北京大学

本科生毕业论文

题目：深度学习程序中数值缺陷检测与修复算法的设计

Algorithm Design for Detecting and Repairing Numerical Bugs in Deep Learning Programs

姓　　名：张煜皓

学　　号：1500012703

院　　系：信息科学技术学院

本科专业：计算机科学与技术

指导导师：熊英飞

二〇一玖 年 五 月

北京大学本科毕业论文导师评阅表

学生姓名	张煜皓	学生学号	1500012703	论文成绩	优
学院（系）	信息科学技术学院			学生所在专业	计算机科学与技术
导师姓名	熊英飞	导师单位/所在研究所	软工所	导师职称	副教授
论文题目（中、英文）	深度学习程序中数值缺陷检测与修复算法设计 Algorithm Design for Detecting and Repairing Numerical Bugs in Deep Learning Programs				

导师评语

（包含对论文的性质、难度、分量、综合训练等是否符合培养目标的目的等评价）

张煜皓的本科毕业论文对于深度学习系统的测试进行了系统性的研究。张煜皓首先在本研项目中研究了深度学习系统缺陷的特点，发表了一篇 ISSTA 论文（CCF A 类）。然后在本科毕业设计中对于其中最重要的一类缺陷：数值缺陷进行了详细研究，提出了数值缺陷的检测与修复方法。不同于传统方法，张煜皓的方法充分利用了深度学习系统需要经过较长时间的训练这一特点，讲测试过程和训练过程融为一体，在没有显著增加训练开销的情况下使得数值缺陷的问题可以更早暴露。总的来说，张煜皓的本科毕业论文所处理的问题新颖和重要，解决方案创新程度高，实现难度大，方法效果较为显著。论文行文流畅，逻辑性强，是一篇优秀的本科毕业论文。

导师签名：

年　　月　　日

版权声明

摘要

深度学习的应用已经渐渐地在很多重要领域中进行部署，如自动驾驶系统、人脸识别系统等。有缺陷的深度学习应用会导致不可挽回的灾难。本文在先前的实证研究基础上，针对深度学习程序缺陷中的一类缺陷——数值缺陷，提出了一种快速高效的检测方法。这种方法能够在深度学习的训练阶段的早期检测到数值错误的出现，为开发者节省了巨大的时间和计算资源的消耗。接着，通过在实证性研究中挖掘数值缺陷和其修复的模式，通过模式匹配的方法，修复已经检测的错误。

本文基于先前实证研究工作提供的数据集中的 13 个数值错误外加 TensorFuzz 工具的提供的 1 个数值错误进行了实验。在所有的 14 个数值缺陷中，我们成功检测了 9 个数值缺陷，达到了 **64.29%** 的检测召回率，并修复了其中的 9 个数值缺陷，达到了 **100.00%** 的修复成功率。相较于开发者常用的正常训练方法，该方法在 2 个缺陷程序上达到了 **59.44** 倍和 **20.08** 倍的加速比，且在 6 个缺陷程序上达到了超过 **3** 倍的加速比；从训练轮次上看，该方法在 8 个缺陷程序上减少了 **67.09%-99.99%** 的训练轮次。此外，在正确的网络模型上应用我们的算法，比正常训练的时间慢了 **41.90%**，这意味着我们算法在正确模型上的检测的额外开销尚可以接受，适合部署到开发环境中给开发者使用。

关键词: 深度学习，数值缺陷，缺陷检测，缺陷修复，TensorFlow

Abstract

Deep learning applications become increasingly popular in important domains such as self-driving systems and facial identity systems. Defective deep learning applications may lead to catastrophic consequences. Based on the previous empirical study of bugs in deep learning applications built on top of TensorFlow, we propose a fast and efficient detection method for numerical bugs in deep learning programs. This method can detect numerical bugs in an early stage of the training process, saving much time and computational resources for developers.

We conducted experiments based on 13 buggy programs provided by the previous empirical study and 1 buggy program provided by TensorFuzz. Out of all 14 numerical bugs, we successfully detected 9 numerical bugs, achieving **64.29%** detection recall. And we successfully repaired 9 numerical bugs, achieving **100.00%** repair recall. Compared to the normal training process adopted by developers, our method accelerates **59.44X** and **20.08X** in 2 buggy programs and achieves **3X** acceleration in 6 buggy programs. In terms of training epochs, our method reduces **67.09%-99.99%** training epochs in 8 buggy programs. Moreover, our method has a **41.90%** overhead when applied to correct deep learning programs, which means our method is suitable to deploy at developing environments for developers to use.

Key Words: Deep Learning, Numerical Bugs, Bug Detection, Bug Repair, TensorFlow

全文目录

摘要 1
Abstract 2
全文目录 3
第一章 绪论 4
第二章 相关工作 7
第三章 研究动机 9
1. 神经网络简介 9
2. 研究基础 10
3. 研究动机实例 11
3.1 动机程序中的数值缺陷简介 11
3.2 动机程序中的数值缺陷检测与修复 12
第四章 算法介绍 15
1. 模式匹配算法 15
2. 触发加速算法 17
2.1 输入修改算法 18
2.2 网络参数引导算法 19
2.3 复合迭代算法 21
第五章 实验设置与结果 23
1. 实验设置 23
1.1 实验数据来源 23
1.2 实验设备 23
1.3 实验方法 23
2. 实验结果 23
2.1 研究问题一 23
2.2 研究问题二 24
2.3 研究问题三 29
第六章 未来工作展望 31
第七章 结论 33
参考文献 34
本科期间的主要工作和成果 37
致谢 38
北京大学学位论文原创性声明和使用授权说明 39

第一章 绪论

深度学习的应用已经渐渐地在很多关键性领域中进行部署，如自动驾驶系统、人脸识别系统等。但是深度学习应用的编程模型和运行方式很大程度上有别于传统的应用。在传统的应用中，程序员们编写一个程序来直接解决目标需求。而在深度学习应用中，程序员们编写的程序体现的是神经网络的结构，以及神经网络训练的过程。在深度学习的开发过程中，也会面对一些在传统软件开发过程中不会遇到的情况：比如设置一个复杂的网络结构（也被称为计算图模型）。此外，神经网络的训练过程中有很多循环、矩阵运算等消耗资源的操作，而最后的结果也很依赖于超参数的设置与微调。许多开源的框架诸如 TensorFlow [1]，Caffe [2]，MXNet [3]，PyTorch [4] 以及 Theano [5] 都被用于搭建这些深度学习的应用。

有缺陷的深度学习应用会导致不可挽回的灾难：比如说一个有缺陷的自动驾驶汽车系统会导致车祸的发生 [6]，一个错误的人脸识别系统会导致巨大的财产损失 [7]。深度学习应用的缺陷也不同与传统程序的缺陷，它们不仅仅会出现在程序代码之中，还可能出现在训练数据（训练数据集的不充分）、模型本身（易被对抗性样本攻击）、底层框架（深度学习开源框架，CUDA 库）以及执行环境（GPU、FPGA 等硬件）之中。在先前的研究中 [8]，作者指出检测和修复深度学习程序中的缺陷是非常有挑战性的工作，并且传统软工领域的方法都不能有效地直接应用到深度学习程序缺陷的检测和修复上。因此，提升深度学习应用的安全性是刻不容缓的，并且这是一个有挑战的具有很大研究价值的课题。

深度学习的程序和框架本质上也是一种严重依赖于浮点运算的应用，这给软件的可靠性和安全性带来了极大挑战。浮点数是广泛使用的来表示实数的一种存储表示，它的计算先天就带有不精确性。比如说它们对于舍入和误差非常敏感，经常会有上溢和下溢的问题，甚至在不同的机器上不能重现同一个错误。而浮点数的错误已经导致了很多巨大的灾难 [9]。在先前的研究 [8] 公布的包含 157 个缺陷程序的数据集中，我们发现有 13 个程序中存在数值缺陷，这意味着在实际情况中，深度学习程序确实会出现浮点数带来的数值错误。

Google Brain 的 Odena 等人提出的 TensorFuzz 工具 [10]，通过模糊测试的方

法能够找到已经训练完毕的深度学习代码中的数值缺陷。但在实证性研究中得出的结论是,通常这类缺陷都是在训练过程中出现的:比如一个 StackOverflow 上的提问[1]就指出在训练到 8000 轮左右的时候出现了数值错误。仔细研究 TensorFuzz 论文中的实例之后，我们发现这个例子在训练时间足够长，训练轮次足够大的时候，也能够在训练过程中触发数值缺陷。综上，我们可以为高效数值缺陷检测工具提出两个具有挑战性的设计要求:

1. 该工具在训练阶段而不是测试阶段捕获数值缺陷引发的错误是一个现实的选择。此外，减少触发数值缺陷所需要的训练时间和训练轮次能够为深度学习开发者节省大量时间。这是因为通常这类深度学习的训练阶段要经历数小时甚至数天的时间，如果当程序运行到数天之后触发了数值错误，那么这几天的运行都是徒劳的，浪费了大量计算资源。

2. 该工具不能有很大的额外开销。因为加速触发数值缺陷的过程注定是动态执行的，如果程序没有数值缺陷，该工具还是会一直执行动态执行直到达到最大训练轮次。所以一个合理的要求就是，如果没有检测到数值缺陷之后应当返回一个训练完成的模型以减小额外开销。

本文克服了实证性研究中所阐述的困难，提出了一种能够高效地在训练阶段检测并修复深度学习程序代码中的数值缺陷的算法，并满足了上述两个具有挑战性的设计要求。首先，我们基于数据集挖掘出了 4 种有可能导致数值缺陷的模式。并且通过模式匹配的方法，寻找出计算图模型中可能出现缺陷的节点（下文也会称之为 “可疑节点”）。假如图中存在 “可疑节点”，则我们的算法能够通过三种触发加速算法：(1) 输入修改算法；(2) 网络参数引导算法；(3) 复合迭代算法来提前数值错误出现的时间。虽然通过模式匹配的方式发现的 “可疑节点” 并不一定会出现数值错误，但只要程序是没有数值缺陷的，我们的算法就能够返回一个正确训练的网络，不需要开发者再次训练。这种方法能够在深度学习的训练阶段的早期检测到数值错误的出现，为开发者节省了巨大的时间和计算资源的消耗。接着，通过已经挖掘到的数值缺陷模式和其被修复的模式，通过模式匹配的方法，修复已经检测的错误。

本文基于此实证研究工作提供的数据集中的 13 个数值错误外加 TensorFuzz 工具的提供的 1 个数值错误进行了实验。在所有的 14 个数值缺陷中，我们成功检测了 9 个数值缺陷，达到了 **64.29%** 的检测召回率，并修复了其中的 9 个数

[1] https://stackoverflow.com/questions/33699174

值缺陷，达到了 **100.00%** 的修复成功率。相较于开发者常用的正常训练方法，该方法在 2 个缺陷程序上达到了 **59.44** 倍和 **20.08** 倍的加速比，且在 6 个缺陷程序上达到了超过 **3** 倍的加速比；从训练轮次上看，该方法在 8 个缺陷程序上减少了 **67.09%-99.99%** 的训练轮次。此外，在正确的网络模型上应用我们的算法，比正常训练的时间慢了 **41.90%**，这意味着我们算法在正确模型上的检测的额外开销尚可以接受，适合部署到开发环境中给开发者使用。

下面简要介绍本文的结构：第三章，介绍了本文的研究动机，包括了深度学习神经网络的简介，对于先前的实证性研究中发现的困难的概述和一个动机实例；第四章，详细介绍了本文的算法；第五章，介绍了本文的实验设置以及实验的结果。

第二章 相关工作

深度学习程序代码缺陷： Zhang 等人 [8] 发表了第一篇对于 TensorFlow 程序缺陷的实证性研究，该研究中指出深度学习缺陷的检测与调试对于传统软工领域提出了重大的挑战，这一工作也是本论文的基础。此外，Google Brain 的 Odena 等人提出的 TensorFuzz 工具 [10]，通过模糊测试的方法能够找到已经训练完毕的深度学习代码中的缺陷，缺陷包括数值缺陷、量化后神经网络中的缺陷等。该方法旨在在已经训练完毕的深度学习代码之中寻找缺陷，本文的方法能够在训练过程中加速触发深度学习程序中的数值缺陷，能够更加快速高效地帮助程序员检测缺陷，更加适合部署到开发环境之中。

深度学习模型鲁棒性： 在有关深度学习模型鲁棒性的文章之中，一部分文章旨在通过特殊的攻击方式生成对抗样本来使得神经网络的准确率降低: Goodfellow 等人首次提出了深度学习模型的白盒攻击（即攻击者拥有所有神经网络的信息）FGSM，实验显示 FGSM 方法能够非常快速地攻击没有防御措施的神经网络 [11]。Carlini 等人提出了非常有效的白盒攻击方法 C&W，能够规避绝大多数包括防御性蒸馏在内的防御手段，但是花费时间较长 [12]。Madry 等人提出了多次使用 FGSM 攻击来进行白盒攻击的模式 PGD，能够非常有效率地攻击带有防御措施的神经网络 [13]。Papernot 等人首次提出了深度学习模型的黑盒攻击（即攻击者仅仅掌握神经网络的输出，而对网络模型的参数一无所知），实验显示没有防御措施的深度学习模型也会很大程度上受到黑盒攻击的影响 [14]。

另一部分文章旨在通过特殊的防御模型来提升神经网络对于对抗样本的鲁棒性: Meng [15] 等人提出了 MagNet 这种防御模型。MagNet 首先检测一个给定的输入是不是对抗性样本，如果是就会先经过一些修改再送入神经网络中进行分类。另一个相似的防御模型是 AE-GAN [16]，它使用了生成对抗网络（GAN）[17] 来生成与给定的对抗性样本相似的图片，并对 GAN 生成之后的图片进行分类。Tramer [18] 等人提出了对抗样本训练的另一个版本：联合对抗样本训练，其主要思想是使用对抗样本训练多个与目标模型结构不同的预训练网络来提升目标模型的鲁棒性。Srisakaokul [19] 等人提出了防御模型 MulDef，它使用对抗样本训练了多个与目标模型结构相同的网络构成一个大的集体，并在分类的时候随

机选择集体中的一个模型返回结果，使用随机性来迷惑白盒攻击者。

深度学习模型测试： Pei 等人的工作 DeepXplore [20] 首次提出了神经元覆盖的概念，并使用神经元覆盖来引导测试数据生成，并使用多模型的差异化测试来生成测试预言 (Test Oracle)。Tian 等人的工作 DeepTest [20] 使用图片的变换来最大化神经元覆盖以求得生成更好的测试数据，并使用变质关系 (Metamorphic Relation) 来作为测试预言。Sun 等人的工作 DeepCover [21] 借鉴了软工领域传统的 MC/DC 覆盖，提出了新的神经元覆盖：SS 覆盖、DS 覆盖和 DV 覆盖。Ma 等人的工作 DeepGauge [22] 在神经元覆盖之外提出了基于一层神经层的覆盖：Top-k 神经元覆盖和 Top-k 神经元模式，并且提出了新的神经元覆盖：k 选择覆盖和边界神经元覆盖。Sun 等人的工作 [23] 提出了使用 Concolic 测试方法在测试要求下生成测试数据，并且首次在深度学习模型测试领域提出使用 Lipschitz 要求来生成测试数据。Ma 等人的工作 DeepCT [24] 提出了组合测试覆盖标准并用它来指导测试数据生成。最近，Li 等人的研究工作也指出这种**神经元覆盖**的测试方法所带来的提升并不本质，甚至带有误导性 [25]。

数值错误缺陷修复： 数值缺陷也是软件工程研究中的一个重点关注的缺陷类型。Franco 等人的实证性研究 [26] 详细统计并讨论了现实中的数值缺陷的类型、出现频率、症状和修复。Chiang 等人提出了一种启发式搜索方法 BGRT 来生成输入使得最大化误差，在测试集的 48 个例子中，BGRT 在 45 个例子上获得了最好的效果 [27]。Barr 等人提出了 Ariadne [28] 这个工具，它使用符号执行的方法来生成输入触发浮点异常。Ariadne 现将程序转换成一些控制流条件和执行路径，通过满足控制流条件与数值运算缺陷的约束来找到那些合适的输入浮点数，并将浮点数带入原始程序中进行测试。Sanchez-Stern 等人提出的 Herbgrind [29] 工具，能够帮助 C/C++ 或者 Fortran 的开发者找到并且定位导致数值错误的根本原因。Herbgrind 能够动态地跟踪操作与程序输出之间的依赖关系来减少假阳性的情况，并且对于错误的计算操作进行抽象，简化程序分支，使得其能够高效地推广到几千行的大程序上。我们的方法与上述方法的本质不同在于，我们的方法是针对于深度学习程序中的数值缺陷检测与修复。上述方法都不能够推广到深度学习程序的场景中。

第三章 研究动机

1. 神经网络简介

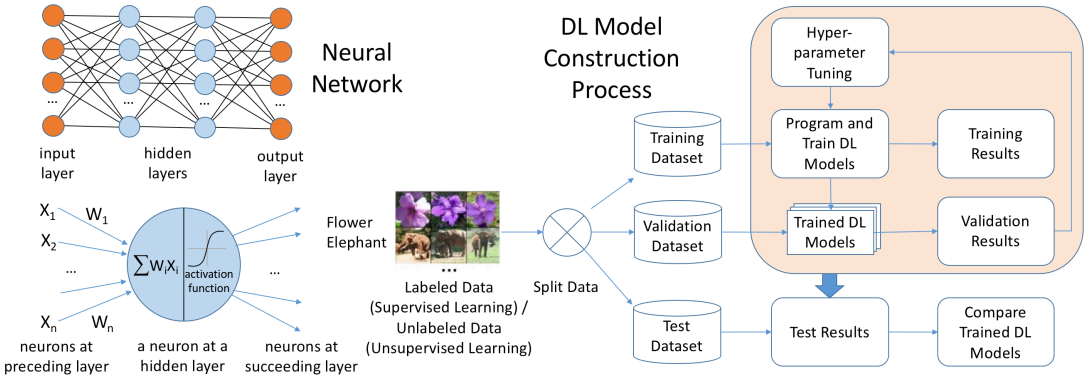


图 1 神经网络和神经元（图左）；深度学习模型的构建过程（图右）

```
# MNIST classifier
1.  import tensorflow as tf
…
2.  x = tf.placeholder(tf.float32, [None, 784])
3.  y_ = tf.placeholder(tf.float32, [None, 10])
4.  x_image = tf.reshape(x, [-1, 28, 28, 1])
5.  W_conv1 = weight_variable([5, 5, 1, 32])
6.  b_conv1 = bias_variable([32])
7.  h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
…
8.  W_fc2 = weight_variable([1, 1, 1024, 10])
9.  b_fc2 = bias_variable([10])
10. h_fc3 = tf.nn.relu(conv2d(h_fc1_drop, W_fc2) + b_fc2)     // node generates error
11. h_pool3 = avg_pool_7x7(h_fc3)
12. y_conv = tf.nn.softmax(tf.reshape(h_pool3, [-1, 10]))
13. cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))         // faulty statement
14. train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
…
15. sess.run(tf.initialize_all_variables())
16. for i in range(20000):
17.   batch = mnist.train.next_batch(50)
18.   if i%100 == 0:
19.     train_accuracy = accuracy.eval(feed_dict={
          x:batch[0], y_: batch[1], keep_prob: 1.0})
20.     print("step %d, training accuracy %g"%(i, train_accuracy))
21.   train_step.run(feed_dict={
        x: batch[0], y_: batch[1], keep_prob: 0.5})
22. print("test accuracy %g"%accuracy.eval(feed_dict={
      x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

DL model training

DL model testing

图 2 StackOverflow #33699174 上一个有缺陷的 TensorFlow 深度学习程序

神经网络（图 1）是一种深度学习计算模型，它能够使用结构化的多层神经网络完成分类和回归这两种任务。每个神经元是一个独立的处理单元，它能够

接受上一层的神经元的输出作为输入，并对此输入作用一个非线性的激活函数，将作用后的值传入下一层神经网络。相邻神经元的连接都用一个 W_i 来定义连接的权值。一个深度学习的模型通常是用反向传播和梯度下降的方法来训练，进而部署到实际场景中。而这些训练的过程和方法通常是非确定性的。[30] 训练阶段实际上是搜索一个权重 W_i 的设置，能够在给定数据集上最小化损失函数。在有监督学习之中，损失函数（loss function[1]）被定义为计算模型输出和期望输出值的差距，并用其来估计误差值。比如说用交叉熵（cross entropy）来计算两个概率分布的距离来作为 loss 是一个很常见的应用。在无监督学习中，loss 可以被定义为底层自动编码器神经网络中的编码和解码表示之间的距离。

图 2展示了一个用 TensorFlow 框架编写的深度学习程序[2]。这个程序包含两个主要的阶段：构建计算图模型阶段和执行阶段。第一，在第 2 行至第 14 行之间，程序员构建了一个计算图模型。第二，程序员创建了一个 session 对象来启动训练图模型的执行。训练阶段能够被进一步分为两个阶段：训练阶段和测试阶段。在训练阶段之中（第 16 行至第 21 行），我们使用一组带标签的训练数据去训练神经网络，并且最小化模型的 loss，即最小化交叉熵。在训练阶段，整个计算图模型会被执行上百万次。当一个模型被训练完成之后，我们会在测试阶段使用测试数据集来测量训练完成的程序的准确率（第 22 行）。

2. 研究基础

先前的研究工作指出，对于程序员和研究人员来讲，检测和修复深度学习程序缺陷有如下困难 [8]：

1. 因为深度学习过程先天就有随机性，所以对于程序正确性的衡量是通过测试一些统计意义上的数值（如准确率、loss 等），而不再是构建一部分测试集让程序在每个测试上都要有正确输出。所以我们需要新的测试技术来支持对于深度学习程序正确性的衡量。

2. 因为神经网络上的计算强度非常之大，参数变量非常多，coincidental correctness [31–35] 会比传统程序出现得更加平凡，但更加难以发现。

[1] https://en.wikipedia.org/wiki/Loss_function. 在下文中，我们将使用 loss 来指代损失函数。

[2] https://github.com/loliverhennigh/All-Convnet-TensorFlow-MNIST-Tutorial/blob/master/all_conv_mnist.py

3. 非确定性在训练阶段是非常常见的，这就意味着复现程序运行结果非常困难。

4. 因为神经网络之间的紧密连接性，传统的 slicing 调试技术 [[36] 就难以起作用，对新的调试技术的研究就显得非常重要。

5. 因为神经网络难以解释的黑盒特性，用户并不能判定在程序执行过程中某个状态上的某个神经元的值是否正确。调试方法通常局限于黑盒地替换超参数或者调换训练数据。

针对上述检测和修复深度学习程序缺陷的困难，本文采用了以下手段规避和解决相应的困难：

- **针对困难 [1, 5]：** 因为针对的是数值缺陷，所以一旦出现 INF，NAN 等值（下文也会称之为“错误值”）就可以判定是否有缺陷，故在这个问题上，还是可以等同于传统程序测试方法定义程序的正确性。并且无论网络的可解释性如何，只要节点中出现错误值，就可以判断程序是否有缺陷。

- **针对困难 [2 ,4]：** 本文采用了一种快速的动态执行技术，能够将程序缺陷被触发的时间大大提前。即使面对巨大的全联接层，这种技术还是能够快速地检测出深度学习程序中的缺陷。

3. 研究动机实例

3.1 动机程序中的数值缺陷简介

图 2展示了一个出现错误的实例，它取自之前的实证性研究，并且在那份研究中被归类为第一类错误：错误的模型参数或机构（Incorrect Model Parameter or Structure）。这类问题通常是由错误的模型参数（如学习率），或是错误的模型（如缺失了计算图节点）引起的。这是一类在 TensorFlow 程序中独特的错误，会引起在程序执行阶段的错误行为。它的缺陷症状通常是过低的正确率和巨大的 loss 值。

通过观察上述错误，实证研究中也提出了以下三个结论：

1. 通常需要很多个训练轮次才能够触发错误的行为，而这轮次个数还依赖于超参数的设置。通常来说错误的训练图节点被执行了上百次，但是触发错误的频率很低。

```
13. - cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
13. + cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv+1e-9))
```

图 3 对图 2缺陷的一个建议修复方法

2. 在实证研究收集的数据集中，错误通常发生在训练阶段。在测试神经网络的过程中，很少发生这种错误。

3. 如果它的缺陷症状是程序异常，那么错误信息通常具有迷惑性。通常来说错误信息中汇报的错误语句通常不是有缺陷的语句本身。StackOverflow 上的讨论体现出即使是在较小的程序片段上的错误定位也是困难的。

在这个实际的例子中，程序在原有的数据集上运行正确，但是在另一个数据集上发生了运行时错误。所以程序员将其发表在 StackOverflow 上寻求帮助。对于出现错误的数据集，在执行到大约 8000 训练轮次的时候，程序发生了 “Relu 节点的输入不是有限的”（“ReluGrad input is not finite”）这个运行时错误。通过修改在第 14 行设置的学习率为 10^{-3}，训练的轮次会大大减少。错误信息将错误报告在 *h_fc3*（第 10 行）这个计算图节点上，但实际上错误节点出现在第 13 行。当 *y_conv* 中有一维为 0 时，*tf.log* 就会执行 $\log 0$ 这个操作，从而产生 NAN 这个错误值。图 3展示了一个在 StackOverflow 上提供的修复。这个修复实际上在计算图模型中隐式地增加了一个 *tf.add* 操作，将 *y_conv* 增加了一个很小的常数值 10^{-9}。

3.2 动机程序中的数值缺陷检测与修复

针对这个实例，下面将介绍我们算法的执行流程：

- 第一步：通过已经挖掘出的模式，进行模式匹配。我们的算法检测到第 12 行的 *tf.nn.softmax*, 和第 13 行的 *tf.log* 是一个可疑的模式。这是因为 *tf.nn.softmax* 可能会输出 0，而 *tf.log* 的定义域中不存在 0，故会出现 NAN 的数值错误。其中 *tf.log* 是一个 “可疑节点”，而 *tf.nn.softmax(y_conv)* 的输入 *y_conv* 则是我们需要引导的值。

- 第二步，针对该模式和其特性，我们想要引导输入 *y_conv* 的最大值与最小值的差距变大。即最大化

$$\max(\mathbf{y\_conv}) - \min(\mathbf{y\_conv}) \tag{3.1}$$

这一步的原因是：*tf.nn.softmax(x)* 进行的操作是针对每个 x_i 计算

$$\frac{e^{x_i}}{\sum e^{x_j}} \tag{3.2}$$

而又因为

$$0 < \frac{e^{x_i}}{\sum e^{x_j}} \leq \frac{e^{x_i}}{\max e^{x_j}}$$

最大化 3.1式导致的结果是最小化 $\frac{e^{x_i}}{\max e^{x_j}}$，导致浮点数只能将其表示为 0（注意这个式子是非负的）。同理 3.2式也会是 0。但因为 *tf.log* 的定义域中不存在 0，故有可能出现 NAN 的数值错误。

- 第三步，我们的算法会在每次训练之中通过三种触发加速算法：（1）输入修改算法；（2）网络参数引导算法；（3）复合迭代算法来尽早地触发程序中的数值错误。

- 第四步，如果在某个训练轮次之中发现了数值错误，则将该错误汇报给程序员，并再返回一个可能的修复模式。反之，如果在跑完所有训练轮次之后还是没有能够发现数值错误，则返回该程序正确的信息，并且返回这个经过训练的网络。

  在这个例子之中，我们的触发加速算法会触发程序中的数值缺陷，因此会将错误报告给程序员。并且，根据我们的修复模板，会将 *tf.log(T)* 修改为 *tf.log(T + ZERO_F)* 或者 *tf.log(tf.clip_by_value(T, ZERO_F, MAX_F))*。其中 *MAX_F* 代表的是比较大的浮点数，比如 10^9；其中 *ZERO_F* 代表的是接近 0 的正值，比如 10^{-9}。

而在这个例子中，如果不使用我们的算法，会经过 281 秒的时间和 22190.67 的训练轮次之后找到数值错误。如果应用了我们的算法之后，最快将会在 91.67 秒的时间，或者最少会在使用 4750 的训练轮次之后找到数值错误。也就是说我们的算法在训练时间上加速了 3.07 倍，或者说能够减少约 78.59% 的训练轮次，这体现出我们的算法能够为开发者减少巨大的时间开销与运行深度学习程序时的算力成本。

针对图 3中的修复，注意到这个 *tf.nn.softmax*, *tf.log* 的模式仍然未消失，因此我们的模式匹配算法仍然不能够判定这是一个没有数值缺陷的程序。于是，把我们的算法应用到修复后的程序上时，仍然需要经历上述的几个步骤。有所不同的是我们的程序在跑完所有的训练轮次之后也未能找到缺陷，所以算法报告

程序中没有数值缺陷，算法结束，并返回经过训练的网络。经过实验，我们发现算法返回的网络与平常训练的网络在测试集上的准确率上几乎没有差别，并且我们的算法仅有 49.26% 的额外开销。这意味着我们的算法可以高效地部署到开发环境之中，为深度学习软件的开发者提供巨大帮助。

第四章 算法介绍

本章将着重介绍我们的算法。在此之前，先介绍一下算法的流程。在上一章的结尾我们已经将该流程实例化到了动机程序之中，下面是这个流程的通用版本：

- 第一步：通过已经挖掘出的模式，进行模式匹配。模式匹配之后也会得出“可疑节点”。
- 第二步，针对该“可疑节点”，我们会分析该模式的特性，并确定按照什么方向引导“可疑节点”的输入走向触发数值错误。这种方向一般都能表示成最大化某个函数的形式，称之为“最大化引导函数”。
- 第三步，我们的算法会在每次训练之中通过三种触发加速算法：（1）输入修改算法；（2）网络参数引导算法；（3）复合迭代算法来尽早地触发程序中的数值错误。
- 第四步，如果在某个训练轮次之中发现了数值错误，则将该错误汇报给程序员，并再返回一个可能的修复模式。反之，如果在跑完所有训练轮次之后还是没有能够发现数值错误，则返回该程序正确的信息，并且返回这个经过训练的网络。

由此可以看到我们的算法主要有两个部分：一、**模式匹配**，我们算法的初期检测“可疑节点”和最后的修复都用到了模式匹配的算法；二、**触发加速**是我们算法的核心。我们设计了三种触发加速算法来尽早地触发程序中的数值错误。

接下来将分两节介绍我们算法的模式匹配和触发加速部分。

1. 模式匹配算法

模式的获得主要是通过观察并分析现有的缺陷形态，并加以总结得出的。这篇论文总共给出了 4 种有可能导致数值缺陷的模式。表 1给出了缺陷模式及其“可疑节点”和引导方向函数。

表 1 缺陷模式及其“可疑节点”和引导方向函数

缺陷模式	可疑节点	输入	引导方向函数（最大化）
exp(x)	*exp(x)*	x	*max(x)*
y = *softmax(x), log(y)*	*log(y)*	x	*max(x) - min(x)*
y = *x / sum(x) (x > 0), log(y)*	*log(y)*	x	*max(x) - min(x)*
log(x)	*log(x)*	x	*max(-x)*

表 2 缺陷模式及其修复模式

缺陷模式	修复模式
exp(x)	*exp(x)* → *clip_value(exp(x), -MAX_F, MAX_F)*
y = *softmax(x), log(y)*	*log(y)* → *log(y + ZERO_F)* *log(y)* → *log(clip_value(y, ZERO_F, MAX_F))*
y = *x / sum(x) (x > 0),* *log(y)*	*log(y)* → *log(y + ZERO_F)* *log(y)* → *log(clip_value(y, ZERO_F, MAX_F))*
log(x)	*log(x)* → *log(clip_value(x, ZERO_F, MAX_F))*

在第三章中，我们介绍了第 2 条模式的来由。接下来我们将依次介绍第 1, 3, 4 条模式的来由：(1)，*tf.exp(x)* 在输入 x 过大的时候会出现运算之后结果太大，而浮点数无法存储的情况，导致 INF 的出现。所以引导的方向也就是让 x 最大化。(3)，与第三章中介绍的 (2) 相类似，*tf.log(y)* 在 y（注意 y 是非负的）中存在 0 时会出现 NAN。而

$$0 < \frac{x_i}{\sum x_j} \leq \frac{x_i}{\max x_j}$$

最大化 $\max(x) - \min(x)$ 式导致的结果是最小化 $\frac{x_i}{\max x_j}$，导致浮点数只能将其表示为 0。同理 y 也会是 0。(4)，*tf.log(x)* 在 x 中存在负数时会出现 NAN。所以引导的方向就是让 x 最小化，即最大化-x。

我们的算法基于这样一个假设：**如果能够最大化引导方向的函数，则一定能**

够在有限步数之内触发深度学习程序的数值缺陷。 即

$$
\begin{aligned}
max(x) > T_0 &\rightarrow exp(x) = INF \\
max(x) - min(x) > T_1 &\rightarrow log(y) = NAN \\
max(x) - min(x) > T_2 &\rightarrow log(y) = NAN \\
max(-x) \geq 0 &\rightarrow log(x) = NAN
\end{aligned}
$$

其中 T_0, T_1, T_2 均为一些固定的阈值。

注意到如果发现了第 2 条（第 3 条）模式匹配到了程序的某一部分的计算图，那么第 4 条模式也能够匹配它。但我们的算法只考虑最严格的那个匹配，所以在这种情况下，我们不再继续考虑第 4 条模式的匹配。

表 2给出了缺陷模式及其可行的修复模式，其中 MAX_F 代表的是比较大的浮点数，比如 10^9；其中 $ZERO\_F$ 代表的是接近 0 的正值，比如 10^{-9}。

2. 触发加速算法

在触发加速阶段,我们实际上想要让一个“可疑节点”的输出出现 INF, NAN 等数值错误。我们可以将这个“可疑节点”的输出刻画成一个函数 $f(in, \theta)$ 其中，in 表示一个能够触发“可疑节点”的输入，θ 表示整个神经网络模型的参数设置。自然地，使得 f 能够出现数值错误，我们可以

1. 寻找一个合适的输入 in。对于这个输入的选择，我们并不局限于测试数据集 $\mathbf{T}$，而是在某个数据 x（$x \in \mathbf{T}$）的高维邻域内寻找一个合适的值即可。下式是把邻域的阈值设为 eps，并用 L2 度量[1]来表示的一个限制：

$$|in - x|^2 \leq eps$$

2. 寻找一个合适的神经网络模型的参数 θ。这个参数的选择应该是一个训练阶段的合法状态。但是如何刻画状态是否合法非常困难：即给出任意一个 θ'，我们无法确定是否存在一个训练流程中的某个阶段，网络的参数是 θ'。但如果我们真实地模拟一个训练过程，并且在不修改最小化 loss 函数这个优化条件的情况下，所有模拟得到的模型参数都会是合法的。训练过程中，

[1] http://mathworld.wolfram.com/L2-Norm.html

只有训练数据批次是能够被我们更改的。通常的训练数据是通过随机挑选的方法进行的，其目的是让每个训练数据被网络训练的次数尽可能平均。我们所更改之后的挑选训练数据的方法也应该保证一定的随机性平均性。

3. 结合以上两种方法。我们能够先固定输入 in_0，修改网络初识参数 θ_0 得到 θ_1，如此进行一轮迭代；接着固定 θ_1 再修改 in_0，得到 in_1；如此反复直到 $f(in_n, \theta_n)$ 的输出出现数值错误为止。下文称这种方法为复合迭代算法。

接下来我们将介绍修改输入 in 的加速触发数值错误的算法，引导网络参数 θ 的加速触发数值错误的算法，和复合迭代算法。

2.1 输入修改算法

该算法主要解决的问题：固定一个从训练数据集 $\mathbf{T}$ 中选定的初始输入 x（令 $in_0 = x$），在网络训练过程中，逐步修改 in_0 至 in_n，使得 $f(in_n, \theta')$（θ' 表示按照原先训练方法得到的合法网络模型参数）出现数值错误。而通过最大化引导方向函数 g，我们能让 “可疑节点”$f(in_n, \theta')$ 出现数值错误。因此这个问题转换成了：固定一个从训练数据集 $\mathbf{T}$ 中选定的初始输入 x（令 $in_0 = x$），在网络训练过程中，逐步修改 in_0 至 in_n，最大化我们的引导方向函数 $g(in_n, \theta')$。

最大化一个函数可以通过沿着梯度上升的方法进行。注意到引导方向函数 $g(in, \theta')$，在 θ’ 暂时固定的时候（视为常数），可以视为一个只含一个变量 in 的函数。我们可以求 g 关于 in 的导数，记为 $\frac{\partial g}{\partial in}(in, \theta')$。第 i 次更新 in_i 的式子如下

$$in_i = in_{i-1} + \frac{\partial g}{\partial in}(in_{i-1}, \theta') * step \tag{4.1}$$

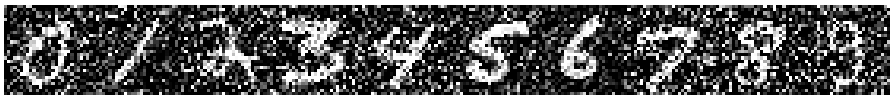
其中 $step$ 是更新的步长，取决于阈值 eps 的设置。

总结一下输入修改算法的算法流程：

- 第一步，选定 $\mathbf{T}$ 中的一个输入 x（通常是从中随机选择一个），记为 in_0。
- 第二步，假设当前进行到第 i 轮（初始情况下 $i = 1$）。按照 4.1式的方式更新 in_i。
- 第三步，检测 $f(in_i, \theta')$ 是否出现数值错误。如果出现，则程序终止；反之进入第四步。

- 第四步，按照原先的方法（通常是随机挑选）挑选一组训练数据批次来训练神经网络。并返回第二步。

(a) 原 MNIST 图片



(b) 输入修改算法修改后的 MNIST 图片

图 4 输入修改算法的触发图片展示

我们可视化了触发编号为 TensorFuzz-1 的深度学习缺陷程序的输入，发现这些输入例子都是非常合理的输入。从图 4中可以看到修改过的输入出现了很多噪声点，不过这并不妨碍人眼对它的识别。

2.2 网络参数引导算法

该算法主要解决的问题：从网络参数的初始状态 θ_0，在网络训练过程中，合理地选择训练批次，逐步修改 θ_0 至 θ_n，并从训练数据集 $\mathbf{T}$ 之中选择输入 x，使得 $f(x,\theta_n)$ 出现数值错误。同理，这个问题也可以被我们的假设转换成：从网络参数的初始状态 θ_0，如何在网络训练过程中，合理地选择训练批次，逐步修改 θ_0 至 θ_n，并从训练数据集 $\mathbf{T}$ 之中选择输入 x，最大化我们的引导方向函数 $g(x,\theta_n)$。

同理，使用梯度上升的方法更新网络的参数从而最大化引导方向函数 $g(x,\theta)$ 也是我们的选择。但是我们不能够直接像先前的输入修改算法一样，通过梯度上升直接修改网络的参数。我们仍然得遵循原先的最小化 loss 函数的原则，通过合理选择训练数据来解决这个难题。

在不考虑随机性的情况下，训练过程会严格按照 loss 函数减小的方向去更新网络的参数。为了描述简单，我们只考虑训练批次中只含有一个训练数据 tr_0 的情况。也就是说在正常训练情况下，网络参数 θ_i 在第 i 步会转变成

$$\theta_i = \theta_{i-1} - \frac{\partial loss}{\partial \theta}(tr_0, \theta_{i-1}) * lr \tag{4.2}$$

其中，4.2式中的 $\frac{\partial loss}{\partial \theta}$ 表示 $loss$ 函数对网络参数 θ 的导数，而 lr 表示每次更新的步长，也就是学习率。

为了最大化引导方向函数 g，我们期望的 θ_i 的更新方向为：

$$\theta_i = \theta_{i-1} + \frac{\partial g}{\partial \theta}(x_i, \theta_{i-1}) * lr \tag{4.3}$$

其中 x_i 是我们从 $\mathbf{T}$ 中选择的当前最大化 g 的输入。我们有相应的启发式规则来最大化引导函数 g，即使不是严格地按照 $\frac{\partial g}{\partial \theta}(x_i, \theta_{i-1})$ 这个梯度进行更新 θ_i 的。注意到 4.3式的右半部分是一个或多个高维向量，因此先定义两个高维向量 $\mathbf{X}$ 和 $\mathbf{Y}$ 之间的点乘 $\cdot$ 操作，以二维为例：

$$\mathbf{X} \cdot \mathbf{Y} \triangleq \sum_i \sum_j \mathbf{X}_{i,j} \mathbf{Y}_{i,j}$$

启发式规则：设实际更新的 θ_i 为 $\Delta\theta$，即 $\theta_i = \theta_{i-1} + \Delta\theta * lr$。我们称 $\Delta\theta$ 能够最大化 g，当且仅当

$$\Delta\theta \cdot \frac{\partial g}{\partial \theta}(x_i, \theta_{i-1}) > 0$$

根据这个规则，只需满足

$$-\frac{\partial loss}{\partial \theta}(tr_0, \theta_{i-1}) \cdot \frac{\partial g}{\partial \theta}(x_i, \theta_{i-1}) > 0 \tag{4.4}$$

即可。接着我们将 4.4式从一个训练数据推广到一个训练数据批次的情况：

$$H(tr) \triangleq \sum_{0 \le j < m} -\frac{\partial loss}{\partial \theta}(tr_j, \theta_{i-1}) \cdot \frac{\partial g}{\partial \theta}(x_i, \theta_{i-1}) > 0 \tag{4.5}$$

若训练数据批次 $tr_0, tr_1, \ldots, tr_{m-1}$ 满足 4.5式，就可以沿着引导函数 g 梯度的方向进行上升。我们通过随机选择 B 组训练数据

$$\begin{aligned} tr^{(0)} &= tr_0^{(0)}, tr_1^{(0)}, \ldots, tr_{m-1}^{(0)} \\ tr^{(1)} &= tr_0^{(1)}, tr_1^{(1)}, \ldots, tr_{m-1}^{(1)} \\ &\cdots \\ tr^{(B-1)} &= tr_0^{(B-1)}, tr_1^{(B-1)}, \ldots, tr_{m-1}^{(B-1)} \end{aligned}$$

并求最优的 $tr^{(best)} = argmax_{x \in \{tr^{(0)}, \ldots, tr^{(B-1)}\}} H(x)$ 的方式来尽可能优地（尽可能最大化引导函数 g）以及尽可能随机地（尽可能使每个训练数据都可以被训练到）选择训练数据批次 $tr^{(best)}$。

总结一下网络参数引导算法的算法流程：

- 第一步，假设当前进行到第 i 轮（初始情况下 $i=1$ 且网络参数处于 θ_0 的状态）。选择 $\mathbf{T}$ 中的一个最好的输入 x_i，使得 $g(x_i,\theta_{i-1})$ 最大化。即

$$x_i = argmax_{x\in\mathbf{T}} g(x,\theta_{i-1})$$

- 第二步，检测 $f(x_i,\theta_{i-1})$ 是否出现数值错误。如果出现，则程序终止；反之进入第三步。

- 第三步，根据选定的 x_i，通过上述方法从初识训练数据集 $\mathbf{T}$ 选取训练数据批次 $tr^{(best)}$ 来训练网络得到参数 θ_i。并返回第一步。

2.3 复合迭代算法

该算法主要解决的问题：从网络参数的初始状态 θ_0，如何在网络训练过程中，合理地选择训练批次，逐步修改 θ_0 至 θ_n，并从训练数据集 $\mathbf{T}$ 之中选择输入 x（记为 in_0），逐步修改 in_0 至 in_n，使得 $f(in_n,\theta_n)$ 出现数值错误。同理，这个问题也可以被我们的假设转换成：从网络参数的初始状态 θ_0，如何在网络训练过程中，合理地选择训练批次，逐步修改 θ_0 至 θ_n，并从训练数据集 $\mathbf{T}$ 之中选择输入 x（记为 in_0），逐步修改 in_0 至 in_n，最大化我们的引导方向函数 $g(in_n,\theta_n)$。

先前两个小节已经介绍了输入修改算法和网络参数引导算法。故在此不再赘述。总结一下复合迭代算法的算法流程：

- 第一步，假设当前进行到第 i 轮（初始情况下 $i=1$ 且网络参数处于 θ_0 的状态）。选择修改后的训练数据集 $\mathbf{T}_{i-1}$（初始情况下 $\mathbf{T}_0=\mathbf{T}$）中的一个最好的输入 in_{i-1}，使得 $g(in_{i-1},\theta_{i-1})$ 最大化。即

$$in_{i-1} = argmax_{x\in\mathbf{T}_{i-1}} g(x,\theta_{i-1})$$

- 第二步，根据输入修改算法提供的方案更新 in_{i-1} 至 in_i：

$$in_i = in_{i-1} + \frac{\partial g}{\partial in}(in_{i-1},\theta_{i-1}) * step$$

  并把 $\mathbf{T}_{i-1}$ 中的 in_{i-1} 替换成 in_i，以供后面的迭代使用。

- 第三步，检测 $f(in_i,\theta_{i-1})$ 是否出现数值错误。如果出现，则程序终止；反之进入第四步。

- 第四步，根据选定的 $x_i=in_i$，通过在网络参数引导算法中介绍的方法选取训练数据批次 $tr^{(best)}$ 来训练网络得到参数 θ_i。并返回第一步。

注意结合上述两种算法还能够带来一个好处：在输入修改算法中，我们的初始输入 x 是事先给定的（一般来说是随机选择一个），因为我们无法得知在如此多训练数据和测试数据之中，哪个输入更能够最大化引导函数 g；而在网络参数引导算法中，我们会从训练数据和测试数据之中选择一个最好的输入，作为引导方向函数 g 的一个输入，这就天然地给了输入修改算法一个初始的输入。

第五章 实验设置与结果

1. 实验设置

1.1 实验数据来源

本次实验数据的来源是 TensorFuzz 中的 1 个缺陷程序和先前的实证研究数据集提供的 13 个缺陷程序，共计 14 个有数值缺陷的程序。这些程序都是搭建在 TensorFlow 的架构之上。表 3中给出了具体的数据来源。

1.2 实验设备

本次实验采用的 CPU 设备是 Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz，GPU 设备是 GeForce GTX 1080 Ti，操作系统是 Ubuntu 16.04.6 LTS (Xenial Xerus)。其中实验运行在 python3.5.2 之上，安装的 TensorFlow 版本为 1.8.0。

1.3 实验方法

在实验过程中，我们尝试回答了三个研究问题：

- **研究问题**一、本文的算法能够检测并修复多少数值缺陷?
- **研究问题**二、在成功检测的数值缺陷之中，本文的算法减少了多少时间与多少训练轮次?
- **研究问题**三、对于正确的程序，本文的算法有多少额外时间开销?

所有的实验都运行了三次取平均值，以减少随机性带来的影响。

2. 实验结果

2.1 研究问题一

从检测的角度来说，本文的算法一共检测到 9 个数值缺陷，详细情况如表 4中所示。我们分析了未能成功检测的数值缺陷，发现这些程序里并没有在第

表 3 实验数据集详情

数据集	TensorFuzz	实证研究数据集	
		Github	StackOverflow
个数	1	5	9
编号	TF-1	IPS-1, IPS-5, IPS-6 IPS-9	IPS-1, IPS-2, IPS-5, IPS-6, IPS-7, IPS-10, IPS-14, IPS-15, IPS-17

表 4 数值缺陷检测结果

数据集	TensorFuzz	实证研究数据集	
		Github	StackOverflow
成功检测 / 总数	1 / 1	3 / 4	5 / 9
成功检测编号	TF-1	IPS-1, IPS-6, IPS-9	IPS-1, IPS-2, IPS-6, IPS-7, IPS-14
未成功检测编号		IPS-5	IPS-5, IPS-10, IPS-15, IPS-17

四章之中介绍的 4 种数值错误的模式。这些未检测出的数值缺陷的原因都是由于过大的学习率或者是由于 loss 函数中缺小了 L2 正则化[1]。

从修复的角度来说，我们将检测出来的 9 个数值错误程序，应用上了第四章之中介绍的修复模板，并重新训练，发现修复后的所有的程序都没有出现数值错误，并且训练结果都是正确的[2]。

2.2 研究问题二

本算法的目标之一就是减少在训练过程中的开销，来加速数值错误的检测，帮助开发者更方便地进行深度学习编程。对于训练过程中的开销问题，我们定义了两种度量值：训练时间和训练轮次。训练时间顾名思义就是从程序开始运行到检测到数值错误的时间。训练轮次是从神经网络初始化之后的第一轮训练开始至检测到数值错误的轮次。

[1] https://en.wikipedia.org/wiki/Regularization_(mathematics)

[2] 因为数据集中有所有缺陷程序修复后的版本，这个正确性是通过与这些数据集中的修复程序做结果上的比较。

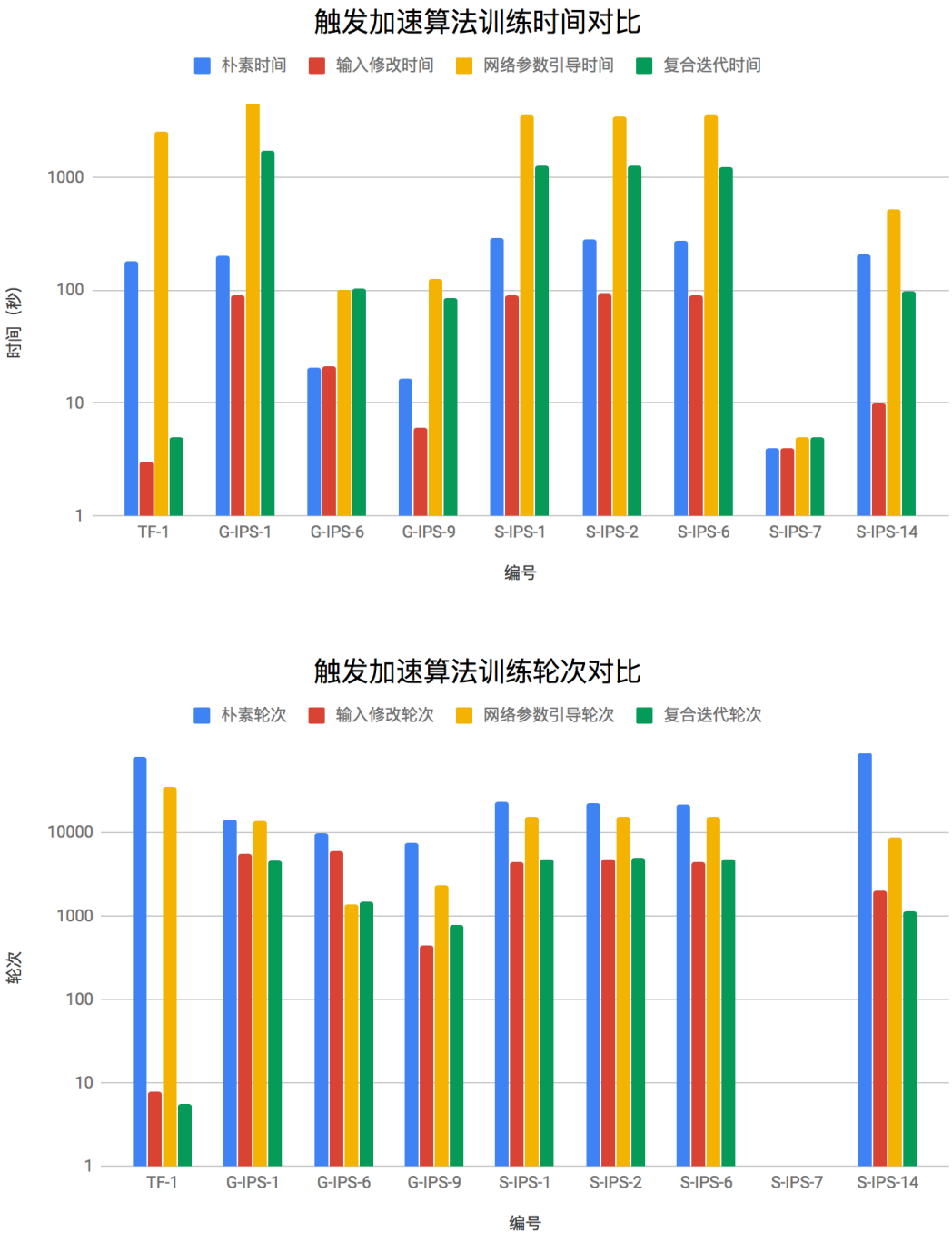


图 5 触发加速算法训练时间与训练轮次对比

表 5 训练时间与训练轮次比较

数据集	编号	训练时间				训练轮次			
		朴素	输入修改	网络参数引导	复合迭代	朴素	输入修改	网络参数引导	复合迭代
TensorFuzz	TF-1	178.33	**3**	2556.33	5	80681.33	8	35732	**5.67**
Github	IPS-1	203.33	**88.67**	4497	1713.67	14178	5627.67	13513.67	**4666.33**
	IPS-6	20.33	**21**	100.33	104.67	9754.33	5891	**1380.33**	1468.67
	IPS-9	16.33	**6**	124.67	84	7600.33	**440**	2318.67	789.67
StackOverflow	IPS-1	292	**90.33**	3526.33	1262.33	23330.33	**4477.33**	15324.67	4800
	IPS-2	281	**91.67**	3437.33	1250.33	22190.67	**4750**	15078.67	4929.33
	IPS-6	273.67	**89.33**	3572.33	1231.67	21684	**4457.33**	15488	4855.67
	IPS-7	**4**	**4**	5	5	**0**	**0**	**0**	**0**
	IPS-14	208	**10**	520	96.67	89047	2014	8751	**1122**

表 6 触发加速算法最优情况下训练时间与训练轮次减少详情

数据集	编号	训练时间			训练轮次		
		朴素	触发加速	加速倍	朴素	触发加速	减少比
TensorFuzz	TF-1	178.33	3	59.44	80681.33	5.67	99.99%
Github	IPS-1	203.33	88.67	2.29	14178	4666.33	67.09%
	IPS-6	20.33	21	0.97	9754.33	1380.33	85.85%
	IPS-9	16.33	6	2.72	7600.33	440	94.21%
StackOverflow	IPS-1	292	90.33	3.23	23330.33	4477.33	80.81%
	IPS-2	281	91.67	3.07	22190.67	4750	78.59%
	IPS-6	273.67	89.33	3.06	21684	4457.33	79.44%
	IPS-7	4	4	1.00	0	0	0.00%
	IPS-14	208	10	20.08	89047	1122	98.74%

我们衡量了第四章介绍的三种触发加速算法的训练时间和训练轮次，并与不使用任何加速技术的朴素结果进行比较。图 5的两个柱状图分别展示了训练时间的比较和训练轮次的比较。表 5是训练时间与训练轮次比较的数值详情，加粗的数字对应的是最优的算法。

从训练时间上来看，输入修改算法在每个数值缺陷程序上都要快于或者等于我们提出的其他两种触发加速算法。仅在 Github 的 IPS-6 中，三种触发加速算法都没有优于朴素结果，这或许是因为神经网络架构太过简单，导致速度的优势被随机的扰动掩盖，并没有体现出我们触发加速算法的优势。在 StackOverflow 的 IPS-7 中，由于该数值缺陷在第一轮就出现，所以我们算法的训练时间与朴素结果相差无几。在其他的数值缺陷程序中，都至少有一个我们的触发加速算法优于朴素结果。

从训练轮次上来看，三种触发加速算法：输入修改、网络参数引导、复合迭代算法都各自在一些数值缺陷程序上使用最少的轮次最早地触发了数值缺陷。同理，由于在 StackOverflow 的 IPS-7 中，由于该数值缺陷在第一轮就出现，所以我们算法的训练轮次与朴素结果是一样的。在其他所有的数值缺陷程序中，我们的触发加速算法都优于朴素结果。

其中，图 6和表 6展示了 2 种衡量值和 9 个数值缺陷程序的 18 种组合情况下，如果选取三种触发加速算法中最好的那个，能够产生训练时间的加速比和减少训练轮次的比例。从上表中可以得知，我们的输入修改算法在最好的 2 个缺陷程序上达到了 59.44 倍和 20.08 倍的加速比，而在其中的 6 个缺陷程序上达

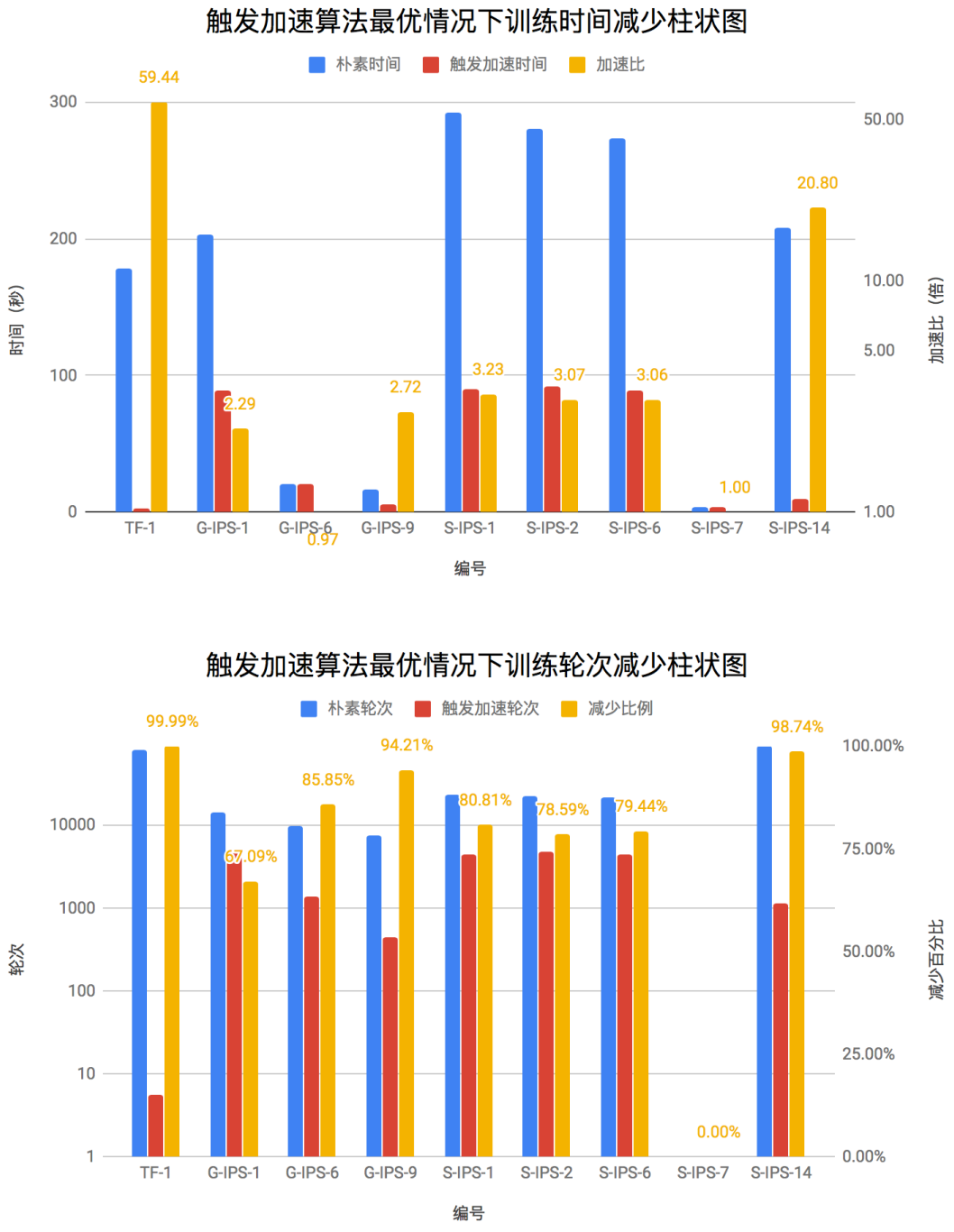


图 6 触发加速算法最优情况下训练时间和训练轮次减少柱状图

到了超过 3 倍的加速比；我们的输入修改算法、网络参数引导算法和复合迭代算法在 8 个缺陷程序上减少了 67.09%-99.99% 的训练轮次。这表明本文的三种触发加速算法都是非常有效的。但因为对于不同的数值缺陷程序，没有一个触发加速算法能够在训练时间和训练轮次两个衡量值上都要明显好于另外两个算法，所以开发者在使用我们的工具之时，还是要仔细谨慎地选择。

2.3 研究问题三

本文所提出的算法确实能够减少发现数值错误所需要的训练时间和训练轮次。但是并不是所有深度学习程序都会存在数值缺陷。如果开发者在一个没有任何缺陷的深度学习程序中应用我们的算法，额外开销又会是如何呢?

因为数据集中的这 9 个缺陷都有其相对应的修复版本，我们就在这些修复版本上进行试验，并比较了使用触发加速算法和直接运行修复版本两者的训练时间开销。值得注意的是，从上一小节就已经得知，从训练时间角度来看，三种触发加速算法中最快的是输入修改算法，故我们的实验就只涉及输入修改算法。

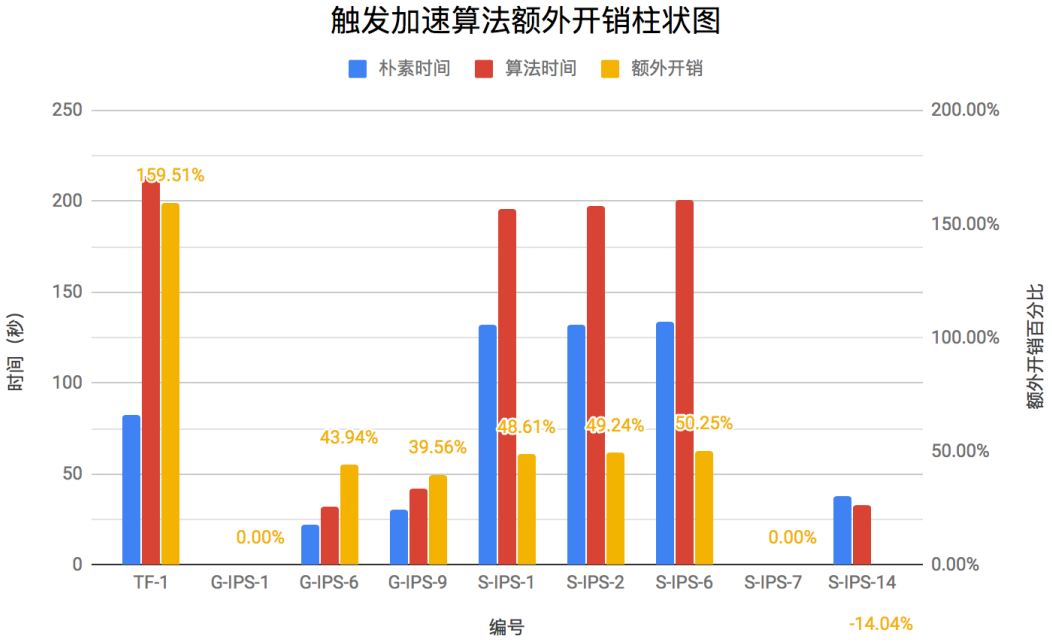


图 7 触发加速算法额外开销柱状图

图 7和表 7中给出了详细的比较结果。其中 Github 的 IPS-1 和 StackOverflow 的 IPS-7 由于在正确的版本中并没有找到相对应的模式，所以实际上我们并没有额外运行触发加速算法，所以统计上的额外开销为 0.00%。而 TensorFuzz 的 TF-1 由于匹配到了两个可能出现数值错误的模式，所以我们的触发加速算法运

表 7 触发加速算法额外开销详细情况

数据集	编号	朴素时间	算法时间	额外开销
TensorFuzz	TF-1	82.33	213.67	159.51%
Github	IPS-1	/	/	0.00%
	IPS-6	22	31.67	43.94%
	IPS-9	30.33	42.33	39.56%
StackOverflow	IPS-1	131.67	195.67	48.61%
	IPS-2	132	197	49.24%
	IPS-6	133.33	200.33	50.25%
	IPS-7	/	/	0.00%
	IPS-14	38	32.67	-14.04%

行了两次，故额外开销达到了 159.51%。最后关注到 StackOverflow 的 IPS-14，我们的触发加速算法的时间竟然优于了朴素运行的算法，这可能是因为在这个例子中我们的算法并没有增加额外的开销，而随机的扰动造成了这次额外开销为-14.04% 的意外。综上，我们算法在正确模型上的检测的额外开销并不大，平均意义下比朴素运行的时间慢了 **41.90%**，适合部署到开发环境中给开发者使用。

第六章 未来工作展望

本文的算法基于对深度学习程序数值缺陷模式的挖掘，来初步定位到计算图模型中的“可疑节点”。但在更广泛的应用场景中，这些挖掘出来的数值缺陷模式可能并不能覆盖所有的缺陷情况。可以肯定的是数值缺陷都出现在数值操作的运算之中。只要我们能够写出这些数值操作的定义域约束，如果前面的网络输出落在定义域内，则这个数值操作就是安全的；反之，这就是个“可疑节点”。因此，我们可以使用静态检测的方法来针对神经网络每一层的结构构建输入输出的约束。并且联合某个数值操作的定义域约束就可以让约束求解器协助我们求解约束。如果找到一组可行解，那么该数值操作就是个“可疑节点”。

但这种进一步的优化仍然需要克服以下困难：

- 因为神经网络紧密连接的特性（正如第三章 2节指出的那样），让这种静态检测能有效地部署到巨大的神经网络之中是一个困难。
- 现有的约束求解器如 Z3 [37] 等并不能够很好地支持非线性函数的约束，而如何为神经网络之中的非线性激活函数：*tanh*、*sigmoid* 和 *softmax* 等写约束是一个困难。

本文的算法无法针对深度学习框架内部的隐式操作挖掘模式。如在训练阶段之中，深度学习框架如何应用学习率更新网络模型参数这个隐式操作就无法挖掘模式。此外，本文的算法也没有对一些深度学习中的专家知识进行挖掘：比如在 loss 函数中通常会加入 L2 正则化来避免神经网络中出现数值缺陷。这也导致了第五章 2.1小节中的 5 个数值缺陷未能被我们的算法发现。

本文的实验所针对的数据集总数只有 14 个，从数量上来看还偏少，今后的工作应该将其拓展到更加大的数据集上：比如自动地爬取 Github 上的开源深度学习项目，对其进行缺陷检测。数据集中的有数值缺陷的程序也只针对于图像识别，比如针对 MNIST 数据集所搭建的 CNN [38]（卷积神经网络）模型。我们的数值缺陷检测算法能够推广到一般的深度学习模型，比如 LSTM [39]（长短期记忆神经网络），GAN [17] 上，尤其是我们的输入修改算法，如果输入换成是自然语言处理中的词嵌入，效果是否还是那么好，是一个非常有研究价值的问题。

截至目前，本文的工作基于作者的人工实验，模式匹配算法和修复都是基于手动的实现，而“可疑节点”、“可疑节点”的输入以及最大化引导函数的选取都是基于人工读代码之后的手动选取。如何自动化地进行模式匹配，自动提取“可疑节点”、“可疑节点”的输入以及最大化引导函数，并将我们的算法插入到待检测的深度学习程序中，也是一个不小的挑战。

本文的实验也只基于 TensorFlow 架构。尽管 TensorFlow 是开源网站 Github 上使用人数最多的深度学习框架，4.8 倍于使用人数第二多的 Pytorch。我们的算法是否能够推广到 Pytorch, MXNet 等其他深度学习框架上，也是一个值得研究的问题。

第七章 结论

本文在先前的实证研究基础上，针对深度学习程序缺陷中的一类缺陷——数值缺陷，提出了一种快速高效的检测方法。这种方法能够在深度学习的训练阶段的早期检测到数值错误的出现，为开发者节省了巨大的时间和计算资源的消耗。接着，通过在实证性研究中挖掘数值缺陷和其修复的模式，通过模式匹配的方法，修复已经检测的错误。本文的实验显示出我们的算法能够以很高的检测召回率检测到数据集中的数值缺陷. 该算法在 2 个缺陷程序上达到了 **59.44** 倍和 **20.08** 倍的加速比，且在 6 个缺陷程序上达到了超过 **3** 倍的加速比；从训练轮次上看，该算法在 8 个缺陷程序上减少了 **67.09%-99.99%** 的训练轮次。此外，在正确的网络模型上应用我们的算法，仅比正常训练的时间慢了 **41.90%**，这意味着我们算法在正确模型上的检测的额外开销很小，也就是说我们的算法可以高效地部署到开发环境之中，为深度学习应用的开发者提供巨大的帮助。

参考文献

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[3] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[6] Wikipedia contributors. List of self-driving car fatalities — Wikipedia, the free encyclopedia, 2019. [Online; accessed 9-April-2019].

[7] Greenberg, Andy. Hackers say they've broken face id a week after iphone x release, 2017. [Online; accessed 9-April-2019].

[8] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. An empirical study on tensorflow program bugs. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*, pages 129–140, 2018.

[9] The explosion of the ariane 5, 2000. [Online; accessed 12-April-2019].

[10] Augustus Odena and Ian J. Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *CoRR*, abs/1807.10875, 2018.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[12] David Wagner Nicholas Carlini. Towards evaluating the robustness of neural networks. *Proc. Security and Privacy (SP)*, pages 39–57, 2017.

[13] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[14] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. *Proc. ASIA CCS*, pages 506–519, 2017.

[15] Dongyu Meng and Hao Chen. Magnet: A two-pronged defense against adversarial examples. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 135–147, 2017.

[16] Shiwei Shen, Guoqing Jin, Ke Gao, and Yongdong Zhang. AE-GAN: adversarial eliminating with GAN. *CoRR*, abs/1707.05474, 2017.

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[18] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[19] Siwakorn Srisakaokul, Zexuan Zhong, Yuhao Zhang, Wei Yang, and Tao Xie. MULDEF: multi-model-based defense against adversarial examples for neural networks. *CoRR*, abs/1809.00065, 2018.

[20] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18, 2017.

[21] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *CoRR*, abs/1803.04792, 2018.

[22] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: multi-granularity testing criteria for deep learning systems. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 120–131, 2018.

[23] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 109–119, 2018.

[24] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. Deepct: Tomographic combinatorial testing for deep learning systems. *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, pages 614–618, 2019.

[25] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. *ICSE' s New Ideas and Emerging Results (NIER)*, 2019.

[26] Anthony Di Franco, Hui Guo, and Cindy Rubio-González. A comprehensive study of real-world numerical bug characteristics. *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 509–519, 2017.

[27] Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. Efficient search for inputs causing high floating-point errors. *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '14, Orlando, FL, USA, February 15-19, 2014*, pages 43–52, 2014.

[28] Earl T. Barr, Thanh Vo, Vu Le, and Zhendong Su. Automatic detection of floating-point exceptions. *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 549–560, 2013.

[29] Alex Sanchez-Stern, Pavel Panchekha, Sorin Lerner, and Zachary Tatlock. Finding root causes of floating point error with herbgrind. *CoRR*, abs/1705.10416, 2017.

[30] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[31] Muriel Daran and Pascale Thévenod-Fosse. Software error analysis: A real case study involving real faults and mutations. *Proceedings of the 1996 International Symposium on Software Testing and Analysis, ISSTA 1996, San Diego, CA, USA, January 8-10, 1996*, pages 158–171, 1996.

[32] István Forgács and Antonia Bertolino. Preventing untestedness in data-flow based testing. *Softw. Test., Verif. Reliab.*, 12(1):29–58, 2002.

[33] Robert M. Hierons. Avoiding coincidental correctness in boundary value analysis. *ACM Trans. Softw. Eng. Methodol.*, 15(3):227–241, 2006.

[34] Brian Marick. The weak mutation hypothesis. *Proceedings of the Symposium on Testing, Analysis, and Verification, TAV 1991, Victoria, British Columbia, Canada, October 8-10, 1991*, pages 190–199, 1991.

[35] Debra J. Richardson and Margaret C. Thompson. An analysis of test data selection criteria using the RELAY model of fault detection. *IEEE Trans. Software Eng.*, 19(6):533–553, 1993.

[36] Baowen Xu, Ju Qian, Xiaofang Zhang, Zhongqiang Wu, and Lin Chen. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes*, 30(2):1–36, 2005.

[37] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

本科期间的主要工作和成果

本科期间参加的主要科研项目
本研基金

1. 国家创新训练项目. 教育部 “国家大学生创新性实验计划”. 熊英飞. 1 年

会议论文

1. Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, Lu Zhang. An empirical study on TensorFlow program bugs. ISSTA 2018 Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, Amsterdam, Netherlands, July 16 - 21, 2018, Pages 129-140

致谢

感谢我的导师熊英飞副教授在研究工作中的悉心指导，在每周的例行讨论中，他总能指点我的迷津，为我指引方向；作为我们班级的班主任，他也能在学习生活方面给予我们支持。

感谢香港科技大学的張成志教授和本校的张路教授在我的本科生科研中给予我的帮助，没有他们就没有我的实证性研究在 ISSTA 上的发表，也没有这篇基于本研论文的毕业论文。

感谢伊利诺伊大学厄巴纳 - 尚佩恩分校的谢涛教授在暑期科研中对我的指导，在暑研中我得以深入了解深度学习模型测试方面的相关工作，并明确我将深度学习程序代码缺陷作为我的毕业设计这一目标。

感谢辛苦的答辩评委们进行阅读并提出宝贵的意见，没有你们这篇论文的质量也不会到今天这个程度，感谢各位能够让我的研究得到一个良好的总结。

感谢我身边的同学，尤其是我的女朋友刘昊棠给予我在平时生活和心理上的帮助，她也从一个深度学习领域的研究者的角度对我的毕业设计提出了很多实用的建议。

感谢我的父母，正因为他们两个月来每周多个电话催促我完成我的毕业设计，这篇工作才会在如此快的时间内高质量完成。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　月　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校 □ 一年/□ 两年/□ 三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　月　日