程序的测试提供了新的思路. DeepGauge 使用类似传统软件的灰盒测试的思想, 针对神经网络的内部结构, 分析测试数据对神经网络结构的不同区域覆盖情况, 从而定义出了一套覆盖指标以评价测试充分程度. DeepGauge 的出现使对神经网络的测试充分性有了可量化的指标. 在其基础上, Xie 等人提出一种可行的测试方法 DeepHunter[7], 其使用覆盖率作为指导自动生成高覆盖率的测试集, 且能在几个经典的 DNN 模型中找到大量错误.

研究表明, 将对抗样本加入到输入中能影响深度学习系统的结果, 其中 FGSM (fast gradient sign method)[24]是

一种基于梯度的攻击方式, 在白盒环境下, 通过求出模型对输入的导数, 然后用符号函数得到其具体的梯度方向, 乘以一个步长得到的“扰动”, 叠加在原来的输入上就得到了在 FGSM 攻击下的样本. FGSM 能在图片上添加一些肉眼难以识别的修改, 但能使 DL 图像识别系统出现错误判断, 其已经被证实能在深度学习系统中引发大量错误. Kim 等人提出的 SADL 方法[25]使用覆盖指标指导深度学习训练, 可将对抗性样本的分类精度提升高达 77.5%.

Li 等人[26]对结构化覆盖标准提出了质疑, 其研究结果表明测试集错分样本数量与神经网络结构化覆盖之间不存在关联关系. 原因是对抗输入和自然输入在输入空间中的共存打破了传统编程中结构化覆盖标准的同质性假设; 覆盖引导搜索和对抗引导搜索策略本质相似, 神经元覆盖并没有提供更多关于模型质量的信息. Li 等人认为有效的覆盖指标应满足: 面向自然应用场景时, 要求具有能够区分相同规模、不同错误率的自然样本集的能力; 面向对抗攻击场景时, 相比面向对抗的搜索策略, 要具有更高效生成对抗样本的能力. 本文提出的覆盖指标可区分相同规模的自然测试样本集之间差异, 并且能高效生成高覆盖率的对抗样本, 可认为是一种有效的覆盖指标.

(2) 深度森林以及应用

深度森林[8,9]作为一种新兴的深度学习模型, 克服了许多 DNN 所具有的缺点, 例如: 需要大量标记数据、高计算性能设备、大量超参数、对参数敏感以及低可解释性. 目前, 深度森林已经在小规模数据集和低算力平台上取得了较高的性能结果, 而 DNN 往往需要更高的训练成本才能达到同样效果, 作为深度学习领域的有力竞争者, 深度森林已有一些实际应用. 例如: 卢喜东等人[27]将恶意代码映射为无压缩的灰度图像, 然后使用深度森林对样本分类以检测恶意代码, 并取得了优于 SPAM-GIST 方法的实验结果. 孙小兵等人[28]使用深度森林来构建缺陷预测模型 (DPDF) 以预测软件缺陷, 并评估了来自 4 个公共数据集 (即 NASA, PROMISE, AEEEM 和 Relink) 的 25 个开源项目的方法. 实验结果表明, 与最佳传统机器学习算法相比, DPDF 将 AUC 值提高了 5%.

然而, 现有的针对深度学习软件的覆盖指标和测试方法只针对 DNN, 深度森林还缺少有效的测试覆盖指标和测试方法. 本文的研究内容旨在为深度森林研究定义并开发一套有效的测试覆盖评价指标, 以及测试数据自动生成方法. 与之前的相关工作相比, 本文首次提出了针对深度森林的覆盖指标, 并使用覆盖指标指导遗传算法生成测试数据. 本文分析深度森林结构上以随机森林为最小单位、层级直接指导预测等特点, 定义了 *RFNC*、*RFLC*、*CFOC* 和 *CFCC* 这 4 种针对性的覆盖指标, 以用于评估深度森林的测试充分程度. 并利用上述指标指导测试数据生成, 设计用于深度森林的测试数据自动生成方法 DeepRanger. 在此前的工作中, 我们对深度森林的测试覆盖率进行了初步探讨[29], 本文在此基础上对相关覆盖指标进行了改进, 设计了测试数据自动生成算法 DeepRanger 并对覆盖指标和算法的有效性进行了实验验证.

7 结论与展望

深度森林是一种有潜力的深度学习模型, 为了评价深度森林的测试充分性, 本文定义了一组测试覆盖指标, 并在 3 个不同复杂程度的深度森林模型上进行实验, 实验结果表明提高测试集规模可以提高覆盖率, 这证实了所提出的覆盖指标可作为深度森林的测试充分性的有效度量. 为更好地利用所提出的覆盖指标, 本文设计了用于深度森林的测试数据自动生成方法 DeepRanger. DeepRanger 可以根据输入的少量原始测试集自动生成高覆盖率的扩展测试集, 与基于随机选择的遗传算法相比较, 在对深度森林模型的测试中能够更快达到更高的覆盖率.

本文是最早对深度森林进行覆盖制导测试方法研究的工作. 未来, 我们将进一步探究对于深度学习软件的测试方法研究, 进一步完善覆盖指标的准确性和有效性, 进一步改进测试工具的性能和实用性. 希望本文的研究不仅对深度森林的测试工作提供有效的方法, 而且能为深度学习软件的研究提供参考.

References:

[1] Yu K, Jia L, Chen YQ, Xu W. Deep learning: Yesterday, today, and tomorrow. Journal of Computer Research and Development, 2013, 50(9): 1799–1804 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2013.20131180]

[2] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436–444. [doi: 10.1038/nature14539]

[3] Du XN, Xie XF, Li Y, Ma L, Liu Y, Zhao JJ. DeepStellar: Model-based quantitative analysis of stateful deep learning systems. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering.

Tallinn: ACM, 2019. 477–487. [doi: 10.1145/3338906.3338954]

[4] Ziegler C. A Google self-driving car caused a crash for the first time: A bad assumption led to a minor fender-bender. 2016. http://www.theverge.com/2016/2/29/11134344/google-selfdriving-car-crash-report

[5] BBC NEWS. Tesla autopilot crash driver “Was Playing Video Game”. 2020. https://www.bbc.com/news/technology-51645566

[6] 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: ACM, 2018. 120–131. [doi: 10.1145/3238147.3238202]

[7] 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]

[8] Zhou ZH, Feng J. Deep forest: Towards an alternative to deep neural networks. In: Proc. of the 26th Int’l Joint Conf. on Artificial Intelligence. Melbourne: IJCAI, 2017. 3553–3559. [doi: 10.24963/ijcai.2017/497]

[9] Zhou ZH, Feng J. Deep forest. National Science Review, 2019, 6(1): 74–86. [doi: 10.1093/nsr/nwy108]

[10] Zhu H, Hall PAV, May JHR. Software unit test coverage and adequacy. ACM Computing Surveys, 1997, 29(4): 366–427. [doi: 10.1145/267580.267590]

[11] Holland JH. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Cambridge: MIT Press, 1992.

[12] 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]

[13] Zhu R, Wang HM, Feng DW. Trustworthy services selection based on preference recommendation. Ruan Jian Xue Bao/Journal of Software, 2011, 22(5): 852–864 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3804.htm [doi: 10.3724/SP.J.1001.2011.03804]

[14] Witten IH, Frank E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. San Francisco: Morgan Kaufmann Publishers Inc., 2000.

[15] Wang Z, Yan M, Liu S, Chen JJ, Zhang DD, Wu Z, Chen X. Survey on testing of deep neural networks. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1255–1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

[16] Goodfellow I, Papernot N. The challenge of verification and testing of machine learning. 2017. http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html

[17] 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: ACM, 2018. 303–314. [doi: 10.1145/3180155.3180220]

[18] 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. Montpellier: IEEE, 2018. 132–142. [doi: 10.1145/3238147.3238187]

[19] Wang JY, Dong GL, Sun J, Wang XY, Zhang PX. Adversarial sample detection for deep neural network through model mutation testing. In: Proc. of the 41st IEEE/ACM Int’l Conf. on Software Engineering. Montreal: IEEE, 2019. 1245–1256. [doi: 10.1109/ICSE.2019.00126]

[20] Hayhurst KJ. A Practical Tutorial on Modified Condition/Decision Coverage. DIANE Publishing, 2001.

[21] Sun YC, Huang XW, Kroening D, Sharp J, Hill M, Ashmore R. Testing deep neural networks. arXiv:1803.04792, 2018.

[22] 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]

[23] Ma L, Juefei-Xu F, Xue MH, Li B, Li L, Liu Y, Zhao JJ. DeepCT: Tomographic combinatorial testing for deep learning systems. In: Proc. of the 26th IEEE Int’l Conf. on Software Analysis, Evolution and Reengineering. Hangzhou: IEEE, 2019. 614–618. [doi: 10.1109/SANER.2019.8668044]

[24] 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.

[25] 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. Montreal: IEEE, 2019. 1039–1049. [doi: 10.1109/ICSE.2019.00108]

[26] 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. Montreal: IEEE, 2019. 89–92. [doi: 10.1109/ICSE-NIER.2019.00031]

[27] Lu XD, Duan ZM, Qian YK, Zhou W. Malicious code classification method based on deep forest. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1454−1464 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5660.htm [doi: 10.13328/j.cnki.jos.005660]

[28] Zhou TC, Sun XB, Xia X, Li B, Chen X. Improving defect prediction with deep forest. Information and Software Technology, 2019, 114: 204–216. [doi: 10.1016/j.infsof.2019.07.003]

[29] Xie RL, Cui ZQ, Jia MH, Wen Y, Hao BS. Testing coverage criteria for deep forests. In: Proc. of the 6th Int'l Conf. on Dependable Systems and Their Applications. Harbin: IEEE, 2020. 513–514. [doi: 10.1109/DSA.2019.00091]

附中文参考文献:

[1] 余凯, 贾磊, 陈雨强, 徐伟. 深度学习的昨天、今天和明天. 计算机研究与发展, 2013, 50(9): 1799−1804. [doi: 10.7544/issn1000-1239.2013.20131180]

[13] 朱锐, 王怀民, 冯大为. 基于偏好推荐的可信服务选择. 软件学报, 2011, 22(5): 852−864. http://www.jos.org.cn/1000-9825/3804.htm [doi: 10.3724/SP.J.1001.2011.03804]

[15] 王赞, 闫明, 刘爽, 陈俊洁, 张栋迪, 吴卓, 陈翔. 深度神经网络测试研究综述. 软件学报, 2020, 31(5): 1255−1275. http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

[27] 卢喜东, 段哲民, 钱叶魁, 周巍. 一种基于深度森林的恶意代码分类方法. 软件学报, 2020, 31(5): 1454−1464. http://www.jos.org.cn/1000-9825/5660.htm [doi: 10.13328/j.cnki.jos.005660]

崔展齐(1984−), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件测试及分析, 智能软件工程.

刘秀磊(1981−), 男, 博士, 教授, CCF 专业会员, 主要研究领域为语义 Web, 本体匹配, 语义搜索, 语义 Sensor, 知识图谱.

谢瑞麟(1996−), 男, 硕士生, CCF 学生会员, 主要研究领域为智能软件工程.

郑丽伟(1979−), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为需求工程, 群体协同, 大数据挖掘.

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.