摘要

随着大数据和计算力（GPU）的快速发展，深度神经网络（DNN）变得越来越流行并开始被应用在一些安全攸关的领域中。但是深度神经网络已经被证明会受到诸如对抗样本之类的外部攻击而产生严重的错误结果，这就意味着需要通过人为的检测手段来预先确保深度神经网络的可靠性。最近在 POPL 2019 会议上的一篇论文提出了新的用于检测深度神经网络对于输入样本鲁棒性的工具——DeepPoly。本论文将介绍对于 DeepPoly 工具所做出的改进工作，包括增强它的适用性和提升它的运行效率。并基于它开发了用于检测神经网络鲁棒性的可视化系统。最后应用改进后的 DeepPoly 工具对 LeNet-5，Autumn 和 ChestNet 三个开源神经网络进行了检测。其中整体的实验结果表明 DeepPoly 能够较好地用于评估规模不大的分类神经网络的鲁棒性，同时也表明目前分类神经网络即使准确率较高但是其抗鲁棒性也可能没有达到可靠的标准。

关键词：深度学习，对抗样本，DeepPoly

Abstract

With the rapid development of big data and computing power (GPU), deep neural networks(DNN) have become increasingly popular and have now started penetrating safety critical domains. However, it has been proved that under external attacks such as adversarial examples, deep neural networks will generate serious error, which means the reliability of deep neural networks needs to be guaranteed in advance by means of artificial detection methods. Recently, a paper presented at POPL 2019 introduced DeepPoly, a new tool for detecting the robustness of a deep neural network to input samples. This paper will introduce the improvements to DeepPoly, including enhancing its applicability and improving its efficiency. In addition, a visual system based on DeepPoly for detecting the robustness of neural networks is developed. Finally, the improved DeepPoly tool was applied to detect LeNet-5, Autumn and ChestNet. And the overall experimental results show that DeepPoly can be used to evaluate the robustness of small classification neural networks, and it also show that the robustness of classification neural networks with high accuracy may not reach the reliable standard.

Keywords: Deep Learning, Adversarial Example, DeepPoly

目录

1 引言

在最近的几年里，由于 GPU 性能的不断增强带动计算机运算浮点矩阵的能力和内存带宽的提升，以及互联网中爆炸式增长的信息使得获取海量的训练数据变得更加便利，深度神经网络（Deep Neural Network）渐渐发展成为非常有前景的智能系统[1]。并且它已经开始被应用在一些诸如无人驾驶[2]（DNN 通过车载摄像头实时拍摄的照片来决定当前车辆的驾驶角度），人脸识别[3]，医疗诊断[4]（DNN 诊断病人是否患有肝炎疾病）等安全攸关的领域中。毋庸置疑，深度神经网络在这些领域中扮演了非常重要的角色，但同时这也意味着保证深度神经网络行为的可靠性将变得异常重要。遗憾的是深度神经网络已经被证明会在某些特定的情况下表现出不鲁棒的行为。举例来说，Szegedy 等人在 2014 年首次提出了对抗样本的概念[5]。在论文中他们指出深度神经网络会将两张仅有细微的像素差异的图片输入分别以高置信度分类成两种不同的结果。具体的实际样例如图 1-1[6]所示，其中将添加细微扰动的输入样本称为对抗样本。

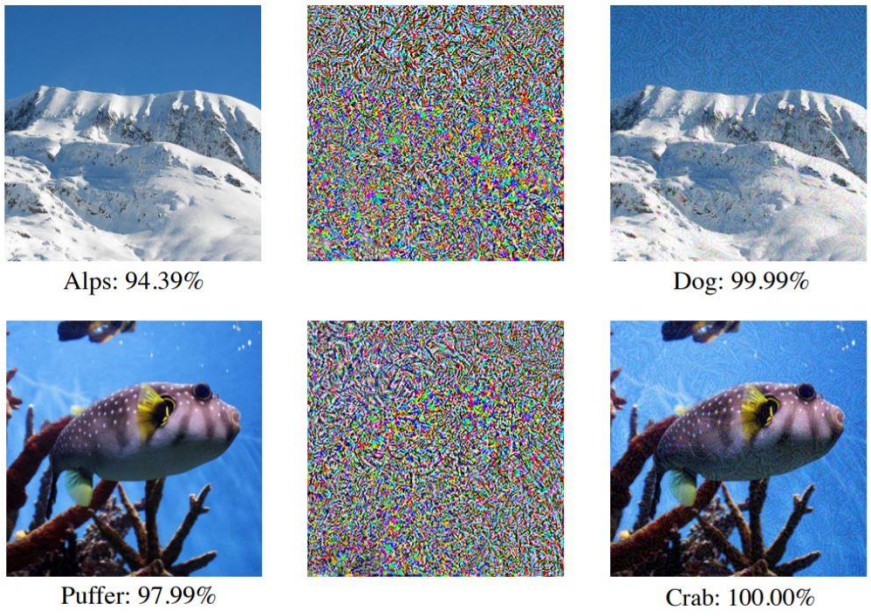

图 1-1 左列：原图。深度神经网络以高置信度且正确地识别出图片中分别是阿尔卑斯山脉和河豚；中间列：在原图上添加的噪声；右列：合成的图片，即对抗样本。可以看到两张图片以高置信度但是错误地被分别分类为狗和螃蟹。

所以必须要有一些相对应的人为检测手段来保证深度神经网络的鲁棒性。但是由于目前的深度神经网络通常都是庞大的模型，其拥有数以千计或万计的权重参数，以至于它不像决策树和广义线性模型等其它的机器学习模型是可解释的[7]。并且 Weng 等人[8]已经证明出精确地得到深度神经网络的最小对抗扰动值的问题，即验证深度神经网络一般意义上的鲁棒性问题不存在时间复杂度在多项式范围内的解法除非NP = P[9]。所以推断整个深度神经网络模型的工作是异常困难的。但目前已经有许多方法和工具能够解决部分的问题。其中最高水平的工作包括 Katz 等人的基于 SMT（Satisfiability

Modulo Theories）求解的方法[10]，Weng 等人的基于线性近似[8]的方法和 Gehr 等人的基于抽象解释[11]的方法。

尽管这些工作取得了一定的进展，但是对于完全解决推断深度神经网络的难题还需要更多的研究。目前尤其还缺少一种能够适应大型神经网络，并且能够处理当今流行的神经网络结构（比如前馈神经网络和卷积神经网络）的分析器。它同时还应该做到足够精确，以证明应用系统所需的相关属性。具体来说，Katz 等人在论文提到的方法已经足够精确但只能处理非常小的神经网络。与此同时 Gehr 等人的方法能够分析大一点的神经网络但是依赖于现有的通用抽象域就会使得它不能在处理大型网络的同时保持结果的精确性。而最近 Weng 等人提出的方法具有更好的可扩展性但是只能处理前馈神经网络并不能处理如今已被广泛使用的卷积神经网络。同时 Katz 等人和 Weng 等人的方法对于在神经网络中广泛使用的浮点运算的处理不够理想，这导致他们的方法会受制于错误的反例[12]。

而最近苏黎世联邦理工学院的 SRI 实验室的成员（Singh 等人）在 POPL 2019 会议上发布了一篇名为《An Abstract Domain for Certifying Neural Networks》的论文，其中他们基于抽象解释的理论和用于神经网络中的非线性激活函数的特定的抽象转换器开发了用于检测深度神经网络对于某个输入样本的鲁棒性程度的工具——DeepPoly。特别地，他们在论文中指出经过了大量的实验数据的对比证明了 DeepPoly 在性能、准确性和适用性的三个方面上相较于之前的检测方法的实际表现有了很大的提高。

本论文中认为这可能会对深度神经网络的可靠性提供新的保障。所以本课题的主要工作是基于此论文中通过抽象解释理论和特定抽象转换器近似深度神经网络中非线性数值运算的原理，深入了解了 DeepPoly 工具的底层代码实现以及相关的使用场景和方法。并应用它来检验目前流行的开源深度神经网络，尝试从检测结果中评价被检测的神经网络在实际使用中的鲁棒性。

但是在整个实验过程中发现 DeepPoly 工具还存在很多值得改进的问题。具体来说，DeepPoly 工具目前仅支持两种主流数据集（MNIST[13]和 CIFAR-10[14]）的检测，并且它所支持的深度神经网络的神经元结构也十分有限。同时虽然 DeepPoly 的运行效率在非完备性的检测方法中已经是顶尖的水平，但是在一些方面仍然有可以进行优化改进的细节。另外在实际应用中还发现目前很难从 DeepPoly 实际给出的检验结果中得出有价值的反馈信息。本论文中认为主要的原因是没有直观有效的评估方式。本课题将二分查找法与 DeepPoly 相结合使得其可以得出深度神经网络对于每个测试样本所能验证的最小对抗扰动值[15]（minimum adversarial distortion）的最大值，并在此基础上提出对于评价深度神经网络鲁棒性的单一数字评估指标（single number evaluation metric）的构想。本论文将详细介绍对于上述相关问题的解决方法。

本论文的行文顺序如下：第 2 部分介绍了与本课题有关且必要的背景知识；在第 3 部分将介绍如何加强 DeepPoly 工具对于一般的神经网络的适用性以及提升它的运行效

率；在第 4 部分将介绍如何将二分查找法与 DeepPoly 工具相结合使得产生的新工具可以计算最大扰动值；在第 5 部分介绍了基于 DeepPoly 工具所开发的一个用于检验深度神经网络鲁棒性的可视化系统；第 6 部分将介绍应用改进后的 DeepPoly 工具和可视化系统检测当今流行的开源深度神经网络的实验结果。最后一部分将对于本课题的工作作出总结。

本课题所有的相关代码实现都已上传至 https://github.com/Callmejp/DiplomaProject 的 github 仓库上。

本课题的主要贡献如下：

- 重构了 DeepPoly 工具的源代码使之能够适用于更多进行图片分类的深度神经网络的检测工作。
- 改进了 DeepPoly 工具的源代码的逻辑细节以进一步提升其运行效率。
- 将二分查找法与 DeepPoly 工具相结合使获取的检测结果更有价值，并基于此工具开发了便于检测神经网络鲁棒性的可视化系统。另外基于检测结果的特点提出评估神经网络鲁棒性的单一数字评估指标的构想。
- 应用 DeepPoly 检测当前流行的开源深度神经网络并对相关的实验结果进行了分析与总结。

2 预备知识

2.1 深度神经网络

目前主流的深度神经网络是层级结构模型，其中共有输入层、隐藏层和输出层三种类型。除了输入层表示神经网络的输入外，其余每层都由n_i个神经元组成，其中$i \in [1, l]$ 且l为神经网络的层数。如图 2-1[16]所示，每个神经元接受前一层所有神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过激活函数（activation function）处理以产生神经元的的输出。正式地，通常可以由 DeepPoly 工具处理的一个层级深度神经网络可以通过l层的组合来表示，即$f_1: \mathbb{R}^m \to \mathbb{R}^{n_1}, \ldots, f_l: \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$。每一层的$f_i$映射都满足如下变换中的一个：

1) 一次仿射变换，对于$A \in \mathbb{R}^{n_i \times n_{i-1}}$, $f_i(x) = Ax + b$（特别地，拥有一个或多个过滤器的卷积操作也是一种仿射变换）。
2) ReLU 激活函数$f(x) = \max(0, x)$。
3) Sigmoid 函数($\sigma(x) = \frac{e^x}{e^x+1}$)或是 Tanh 函数($\tanh(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$)
4) MaxPool 函数，其中它将输入x分割成多部分然后返回每部分的最大值。

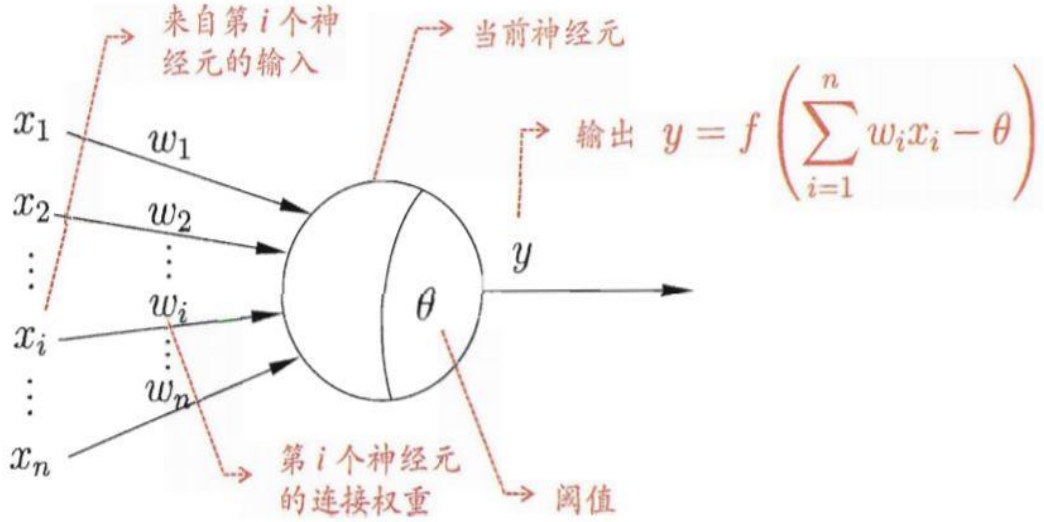

图 2-1 神经元模型结构

2.2 对抗样本

对抗样本(Adversarial Examples)最早是由 Christian Szegedy 等人在 ICLR 2014 会议上发表的论文中提出的的概念，即表示在数据集中通过故意添加细微的干扰所形成的输入样本[5]。其中受干扰之后的输入样本会导致模型（深度神经网络）以高置信度给出一个错误的分类输出。虽然对抗样本在实际的应用中比较少见，但是在特定情况下，

在训练集的不同子集上训练得到的具有不同结构的模型都会对相同的对抗样本实现误分类，这称为对抗样本的泛化性（Transferability）[17]。这就意味着攻击者可以通过相同的训练数据集自行训练出另一个神经网络，并将其作为白盒模型进行计算得到一个通用的对抗样本来攻击其它神经网络。

目前对于对抗样本的攻击主要有三种方向的防御方法[18]：

1) 使用改善过的方法去训练神经网络或在测试的过程中使用改善过的输入。
2) 改变神经网络的结构。比如添加更多的网络层或子网络，修改损失或激活函数等。
3) 当分类预测没有见过的输入时，使用外部模型作为附加网络进行预测。

而对于对抗样本攻击的检测或是对于深度神经网络鲁棒性的检测的方法一般是找到最小对抗扰动值ϵ。

具体来说，通常得到对抗样本的方式是向原图中添加单个像素值最大为ϵ的噪声，即假定η是和原图相同大小的噪声的像素矩阵，有$\|\eta\|_\infty < \epsilon$。

对于某个输入测试样本，当添加的噪声$\|\eta\|_\infty < \epsilon$时，深度神经网络对于此测试样本具有鲁棒性，而当$\|\eta\|_\infty = \epsilon$时，此输入测试样本会影响深度神经网络的判断结果。这时就称ϵ是最小对抗扰动值。

然而目前的检测方法只能尽量接近理论上的最小对抗扰动值，所以本课题中使用最大扰动值来指代检测方法所要验证的目标。

2.3 DeepPoly

2.3.1 代码结构

DeepPoly 是使用 C 语言（底层）和 Python（上层）语言共同编写的工具。底层实现基于同样由 SRI 实验室开发的 ELINA 库，其包含了流行的数值抽象域的优化实现，如用于静态分析的多面体（Polyhedra）、八边形（Octagon）和区域（Zones）。上层代码主要使用了 Python 中的 Tensorflow 和 Numpy 等模块来实现深度神经网络模型文件和测试数据的读取、网络结构的分析以及调用底层 ELINA 库等功能。

2.3.2 功能分析

DeepPoly 主要用于检测深度神经网络在指定的扰动范围内是否对于某个测试样本具有鲁棒性，即不会将受到扰动的样本判断为不同的分类标签。具体来说，假定使用向量p表示测试样本中所有归一化后的像素值（即每个像素值都除以 255.0 的结果），在运行 DeepPoly 程序前用户指定扰动变量$\epsilon \in [0, 1.0]$的值，那么它的运行目标是判断测试的深度神经网络是否将像素值向量在$[\max(p - \epsilon,\ 0.0),\ \min(p + \epsilon,\ 1.0)]$区间内的所有可能的图片都分类为相同的类标签。

2.3.3 使用方法及结果分析

用户运行 DeepPoly 工具前需要确定扰动变量ϵ的值、被测试的神经网络以及用于测试的数据集三个命令行参数（其中抽象域参数在本课题中默认使用 deeppoly 参数）。程序会依次读入数据集的所有测试输入样本。对于每个测试样本，其首先以$[p, p]$的区间表示测试样本的像素值再通过底层 ELINA 库的分析得到一个正确的类标号L_1（即与通过被测试的神经网络模型得到的类标号相同）用于比对，之后以$[\max(p_i - \epsilon,\ 0), \min(p_i + \epsilon,\ 1.0)]$的区间来表示测试样本可能的像素值再通过 ELINA 库的分析得到另一个类标号L_2。假定深度神经网络共有N类输出结果，则最终用户会得到三种结果：

- $L_1 = L_2$且$L_2 \in [0,\ N-1]$。即存在最高置信度的类标号，且与用于对比的类标号相同（检测通过）。
- $L_1 \neq L_2$且$L_2 \in [0,\ N-1]$。即存在最高置信度的类标号，但与用于对比的类标号不同（检测失败）。
- $L_1 \neq L_2$且$L_2 = -1 \notin [0,\ N-1]$。即不存在具有最高置信度的类标号（检测失败）。

当用户得到第一种情况的结果时就可以认为任何对于该测试输入样本的像素值小于ϵ的扰动都不会影响最终深度神经网络的分类结果。

2.3.4 上层 Python 源代码的思路分析

这一小节将按照 DeepPoly 程序中代码运行的顺序来介绍上层 Python 部分的源代码的整体思路，同时这也是为之后介绍改进其源代码的部分进行铺垫。此外下面论述中所提及的代码文件都位于其工程根目录中的 tf-verify 文件夹下。且因为原作者陆续添加了其它抽象域的功能导致本论文所依据的源代码被覆盖，因此具体实现可以参考如下的代码仓库链接：

https://github.com/eth-sri/eran/tree/40ece3413a58f1434cf741adaab169c90083557f。

首先是命令行参数部分的代码。__main__.py 文件中的代码会依次读取神经网络文件的路径、扰动值、使用的抽象域和测试集总共 4 个命令行参数。

其次是读取深度神经网络的模型文件的代码。DeepPoly 工具分别支持 tf、meta 和 pyt 三种格式的文件。其中 tf 和 pyt 格式只是按照原作者指定的格式来描述深度神经网络的文本文件，所以由单独的 read_net_file.py 文件来处理。而 meta 格式是 Tensorflow 框架所用于保存模型的文件格式，所以源代码是通过 Tensorflow 的 API 进行读入。但最终通过两种方式得到的神经网络模型都被用于构建 ERAN 类的实例。

接着是读取模型的操作和变量的代码。ERAN 类在初始化的操作中会调用 tensorflow_translator.py 文件中的方法来依次读取深度神经网络模型中的操作类型和训练好的变量值，并分别保存在 operation_types 和 operations_resources 的两个数组中。接着再将两个数组作为参数初始化 optimizer.py 中 Optimizer 类的实例，并将 Optimizer 实例保存在 ERAN 实例自身的 optimizer 属性中。

最后是读取测试集并对其迭代检验的代码。在__main__.py 中对于每一个测试样本首先对其进行归一化和可选的标准化操作，接着将测试样本的上下边界（上下边界相同）作为参数调用 ERAN 的 analyze_box 方法。在 analyze_box 方法中，ERAN 实例首先调用其 optimizer 属性的 get_deeppoly 方法。这一步的目的在于将神经网络中一层神经元的连续操作（如卷积层包括卷积、偏置相加和激活函数三个不同的操作）合并成一个整体。其中每个整体都对应 deeppoly_nodes.py 文件中不同 Node 类的实例。其次 ERAN 实例将得到的 Node 类的数组用于初始化 analyzer.py 中的 Analyzer 类的实例。最后调用其 analyze 方法得出当前测试样本的类标号，如果类标号不正确则会跳过下一步对于测试样本的鲁棒性检验而直接对下一个测试样本进行检测。

得到正确的类标号之后将 2.3.2 小节中所提到的上下边界区间作为参数再次调用 ERAN 实例的 analyze_box 方法。之后程序的运行步骤与上一段的论述一致并且程序也会运算给出一个标号。最终 DeepPoly 的程序将两次的标号进行比对并给出如 2.3.3 小节中所示的三种结果。

2.3.5 DeepPoly 的不完备性

本小节中会通过一个小型的样例网络作为一个反例来验证此特点，其中推导过程中的详细论证可以参看[12]，这里仅介绍之后会用到的符号：l_k和u_k分别表示第 k 个节点取值范围的上边界和下边界。

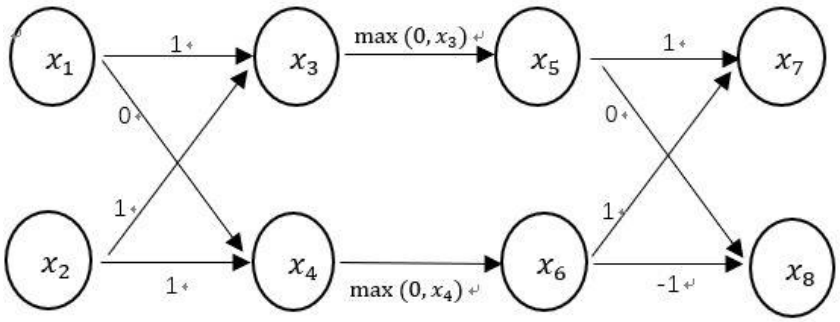

图 2-2 样例网络

如图 2-2，这是具有一个输入层，一个隐藏层和一个输出层的神经网络。其中隐藏层的神经元分为两层：x_3和x_4表示仿射变换的操作而x_5和x_6表示 ReLU 激活函数的操作。其中权重矩阵的值表示在每条有向边上，而 Bias 的值默认为 0。另外样例网络对于输入神经元设定了较大的输入区间以此放大 ReLU 转换时的精度丢失，而证明的目标是 DeepPoly 工具所基于的底层理论无法证明样例网络对于如下指定的输入扰动是具有鲁棒性的。

令$x_1 \in [-5, 3]$和$x_2 \in [1, 3]$为神经网络的两个输入。易得$l_1 = -5, u_1 = 3, l_2 = 1, u_2 = 3$。则经过一次仿射变换之后得到：

$$x_1 + x_2 \leq x_3 \leq x_1 + x_2;$$
$$x_2 \leq x_4 \leq x_2.$$

并且得到上下边界值如下：$l_3 = -4, u_3 = 6, l_4 = 1, u_4 = 3$。接着经过 ReLU 激活函

数的操作，根据参考论文中的推导可以确定：

$$x_3 \le x_5 \le 3 \cdot (x_3 + 4)/5;$$
$$x_4 \le x_6 \le x_4.$$

同样计算得到上下边界值为：$l_5 = -4, u_5 = 6, l_6 = 1, u_6 = 3$。最后经过仿射变换到达输出层：

$$x_5 + x_6 \le x_7 \le x_5 + x_6;$$
$$-x_6 \le x_8 \le -x_6.$$

两个输出神经元的上下边界值为：$l_7 = -3, u_7 = 9, l_8 = -3, u_8 = -1$。因为并没有$l_7 \ge u_8$或$l_8 \ge u_7$恒成立的不等式，故引入额外的$x_9$节点：

$$x_7 - x_8 \le x_9 \le x_7 - x_8.$$

通过不断地使用不等式来替换$x_7 - x_8$，可以得到$x_1 + 3 \cdot x_2 \le x_9$。而$x_1 + 3 \cdot x_2 \in [-2, 12]$，故通过上述推导无法得出$x_7 > x_8$恒成立。然而根据观察可以很快地得到$x_7$恒为正数而$x_8$恒为负数的结果，即事实上神经网络对于如上的输入扰动是具有鲁棒性的，只是因为程序进行底层运算时会造成精度丢失导致会出现错误的反例。

综上所述，通过如上样例网络的推导可以得出对于某个深度神经网络，DeepPoly 工具只能证明其对于输入样本具有鲁棒性而无法证明其对于输入样本不具有鲁棒性。

3 对 DeepPoly 工具的改进

3.1 概况

在这个部分将介绍对 DeepPoly 工具的源代码所做的改进与优化工作。这是应用 DeepPoly 工具检测深度神经网络工作的第一步。主要内容分为三点：

1) 加强 DeepPoly 工具对于一般的用于图片分类的深度神经网络的适用性。
2) 添加相应的处理代码以使得 DeepPoly 能用于 meta 格式的深度神经网络模型文件的检验工作，即使模型文件中包含了随机失活[19]（Dropout）等会影响程序运行的操作。
3) 优化代码的逻辑细节以提升 DeepPoly 工具的运行效率。

其中在第一点的工作中对于代码的重构是具有一般性的，也就是说它可以完美覆盖原先的 DeepPoly 工具的代码功能而不会对其应用在其它神经网络上造成问题。但是第二点的工作中对于代码的改动是临时性的，即只是针对特定神经网络所做的单独的代码改动，会对其应用在其它神经网络上造成问题。

另外本课题训练了一个用于花朵分类的小型卷积神经网络（在论文中用 FlowerNet 指代）用于改进代码时的调试工作，其网络结构依次为 12288 维（输入图片的长、宽和通道数为$64 \times 64 \times 3$）的输入层，卷积核大小为$4 \times 4 \times 3 \times 8$，步长为 1 的卷积层，采样区域为$8 \times 8$的最大池化层（MaxPool），卷积核大小$2 \times 2 \times 8 \times 16$，步长为 1 的卷积层，采样区域为$4 \times 4$的最大池化层以及 1 层全连接层（输出层）。

3.2 加强 DeepPoly 的适用性

本课题主要从以下几点来重构__main__.py 和 optimizer.py 两个部分的代码。

3.2.1 命令行参数

在源代码中，原作者使用了 Python 中的 sys.argv 模块来获取外部用户输入的命令行参数，其中 sys.argv 模块的功能是将程序本身和其接收到的命令行参数打包成 Python 中的 list 类型供当前程序的后续使用。所以 list[0]，即第一个参数表示的是程序本身，其余则依次是用户输入的其它命令行参数。虽然 sys.argv 模块的使用方法相对简明但是它不具备可选参数的功能同时也无法设定默认值来方便用户的使用，所以改进后的代码中采用了更为流行且功能也更加丰富的 argparse 模块来优化程序的命令行参数部分，这是加强 DeepPoly 工具对于其它分类深度神经网络适用性的第一步。最终的必选参数，可选参数和相应的默认值等详细细节如表 3.1 所示。

其中将用于测试的数据集参数设置为用户输入的测试数据集路径使得 DeepPoly 工具可以读取除了 MNIST 和 CIFAR-10 数据集之外的用于训练其它神经网络的数据集。

另外用于标准化输入样本的源代码只是通过条件判断为 MNIST 或是 CIFAR-10 数据集来决定不同的标准化方式，但是通过添加可选的 dimension 参数就能从更加准确的角度来区分一维（黑白）图片和三维（彩色）图片的不同标准化的处理方式。

3.2.2 扩展神经网络结构。

DeepPoly 的程序在分析神经网络结构的时候会将每一层的神经元作为一个整体处理（optimizer.py 文件中的代码逻辑），比如会将权重矩阵与前一层的输入矩阵相乘（MatMul），偏置相加（BiasAdd）和激活函数（ReLU）总共三次运算打包成一个单独的 Node 类的实例来用于后续的进一步分析处理。这可以理解为 DeepPoly 工具只支持预先定义好的常见的神经网络操作的顺序，所以这部分的源代码的实现原理只是通过条件判断来对相对应的 Node 类进行处理，然而在使用 FlowerNet 测试时发现它不存在处理仅有卷积(Conv2D)和激活函数（ReLU）的 Node 类，它只能处理中间还有偏置相加（BiasAdd）的操作。但从实际的应用中来看仅有卷积操作和激活函数也应当是合理的神经网络结构的构建顺序。所以在不破坏其余源代码的功能下通过添加矩阵元素全为 0 的偏置矩阵来构造一个合法的 Node 类从而解决了上述的问题。此处改进在扩展 DeepPoly 工具所支持的神经网络结构的同时显然并不会影响最终的结果，所以它是具有一般性的代码重构。

表 3.1 改进后的 DeepPoly 的命令行参数表

命令行参数名	Python 数据类型	命令行参数的含义	是否为必选参数	默认值	改进说明
network	str	用于测试的神经网络文件。	是	无	
epsilon	float	设定的扰动值。	是	无	
dataname	str	用于测试的数据集。	是	无	设置为测试数据集的路径。
domain	str	使用的抽象域。	否	deeppoly	添加默认值。
normalize	float	正规化参数。	否	0.0	额外添加的可选参数。
dimension	int	图片的维数。	否	1	额外添加的可选参数。

3.3 检测特定神经网络的准备工作

3.3.1 读取 meta 格式的模型文件

DeepPoly 工具读取使用 Tensorflow 训练的神经网络模型文件的代码欠缺一般性。具体来说，在__main__.py 中读取训练好的变量的源代码如下：saver.restore(sess, tf.train.latest_checkpoint(netfolder+'/'))。其中 tf.train.latest_checkpoint 方法只有在该路径

下不仅存在 ckpt 格式的文件（保存模型的训练好的变量值）同时还需存在 checkpoint 文件时方可执行。因为该方法需要通过 checkpoint 文件里记录的训练结果的索引来找到最新的 ckpt 文件。但一般情况下对于训练完成的模型其 checkpoint 文件是多余的，只需要保存神经网络图结构的 meta 文件和保存神经网络变量值的 ckpt 文件即可。同时大多开源的由 Tensorflow 框架训练完成的神经网络也只会提供这两个文件。因此改进后的代码中直接使用提供 ckpt 文件路径名的更加一般的读取方法：saver.restore(sess, '../../networks/lenet/variable.ckpt')。这样仅需要 meta 文件和 ckpt 文件就能还原整个神经网络模型，避免了不必要的冗余。

另外程序还会尝试获取输出层的张量，但是使用固定的张量名明显不能通用于大部分的模型文件。此处的具体改动在后续实际对于开源神经网络的检验工作的准备部分有详细论述。

3.3.2 处理包含无关操作的模型文件

在 tensorflow_translator.py 的代码中程序通过调用 Tensorflow 框架中的 graph_util 模块的 convert_variables_to_constants 方法得到计算图中的所有张量名称。然而以要在之后讨论的 Autumn 神经网络为例，从 DeepPoly 工具的可实用性的角度出发，目前使用 Tensorflow 框架训练的神经网络模型通常包含随机失活等不能被解析的操作。因为这些操作会导致 DeepPoly 程序在运行的时候直接崩溃，所以局限了 DeepPoly 工具的应用范围。然而随机失活操作是在训练的过程中用于减少过拟合风险的手段，在实际测试的过程中通常设定随机失活操作中的 keep_prob 变量值为 1.0，即不使用随机失活操作。因此针对特定的神经网络可以将 tensorflow_translator.py 中与之有关的代码进行修改使得 DeepPoly 跳过它不能解析但是并不会影响最终结果的操作。

具体来说，DeepPoly 的程序通过迭代 graph.get_operations 来依次获得神经网络中的每个操作。那么可以提前输出整个神经网络的所有操作，排除会中断 DeepPoly 工具运行的操作。以之后在 6.3 部分中所要检测的 Autumn 卷积神经网络为例，在图 3-1 中的红色方框包裹的所有操作代表了会中断 DeepPoly 工具运行的随机失活操作，于是根据其在所有操作中出现的位置并通过相应的条件判断忽略这些随机失活的相关操作以此使得 DeepPoly 工具能够成功地对 Autumn 神经网络进行检验。

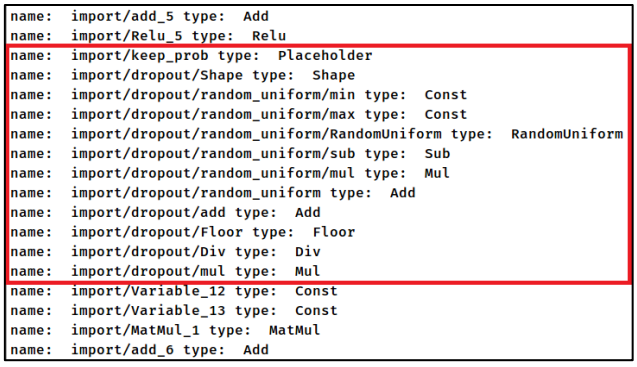

图 3-1 Autumn 模型中包含的部分操作

但是因为不同的神经网络所使用的结构千差万别，所以这部分代码难以进行泛化来适应所有的神经网络结构。但是经过如上的举例分析不难对指定的神经网络模型文件修改出相对应的需求代码。

另外对于神经网络中训练好的模型包含不能被处理的操作的问题还有另一种解决的方法。在 2.3.4 中提到了 DeepPoly 程序能够处理三种类型的格式文件，其中 tf 和 pyt 是预先约定好格式的文本文件。这意味着可以预先按顺序提取出相应的权重矩阵和偏置矩阵，以及相应所使用的网络层和激活函数等细节，再人为地将这些数据和信息进行组合和修改，最后按照指定格式生成被测试的深度神经网络的文本格式文件。这样 DeepPoly 程序会调用 read_net_file.py 来处理此类格式文件而不会造成对于源代码的修改。但是问题的本质并没有被解决，因为提取神经网络中操作的代码对于不同的神经网络的结构同样也是难以进行泛化的。不过在本论文地代码仓库中还是提供了将 6.4 小节提及的 ChestNet 的 Keras 网络模型文件生成 tf 格式文件的样例代码。其中的主要难点是将庞大的权重矩阵按照指定的格式写成文件的一行数组，这点需求需要额外编写相应的代码进行处理。

3.4 优化 DeepPoly 的运行效率

3.4.1 思路阐述

本课题主要从以下两点代码的逻辑来进一步提升 DeepPoly 工具运行时的效率。

其一是得到深度神经网络对于测试样本的正确类标号的过程。源代码是使用上下界相同的区间作为输入传递给底层的 ELINA 库函数进行推导并得出最终准确类标号的结果。但如果测试文件是 meta 格式，即使用 Tensorflow 训练所得到的模型，就可以直接调用 Tensorflow 的 API 快速地还原网络并计算出其输出的类标号。这对于像在第 6 部分所测试的 LeNet-5 和 Autmn 那样规模较大的神经网络来说其平均运行时间可近似地减少一半。

其二是程序判断是否存在具有最高置信度的类标号的过程。其源代码如图 3-2 所示。可以看到如果程序当前还不能确定具有最高置信度的类标号，则所有的类号都会被假定为最高置信度的类标号并与其余的类标号的置信度进行比较来确定其是否为最高置信度的类标号。其中循环结构中包含的 is_greater 函数是其主要的耗时点，因为其涉及到底层代码实现从输出层到输入层的递推运算。

然而，从 2.3.3 小节的结果分析中可以看到 DeepPoly 程序将存在错误的最高置信度的类标号和不存在最高置信度的类标号的两种情况都判定为检验失败。这就意味着只需要判定当前的最高置信度的类标号是否还是正确的类标号就能获得同样的判定结果，即相当于最外层循环的 i 变量是固定的，就等于正确的类标号。

所以可以将正确的类标号，即数据集中预先人为确定好的类标号作为参数传递到

实现上述双循环结构代码的 analyze 方法中就可以使用单循环结构来代替双循环结构。

```
dominant_class = -1
for i in range(output_size):
    flag = True
    for j in range(output_size):
        if i!=j and not self.is_greater(self.man, element, i, j):
            flag = False
    if flag:
        dominant_class = i
        break
```

图 3-2 判断最高置信度类标号的源代码

3.4.2 实验验证

本实验共使用三个均用于 MNIST 数据集分类的深度神经网络进行 DeepPoly 工具的运行效率测试。它们分别是在 6.2 小节中所使用的 LeNet-5 卷积神经网络，以及在 2.3.4 小节的链接页面中所提供的开源的 ConvSmall_Point 和 ConvSmall_PGDK 两个卷积神经网络。

其中 ConvSmall_Point 卷积神经网络包含 1 层卷积核大小为$4 \times 4 \times 16$且步长为 2 的卷积层，1 层卷积核大小为$4 \times 4 \times 32$且步长为 2 的卷积层和一层全连接层，共包含 3604 个神经元（其中卷积核的三个维数依次表示长、宽和过滤器数，因为通道数是直接由上一层的输出所决定的，所以忽略了对于它的介绍）。ConvSmall_PGDK 的网络结构与它相同但它是经过对抗训练所生成的神经网络。而 LeNet-5 的网络结构在 6.2 小节中有详细介绍。

本实验的运行环境为配备 1 核 CPU，2G 物理内存的服务器。

本实验主要分为两次测试。第一部分是对于 ConvSmall_Point 和 ConvSmall_PGDK 两个卷积神经网络的性能测试。因为它们的模型文件是 tf 文件格式，所以无法使用第一步的优化操作，但可以通过输出得到深度神经网络对于测试样本准确类标号的过程所需要的时间来预估第一步优化操作对于整体提升的运行性能。第二部分是对于由 Tensorflow 框架训练出来的 LeNet-5 卷积神经网络的测试。因为它可以同时使用上述的两步优化操作，所以可以与第一部分的实验结果进行对比来更加清晰地展示两步优化操作所分别带来的性能提升。

具体的实验结果如图 3-2，图 3-3 和图 3-4 所示，其中横坐标是每次运行 DeepPoly 程序所设定的ϵ变量值大小，纵坐标是 DeepPoly 验证每个样本所需的平均时间（单位：秒）。

先关注图 3-2，图 3-3 的实验结果。可以看到 DeepPoly 工具的运行效率有明显的

改善。但是整体的平均运行时间并没有减少 10 倍（神经网络输出神经元的个数，即单层循环的次数）左右，经过调试发现是 analyze 方法中的另一个同样是调用底层代码的预处理函数的时间复杂度远高于 is_greater 函数导致第二步的优化操作带来整体的运行效率提升有限。

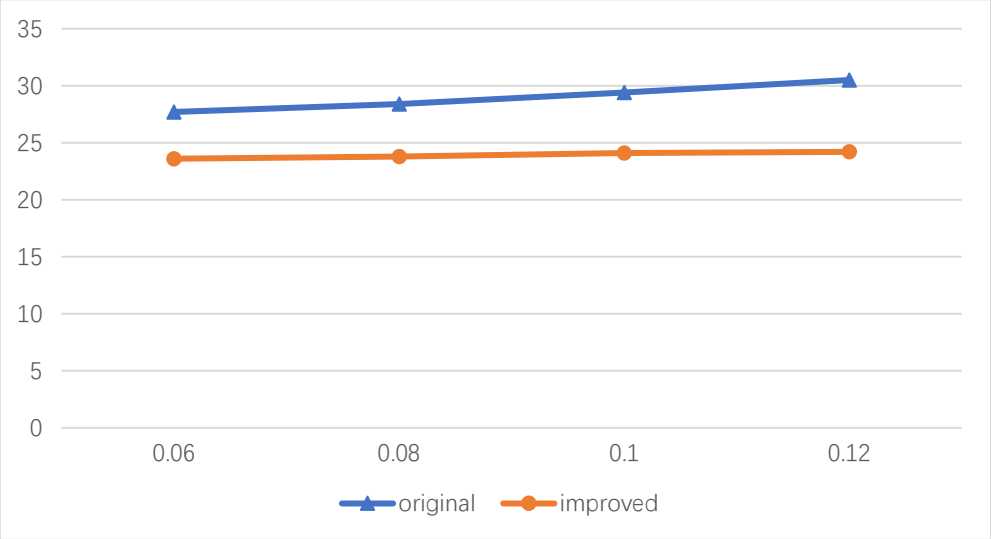

图 3-3 ConvSmall_Point 改进前后的运算时间对比图

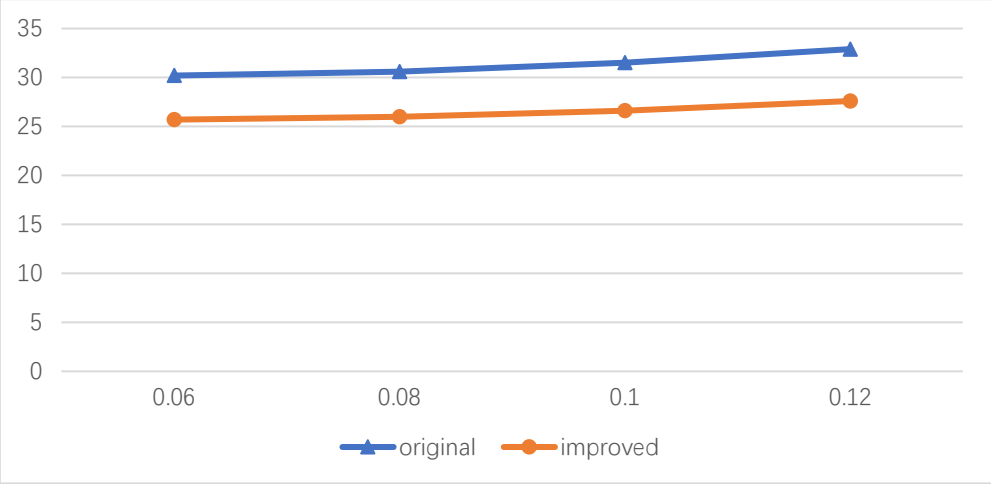

图 3-4 ConvSmall_PGDK 改进前后的运算时间对比图

另外，通过输出得到深度神经网络对于测试样本准确类标号的过程所需要的时间发现其只是略低于第二步得到扰动类标号的过程的时间，且通过 Tensorflow 中 API 来获得准确类标号的时间几乎可以忽略不计。这就意味着当所测试的深度神经网络模型文件是 meta 格式时，通过第一步的优化操作可以将 DeepPoly 验证每个测试样本的平均时间缩短近一倍。这个推断也在第二次的实验结果中得到了验证。

再通过图 3-4 来观察 DeepPoly 在可以同时使用两步优化操作的 LeNet-5 卷积神经网络上进行测试所需时间的前后对比。可以明显地看到平均运行一个测试样本的时间甚至少于原来的一半时间。可见第一步的优化操作对于 DeepPoly 测试网络规模较大的神经网络时有非常大的性能提升。而主要的原因是 DeepPoly 调用的底层代码的关于网

络规模的时间复杂度远高于顺序地运行模型文件得到输出的类标号。

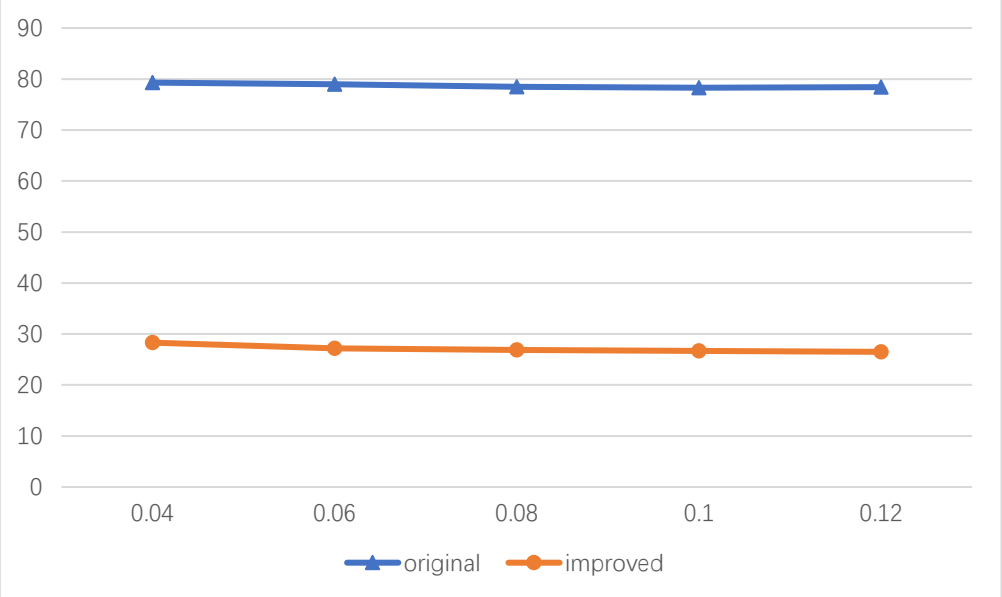

图 3-5 LeNet-5 改进前后的运算时间对比图

4 基于 DeepPoly 计算最大扰动值

4.1 概况

4.1.1 DeepPoly 的检测结果现状

本论文在 2.3.5 小节中通过一个反例论证了 DeepPoly 工具所基于的底层抽象理论具备可靠性但不具有完备性。所以这导致目前 DeepPoly 工具的检测结果很难表现出有价值的信息。因为在确定的扰动变量ϵ下，当它无法证明深度神经网络对于某个测试样本具有鲁棒性时有两种可能：其一是确实存在基于此测试样本的对抗样本的存在；其二是因为它所基于的抽象解释理论在分析深度神经网络的过程中由于精度的损失而导致的错误的反例。

4.1.2 计算最大扰动值的方法

在多项式时间复杂度的约束下，深度神经网络的鲁棒性可以通过检测在数据点x_0附近邻域来验证（比如L_2或L_∞范数球面）。其思想是找到半径为r_0的最大的球，保证邻域内的点不会改变分类器的决策。而r_0参数无疑是检测结果中最重要的点，虽然抽象解释理论因为其不完备性无法直接得到一个最大的r_0，但可以通过多次运行 DeepPoly 程序来逼近r_0。

为了最有效地得到相对准确的r_0参数，本课题使用了二分查找算法进行逼近，同时从实际意义出发再对其进行进一步的简化建模，最终求解r_0的算法只是 DeepPoly 程序本身的时间复杂度的常数倍。确切地说，假定被测试的深度神经网络共有L层，且一层中最多的神经元个数是n_{max}，则求得最大扰动值算法的时间复杂度为$O(n_{max}^3 \cdot L^2)$[12]。

简化建模的过程如下：根据之后 6.1 节的观点可以首先假定$r_0 \in [0, 0.064]$（即 0 到 16 个像素值的范围），其次因为图片的像素值是处在$[0, 255]$区间内的自然数，所以进行归一化之后有意义的最小单元约为 0.04（$\frac{1}{255} \approx 0.0039$）。于是经过如上的简易推导不难将$r_0$的可能性空间压缩到 17 个，同时最终获得的$r_0$参数也是在有现实意义的情况下相对准确。

4.2 实验分析

本小节的实验将继续使用 3.4.2 小节中提到的两个深度卷积神经网络和额外的 ConvSmall_DiffAI 卷积神经网络进行实验分析，其中分别运行 DeepPoly 程序得到三个深度神经网络对于 100 个 MNIST 测试样本的最大可验证的扰动值。然而当首先设置二分查找法的上界为 0.064 后，几乎所有的测试样本的最大可验证的扰动值都达到了 0.064，所以难以从实验结果得到有用的信息。故在下述的实验中讨论的是将其上界调整为 0.12

（约 30 像素值）的实验结果。

在本实验所使用的三个深度卷积神经网络中 ConvSmall_Point 网络没有经过对抗训练，而另外的 ConvSmall_PGDK 是使用 PGD 方法[6]进行对抗训练获得的卷积神经网络，ConvSmall_DiffAI 是使用 DiffAI 方法[20]进行对抗训练所获得的卷积神经网络，且其网络结构与前两个相同。

同时因为实验目标是求得 DeepPoly 工具对于所测试的深度神经网络及其相应的每个测试输入样本所能验证的最大扰动值，所以在命令行参数需要设置的ϵ变量的值在本实验中没有意义，故不作讨论。另外本实验结果忽略了在三个卷积神经网络中本身被误分类的样本，所以共有 99 个测试样本可用于最终的实验评估。详细的实验比对结果如图 4-3 所示，其中横坐标表示每个测试样本的编号，纵坐标表示 DeepPoly 对于该深度神经网络和每个样本所能验证的最大扰动值，单位为像素值。

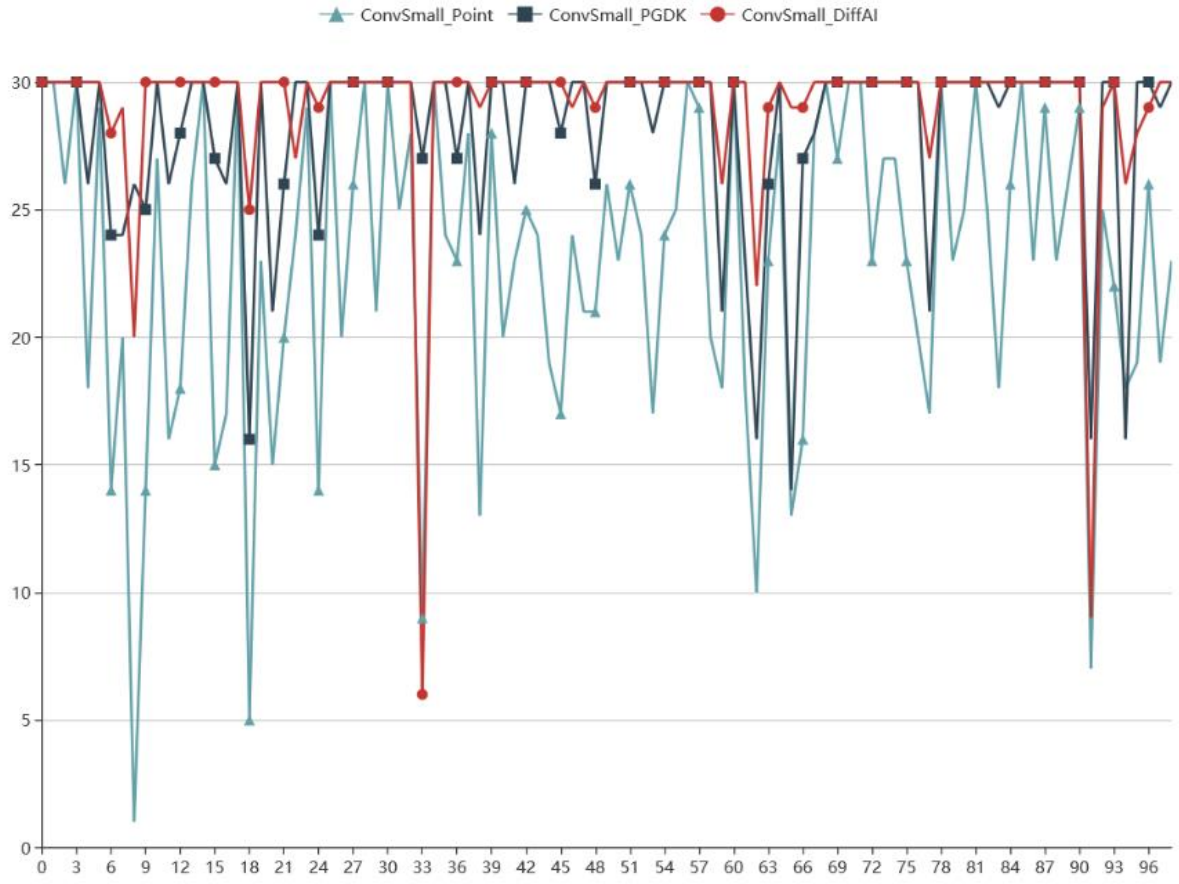

图 4-1 DeepPoly 分别对于三个网络及其 99 个样本所能验证的最大扰动值对比图

其中可以很直观地看出经过对抗训练的两个神经网络对于测试样本所能验证的最大扰动值几乎在每个测试样本上的表现都优于没有经过对抗训练的 ConvSmall_Point 神经网络。

同时还能观察到使用 DiffAI 方法进行对抗训练得到的 ConvSmall_DiffAI 网络的测试结果在大多数的测试样本上的表现上都优于使用 PGD 方法进行对抗训练得到的 ConvSmall_PGDK 网络。

除了借助于图表的可视化展示，更重要的是还可以通过计算每个神经网络的最大扰动值的平均值来从整体上比较每个神经网络的抗鲁棒性。比如经过计算上述的实验结果可以得到 ConvSmall_Point 网络对于所有 99 个测试样本的能被验证的最大扰动平均值为 22.8（单位：像素值），ConvSmall_PGDK 网络的结果为 28.1，ConvSmall_DiffAI 网络的结果为 29.0。这样从整体并且更加精确的角度同样得到了 ConvSmall_DiffAI 的表现略优于 ConvSmall_PGDK 的实验结果。

4.3 对比与小结

原来使用 DeepPoly 工具需要用户手动不断地调整其参数扰动变量ϵ的大小，观察其对于被测试神经网络所能验证具有鲁棒性的样本数的变化。但是这样的实验结果让用户难以准确地衡量神经网络的鲁棒性表现。借助于二分查找法和基于实际意义的建模不难以较小的代价得出更多的反馈信息，即 DeepPoly 工具对于每个测试样本所能验证的最大扰动值，同时省去了部分的人工冗余操作。另外基于此就可以通过单一数字评估指标（所有测试样本的能被验证的最大扰动平均值的平均值）来准确地衡量一个神经网络或对比两个神经网络的鲁棒性。综上所述，将二分查找法与 DeepPoly 工具相结合所得到的检测结果将能提供更多有价值的反馈信息。

5 基于 DeepPoly 的可视化系统

5.1 概况

在命令行的环境下可以直接运行 DeepPoly 程序或是其它深度神经网络的检测工具，但是实验数据一般需要借助于图表等其它的可视化形式才能让用户直观地对其进行观察总结。所以本课题开发了一个网页端的可视化系统用于之后的实际检测工作。

5.2 技术栈

本课题开发的可视化系统是一个前后端分离的网页端系统。前端语言使用 Vue.js 框架，后端语言使用 Python 语言的 Django 框架，而图表的绘制部分使用了百度提供的开源 Echarts 库[21]。其中 Vue.js 的内置指令能够帮助实现响应的数据绑定，便于实现此类逻辑相对简单的系统。而 Django 所基于的 Python 语言则易于调用系统的脚本来运行 DeepPoly 工具。

5.3 系统架构

系统的主要架构如图 5-1 的 UML 时序图所示。

整个系统的难点在于处理 DeepPoly 工具无法立即运算出所有输入样本的实验结果的问题。本小节中将通过介绍如下完整的流程来展示本系统是如何解决这个问题的。

首先用户在表单中添加需要测试的深度神经网络的模型文件并进行提交后，前端的 Vue.js 代码会向 Django 后台发送整个表单的数据。后台接受到这个请求后会立即开启额外的线程用于调用系统脚本，之后再向前端发送确认消息。

其中系统脚本中的逻辑是执行运行 DeepPoly 工具的命令。因为提前对于 DeepPoly 工具的代码进行了修改，所以 DeepPoly 在运行的时候会经过相同的时间间隔不断地将当前得到的实验数据输出到指定的文件中。

前端接收到确认消息后，会不断地向后台发送 Get 请求。后台接收到请求后会读取上述的指定文件，将当前的实验数据加载到内存中并将其打包返回给前端。前端会将接收到的部分实验数据通过 Echarts 库绘制出相应的图表。

其中每次后台读取数据时都会判断 DeepPoly 工具是否已经完成所有的检验工作。当得到正向的条件时会在返回数据中增加一个布尔变量用于传达实验数据已全部传送完毕。前端接收到此消息就会停止向后台发送 Get 请求。

最后前端发送一个 Post 请求用于获取两个神经网络最终的实验结果数组并以此来绘制对比图。这里有必要说明下，因为每个神经网络可能会对某些输入样本本身就是误分类的，所以这些特定的样本不能作为两者鲁棒性对比的依据，所以在绘制单个神经网

络的图表时只需要剔除其中误分类的输入样本即可，但是在用于绘制对比图的数据中系统只保留两者都能正确分类的样本。当后台删除临时存储文件并返回结果后一个完整的流程就完成了。

5.4 功能阐述

系统的整体效果可以参考第 1 部分的代码仓库页面的介绍。其中主要的功能分为两部分。其一是用户可以上传 tf 或 pyt 格式的神经网络模型文件，系统会根据 DeepPoly 工具运行的实验结果不断地绘制出横坐标为测试样本编号，纵坐标为可验证的最大扰动值的折线图。并最终还另外给出两个神经网络的对比图。其二是用户上传使用改进后的 DeepPoly 工具检测神经网络的实验结果文件，系统也会根据文件绘制出与第一点相同的三张折线图用于分析。

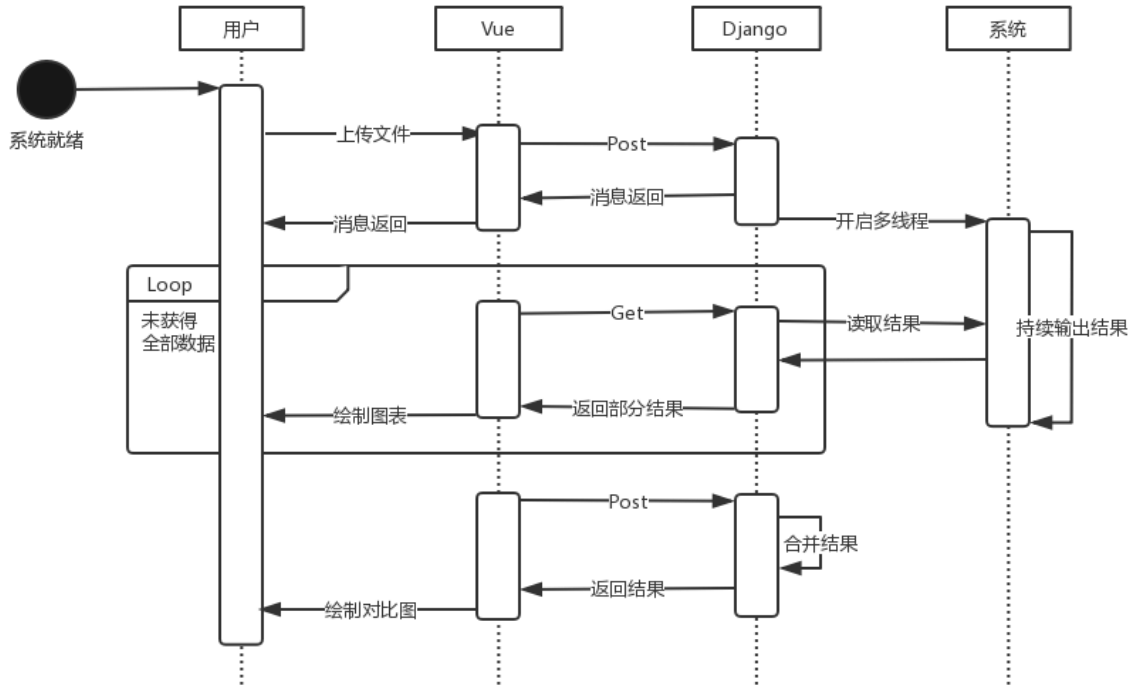

图 5-1 可视化系统的时序图

6 应用 DeepPoly

6.1 概况

在本部分将介绍使用改进后的 DeepPoly 工具检测一些流行的开源深度神经网络对于测试样本的鲁棒性的实验过程，并对相应的实验结果进行分析与总结。检测过程中有一个关键的细节是扰动参数变量ϵ的确定。因为过小的ϵ变量值无法对于深度神经网络的分类结果造成有效的扰动，也就是说对抗样本通常会施加比ϵ更大的扰动就会使得相应的检测结果缺少实际的意义。而过大的ϵ变量值可能会对测试样本产生本质的改变。所以本课题借鉴董胤蓬等人在《Boosting Adversarial Attacks with Momentum》里介绍生成对抗样本的方法中所使用的ϵ变量的最大像素值 16[6]，对其进行归一化（$\frac{16}{255.0} \approx 0.063$）后确定检测过程中$\epsilon$变量值侧重在 0.06 左右。

另外本课题选择如下三个神经网络进行实验分析的原因是：

- LeNet-5 是用于手写数字识别的卷积神经网络，当年美国大多数银行就使用它来识别支票上面的手写数字的，它是早期卷积神经网络中最有代表性的实验系统之一。
- Autumn 是应用于自动驾驶领域中预测当前驾驶角度的卷积神经网络，保障其可靠性的重要性不言而喻。另外它也是回归神经网络，所以也是对 DeepPoly 工具适用性方面的一次尝试。
- ChestNet 是根据胸部 X 光片（之后以胸片简称）来判断病人是否患有肺炎的卷积神经网络，所以预先保证其鲁棒性就能减少之后实际应用所产生的隐患。

6.2 LeNet-5

6.2.1 模型介绍

LeNet-5 是 LeCun 等人在 1998 年提出的经典卷积神经网络[14]。从图 6-1 中我们可以看到 LeNet-5 的输入层为32×32的一维图像，而后是 1 层卷积核大小为$5 \times 5 \times 6$且步长为 1 的卷积层；1 层采样区域为2×2且步长为 1 的平均池化层；1 层卷积核大小为$5 \times 5 \times 16$且步长为 1 的卷积层；1 层采样区域为2×2且步长为 1 的平均池化层；1 层卷积核大小为$5 \times 5 \times 120$且步长为 1 的卷积层；包括输出层在内的两层全连接层。

然而 LeNet-5 的一些细节和如今业界对于神经网络的主流设计有些出入，所以本课题所使用的 LeNet-5 网络[22]整体上借鉴了论文中的结构，但是它做出了如下的几点改进：

- 激活函数不再使用 Sigmoid 或 Tanh 激活函数，而是使用 ReLU 激活函数。
- 以最大池化操作替代平均池化操作。另外 DeepPoly 也无法处理平均池化操

作。

- 因为训练集采用 MNIST 数据集，即输入图像是28×28的矩阵，所以为了适应网络结构在进行卷积操作前对输入图像进行填充。
- 在训练神经网络的过程中使用随机失活的手段以防止模型对于训练集过拟合。

经过完整的训练过程它在 MNIST 的测试集上有 99.1%的准确率。

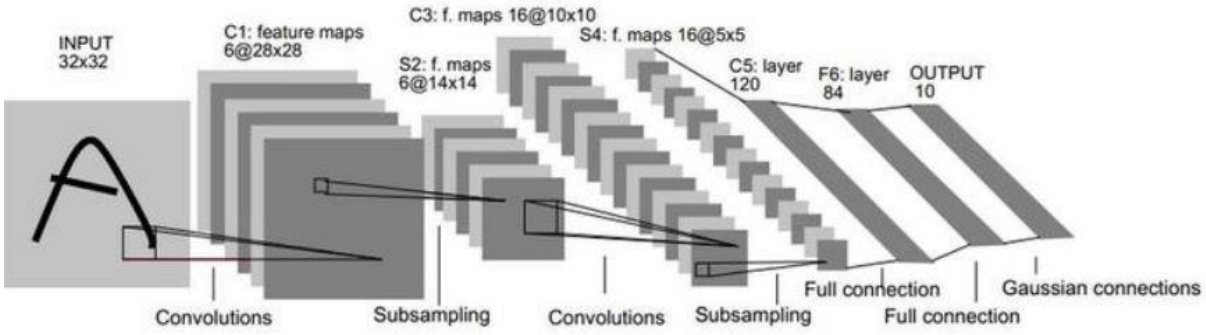

图 6-1 LeNet-5 卷积神经网络的结构

6.2.2 准备工作

首先因为 LeNet-5 的神经网络结构包含随机失活操作，所以使用 3.3 节中提到的方法对其代码做出相应的修改以忽略随机失活操作。此外在__main___.py 的代码中还有一处获取输出张量的代码需要修改，即 ERAN(sess.graph.get_tensor_by_name('logits:0'), sess)。可以看到源代码通过“logits:0”的张量名来获取了一个张量。原作者的意图是获得最后一个用于输出的张量来进行后续的处理。然而不同的神经网络在构建的时候作者的命名方式都是不同的，所以需要在进行检测前预先确认被测试神经网络用于输出的张量的名称，比如该例为“Lenet/fc9_1/Relu:0”。

其次检测实验中先设定$\epsilon = 0.06$并获得 100 个 MNIST 测试样本的整体结果以此来确定实验时ϵ的大小。经过测试发现 DeepPoly 工具能够验证为鲁棒性的测试样本比例仅为 1/98（分类错误的 2 个样本不予考虑，其余 98 个样本中有 1 个样本验证具有鲁棒性）。所以在之后的实验中选用ϵ为 0.02，0.03，0.04，0.05 和 0.06 分别进行测试。

同时为了获得更加多样和严谨的实验结果，本实验还另外引入了 3.4.2 小节中所提到的 ConvSmall_Point 卷积神经网络来进行比对实验。并且本实验还使用了改进后的 DeepPoly 工具计算每个测试输入样本的最大扰动值并使用可视化系统绘制图表进行分析。

6.2.3 实验结果分析

整体的分析结果如图 6-2 和图 6-3 所示。其中图 6-2 的横坐标是ϵ变量值的大小，纵坐标是验证具有鲁棒性的样本数，而图 6-3 的横坐标表示每个测试样本的编号，纵坐标表示 DeepPoly 对于该深度神经网络和每个样本所能验证的最大扰动值。

从图 6-2 中可以看到ϵ每增加 0.01，DeepPoly 对于 LeNet-5 所能验证的具有鲁棒性

的样本比例都会下降超过 50%。而 ConvSmall_Point 神经网络在$\epsilon = 0.06$时仍能对 90 个测试样本具有鲁棒性。其中更加重要的是，ConvSmall_Point 网络只有两层卷积层和一层全连接层，且除去输入层外总共包含 3604 个神经元。而 LeNet-5 网络包含两层卷积层，两层最大池化层和两层全连接层，神经元数量远超过 ConvSmall_Point 网络。但是从在不同的ϵ变量下得出的测试样本的鲁棒性比例来看，ConvSmall_Point 的实验结果远远好于 LeNet-5。这就意味着结构复杂的神经网络并不能保证其在鲁棒性测试上的优异表现。因为其可能会由于对于训练集的部分特征的权重参数分配过大（过拟合）而导致不合理的权重参数被对抗样本所利用。

从图 6-3 中两个神经网络可验证的最大扰动值的曲线也可以看到几乎在每个测试样本的最大扰动值的验证上，ConvSmall_Point 的抗鲁棒性表现都优于 LeNet-5。其中 LeNet-5 和 ConvSmall_Point 的平均最大扰动值分别是 6.8 和 22.9。这就意味着当对输入样本添加约 7 个像素值的噪声扰动，LeNet-5 可能会对几乎所有的干扰测试样本误分类。而 7 个像素值的扰动对于人眼来说其差别是非常细微的，这无疑会给神经网络在实际的应用中带来很大的风险。

同时实验数据中存在一个有趣的细节也可以通过图 6-3 两条折线的变化观察所得到。即虽然 ConvSmall_Point 对于大多数输入样本的可验证的最大扰动值比 LeNet-5 的高，但是它们二者曲线的整体分布形状是十分相似的，这一点通过观察图 4-3 也可以得到验证。这意味着不同的神经网络会对相同的测试输入样本具有较差的抗鲁棒性，这和在本论文中 2.2 小节所提到的对抗样本具有泛化性有较为相似的指向性。所以通过本论文中改进后的 DeepPoly 工具可以非常快速地确定干扰性较强的测试输入样本。如果后续要对于神经网络进行鲁棒性方面的改进就可以重点关注神经网络对于这些特殊的测试输入样本的鲁棒性是否有较大的提升。

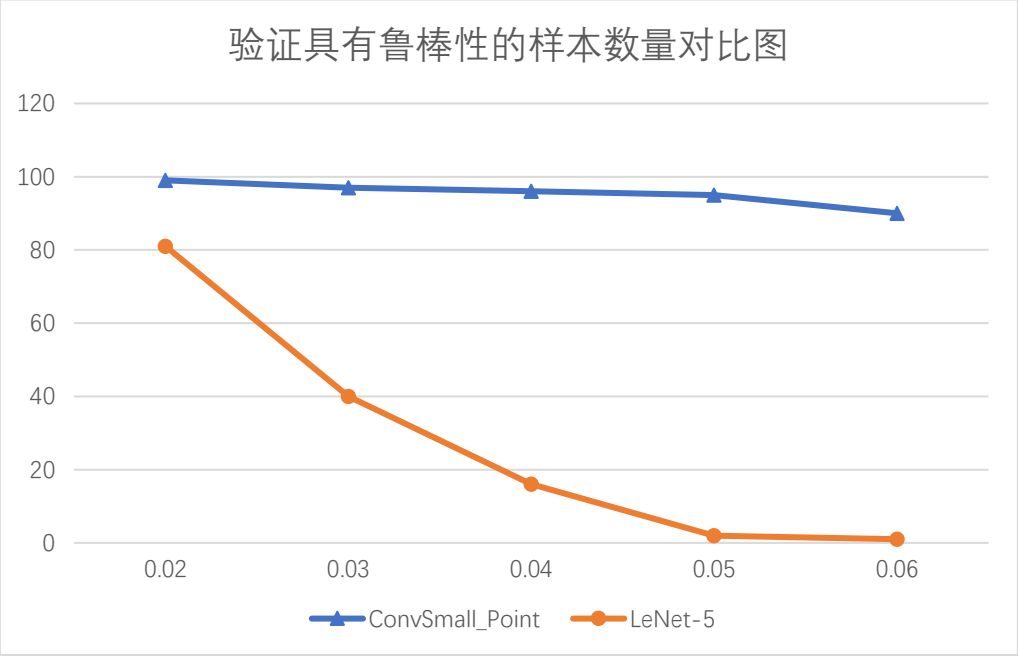

图 6-2 LeNet-5 与 ConvSmall_Point 的对比检测结果

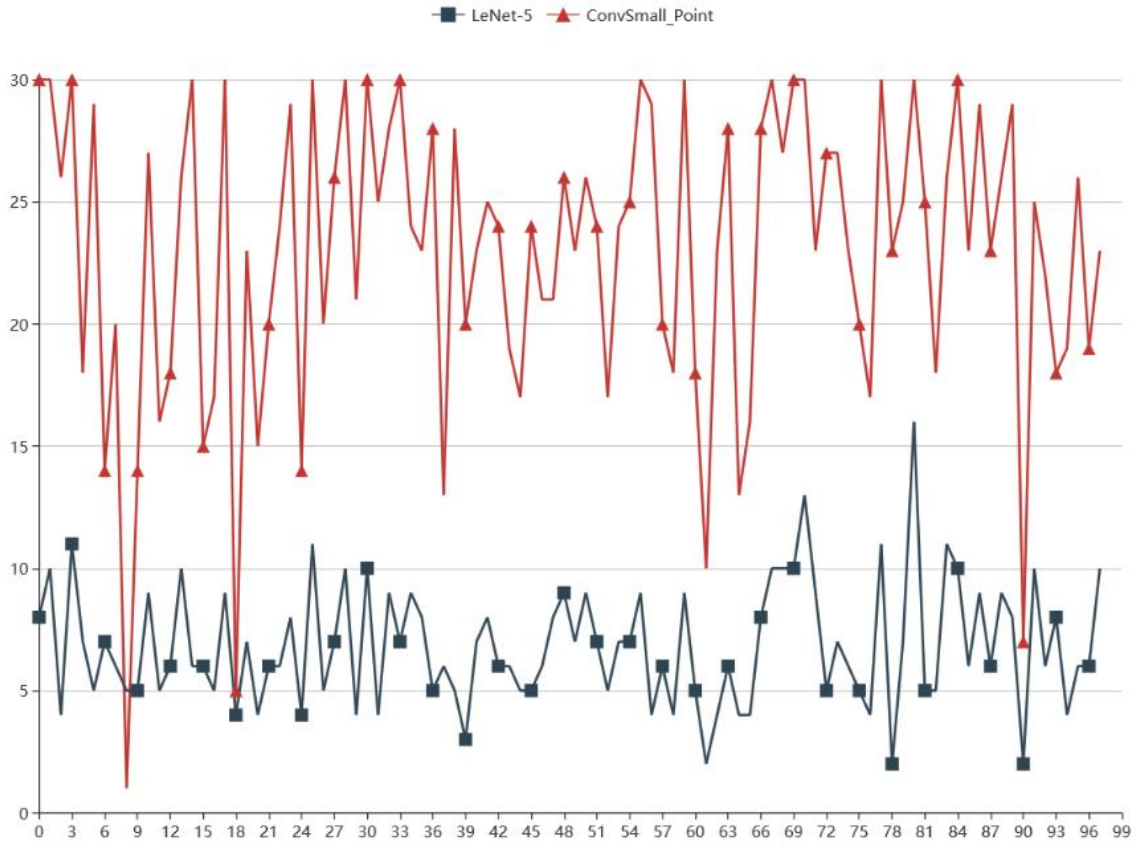

图 6-3 DeepPoly 工具分别对于 LeNet-5 和 ConvSmall_Point 神经网络及其 98 个样本所能验证的最大扰动值对比图

6.3 Autumn

6.3.1 模型介绍

Autumn[23]是 Udacity[24]上发布的用于辅助自动驾驶的神经网络模型。其中 Autumn 的功能是根据车载摄像头拍摄的照片预测当前的驾驶角度。

Autumn 由数据预处理模块和卷积神经网络（CNN）两部分组成。具体来说，Autumn 首先通过 OpenCV 库的 calcOpticalFlowFarneback 方法计算原始图像的光流，之后将得到的结果输入给 CNN 以预测转角。其中 Autumn 的网络架构为：输入的图片大小为$66 \times 200 \times 3$；1 层卷积核大小为$5 \times 5 \times 24$且步长为 2 的卷积层；1 层卷积核大小为$5 \times 5 \times 36$且步长为 2 的卷积层；1 层卷积核大小为$5 \times 5 \times 48$且步长为 2 的卷积层； 1 层卷积核大小为$5 \times 5 \times 64$且步长为 1 的卷积层；1 层卷积核大小为$5 \times 5 \times 64$且步长为 1 的卷积层；5 层采用随机失活操作的全连接层；最后采用反正切函数得到最后的输出值。该模型由 OpenCV、Keras 和 Tensorflow 共同实现[25]。

6.3.2 验证目标及思路

虽然 Autumn 模型是回归神经网络，即只有一个输出神经元，但 DeepPoly 的本质

是通过抽象域和用于特定激活函数的抽象转换器来对测试的神经网络模型函数进行线性近似，也就是说它是沿着神经网络的层级结构从输入层向输出层进行推导得出每层神经元对于指定的输入区间所能验证的取值范围。而 Autumn 网络的鲁棒性可以通过验证当对测试输入样本添加细微的噪声，其输出的驾驶角度是否会有较大的波动来验证。所以实验的目标构想是通过查看并理解 DeepPoly 工具底层运算的 C 语言代码并对其进行改动使得程序在适当的运行时间打印出输出神经元的取值范围，就能将其与正确的确定输出值进行比较判断添加微小扰动的驾驶场景图片是否会导致 Autumn 深度神经网络输出与正确驾驶角度偏离较大的区间值，即表现出不鲁棒的行为。

6.3.3 准备工作

首先因为 Autumn 模型的神经网络结构包含随机失活操作，所以同样使用 3.3 节中提到的方法对其代码做出相应的修改。

其次获取神经元上下界需要改动 ELINA 工程目录下 fppoly 文件夹下 fppoly.c 文件的代码，具体的步骤可以参考引言部分中 github 地址的代码实现。改动完该文件的代码之后需要在系统中重新编译安装 ELINA 库使改动生效。

另外 tensorflow_translator.py 代码中为使用 deepzono 抽象域（工具中包含的另一个抽象域）而准备相关信息的代码会因为读取神经网络操作输出的矩阵维数而报错，可以选择忽略此代码。

6.3.4 实验结果分析

因为 Autumn 的输入层的图片维数和网络的整体架构规模过大致使单独的一个测试样本在拥有 8 核逻辑 CPU 的虚拟机上运行需要约 6 个小时才能得出结果，加上测试的结果与预期目标相差较大，故仅选取如表 6.1 的数据进行展示。在输入区间的上下界相同时，DeepPoly 程序输出神经元的值（此时上下界相同故可看成近似的确定值）与使用 Tensorflow 读取模型得出的结果相同。然而可以看到当添加细微扰动的输入就会造成输出神经元的上下界的数值异常偏离。即使是 1 个像素值的扰动（$\frac{1}{255} \approx 0.0039$）经过整个神经网络的误差传递导致最后的输出上下界也和正确结果相去甚远。

从理论上本论文认为实验结果不理想的原因主要有两点。其一是因为 DeepPoly 工具所基于的理论中有较为关键的一步操作无法在回归神经网络中得到实现，即在测试的神经网络的输出层的后一层人为地新建额外的神经元结点，且每个神经元的输入是前一层（输出层）两个神经元的仿射变换从而通过新神经元的上下约束的不等式来表示两个输出层神经元的差值区间，并将其反推到输入层以提高差值区间下边界的精确度，具体的实现也可以在 2.3.5 小节中的详细推导中查看。然而因为回归神经网络的输出层只有一个神经元导致其无法依靠最后一步反向推导的操作进一步地消除误差。其二是因为 Autumn 卷积神经网络的规模过大，其包含数以万计的神经元和多达 10 层的网络结构致使抽象近似所造成的误差被不断放大。然而不可否认的是这是所有不具备

完备性的鲁棒性检验方法都会遇到的困难。

表 6.1 Autumn 模型部分的实验结果

准确值	扰动变量大小	实际神经元输出下界	实际神经元输出上界
0.00312615	0.02	-2407562.917855	2604364.328327
0.15915938	0.01	-36282.243145	39334.920404
0.15289488	0.005	-4174.664838	4528.129845
0.14972171	0.004	-1471.441848	1595.615299

6.4 ChestNet

6.4.1 模型介绍

ChestNet 是 github 上开源的卷积神经网络[24]，但它的名字是在本论文中便于引用而单独指定的。原作者使用 Keras 和 Tensorflow 框架对其进行训练，其中 ChestNet 原来的输入图片大小为$64 \times 64 \times 3$且其原来的网络架构为：1 层卷积核大小为$3 \times 3 \times 64$且步长为 1 的卷积层；1 层采样区域为2×2且步长为 1 的最大池化层；1 层卷积核大小为$1 \times 1 \times 128$且步长为 1 的卷积层；1 层采样区域为4×4且步长为 2 的最大池化层；1 层卷积核大小为$3 \times 3 \times 256$且步长为 2 的卷积层；1 层采样区域为3×3且步长为 1 的最大池化层；两层带有 ReLU 激活函数的全连接层；1 层带有 Sigmoid 激活函数的输出层。

源代码首先采用了 Keras 框架中的 ImageDataGenerator 对数据集进行缩放、剪切和反转等图像增广操作来扩充数据集。接着采用 flow_from_directory 方法读取训练集和测试集并将它们缩放为指定的64×64的三维图像。最后定义二进制的交叉熵为损失函数，并采用 Adam 梯度下降算法[27]进行模型的训练。

因为其主要的功能是根据胸片判断病人是否患有肺炎，所以可看成类似于二进制的输出。也就是说输出层也只有单个神经元，小于 0 的输出值代表患者正常，而大于 0 的输出值则代表患有肺炎。

6.4.2 准备工作

因为 ChestNet 只提供了网络架构和数据集，所以需要自己进行神经网络的训练工作。经过 5 次的整体数据集（原网络只使用了 1 次的整体数据集迭代）迭代 ChestNet 可以在测试集上达到超过 90%的准确率。然而，在使用 DeepPoly 工具对 ChestNet 进行检测的时候，底层的 C 语言代码会产生段错误。经过仔细的排查以及猜想，判断是由于最大池化函数的步长所造成的，因为原 DeepPoly 工具默认了最大池化函数的步长与采样区域的大小相同而且也通过实验验证了此猜想，于是本实验对神经网络的结构进行了微调。另外由于原 ChestNet 训练后的模型文件在 100MB 左右，远大于在 6.3 小节中只有 20MB 大小的 Autumn 模型，根据之前的实验结果可以判断此神经网络的整体规模已经完全超过了 DeepPoly 工具所能适应的网络模型范围。此外虽然原 ChestNet 网

络将数据集转变为三通道的图片再进行训练，但是没有经过预处理的数据集的图片本身是单通道的，因此没有复杂化的必要。综合以上三点原因，最终本实验所使用的 ChestNet 神经网络架构为：输入图片大小为$64 \times 64 \times 1$的输入层；1 层卷积核大小为$3 \times 3 \times 16$且步长为 1 的卷积层；1 层采样区域为2×2且步长为 2 的最大池化层；1 层卷积核大小为$2 \times 2 \times 32$且步长为 1 的卷积层；1 层采样区域为2×2且步长为 2 的最大池化层；1 层卷积核大小为$4 \times 4 \times 64$且步长为 1 的卷积层；1 层采样区域为3×3且步长为 3 的最大池化层；两层带有 ReLU 激活函数的全连接层；1 层带有 Sigmoid 激活函数的输出层。

从图 6-4 的训练过程中可以看到最终模型对于训练集的准确率为 93.9%，而对于测试集的准确率为 89.74%。整体的结果略低于原先的 ChestNet 神经网络但是现在的模型文件大小只有 4MB，在耗时方面已经达到了可以用于 DeepPoly 工具检测的范围内。

但在实际的检测过程中发现由于 ChestNet 也是单个输出神经元所以整体的实验结果和 6.3 小节中 Autumn 模型的实验结果表现类似，即会得到跨度非常大的输出神经元的上下界区间。

但是 ChestNet 本质上是分类神经网络所以可以将其转变为多输出神经元的网络。所以将最后的输出层进行了微调，将其转变为具有两个输出神经元并带有 Softmax 激活函数的全连接层。并且将三层卷积层的过滤器数分别缩减到 8 个，8 个和 16 个，也去掉了第一层的全连接层。最终经过 10 次整体数据集的迭代训练，其整体表现与单 Sigmoid 输出神经元基本持平，其中对于训练集的准确率为 94.34%，而对于测试集的准确率为 89.26%。

而对于最终的使用 DeepPoly 工具进行检测的过程，本实验直接使用了 Keras 保存模型的 h5 格式文件。虽然 DeepPoly 工程中__main__.py 部分的代码明确只支持三种格式的神经网络文件，但是发现 tensorflow_translator.py 文件中有对于 Keras 模型相对应的处理代码。虽然其中还有一处错误，但是经过对于这两个文件简单的代码改动就可以直接提供 h5 格式文件进行鲁棒性的测试。

```
Epoch 1/4
2019-04-21 22:18:08.064270: I tensorflow/stream_executor/dso_loader.cc:152] successfu
20/20 [==============================] - 3s 155ms/step - loss: 0.2839 - acc: 0.8846
 - 34s - loss: 0.3731 - acc: 0.8355 - val_loss: 0.2839 - val_acc: 0.8846
Epoch 2/4
20/20 [==============================] - 3s 164ms/step - loss: 0.2713 - acc: 0.8910
 - 33s - loss: 0.2147 - acc: 0.9041 - val_loss: 0.2713 - val_acc: 0.8910
Epoch 3/4
20/20 [==============================] - 3s 155ms/step - loss: 0.3043 - acc: 0.9054
 - 33s - loss: 0.1851 - acc: 0.9296 - val_loss: 0.3043 - val_acc: 0.9054
Epoch 4/4
20/20 [==============================] - 3s 167ms/step - loss: 0.3201 - acc: 0.8974
 - 33s - loss: 0.1574 - acc: 0.9390 - val_loss: 0.3201 - val_acc: 0.8974
```

图 6-4 ChestNet 在 Keras 上的训练过程

6.4.3 实验结果分析

因为单 Sigmoid 输出神经元的 ChestNet 网络经过 DeepPoly 程序的底层 C 语言代码的推导得出的上下界范围相较于 Autumn 模型的实验结果稍好但是依然没有很大的参考价值，故仅选取表 6.2 的数据进行展示。其中准确值表示的是采用 Sigmoid 激活函数之前的输出值，所以它的范围超出了$[0,1]$。然而经过 DeepPoly 工具推导出的神经元值的区间基本涵盖了$[0,1]$中大部分的取值范围，难以根据其判断神经网络的鲁棒性。所以综合 Autumn 模型和单 Sigmoid 输出神经元的 ChestNet 模型的实验结果可以看到目前的 DeepPoly 工具的精度近似还远无法满足回归神经网络系统的检测工作。

表 6.2 单 Sigmoid 输出神经元的 ChestNet 模型部分的实验结果

准确值	扰动变量大小	实际神经元输出下界	实际神经元输出上界
-2.362	0.02	-274.4	321.1
-2.362	0.004	-17.6	21.6
-0.806	0.02	-266.8	321.8
-0.806	0.004	-17.9	24.6
-2.2	0.02	-271.3	320.5
-2.2	0.004	-16.4	20.6
-2.06	0.02	-271.2	318.6
-2.06	0.004	-18.1	22.9
-1.81	0.02	-270.0	315.9
-1.81	0.004	-18.0	22.8

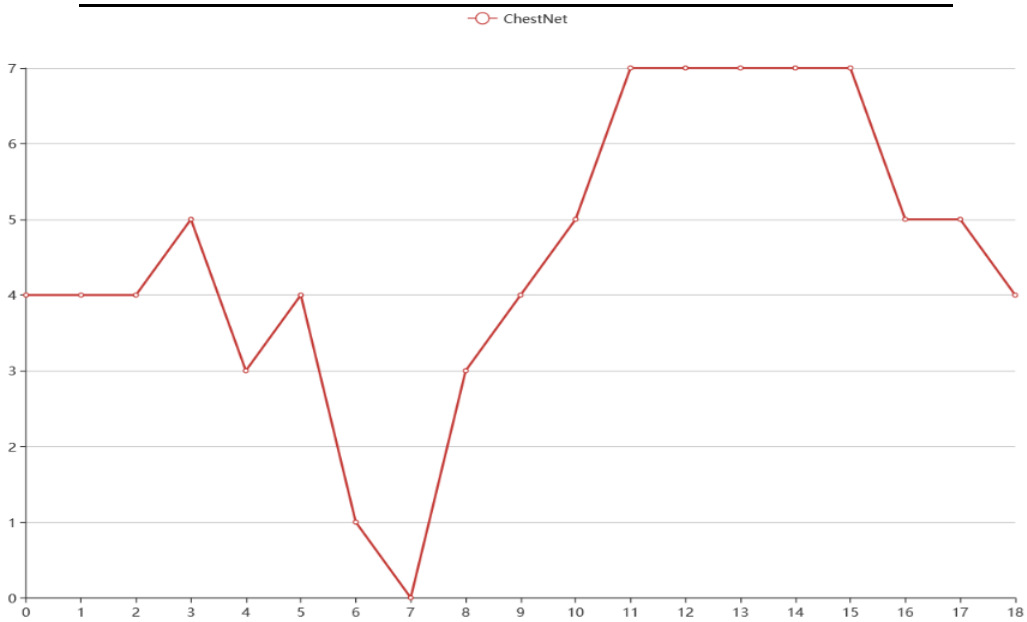

图 6-5 DeepPoly 工具对于简化的 ChestNet 及其 19 个样本所能验证的最大扰动值

另一方面，如图 6-5 所示经过进一步简化并拥有两个输出神经元的 ChestNet 网络的鲁棒性检测的实验结果也不够理想。其中横坐标代表测试样本的编号，纵坐标代表 ChestNet 对于此测试样本的最大扰动值，但是单位仅为 0.1275 像素值（归一化后为 0.0005）。也就是说 ChestNet 对于所有测试样本的可验证最大扰动值都小于 1 个像素值

（$0.1275 * 7 = 0.8925 < 1$）。显然，这样的实验结果无法对于之后的实际应用提供足够的帮助。

并且从之前 6.2 小节的实验结果来看，两个用于 MNIST 数据集分类的卷积神经网络对于每个测试样本的最大扰动值都在 1 个像素值以上。所以造成可验证的最大扰动值较小的原因可能很大程度上还是由于神经网络的规模过大而造成的精度损失。

6.5 对比与小结

综合比较本章中对于 LeNet-5、Autumn 和 ChestNet 三个神经网络的最终的检测实验结果，分别对目前深度神经网络鲁棒性和 DeepPoly 工具做出如下总结：

- 对于没有经过对抗训练的深度神经网络，即使最终在测试集上的准确率表现优异也并不意味着其鲁棒性方面也能同样突出。
- 目前的 DeepPoly 工具能够较好地应用于较小规模的神经网络的检测工作。但是因为其基于非完备性的理论，所以当应用于较大规模的神经网络时，较之于效率，精度是首要影响最终实验结果的点。

7 结论

在本篇论文中主要介绍了使用 DeepPoly 工具检验特定深度神经网络的整个流程及其相应的实验结果。并且在实验过程中重构了 DeepPoly 工具的部分代码来增强它的适用性和运行效率。另外本论文将二分查找法与 DeepPoly 工具相结合并开发了可视化系统来直观地展示它的实验结果并取得了良好的效果。总体的实验结果表明 DeepPoly 工具底层所基于的抽象解释理论致使它难以在回归深度神经网络和较大规模的神经网络中到达实用的拟合精度，但它在较小规模的分类深度神经网络中的性能和精度两方面的表现表明它可以较好地被用于评估部分神经网络的鲁棒性。另外实验结果还表明即使在验证集的准确率上表现优异的神经网络可能不具备合格的鲁棒性。

目前对于深度神经网络鲁棒性的非完备性的检测方法都是对于性能和效率两方面的权衡。但是根据本论文中的实验结果来看，即使是在两方面表现都不俗的 DeepPoly 工具，对于如今大规模神经网络的应用已是常态的现状，也很难起到实用的鲁棒性检测作用。所以尽管目前对于深度神经网络鲁棒性的检测工作取得了一定的进展，但是要想成功地解决整个神经网络推理难题，还需要进行更多的研究。

另外，本课题检测的神经网络数量有限，所以只能得出相对粗糙的结论。有关于目前主流深度神经网络实际鲁棒性的表现还有待于进一步的探究。

参考文献

[1] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.

[2] Bojarski M , Del Testa D , Dworakowski D , et al. End to End Learning for Self-Driving Cars[J]. 2016.

[3] Zhang Y , Zhao D , Sun J , et al. Adaptive Convolutional Neural Network and Its Application in Face Recognition[J]. Neural Processing Letters, 2016, 43(2):389-399.

[4] Ozyilmaz L , Yildirim T . Artificial neural networks for diagnosis of hepatitis disease[C]// International Joint Conference on Neural Networks. IEEE, 2003.

[5] Szegedy C , Zaremba W , Sutskever I , et al. Intriguing properties of neural networks[J]. Computer Science, 2013.

[6] Dong Y , Liao F , Pang T , et al. Boosting Adversarial Attacks with Momentum[J]. 2017.

[7] Lipton Z C . The Mythos of Model Interpretability[J]. Communications of the ACM, 2016.

[8] Weng T W , Zhang H , Chen H , et al. Towards Fast Computation of Certified Robustness for ReLU Networks[J]. 2018.

[9] Paz A , Moran S . Non deterministic polynomial optimization problems and their approximations.[J]. Theoretical Computer Science, 1977, 15(3):251-277.

[10] Katz G, Barrett C, Dill D L, et al. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks[J]. Lecture Notes in Computer Science, 2017.

[11] Gehr T, Mirman M, Drachsler-Cohen D, et al. Ai2: Safety and robustness certification of neural networks with abstract interpretation[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 3-18.

[12] Singh G, Gehr T, Püschel M, et al. An abstract domain for certifying neural networks[J]. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 41.

[13] Lécun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

[14] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images[R]. Technical report, University of Toronto, 2009.

[15] Carlini N , Wagner D . [IEEE 2017 IEEE Symposium on Security and Privacy (SP) - San Jose, CA, USA (2017.5.22-2017.5.26)] 2017 IEEE Symposium on Security and Privacy (SP) - Towards Evaluating the Robustness of Neural Networks[J]. 2017:39-57.

[16] 周志华. 机器学习[M]. 北京: 清华大学出版社, 2016.

[17] Goodfellow I J, Shlens J, Szegedy C. Explaining and harnessing adversarial examples[J]. arXiv preprint arXiv:1412.6572, 2014.

[18] Akhtar N , Mian A . Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey[J]. IEEE Access, 2018:1-1.

[19] Srivastava N , Hinton G , Krizhevsky A , et al. Dropout: A Simple Way to Prevent

Neural Networks from Overfitting[J]. Journal of Machine Learning Research, 2014, 15(1):1929-1958.

[20] Mirman M, Gehr T, Vechev M. Differentiable abstract interpretation for provably robust neural networks[C]//International Conference on Machine Learning. 2018: 3575-3583.

[21] Li D, Mei H, Shen Y, et al. ECharts: A declarative framework for rapid construction of web-based visualization[J]. Visual Informatics, 2018, 2(2): 136-146.

[22] ganyc717: LeNet. [CP/OL]. https://github.com/ganyc717/LeNet, 2017.

[23] Steering angle model: Autumn. [CP/OL]. https://github.com/udacity/self-driving-car/tree/master/steering-models/evaluation, 2016.

[24] udacity: self driving car. [CP/OL] https://github.com/udacity/self-driving-car, 2016.

[25] Zhang M , Zhang Y , Zhang L , et al. [ACM Press the 33rd ACM/IEEE International Conference - Montpellier, France (2018.09.03-2018.09.07)] Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, - ASE 2018 - DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems[C]// 2018:132-142.

[26] ameyas1: CNN_Medical_Pneumonia. [CP/OL]. https://github.com/ameyas1/CNN_Medical_Pneumonia, 2019

[27] Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.