基于分治法的神经网络修复方法

孙 朔[1,3], 严 俊[1,2,3], 晏荣杰[2,3]

[1] 中国科学院软件研究所软件工程技术研究开发中心 北京 中国 100190
[2] 计算机科学国家重点实验室 北京 中国 100190
[3] 中国科学院大学 北京 中国 100049

摘要 神经网络作为一种求解复杂问题的有效方法已经广泛应用于医学影像, 自动驾驶等领域。然而, 神经网络十分脆弱, 对一个样本添加一点肉眼难以察觉的微小扰动就可能导致神经网络做出错误的判断。当神经网络出现了错误的行为, 常用的修复方法是对神经网络进行重训练或者微调, 然而这些方式需要较高的代价而且无法保证完全修复错误行为。在本文中, 我们关注神经网络的完备修复问题, 给定一个待修复的神经网络和一个目标样本集合, 该问题要求修复后的神经网络在目标样本集合上表现出 100%的正确率。

在本文中, 我们基于分治法的思想提出了一种神经网络修复方法。在该方法中, 我们将目标样本集合不断划分为更小的集合, 直到样本集合达到可接受的规模, 之后对于划分得到的每一个集合逐个进行修复得到一个局部补丁, 最后所有的局部补丁进行整合得到对于整个特征空间的补丁。在两个公开数据集上的实验表明我们的方法优于当前最先进的神经网络修复算法。针对对抗攻击和后门攻击生成的目标样本集合, 我们的方法不仅完全修复了神经网络在目标样本集合上的行为, 而且将网络在相同攻击方式生成的测试集上的准确率分别提高了 55.79%和 60.59%。同时, 我们的方法可以避免修复后网络在标准测试集上的准确率大幅度降低。

关键词 错误修复; 神经网络; 分治法; 约束求解
中图法分类号 TP183 DOI 号 10.19363/J.cnki.cn10-1380/tn.2023.05.03

A Neural Network Repair Method Based on Divide-and-Conquer

SUN Shuo[1,3], YAN Jun[1,2,3], YAN Rongjie[2,3]

[1] Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[3] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract As an effective method for solving complex problems, the neural network has been widely used in medical imaging, autonomous driving, and other fields. However, neural networks are very fragile, and adding a tiny perturbation to a sample can cause the neural network to make wrong judgments. When the neural network has erroneous behavior, the common repair method is to retrain or fine-tune the neural network, but these methods require high costs and cannot guarantee complete repair of the erroneous behavior.

In this paper, we focus on the problem of the complete repair of neural networks. Given a neural network to be repaired and a target sample set, the problem requires the repaired neural network to exhibit 100% accuracy on the target sample set. In this paper, we propose a neural network repair method based on the idea of divide and conquer. In this method, we continuously divide the target sample set into smaller sets until the sample set reaches an acceptable size and then repair each set obtained by division one by one to obtain a local patch, and finally integrate all the local patches to get a patch for the entire feature space. Experiments on two public datasets demonstrate that our method outperforms current state-of-the-art neural network repair algorithms. For the target sample set generated by the adversarial attack and backdoor attack, our method not only completely repairs the behavior of the neural network on the target sample set but also improves the accuracy of the network on the test set generated by the same attack method by 55.79% and 55.79%, respectively. 60.59%. At the same time, our method can avoid a large reduction in the accuracy of the repaired network on the standard test set.

Key words bug fixing; neural network; divide and conquer; constraint solving

通讯作者: 严俊, 博士, 研究员, Email: yanjun@ios.ac.cn。
本课题得到国家自然科学基金项目(No. 62132020)和中国科学院前沿科学重点研究计划(No. QYZDJSSW-JSC036)资助。
收稿日期: 2022-09-10; 修改日期: 2022-12-28; 定稿日期: 2023-03-23

1 引言

目前, 神经网络算法已经广泛应用于计算机视觉[1], 语音识别[2], 人脸识别[3]等多个领域。在实际生产中, 由于其较高的准确率, 神经网络技术也已经应用于自动驾驶[4], 医学影像[5]等安全相关的领域中。然而, 神经网络算法往往存在缺陷。例如, 面对对抗样本的脆弱性, 即为一张正常的图片添加一个肉眼无法分辨的扰动, 则添加扰动之后的图片可能会使神经网络做出错误的判断[6]。在如 ACAS Xu[7]等应用于安全相关的领域中的神经网络系统上, 这样的预料之外的行为往往会为用户的生命财产安全带来重大的损失。

在本论文中, 我们关注神经网络修复问题, 即给定一个已经训练好的神经网络以及一个输入样本的集合, 通过对神经网络的参数和结构进行局部调整, 使得修复后的网络在集合中样本上均表现正确。同时, 我们希望修复过程对于神经网络行为的影响尽可能小, 使得修复后网络在其他输入样本上表现与修复前一致。

神经网络修复问题的难点主要表现在以下两点: 非线性激活函数带来的问题复杂性以及约束之间可能存在的冲突。具体而言, 神经网络内部的激活函数通常是非线性函数, 这就导致对问题建模时“修复后的网络在集合中样本上均表现正确”需要被编码为一组非线性的约束, 即使待修复网络是最简单的以整流线性单元 (Rectified Linear Unit, ReLU)为激活函数的全连接神经网络, 直接对问题进行建模也会得到一个可满足性模理论(Satisfiability Modulo Theories, SMT)模型, 而该问题的求解是一个 NP 难问题[8]。另一方面, 在对问题进行编码时, 输入样本集合中的每一个样本均对应于模型中的一组约束, 因此集合中的样本数量越多则模型中的约束数量就越多, 由于输入样本集合中的样本可以满足任意分布, 所以随着模型中约束数量增多, 约束之间存在冲突的可能性也随之提高, 冲突的存在会导致求解器无法找到可行解, 因此无法找到可行的修复方案。

基于分治的思想, 我们提出了针对神经网络修复问题的一种新的方法——DACRepair (Divide And Conquer Repair)。Gopinath 等人的工作[8]指出, 对于一个 ReLU 神经网络来说, 如果两个样本在输入神经网络之后具有相同激活模式, 那么这两个样本的性质具有相似性。受该工作的启发, 我们将目标样本集合按照样本的激活模式划分为多个子集合, 每个子集中包含一定数量的相似样本。受 Fu 等人的工作[10]的启发, 分别对每一个子集计算出一个修复补丁之后, 我们可以将各修复补丁限定于其对应的子空间中。因此, 我们将针对整个输入样本集合的修复分解为分别针对其多个子集的小规模修复, 从而降低了问题求解的复杂度。另外, 由于我们的方法将原问题分解为多个子问题, 所以降低了求解时约束间存在冲突的可能性, 而且通过整合子问题的修复补丁, 我们实际得到的解是针对整个输入空间的一个非线性的修复补丁, 因此 DACRepair 可以处理一些传统的神经网络修复算法无法求解的实例。

本文的主要贡献如下:

(1) 提出了一种新的神经网络修复策略, 基于分治的思想降低问题的规模和求解难度, 从而提高了算法的求解效率和可解问题的范围。

(2) 基于上述策略实现了一个神经网络修复工具, 该修复工具可以通过扩展原神经网络的结构为原神经网络添加一个补丁, 使得扩展之后的神经网络在指定样本点集合上的表现均正确。

(3) 在多个公开数据集上进行实验验证 DACRepair 的性能, 并与当前最新的神经网络修复工具进行比较。实验结果显示 DACRepair 在大部分情况下具有更加优越的性能。

2 相关工作

重训练是最常见的修复方法。通过对原本的训练集进行针对性的数据增强, 可以在训练集中添加一定规模的对抗样本或易错样本, 在数据增强之后的训练集上进行重新训练之后, 神经网络会具有更好的鲁棒性[11]。除了数据增强, 研究者们还提出了可编辑训练[13]和输入选择[14]的策略来提高神经网络的表现。但是这种方法对于神经网络的修复是针对对抗样本数据集上准确率的提高, 无法保障已知的样本上的错误行为全部被修复, 除此之外, 当前常用的神经网络训练集往往具有极大的规模, 重训练往往需要极高的代价。与重训练类似的是微调方法。微调的方法仅在小规模样本集合上重训练网络的部分参数, 因此代价更小, 但是微调的方法往往会导致修复之后的神经网络严重遗忘原有知识[15]。

针对神经网络修复问题, Goldberger 等人提出了 MDNN[16], 将问题编码为 SMT 模型, 并使用 SMT 求解器进行求解。但是 SMT 问题本身是 NP 难问题, 因此仅能处理小规模样本集的问题。2021 年, Sotoudeh 和 Thakur 提出了 PRDNN[17], 将 ReLU 神经元的激活情况与输入值解耦, 通过固定网络的激活情况而仅考虑值的变化, 该方法将神经网络修复问题编码为

线性规划问题。通过线性规划求解器进行求解。但是仅该方法在修复之后无法将解耦的神经元重新耦合成原本的 ReLU 神经元, 因此无法复原为神经网络。另外, 由于该方法对于输入样本集合中的每一个样本均需要计算并保存网络的激活模式和输出层的梯度, 因此该方法在处理大规模样本时需要较多的时间和空间代价。2022 年, Fu 等人提出了 REASSURE[10], 这是一种局部修复的方法, 通过将修复补丁限定在输入空间的一个局部区域内, 该方法保障了修复后的网络对于空间内其他区域的样本保持原本行为不变。但是该方法在原神经网络中添加了大量神经元, 且新增神经元数量与输入样本集合中样本数成正比, 因此难以处理大规模输入样本集合。

神经网络修复问题要求修复后的网络在目标样本集合中的所有样本上均表现正确, 但在某些场景中, 我们需要的是对于某种特定分布的未知样本上准确率提高。Usman 等人提出了 NNRepair[18], 该方法利用神经网络训练集中的被正确分类的样本来收集网络将样本正确分类时表现出的激活模式, 并将这种模式作为 Oracle 来调整网络在被错误分类的样本上的表现。该方法无法保障已知的错误分类样本均能在修复后被正确分类, 但是在实验中修复后的网络在相同分布的未知样本上的准确率提高。

3 预备知识与符号定义

在本节中, 我们首先介绍了神经网络的基本知识, 之后对神经网络修复问题进行了定义。

3.1 预备知识

在本文中, 我们关注神经网络模型, 每个神经网络都可以被视为一个函数, 它接受一个输入样本并输出样本的标签。一个神经网络包括一个输入层、多个隐藏层和一个输出层, 每层包含一组神经元。除输入层外, 在每一层中, 每个节点都与上一层的神经元相连接。 连接到节点的每条边都分配有权重 w。 同时, 每个神经元都被分配了一个偏置 b。 给定一个输入 x, 在第 i 层的输出的特征向量为 $FV(x,i)=\sigma(W_i\cdot FV(x,i-1)+B_i)$。其中$W_i$为连接第$i$层和第$i-1$层的权重矩阵, B_i是第i 层中神经元的权重向量, $FV(x,0)=x$, σ通常是一个非线性的激活函数(例如, 整流线性单元 (ReLU), sigmoid 函数或双曲线切线函数(Tanh))。神经网络的输出层通常是一个线性分类器, 它输出将样本划分为各类别的置信度。 因此, 网络的输出是最高置信度对应的标签。

3.2 问题定义

基于上述关于神经网络的基础知识, 我们可以定义神经网络修复问题:

定义 1.(神经网络修复问题). 给定一个待修复的神经网络 N 以及一个目标样本集合 $S=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$, 其中$x_i$为输入样本, y_i为x_i对应的正确标签。神经网络修复问题是通过调整 N 的参数和结构来得到一个新的神经网络N', 使得对于 S 中的任意样本(x_i,y_i), 满足$N'(x_i)=y_i$。

值得注意的是, 在实际求解的过程中, 我们不仅要求修复后的神经网络N'在 S 中表现出 100%的准确率, 而且希望N'在其他样本上的行为能够尽可能与 N 一致, 即N'在标准测试集上的准确率相比于 N 不要下降太多。

4 算法介绍

在本节中, 我们介绍一种基于分治思路的神经网络修复方法——DACRepair。我们首先简述方法的整体流程, 之后分别介绍样本集合划分, 子问题求解和结果整合三个主要模块。

4.1 算法概述

我们的修复方法的具体流程如图 1 所示。整个流程由三个模块构成: (1)样本集合划分, (2)子问题求解和(3)结果整合。首先, 根据样本在待修复的神经网络上的表现, 我们将目标样本集合划分为多个子集。之后, 我们将划分得到的每一个样本集合作为目标样本集合, 与待修复的神经网络 N 构成规模更小的神经网络修复问题, 并对这些子问题依次进行求解。最后, 我们将求解所有子问题得到的补丁进行整合, 得到一个针对原样本集合的修复补丁, 将补丁添加到待修复网络中之后, 我们获得了修复后的神经网络, 该网络在样本集合中的样本上均能正确表现。

4.2 样本集合分解

样本集合划分模块的目标是将原问题中规模较大的目标样本集合划分为多个规模较小的样本集合, 从而提高求解效率。

根据文献[8,18], 对于一个 ReLU 神经网络来说, 输入网络之后具有相同激活模式的样本之间具有相似性, 因此, 我们以样本输入之后神经网络的激活模式作为样本集合划分的依据。基于神经网络中的激活模式, 我们可以建立一棵决策树, 决策树的每个内部节点代表的决策属性为网络中某一个神经元的激活情况, 每个叶节点表示一个子问题的目标样本集合。

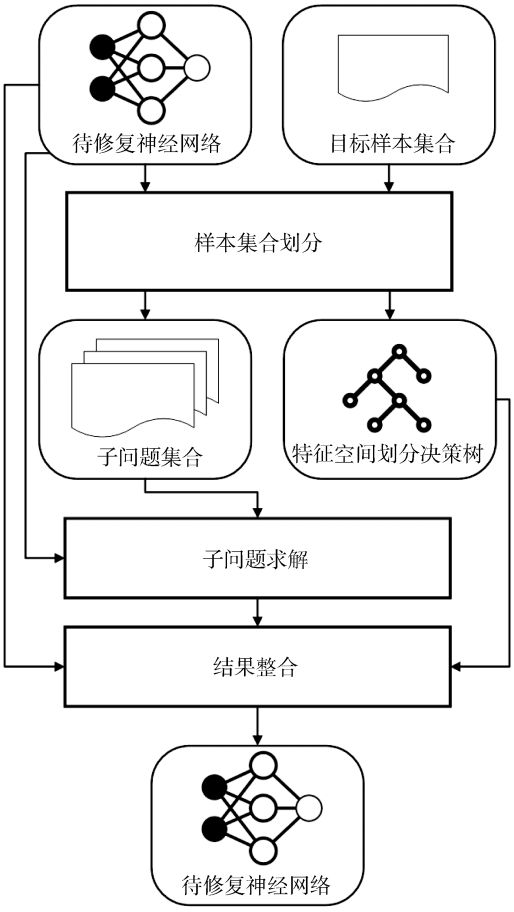

图 1　算法流程

Figure 1　Workflow of our method

选待修复神经网络中的某一层作为待修复层, 令可接受的样本规模为$size_{allowed}$, 则决策树的建立流程如图 2 所示。其中 SC1 和 SC2 分别表示两个终止条件: 当前样本集合的规模小于$size_{allowed}$; 当前集合中所有样本的激活模式相同。决策树以代表整个目标样本集合的根节点开始。如果当前样本集合的规模已经小于$size_{allowed}$, 则该节点成为叶节点。否则, 将集合中的所有样本输入待修复的神经网络中, 统计各个样本输入之后神经网络的激活模式并计算神经网络中各个节点的激活频率。设样本集合规模为 $size$, 设神经网络中节点 i 在集合中所有样本输入之后表现出了t_i次激活, 则节点 i 的激活频率为$\frac{t_i}{size}$。由于我们希望被划分到相同叶节点中的样本具有相似的激活模式, 所以我们选择待修复层中单个节点的激活情况作为决策属性。同时, 由于划分后样本集合的数量决定了子问题的数量, 进而决定了算法的效率, 所以我们希望学习出的决策树是一棵尽量平衡的二叉树, 因此每个节点分裂时两个分支中的样本规模要尽量相近。所以, 我们选择待修复层中激活频率最接近 0.5 的节点的激活情况作为决策属性。如果该集合中所有样本输入神经网络之后待修复层的激活模式相同, 则将当且节点设置为叶节点。否则, 根据样本输入神经网络后该节点的激活情况, 我们可以将目标样本集合相对均匀地划分为两个子集。分别以这两个子集为节点再次重复以上过程, 直到算法终止, 返回一个决策树。

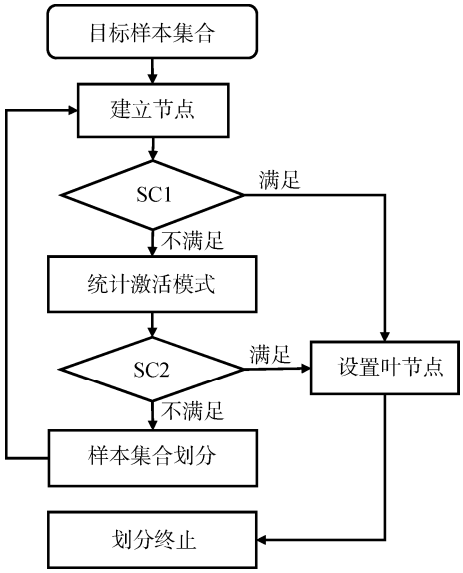

图 2　决策树建立流程

Figure 2　Workflow of building the decision tree

值得注意的是, 在通过决策树的方法对目标样本集合进行划分的过程中, 实际上神经网络待修复层的整个特征空间也被相应的进行了划分。决策树的每一个内部节点的决策属性可以视为特征空间中的一个超平面, 这个超平面将样本集合划分为两个更小的集合, 同时也将整个特征空间划分为两个子空间。由于我们选择激活频率最接近 50%的节点的激活情况作为决策属性, 所以最终得到的决策树必然是一棵完全二叉树。另外, 对于决策树中任意一条从根节点到叶子节点的路径, 路径中包含的所有的决策属性对应的神经元均不同。因此, 决策树的叶节点对应的子空间的集合是完整的特征空间的一个划分。

4.3　子问题求解

在将目标样本集合划分为多个规模较小的集合之后, 我们依次对每个集合对应的子问题进行求解。

子问题的求解思路为将整个问题转化为一个线性规划问题。在每一个子问题中, 我们通过为待修复网络的输出层输出的置信度添加一个修正值来构成一个输出正确的新的输出层, 从而对网络进行修复。

给定一个待修复的神经网络 N 以及一个样本集

合$S=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$, 我们通过添加神经元的方式来构造修正值$Q(x_i)$。构造一层新的隐藏层作为修正层, 修正层中神经元的数量与输出层相同。以待修复网络中的某一层作为待修复层, 待修复层与修正层相连接, 修正层基于待修复层的输出计算出修正值$Q(x_i)$。设待修复层输出的特征向量为$FV(x_i)$, 则:

$$Q(x_i)=FV(x_i)\cdot W_p+B_p \tag{1}$$

其中W_p和B_p为添加到神经网络中的修正层的权重和偏置。

设样本x_i输入 N 之后 N 的输出层输出的置信度为$C(x_i)$, 则修复后的神经网络输出的置信度$C_p(x_i)$为原网络输出的置信度与修正层输出之和, 即:

$$C_p(x_i)=Q(x_i)+C(x_i) \tag{2}$$

修复后的网络输出的置信度需要满足正确性约束, 即正确类别对应的神经元输出的置信度最高, 即:

$$\bigwedge_{i\in\{1,2,\cdots,n\}} argmax\left(C_p(x_i)\right)=y_i \tag{3}$$

因此, 对于子问题中的目标样本集合, 我们为线性模型中添加如公式(3)的线性约束作为W_p和B_p需要满足的约束。

除此之外, 我们希望修复后的神经网络能够与原神经网络尽可能相似, 因此, 我们将线性模型的目标函数设置为修复后网络在 S 中样本点上的置信度与原网络的距离最小。由于样本点的数量较多, 目标函数需要考虑的点通常具有很高的维度, 在高维空间中欧几里得距离无法准确地表征两个点之间的距离, 所以我们使用无穷范数作为距离的度量, 因此目标函数设置为:

$$\text{Min}\left(\left|C_p(x_i)-C(x_i)\right|_\infty\right) \tag{4}$$

4.4 结果整合

在求出每一个子问题的最优解之后, 我们需要将这些子问题的解整合成为原问题的解。如 4.2 中所述, 决策树中的每一个叶节点都对应 4.3 中的某个子问题, 同时也对应特征空间中的某个子区域。因此, 我们可以利用文献[10]中的思路, 将每一个子问题的修复补丁限定在对应的子区域中。同时, 又因为所有子区域的集合是整个特征空间的一个划分, 所以我们可以将所有子区域的补丁整合得到对于整个特征空间的修复补丁。

对于决策树上的任意一个叶节点 L, 设其到根节点的距离为 d, 路径上的决策属性取值为激活的节点集合为 A, 路径上决策属性取值为非激活的节点集合为 I, 则我们可以为这个叶节点建立一个支撑网络用于表征输入样本是否属于该叶节点对应的特征空间:

$$g_L(fv,\epsilon)=ReLU\left(\begin{matrix}\sum_{i\in I}\left(\sigma(fv_i,\epsilon)\right)\\+\sum_{i\in A}\left(1-\sigma(fv_i,\epsilon)\right)\\-d+1\end{matrix}\right) \tag{5}$$

其中:

$$\sigma(fv_i,\epsilon)=ReLU(-\epsilon fv_i+1)-ReLU(-\epsilon fv_i) \tag{6}$$

其中fv为该支撑网络的输入, 即待修复层输出的特征向量; fv_i为特征向量的第 i 个分量, 即待修复层中第 i 个神经元的输出; ϵ为一个超参数, 用于控制支持网络的边界宽度。根据 Fu 的结果[10], 超参数ϵ 足够小时, 其数值变化对于补丁的精度影响不大, 在 DACRepair 中我们将其固定为 1e-4。

$\sigma(fv_i,\epsilon)$表示待修复层中第 i 个神经元的激活情况, 若该神经元激活, 则$\sigma(fv_i,\epsilon)=1$, 否则$\sigma(fv_i,\epsilon)=0$。$g_L(fv,\epsilon)$表示该样本是否在叶节点 L 对应的子空间中。当样本在叶节点 L 对应的子空间中时, $g_L(fv,\epsilon)$的输出值为 1, 否则, $g_L(fv,\epsilon)$的输出值为 0。由于该函数为连续函数, 所以在子空间的边界处存在一部分区域其取值为 0-1 之间的实数, 该区域的面积可以通过ϵ为的取值来进行控制。

基于上述的支持网络, 我们可以将所有子空间中的补丁整合为整个特征空间的补丁。设决策树的所有叶节点为$L_1,L_2,\cdots,L_c$, 对于叶节点L_i, 其对应的子空间中的补丁为$Q_{L_i}(x)$, 支持网络为$g_{L_j}(x,\epsilon)$, 则最后得到的针对整个特征空间的全局补丁为:

$$h(x,\epsilon)=\sum_i\left(\begin{matrix}ReLU\left(\begin{matrix}Q_{L_i}(x)-Q_{L_{i-1}}(x)\\+\varphi_i(x,\epsilon)\end{matrix}\right)\\-ReLU\left(\begin{matrix}-Q_{L_i}(x)+Q_{L_{i-1}}(x)\\+\varphi_i(x,\epsilon)\end{matrix}\right)\end{matrix}\right) \tag{7}$$

其中:

$$\varphi_i(x,\epsilon)=U\cdot\left(\sum_{j\geqslant i}g_{L_j}(x,\epsilon)-1\right) \tag{8}$$

U 是一个足够大的正数, $Q_{L_0}(x)=0$。

最后, 我们将全局补丁的输出值加到待修复网络的输出层的输出值中, 即可得到调整之后满足要求的输出值。

值得注意的是, 各个子空间中的补丁和支持网络以及最后整合全局结果的过程全部由仿射变换和 ReLU 函数的叠加构成。因此, 上述过程形成的全局补丁可以作为一个拓展部分添加到待修复的神经网络中。拓展之后的神经网络仍可以在 Keras, PyTorch 等主流的神经网络框架运行, 并且保持了 ReLU 神经网络的分段线性的性质。

5 实验验证

在本节中, 我们考虑下述研究问题(Research

Question, RQ)来验证 DACRepair 的有效性:

RQ1. DACRepair 能否有效的求解神经网络修复问题?

RQ2. 样本划分过程中设置不同的$size_{allowed}$会带来了什么影响?

RQ3. 选择不同隐藏层作为修复层会对求解效果带来什么影响?

为了回答 RQ1, 我们在四个不同的数据集上进行了实验, 并与当前最新的神经网络修复工具 PRDNN 和 NNRepair 进行了比较。为了回答 RQ2, 我们设置了不同的$size_{allowed}$作为决策树模块的终止条件, 研究了样本集合划分的过程对于问题求解带来的影响。为了回答 RQ3, 我们在求解时将网络中的多个隐藏层作为待修复层进行实验, 通过对比实验结果研究了针对不同隐藏层时修复效果的不同。

5.1 实验设置

为了回答 RQ1, 我们将 DACRepair 与相关工作中提到的神经网络修复方法进行比较, 我们使用了 NNRepair 论文中提供的数据集和神经网络作为目标样本集合和待修复网络。PRDNN 的实验均基于规模较小的数据集, 平均规模均在 700-1,000, 不适合本文方法的适用场景, 因此未用于本次实验中。

NNRepair 提供的四个数据集基于两个公开数据集 MNIST[19]和 CIFAR10[20]以及两种常见的攻击方式 FGSM 对抗攻击[21]和后门攻击[22]。两个数据集和两种攻击方式组合形成了四个目标样本集合, 分别命名为 $MNIST_{ADV}$, $CIFAR10_{ADV}$, $MNIST_{POIS}$ 和 $CIFAR10_{POIS}$。对应于每一个目标样本集合, NNRepair 的论文中提供了一个待修复的神经网络, 这些神经网络的结构如表 1 所示。

$MNIST_{POIS}$ 和 $MNIST_{ADV}$ 具有相同的网络结构, 在表 1 中表示为 MNIST 列; $CIFAR10_{ADV}$ 和 $CIFAR10_{POIS}$ 具有相同的网络结构, 在表 1 中表示为 CIFAR10 列。表 1 中 Conv 表示卷积层, Maxpooling 表示最大池化层, Dense 表示全连接层。神经网络具体的训练参数如学习率, 训练轮次, 优化算法等由于 NNRepair 并未提供, 所以未展示在表 1 中。

表 1 实验中神经网络的结构

Table 1 Structure of neural networks in experiments

隐藏层类型	MNIST	CIFAR10
Conv+ReLU	3x3x2x1	3x3x3x32
Conv+ReLU	3x3x4x2	3x3x32x32
Maxpooling	2x2	2x2
Conv+ReLU	—	3x3x32x64
Conv+ReLU	—	3x3x64x64
Maxpooling	—	2x2
Dense+ReLU	128	512
Dense+Softmax	10	10

四个待修复网络的在目标样本集合, 标准测试集以及与目标样本集合相同攻击方式生成的测试集合上的准确率表现如表 2 所示。

表 2 四个待修复神经网络的准确率

Table 2 The accuracy of the four neural networks to be repaired

	Size	TS (%)	ST (%)	GT (%)
$MNIST_{ADV}$	10,000	29.92	97.87	28.37
$CIFAR10_{ADV}$	50,000	34.39	81.04	35.96
$MNIST_{POIS}$	60,000	98.99	98.63	10.38
$CIFAR10_{POIS}$	50,000	96.97	72.26	15.89

在表 2 中, size 列表示目标样本集合的规模, TS 列, ST 列和 GT 列分别表示待修复网络在目标样本集合, 标准测试集以及相同攻击方式成成的测试集上的准确率。 可以发现, 这些神经网络在标准测试集上均表现出了较高的准确率。但在与目标集合中样本相同分布的数据集中这些神经网络均表现出了很低的准确率。在对抗攻击对应的两个目标样本集合中, 神经网络表现出了很低的准确率。而在后门攻击对应的两个目标样本集合中, 神经网络准确率较高。这是因为后门攻击往往通过对训练集数据进行“投毒”的方式进行。即在标准的训练集中添加少量的添加了后门的数据, 从而使得训练得到的网络存在可以利用的后门。这里目标样本集合模仿了场景中被投毒的训练集, 所以神经网络表现出了较高的准确率。

为了回答 RQ2, 我们在 $MNIST_{ADV}$ 数据集和 $CIFAR10_{ADV}$ 数据集上使用 100, 200, 300, 500, 1,000, 1,500, 2,000, 3,000, 5,000 作为$size_{allowed}$的值进行了实验。

为了回答 RQ3, 我们在四个数据集上分别将最后一层隐藏层和倒数第二层隐藏层作为待修复层进行了实验。

所有的实验均运行一台 CPU 为 Intel E5-2680 v4 @ 2.40GHz, 内存 64G 的服务器。我们使用 Gurobi[23]作为求解器来求解实验中的线性规划问题。

我们将 DACRepair 与相关工作中介绍的神经网络修复方法 PRDNN 和 NNRepair 进行了比较。MDNN 和 REASSURE 并没有提供可以执行的代码, 因此没有办法进行比较。PRDNN 难以处理大规模的

神经网络修复问题, 在我们的实验中未给出可比较的结果。NNRepair 虽然没有提供可以执行的代码, 但是提供了实验数据以及实验结果, 因此我们可以在其提供的实验数据上进行实验, 并与其论文中的实验结果进行比较。

DACRepair, PRDNN 和 NNRepair 的方法在求解时均需要指定待修复神经网络中的某一层全连接层作为待修复层, 实验用网络中的最后一层和倒数第二层隐藏层为全连接层, 因此在实验中我们分别将这两层作为待修复层进行修复。由于 NNRepair 没有提供可运行代码, 因此我们展示在其论文中的表现最好的结果与 DACRepair 进行比较。关于超参数对于 DACRepair 性能的影响, 我们将在在 5.2.2 和 5.2.3 中进行研究。

5.2 实验结果与分析

根据实验结果, 我们可以对三个研究问题进行回答并对 DACRepair 的有效性进行验证。

5.2.1 RQ1 算法的求解能力

DACRepair 和 NNRepair 在四个数据集上的求解效果如表 3 所示。表中括号里的–1 和–2 分别表示将最后一层隐藏层和倒数第二层隐藏层作为待修复层。在每一种方法对应的三列数据中, E 列表示修复后的神经网络在目标样本集合上的准确率。D 列表示修复后的神经网络在标准测试集上准确率的降低。G 列表示修复后的神经网络在与目标样本集合中数据具有相同分布的其他样本上准确率的提升。NNRepair 和 DACRepair 中均包含多个超参数, 表中仅展示了表现最好的超参数下的求解效果。DACRepair 中主要的超参数为$size_{allowed}$的设置与待修复层的选择, 对于这两个超参数的影响最佳选择, 我们将在 RQ2 和 RQ3 中进行讨论。

表 3 DACRepair 和 NNRepair 的修复效果比较
Table 3 Comparison of repair results of DACRepair and NNRepair

	Method	E (%)	D (%)	G (%)
$MNIST_{ADV}$	DACRepair(–1)	100	17.02	38.27
	DACRepair(–2)	100	**2.05**	**55.79**
	NNRepair	32.67	3.14	10.4
$CIFAR10_{ADV}$	DACRepair(–1)	100	12.57	**1.46**
	DACRepair(–2)	100	6.08	1.19
	NNRepair	34.76	**0.27**	0.27
$MNIST_{POIS}$	DACRepair(–1)	100	0.07	50.79
	DACRepair(–2)	100	**0.01**	**60.59**
	NNRepair	96.36	3.11	45.56
$CIFAR10_{POIS}$	DACRepair(–1)	100	0.94	**24.39**
	DACRepair(–2)	100	**0.57**	3.79
	NNRepair	96.08	0.61	3.77

如第二节中所述, PRDNN 难以处理大规模的神经网络修复问题。在我们的实验中, PRDNN 在将倒数第二层隐藏层作为修复层时在四个问题的求解中均出现了内存溢出的问题。其中 PRDNN 与 DACRepair 在求解 $MNIST_{ADV}$ 和 $CIFAR10_{ADV}$ 时各自的内存消耗随问题规模的变化如图 3 所示。

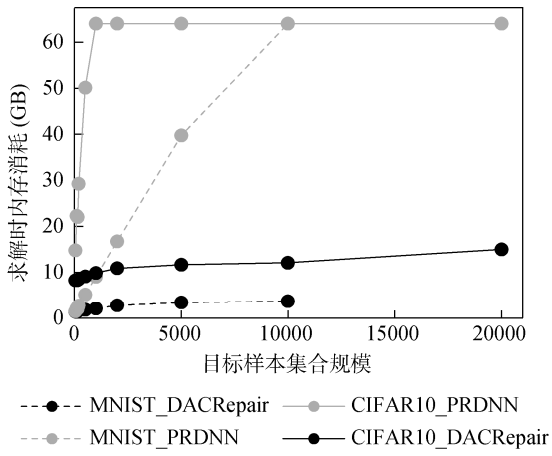

图 3 PRDNN 与 DACRepair 内存消耗的变化比较
Figure 3 Comparison of Change of memory consuming of PRDNN and DACRepair

观察图 3 可以发现, 以倒数第二层隐藏层作为修复层时, PRDNN 在求解时消耗的内存与目标数据集的规模成正比, 在目标数据集规模变大时内存消耗迅速提高, 因此对于实验中的问题无法求解。在将最后一层隐藏层作为修复层时, PRDNN 在求解其中三个问题时出现了无解的情况。仅在求解 $MNIST_{POIS}$ 的时候得到了一个可行解。因此 PRDNN 的实验结果未被展示在表 2 中。

观察表 2 中的数据可以发现, 相比于 NNRepair, DACRepair 在三个问题的求解中得到了比 NNRepair 更好的结果。DACRepair 不仅完全修复了神经网络在目标样本集合上的行为, 而且对于标准测试集上的准确率降低更少, 且在相同分布的数据集上准确率提高更多。由于利用了训练集中的正样本, 所以 NNRepair 能够更加有效的控制修复后网络在标准数据集上的准确率损失。在 $CIFAR10_{ADV}$ 数据集上, 虽然 NNRepair 的修复带来的标准测试集上准确率的下降更小, 但是 NNRepair 仅修复了目标样本集合中三分之一的样本, 且修复后网络在各测试集上的准确率变化均较小(小于 0.5%)。

综合以上观察, 我们可以回答 RQ1。DACRepair 可以有效的求解较大规模的神经网络修复问题。相

比于现有工作, DACRepair 可以处理当前的神经网络修复工具无法处理的一些较大规模的目标样本集合, 修复后的神经网络能够在目标样本集合上表现出 100% 的准确率。与此同时, 在大部分情况下 DACRepair 修复后的网络在标准测试集上的精度损失较小, 且在与目标样本集合中样本相同分布的其他样本上准确率提高较大。

5.2.2 RQ2 子问题规模的影响

在 $\text{MNIST}_{\text{ADV}}$ 和 $\text{CIFAR10}_{\text{ADV}}$ 中设置不同的 $size_{allowed}$, 实验结果如图 4、图 5 所示。

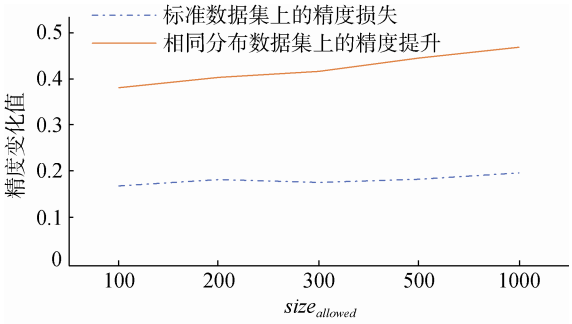

图 4 $\text{MNIST}_{\text{ADV}}$ 实验中不同$size_{allowed}$取值的影响

Figure 4 Influence of Different Value of $size_{allowed}$ in $\text{MNIST}_{\text{ADV}}$ Experiment

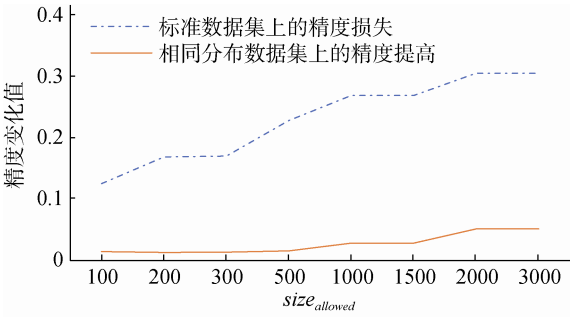

图 5 $\text{CIFAR10}_{\text{ADV}}$ 实验中不同$size_{allowed}$取值的影响

Figure 5 Influence of Different Value of $size_{allowed}$ in $\text{CIFAR10}_{\text{ADV}}$ Experiment

从图中可以看出, 随着$size_{allowed}$取值变大, 修复后神经网络在标准数据集上的精度损失和在相同分布数据集上的精度提升均呈上升趋势。这是因为, 随着$size_{allowed}$增大, 每一个子问题中包含的负样本数量随之增加, 同时, 每一个补丁作用到的空间也随之增大, 然而线性规划问题中的变量数并没有增加, 这也就意味着补丁的拟合能力并没有变化, 所以, 随着子问题中样本点数量的增加和补丁作用范围的扩大, 线性规划得到的补丁不可避免地造成更大的副作用, 但同时针对于相同分布的数据的泛化能力也随之提高。因此, 在补丁的泛化能力和副作用之间存在权衡。更大的样本集合规模会带来更好的泛化能力, 但也会导致修复后的网络在标准数据集上准确率下降更加严重。

值得注意的一点是, 我们在实验时设置了 $size_{allowed}$ 的变化范围为 100 到 5000, 但当 $size_{allowed}$ 被设置的过大时(在 $\text{MNSIT}_{\text{ADV}}$ 实验中大于 1000, 在 $\text{CIFAR10}_{\text{ADV}}$ 中大于 3000), 分解得到的子问题中会存在线性规划问题无解的情况。在每个子问题求解过程中, 算法通过生成一个线性补丁来调整神经网络。但是由于线性补丁中的变量数有限, 所以其拟合能力也存在限制, 当需要修复的样本点数量超过了线性补丁的拟合能力, 就会出现子问题无解的情况。

综合以上观察分析, 我们可以回答 RQ2。由于每个子问题生成的线性补丁的拟合能力相对固定, 所以当子问题规模变大时, 获得的线性补丁会具有更好的泛化能力, 但同时也会导致修复后网络在标准数据集上的准确率下降更加严重。另一方面, 由于线性补丁的拟合能力有限, 所以当子问题的规模超过了线性补丁可以接受的最大规模时, 子问题求解会遇到无解的情况。

5.2.3 RQ3 不同修复层的影响

为了回答问题 3, 我们在四个数据集上分别以最后一层隐藏层和倒数第二层隐藏层作为修复层进行了实验, 比较其求解时间和修复后神经网络的准确率变化。实验结果如表 4 所示。

表 4 不同修复层对修复算法的影响

Table 4 The Influence of different repaired layers on the repair algorithm

	L	T (s)	D (%)	G (%)
$\text{MNIST}_{\text{ADV}}$	–1	**56.99**	17.02	38.27
	–2	202.91	**2.05**	**55.79**
$\text{CIFAR10}_{\text{POIS}}$	–1	**2207.37**	12.57	**1.46**
	–2	7789.79	**6.08**	1.19
$\text{MNIST}_{\text{ADV}}$	–1	5995.17	0.07	50.79
	–2	**230,01**	**0.01**	**60.59**
$\text{CIFAR10}_{\text{POIS}}$	–1	**6349.77**	0.94	**24.39**
	–2	13472.65	**0.57**	3.79

其中 L 列表示神经网络中的修复层, –1 表示最后一层隐藏层, –2 表示倒数第二层隐藏层。T 列表示求解时间, D 列表示修复后的神经网络在标准测试集上准确率的降低。G 列表示修复后的神经网络在与目标样本集合中数据具有相同分布的其他样本上准确率的提升。观察表中数据可以发现, 在大部分情况下,

以倒数第二层隐藏层作为修复层进行修复之后得到的神经网络在标准数据集上的精度损失更小, 而且在相同分布的数据集上的准确率提高更大。但是相对的, 以倒数第二层隐藏层作为修复层需要更多的求解时间。

这样的结果与待修复的神经网络的结构有关。通常情况下, 对于一个神经网络分类器来说, 更接近输出层的隐藏层输出的特征向量往往代表的抽象层次更高, 包含的信息更少, 后面的隐藏层也往往具有更少的神经元。在本次实验的四个待修复神经网络中, 倒数第二层隐藏层中包含的神经元数量均比最后一层隐藏层中更多, 这就意味着在子问题求解时以倒数第二层隐藏层作为修复层建立的线性模型中具有更多的变量, 所以得到的线性补丁就具有了更强的拟合能力, 因此最终得到的修复后的神经网络具有更好的表现。但是更多的变量也意味着子问题中线性模型的求解具有更高的复杂度, 会带来更多的时间和空间的负担。若选择的修复层中神经元数量过多, 那么就会导致线性规划问题的复杂度过高, 从而可能会引起超时或者内存溢出的问题。

为了孤立地研究修复层的位置对于修复效果的影响, 我们重新训练了三个 toy 级别的 MNIST 神经网络, 并针对对抗样本进行了修复。

三个神经网络均为全连接神经网络, 均包含 3 个隐藏层, 每个神经网络内部各隐藏层中神经元数量相同, 分别为 64, 128 和 256。三个神经网络根据隐藏层中神经元数量分别命名为 $MNIST_{64}$, $MNIST_{128}$ 和 $MNIST_{256}$, 在数据集上的准确率信息如表 5 所示。

表 5 四个待修复神经网络的准确率

Table 5 The accuracy of the four neural networks to be repaired

	Size	TS (%)	ST (%)	GT (%)
$MNIST_{64}$	10,000	34.38	97.45	32.15
$MNIST_{128}$	10,000	44.29	97,39	43.04
$MNIST_{256}$	10,000	54.16	97.63	52.80

表 5 中 Size 列表示目标样本集合的规模, TS 列, ST 列和 GT 列分别表示待修复网络在目标样本集合, 标准测试集以及相同攻击方式成成的测试集上的准确率。实验中修复后三个神经网络在目标样本集合上准确率均达到了 100%, 其他的详细信息如表 6 所示。

其中 L 列表示神经网络中的修复层, –1 表示最后一层隐藏层, –2 表示倒数第二层隐藏层, –3 表示第一层隐藏层。T 列表示求解时间, D 列表示修复后的神经网络在标准测试集上准确率的降低。G 列表示修复后的神经网络在与目标样本集合中数据具有相同分布的其他样本上准确率的提升。

表 6 修复层位置对修复效果的影响

Table 6 The Influence of different location of repaired layers on the repair algorithm

	L	T (s)	D (%)	G (%)
$MNIST_{64}$	–1	33.06	58.53	18.85
	–2	34.22	37.83	20.35
	–3	33.54	**8.67**	**23.80**
$MNIST_{128}$	–1	53.94	50.06	15.00
	–2	54.05	33.43	15.61
	–3	53.75	**7.73**	**20.70**
$MNIST_{256}$	–1	102.72	54.21	12.95
	–2	96.24	41.23	13.94
	–3	92.84	**7.51**	**15.94**

观察表 4 中数据可以发现, 在神经元数量相同的情况下, 选择的待修复层位置越靠近输入层, 修复后的神经网络性能越好。在三个实验网络中, 以第一层隐藏层作为修复层均可以得到最好的修复效果, 修复后的神经网络在标准测试集上的准确率损失最低, 而且在在与目标样本集合中数据具有相同分布的其他样本上准确率提升最高。由于神经网络的不可解释性, 我们给出一个可能的解释为待修复层越接近输入层, 其输出的特征向量的抽象层次就越低, 信息损失也越少, 因而修复算法在计算时可以利用的信息就越多, 修复效果就越好。

除此之外, 比较三个网络的修复时间我们可以发现, 待修复层的位置对于修复时间的影响不大, 但是修复层神经元数量对于修复时间有明显的影响。随着修复层神经元数量的增大, 修复时构建的线性规划模型中的变量数增大, 进而导致问题求解的用时提高。

综合上述分析, 我们可以回答 RQ3。待修复层会从位置和神经元数量两个方面影响修复效果。待修复层的位置越接近输入层, 则特征向量的抽象层次越低, 修复后网络在标准数据集和相同攻击方式的其他样本集合上的表现就会越好。待修复层的神经元越多, 则各子问题中得到的线性补丁的拟合能力就越强, 进而修复后神经网络的表现就越好。同时, 选择神经元多的隐藏层作为修复层会导致问题求解的复杂度提高, 带来更高的时间和空间代价, 修复层中的神经元过多也可能会导致求解过程超时或内存溢出。

综合 RQ2 和 RQ3 的结论, 我们可以发现, 超参数的选择对于修复过程的影响存在一个权衡。具体影响如表 7 所示。

表 7　超参数对于 DACRepair 的影响

Table 7　Effect of hyperparameters on DACRepair

	T	D	G
$size_{allowed}$	—	正相关	正相关
$location_{repaired}$	—	正相关	负相关
$size_{repaired}$	正相关	负相关	—

表中的 T 列表示求解时间, D 列表示修复后的神经网络在标准测试集上准确率的降低。G 列表示修复后的神经网络在与目标样本集合中数据具有相同分布的其他样本上准确率的提升。$location_{repaired}$ 和 $size_{repaired}$ 分别表示待修复层的位置和神经元数量, $location_{repaired}$ 行中正相关表示距离输入层越远数值越大, 负相关表示距离输入层越远数值越小。综合上述影响进行分析, 待修复层的选择需要考虑其位置和神经元数量, 推荐的选择是在计算资源可以接受的程度内选择神经元多且靠近输入层的隐藏层作为待修复层。$size_{allowed}$ 的选择需要考虑到待修复层的神经元数量, 因为待修复的神经元数量决定了线性规划模型的变量数, 进而影响了子问题可以求解的规模。实验中 $size_{allowed}$ 经验性的推荐值为二倍的线性模型中变量数。

6　结论

在本文中, 我们提出了一种基于分治思想的神经网络修复算法。为了提高算法可以求解的问题规模, 我们通过将目标样本集合不断划分为更小的集合, 我们将原本的大规模问题划分为多个小的子问题进行求解。基于 Fu 等人的思路, 我们将求解各个子问题得到的线性补丁整合成为针对整个特征空间的非线性补丁, 同时保障了修复后的神经网络仍然保持神经网络的基本结构以及 ReLU 神经网络的分段线性性质。我们与目前最先进的神经网络修复算法进行了比较, 实验结果证明 DACRepair 能够处理现有算法无法处理的大规模问题, 而且修复后神经网络的性能优于以提高网络性能为目标的修复算法修复后的神经网络。另外, 该方法未来可以与 Sun 等人的工作 RIPPLE[24]结合, 改进 RIPPLE 因为目标样本集合规模过大导致的求解效率较低的问题。

但是 DACRepair 仍然存在局限性。首先, 超参数的选择方面, 不恰当的超参数选择不仅可能影响算法的求解效率, 甚至可能导致算法超时, 内存溢出甚至无解, 但是子问题中目标样本集合的 $size_{allowed}$ 的取值以及待修复神经网络中修复层的选择目前仍然无法自动实现。另外, 在划分样本集合的过程中, 决策树的建立过程仅利用了样本的激活模式的信息, 没有充分利用样本的空间位置, 正确类别等信息, 因此划分的过程缺乏理论保障。在未来的工作中, 克服上述的局限性将是我们工作的方向。

参考文献

[1] Krizhevsky A, Sutskever I, Hinton G E. ImageNet Classification with Deep Convolutional Neural Networks[J]. *Communications of the ACM*, 2017, 60(6): 84-90.

[2] Yuan Z L, Lu Y Q, Wang Z G, et al. Droid-Sec: Deep Learning in Android Malware Detection[C]. *The 2014 ACM conference on SIGCOMM*, 2014: 371-372.

[3] Schroff F, Kalenichenko D, Philbin J. FaceNet: A Unified Embedding for Face Recognition and Clustering[C]. *2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015: 815-823.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, et al. End to end learning for self-driving cars [EB/OL]. 2016: CoRR, abs/1604.07316.

[5] Shahid N, Rappon T, Berta W. Applications of Artificial Neural Networks in Health Care Organizational Decision-Making: A Scoping Review[J]. *PLoS ONE*, 2019, 14(2): e0212356.

[6] Huang L, Joseph A D, Nelson B, et al. Adversarial Machine Learning[C]. *The 4th ACM workshop on Security and artificial intelligence*, 2011: 43-58.

[7] Julian K D, Kochenderfer M J, Owen M P. Deep Neural Network Compression for Aircraft Collision Avoidance Systems[J]. *Journal of Guidance, Control, and Dynamics*, 2019, 42(3): 598-608.

[8] Katz G, Barrett C, Dill D L, et al. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks[C]. *International Conference on Computer Aided Verification*, 2017: 97-117.

[9] Gopinath D, Converse H, Pasareanu C, et al. Property Inference for Deep Neural Networks[C]. *2019 34th IEEE/ACM International Conference on Automated Software Engineering*, 2020: 797-809.

[10] Fu F S, Li W C. Sound and Complete Neural Network Repair with Minimality and Locality Guarantees[EB/OL]. 2021: arXiv: 2110.07682. https://arxiv.org/abs/2110.07682

[11] Dreossi T, Ghosh S, Yue X Y, et al. Counterexample-Guided Data Augmentation[EB/OL]. 2018: arXiv: 1805.06962. https://arxiv.org/abs/1805.06962

[12] Ren X H, Yu B, Qi H, et al. Few-Shot Guided Mix for DNN Repairing[C]. *2020 IEEE International Conference on Software Maintenance and Evolution*, 2020: 717-721.

[13] Sinitsin A, Plokhotnyuk V, Pyrkin D, et al. Editable Neural Networks[EB/OL]. 2020: arXiv: 2004.00345. https://arxiv.org/abs/2004.00345

[14] Ma S Q, Liu Y Q, Lee W C, et al. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection[C]. *The 2018 26th ACM Joint Meeting on European*

Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018: 175-186.

[15] Kirkpatrick J, Pascanu R, Rabinowitz N, et al. Overcoming Catastrophic Forgetting in Neural Networks[J]. *Proceedings of the National Academy of Sciences of the United States of America*, 2017, 114(13): 3521-3526.

[16] Matthew Sotoudeh and Aditya V Thakur. Provable repair of deep neural networks[C]. *In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021: 588-603.

[17] Sotoudeh M, Thakur A V. Provable Repair of Deep Neural Networks[C]. *The 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021: 588-603.

[18] Usman M, Gopinath D, Sun Y C, et al. NNrepair: Constraint-Based Repair of Neural Network Classifiers[EB/OL]. 2021: arXiv: 2103.12535. https://arxiv.org/abs/2103.12535

[19] Yann LeCun. The mnist database of handwritten digits. http://yann.lecun. com/exdb/mnist. 1998.

[20] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images[M], 2009.

[21] Goodfellow I J, Shlens J, Szegedy C. Explaining and Harnessing Adversarial Examples[EB/OL]. 2014: arXiv: 1412.6572. https://arxiv.org/abs/1412.6572

[22] Gu T Y, Liu K, Dolan-Gavitt B, et al. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks[J]. *IEEE Access*, 2019, 7: 47230-47244.

[23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. https://www.gurobi.com. 2021.

[24] Sun S, Yan J, Yan R J. Layer-Specific Repair of Neural Network Classifiers[C]. *International Conference on Artificial Neural Networks*, 2022: 550-561.

孙朔 于 2020 年在中国科学技术大学少年班学院获得学士学位。现在中国科学院软件所软件工程专业攻读博士学位。研究领域为智能软件中的漏洞检测与修复。Email: sunshuo20@otcaix.isca.ac.cn

晏荣杰 于 2007 年在中国科学院软件研究所获得博士学位。现任中国科学院软件研究所副研究员。研究领域为嵌入式、自治系统的分析与设计。Email: yrj@ios.ac.cn

严俊 于 2007 年在中国科学院研究生院获得博士学位。现任中国科学院软件研究所研究员。研究领域为程序测试与分析。Email: yanjun@ios.ac.cn