结合 Doc2Vec 和 BERT 嵌入技术的补丁验证方法

黄颖，姜淑娟，蒋婷婷

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210900207

结合 Doc2Vec 和 BERT 嵌入技术的补丁验证方法

黄 颖 姜淑娟 蒋婷婷
中国矿业大学矿山数字化教育部工程研究中心 江苏 徐州 221116
中国矿业大学计算机科学与技术学院 江苏 徐州 221116
(TS19170033A31@cumt.edu.cn)

摘 要 自动程序修复是近年来的研究热点并取得了一定的进展。现有的自动程序修复方法大多利用测试套件来验证补丁正确性。然而,使用测试套件验证自动程序修复方法生成的大量候选补丁不仅会造成巨大的开销,不完美的测试套件还会导致补丁的过拟合问题,因此如何提高补丁验证效率、有效验证补丁正确性成为亟待解决的问题。为了降低补丁验证开销并提高补丁正确率,提出了结合两种嵌入技术验证补丁正确性的方法。该方法首先利用 Doc2Vec 计算补丁与错误代码的相似性,然后使用一个基于 BERT 模型的分类器过滤通过相似性筛选出的补丁中的错误补丁。为了验证所提方法的有效性,基于 5 个开源的 Java 缺陷库进行实验,结果表明该方法能够有效地验证补丁的正确性并提高验证效率。

关键词:自动程序修复;补丁验证;代码相似性;嵌入技术
中图法分类号 TP311

Patch Validation Approach Combining Doc2Vec and BERT Embedding Technologies

HUANG Ying,JIANG Shu-juan and JIANG Ting-ting
Engineering Research Center of Mine Digitalization of Ministry of Education,China University of Mining and Technology,Xuzhou,Jiangsu 221116,China
School of Computer Science and Technology,China University of Mining and Technology,Xuzhou,Jiangsu 221116,China

Abstract Automatic program repair is a research hotspot in recent years and has made some progress. Most of the existing automatic program repair methods use the test suite to validate patch correctness. However,using the test suite to validate a large number of candidate patches will not only bring huge costs,but also lead to the overfitting problem of patches. Therefore,how to improve the efficiency of patch validation and effectively validate patch correctness has become an urgent problem. In order to reduce the cost and improve the patch accuracy,this paper proposes an approach combining two embedding techniques to validate patch correctness. Firstly,this approach uses Doc2Vec model to calculate the similarity between the patch and the error code,then it uses the classifier based on BERT model to filter out the error patches from the patches screened by the similarity. To evaluate the effectiveness of this approach,experiments are carried out based on five open source Java benchmarks. Experimental results show that this approach can effectively validate patch correctness and improve the efficiency of patch validation.

Keywords Automatic program repair,Patch validation,Code similarity,Embedding technology

1 引言

随着计算机科学技术的不断发展,软件产业已经逐渐深入到社会生产和人类生活的方方面面,然而软件缺陷却难以避免。软件缺陷会使得软件不能满足既定的功能需求,甚至会导致软件难以正常执行,造成经济以及其他方面的重大损失。因此,软件的维护对于促进社会发展必不可少。软件自动修复旨在通过自动生成正确补丁来减少修复软件缺陷的工作量,同时在一定程度上解决传统软件缺陷修复方法在修复软件缺陷上面临的能力不足和成本昂贵、不可控条件下无法修复等问题[1]。

近年来,为了自动化软件修复过程,减少软件维护的工作量,开发人员提出了多种自动程序修复方法。其中,Weimer 等[2]和 Le Goues 等[3]应用基于遗传规划的方法自动生成补丁,是自动程序修复研究领域的开创性研究工作;Kim 等[4]基于总结的代码修改模板提出了 PAR 方法;针对补丁的准确性及搜索空间的效率问题,Nguyen 等[5]提出了基于语义的方法 Semfix,该方法使用符号执行技术和约束求解器改善了修复效果;White 等[6]则通过深度学习代码相似度对程序修复成分进行排序和转换,从而自动生成修复补丁。总之,以上方法都是通过自动生成候选补丁并对其进行测试来得到最终的正确补丁,实现程序缺陷的自动修复。

自动程序修复包括缺陷定位、补丁生成、补丁验证和确认

到稿日期:2021-09-24 返修日期:2022-03-11
基金项目:国家自然科学基金(61673384)
This work was supported by the National Natural Science Foundation of China(61673384).
通信作者:姜淑娟(shujjiang@cumt.edu.cn)

阶段[7]。对于补丁的验证和确认，目前的大部分自动程序修复方法都是使用测试套件来对生成的修复补丁进行验证。然而，不同的源代码可能有着相同的语义，测试套件并不一定能够完全过滤掉错误补丁，从而导致过拟合。同时，使用测试套件对每个候选补丁进行多次测试时，需要对程序进行多次重编译与运行，这一过程会耗费大量时间。因此，单纯使用测试用例对补丁进行验证并不能很好地识别出正确补丁，还会造成大量验证开销。

为了在验证阶段提高生成的候选补丁的正确率并减小验证开销，有开发人员建议对测试套件进行改进，如 Yang 等[8]通过生成更好的测试套件来增强补丁验证，Xin 等[9]则通过尝试添加测试用例来扩展测试套件覆盖范围；也有学者尝试针对生成的候选补丁进行改进，Long 等[10]通过学习正确代码的概率，应用独立模型按照可能的正确性对候选补丁进行排序，并丢弃可能过拟合的补丁。本文按照后者的思路，提出利用嵌入模型选择正确补丁并过滤错误补丁来辅助补丁验证。

有研究[11]表明，语法层面提取的代码特征可用于预测过拟合补丁，即可以利用嵌入模型捕捉代码自然性特征。而相似性在基于冗余的程序修复中具有显著作用[12]，Tian 等[13]证明了正确补丁相比错误代码更改较小，同时嵌入模型可以有效提取代码相似性以识别正确补丁并过滤错误补丁，因此可通过计算错误代码和候选补丁之间的相似性来评估补丁的正确性。但是文献[13]中的结果表明，各个嵌入模型识别正确补丁与过滤错误补丁的能力各有优劣，因此本文提出结合两个嵌入模型来弥补双方的不足。

本文方法的基本思想是利用词嵌入模型提取错误代码与补丁代码的语法语义特征。首先利用一个嵌入模型计算两者之间的相似性，根据相似性对候选补丁进行排序并根据设置的阈值选择可能正确的补丁，之后在选择出的可能正确的补丁中，再利用基于另一个嵌入模型的分类器过滤可能过拟合的补丁，最后达到提高补丁验证效率的目的。为了验证本文方法的有效性，首先在多个自动程序修复工具生成的补丁集合上进行实验，比较了本文方法与只使用单个嵌入模型的方法的优劣。之后，将本文方法应用于修复工具 SimFix[14]，AVATAR[15]，FixMiner[16]，TBar[17]以及 jKali[18]生成的补丁集合上，结果表明，本文提出的方法可以过滤生成的错误补丁并选择正确补丁，提高结果的精度。

本文的主要贡献是提供了一种结合两种嵌入技术来评估补丁正确性的方法。与单个嵌入技术相比，本文结合了在识别正确补丁方面性能更好的 Doc2Vec[19]模型以及在过滤错误补丁方面有着更好性能的 BERT[20](Bidirectional Encoder Representations from Transformers)模型。本文提供了一个同时兼顾两方面性能的方法，解决了单个嵌入技术在过滤错误补丁与识别正确补丁两个方面的性能差异过大的问题，并使用 5 种程序自动修复工具进行验证，证明了其有效性。并且，这一方法可独立用于评估多种软件自动修复方法所生成补丁的正确性。

2 研究背景

2.1 自动程序修复工具

典型的自动程序修复技术通常以一个错误程序和一个测试集作为输入，输出补丁来修复故障。现有的自动程序修复技术存在补丁搜索空间过大以及过拟合的问题。比如由于过拟合问题，生成和验证修复工具 jKali[18]生成的补丁绝大多数都是不正确的。

针对存在的问题，Jiang 等[14]结合了现有补丁以及源代码这两种用于生成补丁的数据源，以减少两种数据源各自的局限性，并实现了原型工具 SimFix[14]。SimFix 方法的基本思想是两种数据源各描述一个搜索空间，然后取两个搜索空间的交集，从而得到最终的搜索空间。在最终的搜索空间中，系统使用基本的启发式搜索方法搜索补丁。Jiang 等[14]在 Defects4J 基准集[21]上评估了 SimFix，成功修复 34 个 bug，其中 13 个 bug 没有被 SimFix 之前的修复方法修复过。

基于模板的方法是自动程序修复中提高补丁正确性的重要方法。从常见的补丁中挖掘修复模式可以得到针对性的修复补丁，从而提高补丁正确性。AVATAR[15]，FixMiner[16]以及 TBar[17]3 种程序自动修复工具都是通过提取修复模式来生成补丁。其中 AVATAR[15]工具通过将给定的一组编辑脚本进行相似脚本分组，来推断编辑操作的公共子集，最终得到的修复模式是关于特定抽象语法树节点类型的修复动作的编辑脚本，它是第一个利用静态分析冲突的修复模式作为修复成分的方法。FixMiner[16]是一种模式挖掘方法，它利用富编辑脚本的不同树表示来识别相似更改，实验表明，该方法可以有效利用富编辑脚本中的变化信息来挖掘准确的修复模式，将其集成到自动修复工具中也能有效修复多个补丁。Liu 等[17]研究了多种修复模式并对其进行分类与评估，在此基础上实现了原型工具 TBar。TBar 根据可疑语句的抽象语法树上下文信息来选择修复模式。与其他方法相比，以上 3 种基于模板的方法可以修复更多数量的软件缺陷。

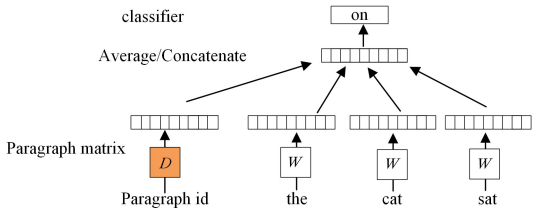
因此，本文将利用这几种工具生成的合理补丁集合来验证本文方法的有效性。

2.2 词嵌入模型

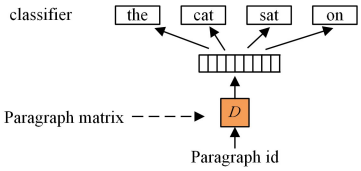
神经网络嵌入是用连续向量表示离散变量的方法，而词嵌入是嵌入在机器翻译中的成功应用。在自然语言处理(Natural Language Processing，NLP)中，词嵌入技术可用于语言建模和特征学习，其中，词汇中的单词或短语被映射到低维空间中的实数向量，如 Word2Vec[19]，Doc2Vec[19]和 BERT[20]已被成功地应用于不同的语义相关任务。

2.2.1 Doc2Vec

Word2Vec[19]以完全无监督的方式从输入的纯文本中学习连续的单词嵌入，输出一组表示语料中词语的特征向量。Word2Vec 使用两种神经网络模型，分别为连续词袋(Continuous Bag-of-Word，CBOW)模型和 Skip-gram 模型。其中，CBOW 模型[22]根据上下文预测当前单词，而 Skip-gram 模型[22]预测当前单词的相邻单词。Doc2Vec[19]是建立在 Word2Vec 基础上的用于训练任意长度的文档的向量表示的模型，它使用的两种神经网络模型，即段落向量的分布式记忆模型(the Distributed Memory Version of Paragraph Vectors，PV-DM)和段落向量的分布式词袋模型(the Distributed Bag of Words Version of Paragraph Vector，PV-DBOW)分别是对 CBOW 与 Skip-gram 两个模型的扩展，即添加了另一个表示文档 id 的特征向量，两个模型的示意图如图 1 所示。



(a)段落向量的分布式记忆 PV-DM



(b)段落向量的分布式词袋 PV-DBOW

图 1　Doc2Vec 的两种神经网络模型

Fig. 1　Two neural network models of Doc2Vec

利用 Doc2Vec 模型计算代码相似性时，对于代码长度没有要求，而且得到的相似度往往偏高，因此难以忽略正确的补丁，这也是本文实验用 Doc2Vec 模型来筛选正确补丁的原因。但是该模型也容易将错误补丁识别为正确补丁，因此需要结合其他模型一起用于补丁验证。

2.2.2　BERT

BERT[20]是一种语言表征模型，由 Google 的 AI 语言团队引入。BERT 首先使用无监督的方式从大规模无标注语料中训练，获得文本的语义表示，然后将该语义表示在特定的自然语言处理任务中进行微调，最终应用于该自然语言处理任务，如情感分析，问题回答等。BERT 的输入是由文本中各个字或单词的原始词向量、文本向量以及位置向量求和得到的向量，输出为各个字或单词融合了全文语义信息后的向量表示。

为了更好地刻画出语言的本质，以便后续针对特定 NLP 任务进行微调，引入该模型的团队提出了两个预训练任务：1)遮蔽语言模型(Masked Language Model，MLM)；2)下一句预测(Next Sentence Prediction)任务。

MLM 任务指将一句话输入给 BERT 之前，随机将这几句话中 15%的单词或字用[MASK]标记替换，然后要求模型根据剩余未被替换的词汇预测出被遮蔽的原单词或字，最终的损失函数只计算被遮蔽的那个单词或字；下一句预测任务是为了预训练句子之间的关系模型，即给定同一篇文章中的两个句子，判断第二句是否紧跟第一句之后。BERT 模型对 MLM 任务和下一句预测任务进行联合训练，使其尽可能全面、准确地刻画输入文本的整体信息，以便为针对特定 NLP 任务进行微调提供更好的模型参数初始值。BERT 模型针对句子对任务的微调过程如图 2 所示。

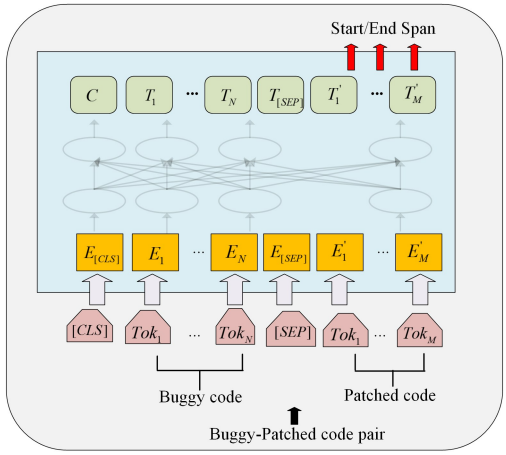


图 2　BERT 的微调过程

Fig. 2　Fine-tuning procedure of BERT

BERT 模型对输入的代码长度有一定限制，但它的深度双向预测模型使其在训练时能更好地利用上下文信息，且有一定纠错能力。它对补丁进行分类时既能有效过滤错误补丁，也能在一定程度上识别正确补丁，但过滤错误补丁的能力更强，因此本文将它与 Doc2Vec 模型相结合来进行实验。

3　本文方法

3.1　方法的总体框架

本文方法共分为 4 个阶段，整体框架如图 3 所示。

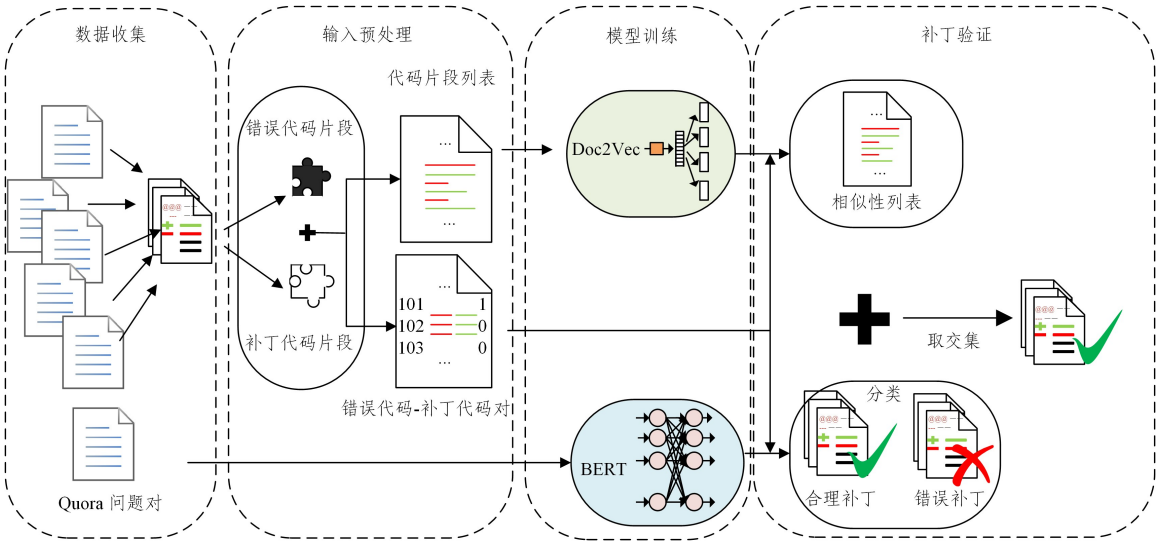


图 3　本文方法的整体框架

Fig. 3　Overall framework of our method

第一阶段是数据收集，这一阶段主要收集用于训练模型的数据集以及用于验证的补丁集合；第二阶段是模型输入预处理，这一阶段对输入模型的代码进行处理，并将数据转换为模型可接受的形式；第三阶段是嵌入模型的训练；第四阶段则是使用学习好的嵌入模型对生成的补丁集合进行验证，选择正确补丁并过滤错误补丁。

3.2 数据收集

本文针对不同的模型选择了不同的训练数据。Doc2Vec 模型使用 Bears[23]，Bugs. jar[24]，Defects4J[21]，ManySStubBs4J[25]以及 QuixBugs[26]这 5 个基准集中的补丁来训练代码嵌入；BERT 模型使用自然语言数据集 Quora 问题对数据集进行训练。最后对模型进行评估时则使用了 Liu 等[27]生成的补丁集合。

3.3 模型输入预处理

在对补丁进行验证时，两个模型的输入都需要将错误代码片段以及需要验证的补丁代码片段分别表示成单行文本，并将对应的错误代码片段与补丁代码片段的单行文本表示为错误代码-补丁代码对，从而对错误代码-补丁代码对计算相似性值或者进行分类，以判断补丁是否正确。

除此之外，还需要对 Doc2Vec 模型的训练数据进行预处理。在使用 5 个基准集中的补丁代码片段进行训练时，首先同样需要分别保留补丁中的错误代码行与修复代码行，得到错误代码片段和补丁代码片段。与需要验证的补丁不同的是，此处不是得到错误代码-补丁代码对，而是将得到的两种代码片段表示成单行文本，并将代码片段列表作为 Doc2Vec 模型的输入。

3.4 模型训练

本文使用 5 个基准集中的补丁代码片段来训练 Doc2Vec 模型，训练中使用的神经网络模型为段落向量的分布式词袋 PV-DBOW 模型，这一模型相比 PV-DM 模型所需的训练内存更小，因为它只储存 softmax 权重，而无须储存词向量。

第二个嵌入模型考虑的是一个基础的预先训练好的 12 层 BERT 模型，这一模型增加了一个单线性层用于分类，实验中将该层作为分类器。输入训练数据后，预训练好的 BERT 模型以及未训练的分类器会根据特定的句子对分类任务进行训练。

3.5 补丁验证

这一阶段首先使用训练好的 Doc2Vec 模型计算自动程序修复工具生成的补丁代码与错误代码片段的余弦相似性，并根据相似性大小将所有补丁按照降序排序，并选择相似性高于阈值的补丁；同时使用 BERT 模型对自动程序修复工具生成的错误代码-补丁代码对分类，将使用 Doc2Vec 模型挑选出的补丁根据分类结果过滤掉错误补丁。这一过程类似于取两种模型得到的正确补丁的交集。

4 实验

本文实验的运行环境为 64 位 Windows 与 Linux 系统，4 个 2.20GHz 的 Intel(R) Xeon(R) Gold 5120 CPU，24 GB 物理内存，1 个 GeForce RTX 2080 Ti GPU，编译环境为 Python3。

4.1 数据集

表 1 列出了训练 Doc2Vec 模型使用的 5 个 Java 基准集 Bears[23]，Bugs. jar[24]，Defects4J[16]，ManySStubBs4J[25]，QuixBugs[26]以及评估模型时使用的 Liu 等[27]生成的补丁集合，并列出了各个数据集包含的补丁数目。

表 1 各数据集包含的补丁数目

Table 1 Number of patches contained in each dataset

Subjects	Correct patches	Incorrect patches	Labelled datasets	Total
Bears	251	0	—	251
Bugs. jar	1158	0	—	1158
Defects4J	864	0	—	864
ManySStubBs4j	34051	0	—	34051
QuixBugs	40	0	—	40
Liu et al.	125	474	Yes	599
Total	36489	474	—	36963

本文使用 Quora 问题对数据集来对 BERT 模型的句子对分类方法进行微调。这一数据集收集了 Quora 平台上的问题对，目的是挖掘具有相同意图的问题，以求快速找到高质量回复。这一目的与本文找到与错误代码片段相似的补丁代码的目的相同，因此在实验中选择这一数据集微调 BERT 模型。

4.2 评价标准

为了对本文方法进行评价，本文采用以下几个指标对实验结果进行分析：

$$R+=\frac{TP}{TP+FN},R-=\frac{TN}{TN+FP}$$

$$P=\frac{TP}{TP+FP},F1=\frac{2\times P\times (R+)}{P+(R+)}$$

其中，TP 表示准确识别的正确补丁数量；FN 表示错误识别的正确补丁数量；TN 表示准确识别的错误补丁数量；FP 表示错误识别的错误补丁数量。而度量 $R+$ 为识别正确补丁的召回率，表示生成的正确补丁中有多少补丁被预测为正确补丁；$R-$ 为过滤错误补丁的召回率，表示生成的错误补丁中被过滤的补丁概率；P 为准确率，表示所有被预测为正确补丁且真正为正确补丁的概率。这几个度量的值越高，效果越好。

4.3 实验设计

为了验证本文提出的结合两个嵌入模型验证补丁正确性方法的有效性，本文提出了两个研究问题：

RQ1：结合了两个嵌入模型的方法能否有效地选择正确补丁并过滤错误补丁？

RQ2：使用本文的方法能否提高自动程序修复工具生成的合理补丁集合中的补丁正确率？

为了回答以上两个问题，本文从两个方面进行研究。首先使用 Liu 等的补丁集合评估本文方法的性能，并与单个模型的效果进行比较，然后利用模型验证 5 个程序自动修复工具生成的补丁。训练模型时，Doc2Vec 模型的词向量大小为 200，epoch 为 500，而 Devlin 等[20]的报告表明，BERT 模型的批大小(Batch_size)设为 32、epoch 设置为 3 较为理想，但由于环境限制，实验中设置 Batch_size 为 16，而对于 epoch，比较之后，将其设为 2 而不是 3，且将学习率设置为 2×10^{-5}。

4.4 实验结果与分析

4.4.1 与单个嵌入技术比较

为了回答 RQ1，本文在 Liu 等[27]的 15 个自动程序修复

工具生成的补丁集合上进行了验证。我们在补丁集合上分别评估了 Doc2Vec 模型、BERT 模型以及本文方法，即两者相结合的方法，并将三者的效果进行了对比。至于实验结果的处理，对于 Doc2Vec 模型，实验中选择的相似性阈值是下四分位数；对于本文方法，考虑到 BERT 模型过滤错误补丁的能力较强，因此将分类的可能性阈值降低，避免过滤大量正确补丁。实验中选定了 6 个阈值来验证本文方法的有效性，实验结果如表 2 所列。阈值为 0.1 时，度量 F1 值最高，因此最终将可能性阈值设为 0.1。

表 2　BERT 的阈值

Table 2　Threshold of BERT

Threshold	$R+$	$R-$	P	$F1$
0.6	0.4377	0.6513	0.6967	0.5376
0.5	0.4534	0.6396	0.5495	0.5495
0.4	0.4639	0.6264	0.6944	0.5559
0.3	0.4314	0.5730	0.6498	0.5270
0.2	0.4785	0.5423	0.6576	0.5556
0.1	0.5466	0.4731	0.6558	0.5972

从表 3 可以看出，Doc2Vec 模型识别正确补丁的召回率在 3 种方法中最高，达到了 0.7717，但是过滤错误补丁的召回率却只有 0.2893，说明 Doc2Vec 模型过滤错误补丁的能力十分有限；BERT 模型分类器则能更有效地过滤错误补丁，但是识别的正确补丁数不到所有正确补丁的一半；与前两者相比，虽然本文方法各个度量的值都不是最高的，但是却弥补了两种模型的不足，不仅能识别大部分正确补丁，也能过滤掉大量错误补丁，而且准确率也与两种模型相差无几，这就可以回答 RQ1，即本文方法可以有效地选择正确补丁并过滤错误补丁。

表 3　过滤错误补丁的结果

Table 3　Results of filtering incorrect patches

Model	$R+$	$R-$	P	$F1$
Doc2Vec	0.7717	0.2893	0.6652	0.7145
BERT	0.4042	0.6654	0.6892	0.5096
Doc2Vec+BERT	0.5466	0.4731	0.6558	0.5972

4.4.2　验证修复工具生成的合理补丁集合

我们将 SimFix 工具生成的合理补丁集合中的重复补丁去掉，并将本文方法应用到该集合上，获得的结果如表 4 所列。表 4 中 $\#CP$ 表示正确补丁个数，$\#IP$ 表示错误补丁个数，第 2－第 4 列结果表示为 x/y，其中 x 为自动修复工具生成的补丁数目或准确率，y 为使用本文方法过滤后得到的补丁数目或准确率。

表 4　在修复工具补丁集合上的评估结果

Table 4　Evaluation results on patches of tools

Datasets	$\#CP$	$\#IP$	P	$R+$	$R-$	top10
SimFix	24/17	43/34	0.3582/0.4146	0.7083	0.4419	7
AVATAR	21/16	42/24	0.3333/0.4000	0.7619	0.4286	3
FixMiner	14/10	24/12	0.3684/0.4545	0.7143	0.5000	4
TBar	26/18	53/27	0.3291/0.4000	0.6923	0.4906	3
jKali	6/2	19/2	0.2400/0.5000	0.3333	0.8947	2

从表 4 中可以看到，本方法对 SimFix 工具生成的合理补丁集合的识别正确补丁的召回率达到了 0.7083，同时过滤掉 44.19%的错误补丁，准确率相比原来的 35.82%也提高至 41.46%。结果表明，本文方法可以提高 SimFix 工具生成的合理补丁集合中的补丁正确率。同样，对于 AVATAR，FixMiner 以及 TBar 这 3 个修复工具生成的合理补丁集合，本文方法可以识别 65%80%的正确补丁，同时可以过滤掉 40%50%的错误补丁，准确率也都有所提升。而对于 jKali 生成的合理补丁集合，识别的正确补丁只有 1/3，而过滤掉的错误补丁高达 89.47%，其原因是该工具生成的正确补丁过少，准确率过低，使用本文方法后，准确率从 24%提升到了 50%。例如，对于 Defects4J 基准集中的 Chart 项目中的缺陷 Chart-25，SimFix 生成了一个错误补丁，而本文方法则将其识别为错误补丁并进行了过滤。

之后，我们还将 Doc2Vec 模型得到的相似性与 BERT 分类器得到的分类可能性相加来排序 SimFix 工具生成的合理补丁集合，结果显示，在 top5 中，正确补丁有 3 个；在 top10 中，正确补丁有 7 个。对于 AVATAR，FixMiner 以及 TBar 这 3 个修复工具，top10 中正确补丁也占 1/3 以上，而对于 jKali，识别出来的两个正确补丁都在 top5 中。这回答了 RQ2，即本文方法可以有效地排序候选补丁，提高自动程序修复工具生成的合理补丁集合中的补丁正确率。

5　有效性分析

我们从内部有效性和外部有效性两个方面对本文方法的有效性进行了讨论。

对本文方法的内部有效性的影响主要来自嵌入模型在验证的补丁集合上的结果的准确性，本文按照相关文献复现了相应的模型，保证了结果的准确性。

对外部有效性的影响主要在于训练模型的数据集的规模对模型效果的影响。本文选择了 5 个基准集中的 36364 个补丁代码来训练 Doc2Vec 模型，尽管数据集不大，但是该数据集中包含多种类型的补丁代码；同时增加了模型训练时的迭代次数，在一定程度上弥补了不足。

6　相关工作

近年来，自然语言处理嵌入技术领域已经有了一定的发展。如之前提到的 Doc2Vec[19] 和 BERT[20]，其中，Doc2Vec 已经被应用于各种软件工程任务。例如，Ndichu 等[28] 使用 Doc2Vec 学习 AST 级别的代码结构表示来预测基于 Java 脚本的攻击。但是实验表明，Doc2Vec 在面对混淆的数据集时，难以准确检测出恶意代码，这与本文发现的 Doc2Vec 模型难以过滤大量错误补丁相似。而本文方法使用 BERT 模型对 Doc2Vec 模型的结果再次进行了过滤，从而弥补了 Doc2Vec 的不足，保证能够检测出大量错误补丁。在 Doc2Vec 之后，Alon 等[29] 提出了 code2vec，它是一种基于注意力的神经代码嵌入模型，考虑代码中的结构信息，通过在 AST 路径上训练代码令牌实现。它的嵌入早已被用来预测代码片段的语义属性，如方法名的预测。最近，Hoang 等[30] 又提出了 CC2Vec，它是一个专门的分层注意力神经网络模型，专门用于学习有

相关提交信息引导的代码更改的向量表示。CC2Vec 在提交消息生成、错误修复补丁识别和及时缺陷预测方面具有很好的性能。

之前的研究很大程度上忽略了对补丁正确性的分析，Qi 等[31]对补丁正确性进行分析研究之后发现了过拟合问题。他们对 3 个生成和验证程序修复系统(即 GenProg，RSRepair 和 AE)报告的补丁进行了系统分析，结果表明生成的绝大多数补丁是不正确的，只是在错误程序的测试套件中过度拟合了测试输入。为了提高补丁的正确率，方法之一是增强测试套件，即需要更多的测试。Xiong 等[32]提出利用新测试用例的自动生成，以及测量错误源程序和修复程序上失败测试的行为相似性来评估 APR 系统的补丁正确性。Tian 等[13]研究了代码更改的表示学习对预测补丁正确性的影响。研究比较了 Doc2Vec，BERT，code2vec 以及 CC2Vec 这 4 个表示学习模型用于指导补丁正确性预测的效果，为嵌入在补丁正确性预测这一方向的研究提供了广泛的实验结果。结果同样表明 Doc2Vec 存在难以过滤错误补丁的问题，而 BERT 难以识别正确补丁。本文结合了 Doc2Vec 和 BERT 两种嵌入技术，弥补了两种嵌入技术的不足并结合了两者的优势，既能识别大量正确补丁，又能过滤大量错误补丁。将本文方法应用在使用测试套件验证补丁的 5 个程序自动修复工具生成的补丁上，表明本文方法可以有效验证补丁正确性并提高验证效率。

结束语 本文提出了结合两种嵌入技术来验证补丁正确性的方法，并在 15 种自动程序修复工具生成的补丁集合上进行了验证。实验表明，两种嵌入模型在自动程序修复工具生成的补丁集合上的性能各有优势，但是不足之处也比较明显，而本文方法可以弥补两者的不足，达到一个比较理想的效果，既可以识别大部分正确补丁，也可以过滤掉大量错误补丁。对 5 个自动修复工具生成的补丁进行验证，结果表明本文方法可以识别 60%80%的正确补丁，过滤掉 40%50%的错误补丁，提高了准确率，并且根据相似性与分类可能性之和对补丁进行排序时，top10 的补丁中有 1/3 以上的正确补丁，尤其是对于 SimFix 来说，top10 的补丁中正确补丁有 7 个，这证明本文方法可以有效验证补丁正确性，提高补丁验证的效率。

参 考 文 献

[1] GAZZOLA L, MICUCCI D, MARIANI L. Automatic software repair: a survey[J]. IEEE Transactions on Software Engineering, 2017, 45(1): 34-67.

[2] WEIMER W, NGUYEN T V, LE GOUES C, et al. Automatically finding patches using genetic programming[C]// International Conference on Software Engineering. IEEE Computer Society, 2009: 364-374.

[3] LE GOUES C, NGUYEN T V, FORREST S, et al. Genprog: a generic method for automatic software repair[J]. IEEE Transactions on Software Engineering, 2011, 38(1): 54-72.

[4] KIM D, NAM J, SONG J, et al. Automatic patch generation learned from human-written patches[C]// International Conference on Software Engineering. IEEE Computer Society, 2013: 802-811.

[5] NGUYEN H D T, QI D, ROYCHOUDHURY A, et al. Semfix: program repair via semanticanalysis[C]// International Conference on Software Engineering. IEEE Computer Society, 2013: 772-781.

[6] WHITE M, TUFANO M, MARTINEZ M, et al. Sorting and transforming program repair ingredients via deep learning code similarities[C]// International Conference on Software Analysis, Evolution and Reengineering. IEEE Computer Society, 2019: 479-490.

[7] WANG Z, GAO J, CHEN X, et al. Automatic program repair techniques: a survey [J]. Chinese Journal of Computers, 2018, 41(3): 588-610.

[8] YANG J, ZHIKHARTSEV A, LIU Y, etal. Better test cases for better automated program repair[C]// Joint Meeting on Foundations of Software Engineering. ACM, 2017: 831-841.

[9] XIN Q, REISS S P. Identifying test-suite-overfitted patches through test case generation[C]// International Symposium on Software Testing and Analysis. ACM, 2017: 226-236.

[10] LONG F, RINARD M. Automatic patch generation by learning correct code[C]// ACM SIGPLAN-SIGACT Symposium on Principles of Programming Language. ACM, 2016: 298-312.

[11] YE H, GU J, MARTINEZ M, et al. Automated classification of overfitting patches with statically extracted code features[J]. arXiv: 1910, 12057, 2019.

[12] CHEN Z, MONPERRUS M. The remarkable role of similarity in redundancy-based program repair[J]. arXiv: 1811. 05703, 2018.

[13] TIAN H, LIU K, KABORÉ A K, et al. Evaluating representation learning of code changes for predicting patch correctness in program repair[C]// International Conference on Automated Software Engineering. IEEE Computer Society, 2020: 981-992.

[14] JIANG J, XIONG Y, ZHANG H, et al. Shaping program repair space with existing patches and similar code[C]// International Symposium on Software Testing and Analysis. ACM, 2018: 298-309.

[15] LIU K, KOYUNCU A, KIM D, et al. Avatar: Fixing semantic bugs with fix patterns of static analysis violations[C]// International Conference on Software Analysis, Evolution and Reengineering. IEEE Computer Society, 2019: 1-12.

[16] KOYUNCU A, LIU K, BISSYANDÉ T F, et al. Fixminer: Mining relevant fix patterns for automated program repair[J]. arXiv: 1810. 01791, 2018.

[17] LIU K, KOYUNCU A, KIM D, et al. Tbar: revisiting template-based automated program repair[C]// International Symposium on Software Testing and Analysis. ACM, 2019: 31-42.

[18] MARTINEZ M, MONPERRUS M. Astor: A program repair library for java[C]// International Symposium on Software Testing and Analysis. ACM, 2016: 441-444.

[19] LE Q, MIKOLOV T. Distributed representations of sentences and documents[C]// International Conference on Machine Learning. ACM, 2014: 1188-1196.

[20] DEVLIN J, CHANG M W, LEE K, et al. Bert: pre-training of deep bidirectional transformers for language understanding[J]. arXiv: 1810. 04805, 2018.

[21] JUST R, JALALI D, ERNST M D. Defects4J: a database of

existing faults to enable controlled testing studies for Java programs[C]// International Symposium on Software Testing and Analysis. ACM，2014：437-440.

[22] MIKOLOV T，CHEN K，CORRADO G，et al. Efficient estimation of word representations in vector space[J]. arXiv：1301.3781，2013.

[23] MADEIRAL F，URLI S，MAIA M，et al. Bears：an extensible java bug benchmark for automatic program repair studies[C]// International Conference on Software Analysis，Evolution and Reengineering. IEEE Computer Society，2019：468-478.

[24] SAHA R K，LYU Y，LAM W，et al. Bugs. jar：a large-scale，diverse dataset of real-world java bugs[C]// International Conference on Mining Software Repositories. ACM，2018：10-13.

[25] KARAMPATSIS R M，SUTTON C. How often do single-statement bugs occur? The ManySStuBs4J dataset[C]// International Conference on Mining Software Repositories. ACM，2020：573-577.

[26] LIN D，KOPPEL J，CHEN A，et al. QuixBugs：A multi-lingual program repair benchmark set based on the Quixey challenge [C]// International Conference on Systems，Programming，Languages，and Applications：Software for Humanity. ACM，2017：55-56.

[27] LIU K，WANG S，KOYUNCU A，et al. On theefficiency of test suite based program repair：A systematic assessment of 16 automated repair systems for java programs[C]// International Conference on Software Engineering. ACM，2020：615-627.

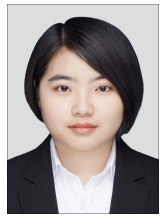
[28] NDICHU S，KIM S，OZAWA S，et al. A machine learning approachto detection of JavaScript-based attacks using AST features and paragraph vectors[J]. Applied Soft Computing，2019，84：105721.

[29] ALON U，ZILBERSTEIN M，LEVY O，et al. code2vec：Learning distributed representations of code[J]. Proceedings of the ACM on Programming Languages，2019，3(POPL)：1-29.

[30] HOANG T，KANG H J，LO D，et al. CC2Vec：Distributed representations of code changes[C]// International Conference on Software Engineering. ACM，2020：518-529.

[31] QI Z，LONG F，ACHOUR S，et al. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems[C]// International Symposium on Software Testing and Analysis. ACM，2015：24-36.

[32] XIONG Y，LIU X，ZENG M，et al. Identifying patch correctness in test-based program repair[C]// International Conference on Software Engineering. ACM，2018：789-799.



HUANG Ying，born in 1996，postgraduate. Her main research interests include automatic program repair and so on.

JIANG Shu-juan，born in 1966，Ph. D，professor，Ph.D supervisor，is a member of China Computer Federation. Her main research interests include software analysis and test，and compilation techniques.

（责任编辑：何杨）