Neural Bug Finding - A Study of Opportunities and Challenges

Abstract

Static analysis is one of the most widely adopted techniques to find software bugs before code is put in production. Designing and implementing effective and efficient static analyses is difficult and requires high expertise, which results in only a few experts able to write such analyses. This paper explores the opportunities and challenges of an alternative way of creating static bug detectors: neural bug finding. The basic idea is to formulate bug detection as a classification problem, and to address this problem with neural networks trained on examples of buggy and non-buggy code. We systematically study the effectiveness of this approach based on code examples labeled by a state-of-the-art, static bug detector. Our results show that neural bug finding is surprisingly effective for some bug patterns, sometimes reaching a precision and recall of over 80%, but also that it struggles to understand some program properties obvious to a traditional analysis. A qualitative analysis of the results provides insights into why neural bug finders sometimes work and sometimes do not work. We also identify pitfalls in selecting the code examples used to train and validate neural bug finders, and propose an algorithm for selecting effective training data.

静态分析是在代码投入生产之前发现软件错误的最广泛的技术之一。设计和实施有效和高效的静态分析是困难的，需要高度的专业知识，这导致只有少数专家能够编写这样的分析。本文探讨了一种创建静态 bug 检测器的替代方法的机遇和挑战: 神经 bug 发现。基本思想是将 bug 检测制定为分类问题，并通过对错误代码和非错误代码示例进行训练的神经网络来解决这个问题。我们基于由最先进的静态 bug 检测器标记的代码示例系统地研究了这种方法的有效性。我们的结果表明，神经错误发现对一些错误模式非常有效，有时达到 80% 以上的精度和召回率, 但也很难理解传统分析中明显的一些程序属性。对结果的定性分析提供了对为什么神经缺陷发现器有时工作，有时不工作的见解。我们还发现了在选择用于训练和验证神经缺陷发现者的代码示例时存在的陷阱，并提出了一种选择有效训练数据的算法。

I. INTRODUCTION

A popular way of finding software bugs early during the development process is static analysis tools that search a code base for instances of common bug patterns. These tools, which we here call *bug detectors*, often consist of a scalable static analysis framework and an extensible set of checkers that each search for instances of a specific bug pattern. Examples of bug detectors include the pioneering FindBugs tool [1], its successor SpotBugs1, Google’s Error Prone tool [2], and the Infer tool by Facebook [3].

开发过程早期发现软件错误的一种流行方法是静态分析工具，它在代码库中搜索常见错误模式的实例。这些工具，我们这里称之为错误检测器，通常由一个可扩展的静态分析框架和一组可扩展的检查器组成，每个检查器都搜索特定错误模式的实例。Bug 探测器的例子包括开拓性的 findbug 工具 [1] 、其继任者 spot bugs1 、谷歌的易出错工具 [2] 和 Facebook [3 的推断工具。

Despite the overall success of static bug detection tools, there still remains a lot of potential for improvement. A recent study that applied state-of-the-art bug detectors to a set of almost 600 real-world bugs shows that over 95% of the bugs are currently missed [4]. The main reason is that various different bug patterns exist, each of which needs a different bug detector. These bug detectors must be manually created, typically by program analysis experts, and they require significant finetuning to find actual bugs without overwhelming developers with spurious warnings. Bug detectors often require hundreds of lines of code each, even for bug patterns that seem trivial to find at first sight and when being built on top of a static analysis framework.

尽管静态 bug 检测工具总体上取得了成功，但仍有很大的改进潜力。最近一项将最先进的 bug 检测器应用于一组近 600 个真实世界的 bug 的研究表明，目前有超过 95% 的 bug 被遗漏了 [4]。主要原因是存在各种不同的 bug 模式，每种模式都需要不同的 bug 检测器。这些错误检测器必须手动创建，通常由程序分析专家创建，它们需要大量的改进才能找到实际的错误，而不会让开发人员产生虚假警告。错误检测器通常需要每一行数百行代码，即使对于乍一看和构建在静态分析框架之上的 Bug 模式来说似乎微不足道。

This paper studies a novel way of creating bug detectors: *neural bug finding*. Motivated by the huge success of neural networks for various software engineering tasks [5], we ask a simple question: Can we automatically learn bug detectors from data, instead of implementing program analyses manually? Giving a positive answer to this question has the potential of complementing existing bug detectors with additional checkers that address previously ignored bug patterns. Moreover, it may enable non-experts in program analysis, e.g., ordinary software developers, to contribute to the creation of bug detectors.

本文研究了一种新的缺陷检测器的创建方法:神经缺陷发现。在各种软件工程任务[5]的神经网络取得巨大成功的激励下，我们提出了一个简单的问题:我们能否从数据中自动学习bug检测器，而不是手工实现程序分析?对这个问题给出一个积极的答案，有可能用额外的检查程序来补充现有的bug检测器，这些检查程序处理以前被忽略的bug模式。此外，它还可以使程序分析中的非专家，例如普通的软件开发人员，为bug检测器的创建做出贡献。

Given the importance of bug detection and the power of neural networks, the intersection of these two areas so far has received surprisingly little attention. Existing work focuses on learning-based defect prediction [6], which ranks entire files by their probability to contain any kind of bug, whereas we here aim at pinpointing code that suffers from a specific kind of bug. Other work addresses the problems of predicting code changes [7], predicting identifier names [8]–[10], and predicting how to complete partial code [11]–[13], which are complementary to detecting bugs. The perhaps closest existing work is DeepBugs [14], which trains a neural network to find name-related bugs, and learning-based techniques for identifying security vulnerabilities [15]–[17]. While these approaches show that neural bug finding is possible for a specific class of bugs, we here study the potential of neural bug finding in much more detail and for a broader range of code issues.

考虑到bug检测的重要性和神经网络的强大功能，这两个领域的交集到目前为止几乎没有受到关注。现有的工作侧重于基于学习的缺陷预测[6]，它根据包含任何类型缺陷的概率对整个文件进行排序，而我们在这里的目标是精确定位受特定类型缺陷影响的代码。其他工作包括预测代码更改[7]、预测标识符名称[8]-[10]以及预测如何完成部分代码[11]-[13]，这些都是对bug检测的补充。目前最接近的工作可能是DeepBugs[14]，它训练一个神经网络来发现与名称相关的bug，以及基于学习的技术来识别安全漏洞[15]-[17]。虽然这些方法表明，神经缺陷的发现对于特定类别的缺陷是可能的，但我们在这里更详细地研究了神经缺陷发现的潜力，并针对更广泛的代码问题进行了研究。

Automatically learning bug detectors requires addressing two problems: (1) Obtaining sufficient training data, e.g., consisting of buggy and non-buggy code examples. (2) Training a model that identifies bugs, e.g., by distinguishing buggy code from non-buggy code. The first problem could be addressed by automatically seeding bugs into code, by extracting buggy code examples from version histories, or by manually labeling code examples as instances of specific bug patterns. In this work, we sidestep the first problem and study whether given sufficient training data, the second problem is tractable. Our work therefore does not yield a ready-to-deploy bug detection tool, but rather novel insights into what kinds of bugs neural bug finding can and cannot find. We believe that thoroughly studying this question in isolation is an important step forward toward the ultimate goal of neural bug finding.

自动学习bug检测器需要解决两个问题:(1)获取足够的训练数据，例如包含bug和非bug代码示例。(2)训练识别bug的模型，例如通过区分有bug的代码和无bug的代码。第一个问题可以通过自动将bug播种到代码中、通过从版本历史中提取有bug的代码示例，或者通过手动将代码示例标记为特定bug模式的实例来解决。在这项工作中，我们避开了第一个问题，研究了是否有足够的训练数据，第二个问题是可处理的。因此，我们的工作并没有提供一个随时可以部署的bug检测工具，而是对神经bug发现可以发现和不能发现哪些类型的bug提出了新的见解。我们相信，孤立地彻底研究这个问题是朝着发现神经缺陷的最终目标迈出的重要一步。

To study the potential of learned bug detectors while sidestepping the problem of obtaining labeled training data, we use an existing, traditionally developed bug detector as a generator of training data. To this end, we run the existing bug detector on a corpus of code to obtain warnings about specific kinds of bugs. Using these warnings and their absence as a ground truth, we then train a neural model to distinguish code with a particular kind of warning from code without such a warning. For example, we train a model that predicts whether a piece of code uses reference equality instead of value equality for comparing objects in Java. This setup allows us to assess to what extent neural bug finding can imitate existing bug detectors.

为了研究学习错误检测器的潜力，同时避免获取标记的训练数据，我们使用一个现有的，传统开发的错误检测器作为训练数据的生成器。为此，我们在一个代码库上运行现有的bug检测器，以获得关于特定类型bug的警告。利用这些警告和它们的缺失作为基本事实，然后我们训练一个神经模型来区分具有特定类型警告的代码和没有此类警告的代码。例如，我们训练一个模型来预测一段代码是否使用引用相等而不是值相等来比较Java中的对象。这个设置允许我们评估神经缺陷发现可以在多大程度上模拟现有的缺陷检测器。

One drawback of using an existing bug detector as the data generator is that some warnings may be spurious and that some bugs may be missed. To mitigate this problem, we focus on bugs flagged by bug detectors that are enabled in production in a major company and that empirically show false positive rates below 10% [18]. Another drawback is that the learned bug detectors are unlikely to outperform the static analyzers they learn from. However, the purpose of this work is to study whether training a model for neural bug finding is feasible, whereas we leave the problem of obtaining training data beyond existing static analyzers as future work.

使用现有的bug检测器作为数据生成器的一个缺点是，有些警告可能是假的，有些bug可能会被忽略。为了缓解这个问题，我们关注由bug检测器标记的bug，这些bug检测器在一家大公司的生产中启用，并且经验上显示错误阳性率低于10%[18]。另一个缺点是，所学习的bug检测器不太可能胜过它们所学习的静态分析程序。然而，这项工作的目的是研究训练一个用于神经缺陷发现的模型是否可行，而我们把获取训练数据的问题留给了现有的静态分析仪作为未来的工作。