软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007127]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

完备神经网络验证加速技术综述*

刘宗鑫[1,2], 杨鹏飞[1], 张立军[1,2], 吴志林[1,2], 黄小炜[3]

[1](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](中国科学院大学, 北京 100049)

[3](University of Liverpool, Liverpool L69 3BX, UK)

通信作者: 张立军, E-mail: zhanglj@ios.ac.cn

摘　要: 人工智能技术已被广泛应用于生活中的各个领域. 然而, 神经网络作为人工智能的主要实现手段, 在面对训练数据之外的输入或对抗攻击时, 可能表现出意料之外的行为. 在自动驾驶、智能医疗等安全攸关领域, 这些未定义行为可能会对生命安全造成重大威胁. 因此, 使用完备验证方法证明神经网络的性质, 保障其行为的正确性显得尤为重要. 为了提高验证效率, 各种完备神经网络验证工具均提出各自的优化方法, 但并未充分探索这些方法真正起到的作用, 后来的研究者难以从中找出最有效的优化方向. 介绍神经网络验证领域的通用技术, 并提出一个完备神经网络验证的通用框架. 在此框架中, 重点讨论目前最先进的工具在约束求解、分支选择与边界计算这 3 个核心部分上的所采用的优化方法. 针对各个工具本身的性能和核心加速方法, 设计一系列实验, 旨在探究各种加速方式对于工具性能的贡献, 并尝试寻找最有效的加速策略和更具潜力的优化方向, 为研究者提供有价值的参考.

关键词: 完备验证; 可满足性模理论; 人工智能安全; 形式化方法; 鲁棒性

中图法分类号: TP311

中文引用格式: 刘宗鑫, 杨鹏飞, 张立军, 吴志林, 黄小炜. 完备神经网络验证加速技术综述. 软件学报. http://www.jos.org.cn/1000-9825/7127.htm

英文引用格式: Liu ZX, Yang PF, Zhang LJ, Wu ZL, Huang XW. Survey on Acceleration Techniques for Complete Neural Network Verification. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7127.htm

Survey on Acceleration Techniques for Complete Neural Network Verification

LIU Zong-Xin[1,2], YANG Peng-Fei[1], ZHANG Li-Jun[1,2], WU Zhi-Lin[1,2], HUANG Xiao-Wei[3]

[1](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

[3](University of Liverpool, Liverpool L69 3BX, UK)

Abstract: Artificial intelligence (AI) has been widely applied to various aspects of lives. However, as the primary technique of realizing AI, neural networks can exhibit undefined behavior when faced with inputs outside of their training data or under adversarial attacks. In safety-critical fields such as autonomous driving and intelligent healthcare, undefined behavior can pose significant threats to human safety. Therefore, it is particularly crucial to employ complete verification methods to verify the properties of neural networks and ensure the correctness of the behavior. To enhance the verification efficiency, various complete neural network verification tools have proposed their optimization methods. However, the true influence of these methods has not been thoroughly explored, making it challenging for researchers to identify the most effective optimization directions. This paper introduces the common techniques in neural network verification and presents a universal framework for complete neural network verification. Within this framework, it focuses on discussing

* 基金项目: 中国科学院基础研究青年团队 (CASYSBR-040); 中国科学院软件研究所新培育方向项目 (ISCAS-PYFX-202201)
本文由“形式化方法与应用”专题特约编辑曹钦翔副教授、宋富研究员、詹乃军研究员推荐.
收稿时间: 2023-09-10; 修改时间: 2023-10-30, 2023-12-13; 采用时间: 2023-12-20; jos 在线出版时间: 2024-01-05

the optimization methods employed by state-of-the-art tools for constraint solving, branch selection, and boundary computation. Meanwhile, a series of experiments are designed to investigate the contributions of various acceleration techniques to the performance of each tool and to explore the most effective acceleration strategies and more promising optimization directions. Finally, valuable references can be provided for researchers in this field.

Key words: complete verification; satisfiability modulo theories; artificial intelligence security; formal methods; robustness

1 引 言

人工智能技术已被广泛应用于图像识别[1]、医学系统[2]、飞行控制[3]和自动驾驶[4]等各种领域. 然而, 神经网络作为实现人工智能技术的主要手段, 其运行过程中的各种性质却难以得到保证. 攻击者可以对神经网络的输入进行微小的扰动致使模型输出错误结果[5]. 此外, 在面对训练数据分布之外的输入时, 神经网络也可能产生出乎意料的行为. 在自动驾驶汽车或医学诊断等关键应用中, 这些问题可能对人们的生命安全构成重大威胁. 例如, 自动驾驶汽车目标识别系统受到对抗性攻击可能会导致汽车将行人错误地识别为路标; 过度拟合的医学诊断系统可能会产生错误的诊断结果. 因此, 验证神经网络模型的性质 (鲁棒性、安全性等), 确保模型在存在扰动等特殊环境下正确且一致地工作对推动人工智能技术在现实中的应用至关重要. 目前对神经网络安全性的评估主要有基于测试的方法与基于形式化验证的方法两类. 基于测试的方法只考虑有限的输入集, 无法覆盖所有可能的输入, 因此可能存在未检测到的错误或漏洞, 难以进行精准性验证. 为确保神经网络始终具有所需要的性质, 我们需要使用形式化的方法对神经网络的性质进行严格验证.

1.1 神经网络验证技术

神经网络验证问题最早于 2010 年被 Pulina 等人[6]提出, 作者使用基于神经元输入划分的精化方法, 首次将形式化方法中的思想与技术用于神经网络的安全评估中. 2017 年, Katz 等人[7]提出并实现了基于可满足性模理论 (satisfiability modulo theories, SMT) 的验证工具 Reluplex 并证明 ReLU 神经网络的安全性质验证问题是 NP 完全的; Ehlers[8]独立实现工具 Planet, 分析求解中冲突出现的原因并利用冲突驱动的子句学习 (conflict-driven clause learning, CDCL) 框架进行剪枝; Huang 等人[9]基于 z3[10]实现了自动化验证框架用于检测网络在输入图像面临划痕、光照等条件改变时的鲁棒性. 这 3 个工作被视为 SMT 方法在神经网络验证中的先驱. 自此, 神经网络领域迅速发展, 涌现了许多神经网络验证工具和各种各样的验证方法, 如抽象解释[11−16], 符号传播[17−23], SMT[24−26], 凸优化[27,28], 数学规划[29−31], 基于反例的抽象精化方法[32−34]和利普希茨常数[18,35,36]等.

在神经网络形式化验证领域, 可靠性 (soundness) 和完备性 (completeness) 是两个重要概念. 所有验证工具都必须尽可能保证其结果的可靠性, 但在验证的完备性上, 不同工具会做出不同的取舍. 如果只允许工具返回性质成立或者不成立, 那么可靠性意味着, 工具返回性质成立, 则网络在给定约束下一定具有该性质, 但工具返回性质不成立时, 性质有可能成立的. 而完备性则指如果性质成立, 那么使用工具一定可以成功证明. 非完备的方法通常速度较快, 但不够精确; 完备的方法精确, 但效率较低. 完备与非完备的方法并非对立关系, 通常完备的方法中往往使用非完备方法提供的信息进行加速, 大部分非完备的方法结合分支限界 (branch and bound, BaB) 技术也可以拓展为完备方法.

在本篇综述中我们主要关注可靠且完备的验证工具. 完备的验证方法的核心通常由约束求解、边界计算与分支选择 3 个部分组成, 其中后两个部分合称为分支限界. 在验证开始时, 我们首先进行边界计算, 也就是计算神经网络中每个节点的上下界, 越紧的上下界越有利于对性质的证明. 在进行边界计算时, 通常需要对网络进行一定程度的抽象 (上近似), 所以计算出的边界是真实边界的超集. 如果通过边界计算无法证明性质是否成立, 则需要使用更精确的约束求解方法. 约束求解是指在给定一组约束条件下, 寻找满足这些约束条件的变量取值的过程. 在神经网络形式化验证中, 通常将神经网络的输入、输出和中间状态视为变量, 并将神经网络表示为一系列约束条件的组合, 这些条件包括输入的边界、输出要满足的性质以及网络节点之间的关系等. 通过求解这些约束条件, 可以判断神经网络是否会在给定的输入约束下产生违反性质的输出. 如果约束求解依然无法判断性质是否成立, 则需要

进入分支选择步骤. 分支选择是通过启发式地枚举某个神经元的激活状态, 例如非线性激活函数 (如 ReLU 函数) 的激活与否, 来添加更严格的约束, 将抽象后的网络进行一定程度的精化, 使网络行为更贴近原始网络. 最坏情况下, 当所有节点的状态都被确定后, 网络将变成完全线性的网络, 此时可以使用线性规划方法进行精确求解. 需要注意的是, 有些工具, 如 α,β-CROWN (alpha-beta-CROWN) 的边界计算方法同时可以用于判断约束是否满足, 因此可以没有明确的约束求解部分.

在边界计算时, 可以使用抽象解释对网络进行抽象, 并使用凸优化等方法求得更紧的边界. 抽象解释通过对输入和输出进行抽象 (上近似) 以及构建抽象域来简化约束, 加快验证的速度, 但可能会丢失具体信息从而使验证不完备. 凸优化方法则利用对偶理论、拉格朗日乘子法等数学方法来优化神经网络的鲁棒性边界, 从而得到更加紧的上下界提升证明的精度. 其中, 对偶理论将优化问题转化为对偶问题, 并在对偶问题上求解来得到原问题的最优解; 拉格朗日乘子法则是将约束条件转化为目标函数的一部分, 并使用拉格朗日乘数来建立约束条件与目标函数之间的关系, 在优化边界的同时保证其上下界满足约束.

神经网络验证领域通常使用数学工具进行约束求解. 这些工具包括线性规划 (linear programming, LP)、混合整数规划 (mixed integer linear programming, MILP)、单纯形法 (simplex) 及其拓展 Reluplex 等. 线性规划是在线性约束下求得线性目标函数的一组最小化或最大化解的方法, 而混合整数线性规划是将线性规划问题中某些变量限定为整数. 单纯形法则先找到一个初始的可行解, 然后不断寻找相邻的解向量, 直到所有变量都满足给定约束. 此外, 还可以通过将验证问题转化为布尔逻辑公式, 并利用 SMT 求解器来求解. 这种方法通常能够高效地处理多种类型的约束条件.

分支求解主要依靠于启发式[37]的节点选择, 如根据节点的上下界范围, 选择节点进行分支后对输出产生的影响等因素选择某个神经元进行分支等. 后续实验表明, 合理的分支选择方法可以大幅度提升工具的性能.

1.2 完备验证技术优化

完备的方法可以精准地验证神经网络的性质, 但计算开销较大, 最坏情况下需要遍历所有神经元的状态组合, 占用大量的时间和计算资源. 为了克服这些问题, 研究者们提出了各种方法加速验证过程. 例如使用拉格朗日乘子法在计算边界的同时进行约束求解, 优先选择层数考前的节点分支等. 此外, 并行处理与神经网络攻击方法[38−42]也可以加速性质的证明. 还有一类反例引导的抽象精化 (counterexample-guided abstraction refinement, CEGAR) 方法, 从网络层面进行抽象, 从而减少节点加速证明. 最后, 有些工作更关注神经网络的增量验证[43,44]. 即神经网络的结构或者权值发生轻微扰动后, 利用验证原始网络过程中的信息提高第 2 次验证的效率.

目前关于神经网络验证领域的综述[45−48]着重于方法的广度, 在方法的深度的介绍上稍有欠缺. 这些综述往往只介绍每个工具最为核心的方法, 对其他加速手段只是简单提及. 此外, 在介绍方法时通常只给出相关的公式, 但缺乏简单的例子辅助理解. 更为关键的是, 这些文章很少对工具的性能进行实际评估, 只停留在方法的介绍层面. VNN-COMP[49,50]是神经网络验证领域的竞赛, 旨在促进对神经网络验证和验证工具的研究和发展. 竞赛中对比了多种工具的总体性能, 但缺乏对工具核心加速方法的探索, 仅得出性能较好的工具往往采用基于 GPU 的边界传播与分支限界框架的粗略结论[50], 无法得知这些工具所使用的各种技术真正起到的效果, 研究者难以从中找出最有效的优化方向.

本综述聚焦于完备神经网络验证领域. 在宏观角度, 我们将介绍完备神经网络验证的通用技术, 并提出一个通用框架帮助读者全面了解完备神经网络验证流程. 在微观角度, 我们将关注目前最先进的工具在框架中最为核心的 3 个部分, 即约束求解、分支选择与边界计算中所提出的优化方法. 我们对这些方法进行详细的阐述, 并给出简单的例子以方便读者理解. 我们还针对各个工具本身的性能和所使用的核心加速方式设计了各种实验. 这些实验旨在找出最有效的加速方法和更具潜力的优化方向, 为研究者提供一些有价值的参考.

1.3 文章结构

我们将在第 2 节给出神经网络问题的形式化定义. 第 3 节提出完备验证工具的通用验证框架, 并对较通用的

验证方法进行介绍. 第 4 节对每个工具的关键算法以及采用的优化方法进行介绍并给出易于理解的例子. 第 5 节则在统一的设备上对各个工具的总体性能进行评估, 并分别探讨其优化方法的有效性. 最后一节对全文进行总结, 并给出可能的优化方向.

2 形式化定义

为了方便说明, 在本文中我们用粗体表示矩阵, 如 $\mathbf{W}$; 用 $\mathbf{W}_{i,:}$ 表示矩阵 $\mathbf{W}$ 的第 i 行, $\mathbf{W}_{:,j}$ 表示矩阵的第 j 列, $\mathbf{W}_{i,j}$ 表示矩阵第 i 行的第 j 个元素; 加粗斜体表示向量, 如 $\boldsymbol{x}$. 用下标的形式表示向量中的某个元素, 如 $\boldsymbol{x}_j$ 表示向量 $\boldsymbol{x}$ 中的第 j 个元素; 用 $[L]$ 来表示集合 $\{1,2,\ldots,L\}$.

2.1 前馈式神经网络

一个输入为 $\boldsymbol{x}\in\mathbb{R}^{n_0}$ 的前馈式神经网络 $f:\mathbb{R}^{n_0}\to\mathbb{R}^{n_L}$ 由 L 层组成, 每一层 $i\in[L]$ 中有 n_i 个神经元, 且有对应的权重矩阵 $\mathbf{W}^{(i)}\in\mathbb{R}^{n_i\times n_{i-1}}$ 和偏置向量 $\boldsymbol{b}^{(i)}\in\mathbb{R}^{n_i}$. 其中输入层可以视为第 0 层, 最后一层被称为输出层, 第 1 层到第 $L-1$ 层被称为隐藏层. 我们用 $\text{node}_j^{(i)}$ 表示第 i 层的第 j 个神经元.

对于第 $i\in[L]$ 层神经元, 定义函数 $\boldsymbol{z}^{(i)}:\mathbb{R}^{n_0}\to\mathbb{R}^{n_i}$ 与函数 $\hat{\boldsymbol{z}}^{(i)}:\mathbb{R}^{n_0}\to\mathbb{R}^{n_i}$, 根据第 i 层与第 $i-1$ 层的仿射关系有 $\boldsymbol{z}^{(i)}(\boldsymbol{x})=\mathbf{W}^{(i)}\hat{\boldsymbol{z}}^{(i)}(\boldsymbol{x})+\boldsymbol{b}_i$, $\hat{\boldsymbol{z}}^{(i)}(\boldsymbol{x})=\sigma(\boldsymbol{z}^{(i)}(\boldsymbol{x}))$ 并且 $f(\boldsymbol{x})=\boldsymbol{z}^{(L)}(\boldsymbol{x})$, $\hat{\boldsymbol{z}}^{(0)}(\boldsymbol{x})=\boldsymbol{x}$. 其中 σ 为激活函数, 在本文中主要关注激活函数为 ReLU 的情形. 当不引起歧义时, 我们使用 $\boldsymbol{z}_j^{(i)}$, $\hat{\boldsymbol{z}}_j^{(i)}$ 分别表示在输入为 $\boldsymbol{x}$ 时, 第 i 层第 j 个神经元激活前与激活后的值.

在神经网络验证时, 往往维护每个神经元添加松弛后的数值上下界与符号上下界, 这里我们用 $\boldsymbol{u}_j^{(i)}$, $\boldsymbol{l}_j^{(i)}$ 与 $\hat{\boldsymbol{u}}_j^{(i)}$, $\hat{\boldsymbol{l}}_j^{(i)}$ 分别表示第 i 层的第 j 个神经元在给定输入约束下激活前与激活后的数值上下界. 用 $\bar{f}_j^{(i)}(\boldsymbol{z}^{(k)})$, $\underline{f}_j^{(i)}(\boldsymbol{z}^{(k)})$ 与 $\bar{\hat{f}}_j^{(i)}(\boldsymbol{z}^{(k)})$, $\underline{\hat{f}}_j^{(i)}(\boldsymbol{z}^{(k)})$ 分别表示激活前与激活后的符号上下界关于 $\boldsymbol{z}^{(k)}$ 的表达式.

2.2 神经网络验证问题

首先, 我们给出神经网络验证问题较为通用的定义.

定义 1 (神经网络验证问题). 神经网络验证问题是对给定元组 $\langle f,C,\mathcal{P}\rangle$, 验证 $\forall\boldsymbol{x}\in C.f(\boldsymbol{x})\in\mathcal{P}$ 是否成立, 其中,

- f 为给定的前馈神经网络.
- 输入约束 $C=\left\{\boldsymbol{x}\mid\wedge_{i\in[n_0]}\boldsymbol{l}_i^{(0)}\leqslant\boldsymbol{x}_i\leqslant\boldsymbol{u}_i^{(0)}\right\}$, 其中 $\boldsymbol{l}^{(0)},\boldsymbol{u}^{(0)}\in\mathbb{R}^{n_0}$ 为输入的上下界约束.
- 待验证的性质 $\mathcal{P}=\left\{f(\boldsymbol{x})\mid\wedge_i\vee_j((\boldsymbol{a}^{i,j})^{\mathrm{T}}f(\boldsymbol{x})+c^{i,j}<0)\right\}$, 其中 $i,j\in\mathbb{Z}^+$ 为正整数, 向量 $\boldsymbol{a}^{i,j}\in\mathbb{R}^{n_L}$, $c^{i,j}\in\mathbb{R}$ 为常数.

如果 $\forall\boldsymbol{x}\in C.f(\boldsymbol{x})\in\mathcal{P}$ 成立, 我们称神经网络 f 在给定输入约束 C 下满足性质 $\mathcal{P}$. 否则, $\exists\boldsymbol{x}^*\in C.f(\boldsymbol{x}^*)\notin\mathcal{P}$ 我们称 $\boldsymbol{x}^*$ 为神经网络 f 在输入约束 C 下关于性质 $\mathcal{P}$ 的反例.

最广为人知的神经网络验证问题当属神经网络的局部鲁棒性验证. 在分类问题中, 神经网络 f 会将输入 $\boldsymbol{x}$ 分类至标签 $C_f(\boldsymbol{x})=\arg\max_{i\in[n_L]}f(\boldsymbol{x})_i$. 给定输入 $\boldsymbol{x}^{(0)}$, 神经网络 f 在 $\boldsymbol{x}^{(0)}$ 附近半径为 ε 的 l_p 范数球 $\mathcal{B}_p(\boldsymbol{x}^{(0)},\varepsilon)$ 内是局部鲁棒的, 当且仅当 $\forall\boldsymbol{x}\in\mathcal{B}_p(\boldsymbol{x}^{(0)},\varepsilon)$, $C_f(\boldsymbol{x})=C_f(\boldsymbol{x}^{(0)})$. 在可靠且完备的验证中, 我们主要关注 l_∞ 范数下的局部鲁棒性. 该验证问题可以表示为 $\langle f,C,\mathcal{P}\rangle$, 其中 $C=\mathcal{B}_\infty(\boldsymbol{x}^{(0)},\varepsilon)=\left\{\boldsymbol{x}\mid\wedge_{i\in[n_0]}\boldsymbol{x}_i^{(0)}-\varepsilon\leqslant\boldsymbol{x}_i\leqslant\boldsymbol{x}_i^{(0)}+\varepsilon\right\}$ 并且 $\mathcal{P}=\{f(\boldsymbol{x})\mid\wedge_{i\in[n_L],i\neq C_f(\boldsymbol{x}^{(0)})}-f(\boldsymbol{x})_{C_f(\boldsymbol{x}^{(0)})}+f(\boldsymbol{x})_i<0\}$.

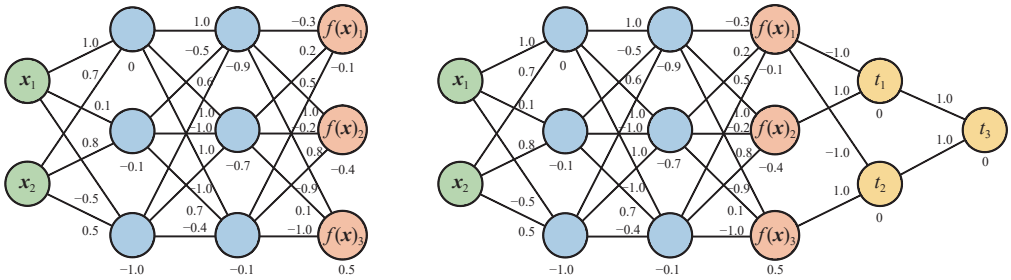
对于具有多个合取约束的性质 $\mathcal{P}$, 我们可以将复杂的性质 $\mathcal{P}$ 进行拆分, 即 $\mathcal{P}=\bigcap_i\mathcal{P}_i$. 此时, 原验证问题 $\langle f,C,\mathcal{P}\rangle$ 中性质 $\mathcal{P}$ 成立, 当且仅当每个子验证问题 $\langle f,C,\mathcal{P}_i\rangle$ 中性质 $\mathcal{P}_i$ 成立.

在处理性质时, 我们可以将性质约束编码到网络中. 如果我们要验证图 1(a) 所示的网络是否具有性质 $\mathcal{P}=\{\boldsymbol{x}\mid f(\boldsymbol{x})_1-f(\boldsymbol{x})_2<0\vee f(\boldsymbol{x})_1-f(\boldsymbol{x})_3<0\}$, 我们可以使用下面的方式进行编码. 如图 1(b) 所示, 我们构造 3 个神经元, 其中 $t_1=\text{ReLU}(f(\boldsymbol{x})_2-f(\boldsymbol{x})_1)$, $t_2=\text{ReLU}(f(\boldsymbol{x})_3-f(\boldsymbol{x})_1)$ 并且 $t_3=\text{ReLU}(t_1+t_2)$. 当 $f(\boldsymbol{x})_1-f(\boldsymbol{x})_2<0$ 或 $f(\boldsymbol{x})_1-f(\boldsymbol{x})_3<0$ 成立时, $t_3>0$; 否则 $t_3\leqslant 0$. 我们将编码了性质 $\mathcal{P}$ 的神经网络记为 $f_{\mathcal{P}}$, 那么 $\langle f,C,\mathcal{P}\rangle$ 成立当且仅当 $\forall\boldsymbol{x}\in C.f_{\mathcal{P}}(\boldsymbol{x})>0$. 通过这种方式神经网络验证问题则可以转换为证明在给定输入约束下神经网络输出是否始终大于 0.

由于绝大部分性质都可以使用类似的方式进行编码成单输出的神经网络形式[51], 所以在后文中除非特殊说

明, 我们均采用以下假设.

假设 1 (网络结构). 我们讨论的神经网络 f 的输出层只具有一个输出, 即 $f:\mathbb{R}^{n_0}\to\mathbb{R}$.



(a) 神经网络 f (b) 神经网络 f_p

图 1 将验证问题编码至神经网络

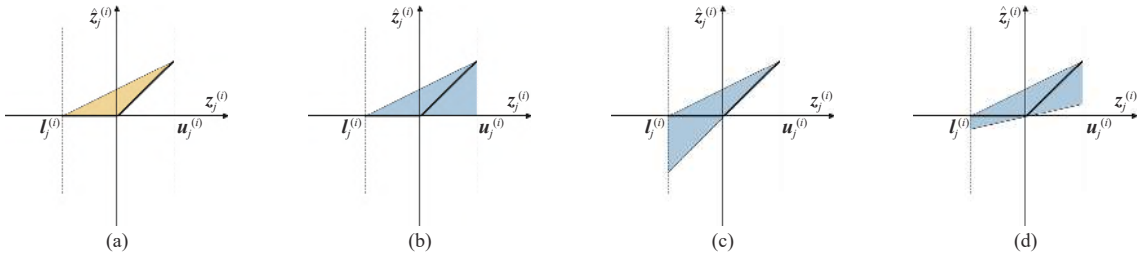
3 神经网络验证通用技术

在本节中, 我们对神经网络验证领域中的通用技术: 激活函数抽象, 拉格朗日乘子法, 符号区间反向传播方法, 约束求解方法以及分支限界方法进行简要的介绍. 最后, 我们给出神经网络验证的通用框架以帮助读者了解完备神经网络验证的流程, 方便之后对于加速技术的探讨.

3.1 激活函数抽象

对于 ReLU 节点 $\text{node}_j^{(i)}$, 若算得其输入的上下界 $\boldsymbol{l}_j^{(i)}<0<\boldsymbol{u}_j^{(i)}$ 则无法判断节点的激活状态, 此时需要对该节点进行抽象, 否则无法使用线性规划方法进行求解. 对于 ReLU 函数而言, 常用的抽象方法主要有使用三角形松弛和使用 Zonotope[52]松弛两大类.

图 2 展示了 4 种三角形松弛. 通常认为较小的面积通常意味着更少的误差, 所以较理想的三角形松弛如图 2(a) 所示. 但这种松弛方式使用两个线性约束来约束下界 (即下界仍然是分段函数), 在一些基于反向传播的算法 (如 CROWN, DeepPoly 等) 中会导致约束数量指数级上升, 所以实际应用中, CROWN 与 DeepPoly 主要使用图 2(b)、图 2(c) 的松弛方式来避免约束数量的指数增长, 并且选择使用二者中面积较小的松弛. 事实上, 较小的面积并不总能获得最紧的边界 [53], 但选择小面积的启发式由于容易实现且效果相对较好而被广泛接受.



(a) (b) (c) (d)

图 2 三角形松弛

Zonotope 可以高效地进行仿射变换, 所以在神经网络验证领域中 Zonotope 被广泛应用于描述神经网络输入输出的空间. 图 3 展示了 3 种 Zonotope 选择. 与三角形松弛的选择类似, 已有研究[54]表明很难有在所有情况下都是最优的 Zonotope 抽象, 因为每种 Zonotope 抽象都会排除一些其他抽象不会排除的输入范围.

3.2 拉格朗日乘子法

许多工具使用拉格朗日乘子[55]技术将约束条件转换为最优化问题的一部分. 假设最优化问题 $\min_x f(\boldsymbol{x})$ s.t. $g(\boldsymbol{x})\leqslant\boldsymbol{0}$ 有 n 组约束, 那么 $g(\boldsymbol{x})\in\mathbb{R}^n$, 并且第 j 个约束被满足当且仅当 $g(\boldsymbol{x})_j\leqslant 0$, 其中 $\boldsymbol{0}$ 表示全 0 向量. 我们可以通过引入

非负的拉格朗日乘子向量 $\boldsymbol{\lambda}\in\mathbb{R}^n$ 将约束条件加入目标函数中, 从而得到下面形式的最优化问题.

$$\min_{\boldsymbol{x}}\max_{\boldsymbol{\lambda}\geqslant 0} f(\boldsymbol{x})+\boldsymbol{\lambda}^{\mathrm{T}}g(\boldsymbol{x})=\min_{\boldsymbol{x}}\max_{\boldsymbol{\lambda}\geqslant 0} f(\boldsymbol{x})+\boldsymbol{\lambda}_1 g(\boldsymbol{x})_1+\boldsymbol{\lambda}_2 g(\boldsymbol{x})_2+\ldots+\boldsymbol{\lambda}_n g(\boldsymbol{x})_n \tag{1}$$

当 $g(\boldsymbol{x})$ 中第 j 个约束被满足时, 不等式 $g(\boldsymbol{x})_j\leqslant 0$ 成立, 只需要令 $\boldsymbol{\lambda}_j=0$ 就可以使得内部的最优化目标 $\max_{\boldsymbol{\lambda}\geqslant 0} f(\boldsymbol{x})+\boldsymbol{\lambda}^{\mathrm{T}}g(\boldsymbol{x})$ 取到最大值. 而如果当前的 $\boldsymbol{x}$ 无法使得 $g(\boldsymbol{x})$ 中第 j 个约束被满足, 即 $g(\boldsymbol{x})_j>0$ 时, 内部最大化的目标 $\max_{\boldsymbol{\lambda}\geqslant 0} f(\boldsymbol{x})+\boldsymbol{\lambda}^{\mathrm{T}}g(\boldsymbol{x})$ 会使得 $\boldsymbol{\lambda}_j$ 无限增大. 由于最终的优化目标是让函数 $f(\boldsymbol{x})+\boldsymbol{\lambda}^{\mathrm{T}}g(\boldsymbol{x})$ 的最大值最小, 所以需要对 $\boldsymbol{x}$ 进行优化, 直到 $g(\boldsymbol{x})_j=0$ 成立以阻止 $\boldsymbol{\lambda}_j$ 的无限增大. 在问题有最优解时, 如果优化结束后向量 $\boldsymbol{\lambda}$ 的第 j 项 $\boldsymbol{\lambda}_j>0$, 那么在最优化过程中, 由于约束 $g(\boldsymbol{x})_j$ 的存在, 引发了最优化算法对 $\boldsymbol{x}$ 与 $\boldsymbol{\lambda}$ 一系列的优化步骤. 我们称这种 $\boldsymbol{\lambda}_j>0$ 的乘子向量所对应的约束 $g(\boldsymbol{x})_j$ 为激活约束.

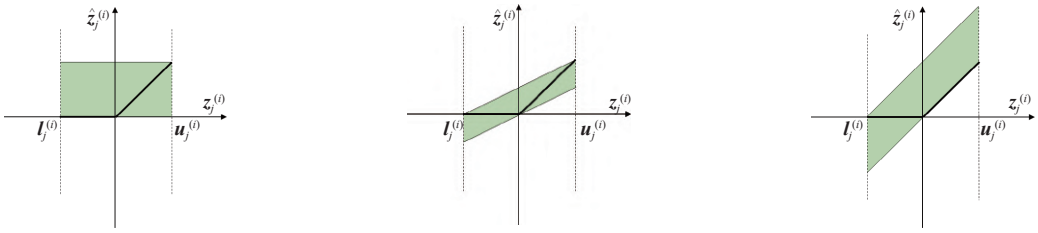


图 3 Zonotope 松弛

拉格朗日乘子法不仅可以用来求解最优化问题, 还可以判断约束是否满足, 因此很多神经网络验证工具都将拉格朗日乘子法作为算法核心.

3.3 边界计算

最基础的边界计算可以使用区间计算方法. 区间计算非常简便, 例如 $a\in[-1,1]$, $b\in[-2,2]$ 则 $a+b\in[-3,3]$, 即直接将 a 和 b 的上下界分别相加. 然而由于神经网络具有大量的节点并且具有高度非线性的特征, 使用区间计算所得到的边界误差往往较大. 为了减小误差, 研究者提出了符号传播[17,22,56]方法. 目前, 最广为使用的符号传播方法是 CROWN 与 DeepPoly 算法中提出的符号区间反向传播方法.

符号区间反向传播方法属于不完备的验证方式, 但却是多种完备验证方式的基础. 符号区间反向传播方法对于每个神经元都维护一对具体上下界和一对符号上下界. 其中符号上下界是该层节点相对于上一层节点的线性表达式. 当符号上下界具体化以得到具体上下界时, 首先将符号上下界利用之前层节点的符号上下界进行迭代, 直到得到当前层节点的符号上下界关于输入的线性组合. 为了区分激活前后的节点, 文中使用 $\overline{f}_j^{(i)}(\boldsymbol{z}^{(k)}),\underline{f}_j^{(i)}(\boldsymbol{z}^{(k)})$ 与 $\overline{\hat{f}}_j^{(i)}(\boldsymbol{z}^{(k)})$, $\underline{\hat{f}}_j^{(i)}(\boldsymbol{z}^{(k)})$ 分别表示第 i 层第 j 个神经元激活前与激活后的符号上下界关于 $\boldsymbol{z}^{(k)}$ 的表达式.

对于 $\hat{\boldsymbol{z}}^{(i)}=\mathrm{ReLU}(\boldsymbol{z}^{(i)})$, 我们添加三角形松弛来维护 $\hat{\boldsymbol{z}}^{(i)}$ 的符号上下界, 即 $\underline{\hat{f}}(\boldsymbol{z}^{(i)})\leqslant\hat{\boldsymbol{z}}_j^{(i)}\leqslant\overline{\hat{f}}(\boldsymbol{z}^{(i)})$, 其中,

$$\underline{\hat{f}}_j^{(i)}(\hat{\boldsymbol{z}}^{(i)})=\underline{\boldsymbol{\alpha}}_j^{(i)}\boldsymbol{z}_j^{(i)}+\underline{c}_j^{(i)},\overline{\hat{f}}_j^{(i)}(\hat{\boldsymbol{z}}^{(i)})=\overline{\boldsymbol{\alpha}}_j^{(i)}\boldsymbol{z}_j^{(i)}+\overline{c}_j^{(i)} \tag{2}$$

并且:

$$\begin{cases}\underline{\boldsymbol{\alpha}}_j^{(i)}=\overline{\boldsymbol{\alpha}}_j^{(i)}=0,\ \underline{\boldsymbol{c}}_j^{(i)}=\overline{\boldsymbol{c}}_j^{(i)}=0, & \text{if } \boldsymbol{u}_j^{(i)}\leqslant 0\\ \underline{\boldsymbol{\alpha}}_j^{(i)}=\overline{\boldsymbol{\alpha}}_j^{(i)}=1,\ \underline{\boldsymbol{c}}_j^{(i)}=\overline{\boldsymbol{c}}_j^{(i)}=0, & \text{if } \boldsymbol{l}_j^{(i)}\geqslant 0\\ \underline{\boldsymbol{\alpha}}_j^{(i)}=a\in[0,1],\ \underline{\boldsymbol{c}}_j^{(i)}=0;\ \overline{\boldsymbol{\alpha}}_j^{(i)}=\dfrac{\boldsymbol{u}_j^{(i)}}{\boldsymbol{u}_j^{(i)}-\boldsymbol{l}_j^{(i)}},\ \overline{\boldsymbol{c}}_j^{(i)}=-\dfrac{\boldsymbol{u}_j^{(i)}}{\boldsymbol{u}_j^{(i)}-\boldsymbol{l}_j^{(i)}}\boldsymbol{l}_j^{(i)}, & \text{if } \boldsymbol{l}_j^{(i)}<0<\boldsymbol{u}_j^{(i)}\end{cases} \tag{3}$$

当 $a=0$ 时, 表示的是图 2(b) 所示的三角形松弛, 此时符号下界为 $\underline{\hat{f}}_j^{(i)}(\hat{\boldsymbol{z}}^{(i)})=0$; 同理, 当 $a=1$ 时, 表示的是图 2(c) 所示的三角形松弛, 此时符号下界为 $\underline{\hat{f}}_j^{(i)}(\hat{\boldsymbol{z}}^{(i)})=\boldsymbol{z}^{(i)}$.

对于具有 L 层的神经网络 f, 其输出层有 $\underline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)})\leqslant f(\boldsymbol{x})\leqslant\overline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)})$, 其中,

$$\overline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)}) = \underline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)}) = \mathbf{W}^{(L)}\hat{\boldsymbol{z}}^{(L-1)} + \boldsymbol{b}^{(L)} \tag{4}$$

$\overline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)})$, $\underline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)})$ 为输出 $f(\boldsymbol{x})$ 的符号上下界关于第 $L-1$ 层节点激活后的值 $\hat{\boldsymbol{z}}^{(L-1)}$ 的表达式. 假设前 $L-1$ 层节点的上下界已计算结束, 为了获取的第 L 层节点关于 $\boldsymbol{z}^{(L-1)}$ 的符号上下界 $\underline{f}^{(L)}(\boldsymbol{z}^{(L-1)})$, $\overline{f}^{(L)}(\boldsymbol{z}^{(L-1)})$, 我们采用方程 (3) 中描述的方式对第 $\hat{\boldsymbol{z}}^{(L-1)}$ 进行松弛, 随后带入方程 (4) 中, 即可得到输出的符号上下界关于第 $L-1$ 层节点激活前的值 $\boldsymbol{z}^{(L-1)}$ 的表达式 $\underline{f}^{(L)}(\boldsymbol{z}^{(L-1)})$, $\overline{f}^{(L)}(\boldsymbol{z}^{(L-1)})$:

$$\begin{cases} \underline{f}^{(L)}(\boldsymbol{z}^{(L-1)}) = \mathbf{W}^{(L)}(\underline{\mathbf{D}}^{(L-1)}\boldsymbol{z}^{(L-1)} + \underline{\boldsymbol{b}}^{(L-1)}) + \boldsymbol{b}^{(L)} \leqslant \underline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)}) \\ \overline{f}^{(L)}(\hat{\boldsymbol{z}}^{(L-1)}) \leqslant \mathbf{W}^{(L)}(\overline{\mathbf{D}}^{(L-1)}\boldsymbol{z}^{(L-1)} + \overline{\boldsymbol{b}}^{(L-1)}) + \boldsymbol{b}^{(L)} = \overline{f}^{(L)}(\boldsymbol{z}^{(L-1)}) \end{cases} \tag{5}$$

$\underline{\mathbf{D}}^{(L-1)}$ 与 $\overline{\mathbf{D}}^{(L-1)}$ 是对角矩阵, 如果 $\mathbf{W}^{(L)}_{1,j} \geqslant 0$, 则 $\underline{\mathbf{D}}^{(L-1)}_{j,j} = \underline{\alpha}^{(L-1)}_j$, $\overline{\mathbf{D}}^{(L-1)}_{j,j} = \overline{\alpha}^{(L-1)}_j$, $\underline{\boldsymbol{b}}^{(L-1)}_j = \underline{\boldsymbol{c}}^{(L-1)}_j$, $\overline{\boldsymbol{b}}^{(L-1)}_j = \overline{\boldsymbol{c}}^{(L-1)}_j$; 否则 $\underline{\mathbf{D}}^{(L-1)}_{j,j} = \overline{\alpha}^{(L-1)}_j$, $\overline{\mathbf{D}}^{(L-1)}_{j,j} = \underline{\alpha}^{(L-1)}_j$, $\underline{\boldsymbol{b}}^{(L-1)}_j = \overline{\boldsymbol{c}}^{(L-1)}_j$, $\overline{\boldsymbol{b}}^{(L-1)}_j = \underline{\boldsymbol{c}}^{(L-1)}_j$. 注意根据假设 1, 这里 $\mathbf{W}^{(L)} \in \mathbb{R}^{1\times n_{L-1}}$, 即 $\mathbf{W}^{(L)}$ 是一个 1 行 n_{L-1} 列的矩阵, 所以只需要根据第 j 列的值就可以确定 $\underline{\mathbf{D}}^{(L-1)}$ 与 $\overline{\mathbf{D}}^{(L-1)}$; 同理, $\underline{\mathbf{D}}^{(L-2)}$ 与 $\overline{\mathbf{D}}^{(L-2)}$ 的值由分别由其前面所有矩阵相乘得到的结果中第 j 列的系数确定.

按照上述过程不断迭代, 最终可以得到输出 $f(\boldsymbol{x})_j$ 的符号上下界关于输入的表达式:

$$\underline{f}^{(L)}(\boldsymbol{x}) \leqslant f(\boldsymbol{x}) \leqslant \overline{f}^{(L)}(\boldsymbol{x}) \tag{6}$$

且输出层的符号上下界具有如下形式:

$$\begin{cases} \underline{f}^{(L)}(\boldsymbol{x}) = \mathbf{W}^{(L)}\underline{\mathbf{D}}^{(L-1)}\mathbf{W}^{(L-1)}\underline{\mathbf{D}}^{(L-2)}\cdots\underline{\mathbf{D}}^{(1)}\mathbf{W}^{(1)}\boldsymbol{x} + \underline{\boldsymbol{b}} \\ \overline{f}^{(L)}(\boldsymbol{x}) = \mathbf{W}^{(L)}\overline{\mathbf{D}}^{(L-1)}\mathbf{W}^{(L-1)}\overline{\mathbf{D}}^{(L-2)}\cdots\overline{\mathbf{D}}^{(1)}\mathbf{W}^{(1)}\boldsymbol{x} + \overline{\boldsymbol{b}} \end{cases} \tag{7}$$

方程 (7) 中的 $\bar{b}$ 与 $\underline{b}$ 为过程中产生的所有常数项的和. 则神经网络输出的数值上下界为:

$$\boldsymbol{l}^{(L)} = \min_{\boldsymbol{x}\in C} \underline{f}^{(L)}(\boldsymbol{x}),\ u^{(L)} = \min_{\boldsymbol{x}\in C} \overline{f}^{(L)}(\boldsymbol{x}) \tag{8}$$

在实际计算时, 首先需要将第 1 层的每个神经元当作输出, 获得其符号上下界, 随后将符号上下界具体化, 得到 $\boldsymbol{l}^{(1)}$ 以及 $\boldsymbol{u}^{(1)}$. 再将第 2 层的每个神经元当作输出, 执行上述符号区间反向传播的过程, 获得第 2 层每个神经元的具体上下界 $\boldsymbol{l}^{(2)}$, $\boldsymbol{u}^{(2)}$. 重复上述过程, 最后得到最后一层的具体上下界 $\boldsymbol{l}^{(L)}$, $\boldsymbol{u}^{(L)}$.

我们通过计算图 4 中 $\text{node}_1^{(2)}$ (即第 2 层第 1 个节点) 的上下界为例来说明符号区间反向传播方式计算边界的过程. 首先, 计算第 1 层节点的边界. 根据仿射关系很容易得出 $\text{node}_1^{(1)}$ 激活前的值与输入层节点的约束:

$$\boldsymbol{x}_1 + 0.7\boldsymbol{x}_2 \leqslant \boldsymbol{z}_1^{(1)} \leqslant \boldsymbol{x}_1 + 0.7\boldsymbol{x}_2 \tag{9}$$

其中, $\boldsymbol{x}_1 \in [-1,1], \boldsymbol{x}_2 \in [-1,1]$. 通过区间运算, 可以得到 $\boldsymbol{z}_1^{(1)} \in [-1.7, 1.7]$. 重复这个步骤, 我们可以获得第 1 层所有节点的边界信息, 可以发现 $\text{node}_3^{(1)}$ 的上界 $\boldsymbol{u}_3^{(1)} \leqslant 0$, 这意味着所以无论输入在给定范围内如何变化, 该节点一定非激活, 所以在随后的边界计算中, 我们可以忽略该节点.

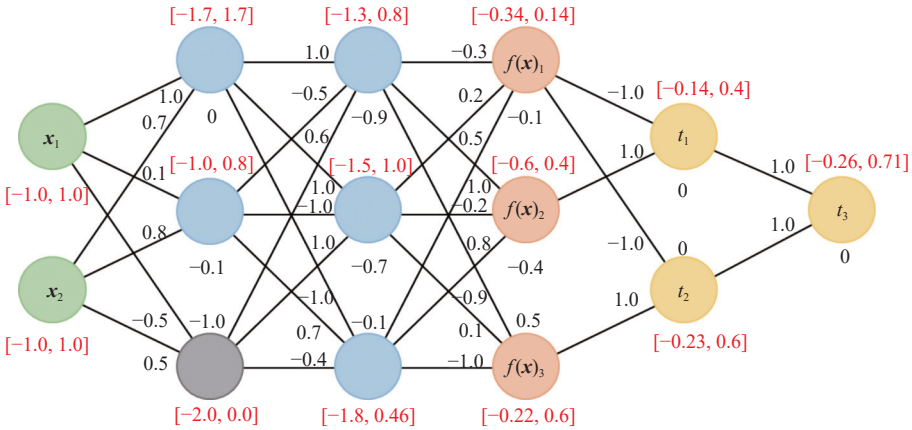


图 4 神经网络 $f_{\mathcal{P}}$ 的边界信息

接下来考虑 $\text{node}_1^{(2)}$, 根据 $\text{node}_1^{(2)}$ 激活前的值与第 1 层节点的仿射关系有:

$$\hat{z}_1^{(1)} - 0.5\hat{z}_2^{(1)} - 0.9 \leqslant z_1^{(2)} \leqslant \hat{z}_1^{(1)} - 0.5\hat{z}_2^{(1)} - 0.9 \tag{10}$$

随后处理激活函数 $\hat{z}_1^{(1)} = \text{ReLU}(z_1^{(1)})$, 此时使用图 2(b) 所示的三角形松弛, 即:

$$0 \leqslant \hat{z}_1^{(1)} \leqslant 0.5z_1^{(1)} + 0.85 \tag{11}$$

同理, 对于节点 $\text{node}_2^{(1)}$ 采用同样的松弛, 即:

$$0 \leqslant \hat{z}_2^{(1)} \leqslant 0.44z_2^{(1)} + 0.44 \tag{12}$$

接下来利用符号区间计算, 将方程 (11) 与方程 (12) 带入方程 (10) 中, 可以得到:

$$-0.22z_2^{(1)} - 1.12 \leqslant z_1^{(2)} \leqslant 0.5z_1^{(1)} - 0.05 \tag{13}$$

最后, 带入 $z_1^{(1)}$, $z_2^{(1)}$ 与输入的关系, 可以得到:

$$-0.22(0.1\boldsymbol{x}_1 + 0.8\boldsymbol{x}_2) - 1.12 \leqslant z_1^{(2)} \leqslant 0.5(\boldsymbol{x}_1 + 0.7\boldsymbol{x}_2) - 0.05 \tag{14}$$

利用区间算数, 最终得到 $-1.296 \leqslant z_1^{(2)} \leqslant 0.8$.

重复上述过程, 最终可以获得所有节点的边界. 我们在图 4 中用红色区间的形式表示这种方式计算出的每个节点 (激活前) 的边界信息. 多数情况下使用三角形松弛进行符号传播, 但文献 [22] 证明了符号传播过程可以拓展到任意的松弛.

3.4 约束求解

在神经网络验证领域中, 通常使用线性规划或者混合整数线性规划来进行约束求解. 线性规划会对 ReLU 节点添加松弛, 通过符号边界传播过程缩紧上下界并通过分支限界过程逐步去除松弛, 最坏情况是枚举出所有 ReLU 节点的状态, 此时神经网络验证问题则可以等价地编码成线性规划问题. 混合整数线性规划则是对每个节点 $\text{node}_j^{(i)}$ 引入整数变量 $\boldsymbol{\delta}_j^{(i)}$ 用来编码该神经元激活 ($\boldsymbol{\delta}_j^{(i)} = 1$) 还是非激活 ($\boldsymbol{\delta}_j^{(i)} = 0$).

具体来讲, 使用线性规划求解神经网络验证问题时通常按照下面的方式进行编码. 首先, 最优化目标是让输出最小:

$$f^* = \min_{x,z,\hat{z},a} f(\boldsymbol{x}) \tag{15}$$

对于输入输出, 有如下的约束:

$$f(\boldsymbol{x}) = \boldsymbol{z}^{(L)};\ \hat{\boldsymbol{z}}^{(0)} = \boldsymbol{x}, \boldsymbol{x} \in C \tag{16}$$

对于第 L 层节点激活前的值与第 $L-1$ 层节点激活后的值, 有如下约束:

$$\boldsymbol{z}^{(i)} = \mathbf{W}^{(i)}\hat{\boldsymbol{z}}^{(i-1)} + \boldsymbol{b}^{(i)},\ i \in [L] \tag{17}$$

而对于隐藏层节点激活前后的值, 有如下约束:

$$\hat{\boldsymbol{z}}^{(i)} = \boldsymbol{z}^{(i)} + \boldsymbol{a}^{(i)},\ i \in [L-1];\ 0 \leqslant \hat{z}_j^{(i)};\ z_j^{(i)} \leqslant \hat{z}_j^{(i)} \leqslant \frac{\boldsymbol{u}_j^{(i)}}{\boldsymbol{u}_j^{(i)} - \boldsymbol{l}_j^{(i)}}(z_j^{(i)} - \boldsymbol{l}_j^{(i)}) \tag{18}$$

其中, $\boldsymbol{a}_j^{(i)} > 0$ 是松弛变量. 如果可以通过边界计算、分支限界等方法确定 $\text{node}_j^{(i)}$ 的激活状态, 那么当 $\text{node}_j^{(i)}$ 处于激活状态时, 需要添加约束:

$$\boldsymbol{a}_j^{(i)} = 0;\ z_j^{(i)} \geqslant 0 \tag{19}$$

而当 $\text{node}_j^{(i)}$ 处于非激活状态时, 需要添加约束:

$$\hat{z}_j^{(i)} = 0;\ z_j^{(i)} \leqslant 0 \tag{20}$$

使用混合整数求解器求解问题神经网络验证问题时, 通常按照下面的方式进行编码. 最优化目标同样是让输出最小:

$$f^* = \min_{x,z,\hat{z},\delta} f(\boldsymbol{x}) \tag{21}$$

对输入输出和相邻两层间节点间约束关系的编码分别与公式 (16) 和公式 (17) 相同. 对于第 L 层节点激活前

后的值, 有如下约束:

$$z_j^{(i)} \leqslant \hat{z}_j^{(i)},\ 0 \leqslant \hat{z}_j^{(i)},\ \hat{z}_j^{(i)} \leqslant \boldsymbol{u}_j^{(i)}\boldsymbol{\delta}_j^{(i)},\ \hat{z}_j^{(i)} \leqslant z_j^{(i)} - \boldsymbol{l}_j^{(i)}(1-\boldsymbol{\delta}_j^{(i)}),\ \boldsymbol{\delta}_j^{(i)} \in \{0,1\} \tag{22}$$

可以看到, 当节点处于激活状态时, $\boldsymbol{\delta}_j^{(i)} = 1$, 此时有 $0 \leqslant z_j^{(i)} = \hat{z}^{(i)} \leqslant \boldsymbol{u}_j^{(i)}$; 而当节点处于非激活状态时, $\boldsymbol{\delta}_j^{(i)} = 0$, 有 $z_j^{(i)} \leqslant 0 = \hat{z}_j^{(i)} \leqslant z_j^{(i)} - \boldsymbol{l}_j^{(i)}$.

通常在验证时, 验证工具会将原问题的反, 即 $\exists \boldsymbol{x}^* \in C.f(\boldsymbol{x}^*) \notin \mathcal{P}$ 编码为约束求解问题进行验证. 当这组约束可以满足时 (SAT), 说明找到了 $\mathcal{P}$ 的反例, 性质 $\mathcal{P}$ 不成立; 否则, 这组约束不可满足 (UNSAT), 即不存在反例, 性质 $\mathcal{P}$ 成立.

3.5 分支限界方法

图 5 展示了结合分支限界方法进行神经网络验证的大致过程. 首先, 使用非完备的方法 (CROWN, DeepPoly 等) 计算出网络中每个神经元的边界, 并将一些可以确定激活状态的节点从分支选择的候选中排除. 如果仅通过边界信息就可以判断出性质 $\mathcal{P}$ 成立与否, 则直接返回结果, 如果不能判断, 则进行约束求解. 若仍无法判断性质, 则进入分支限界过程. 分支限界的过程可以用树形结构进行表示, 我们称这种表示分支限界过程的树形结构为搜索树.

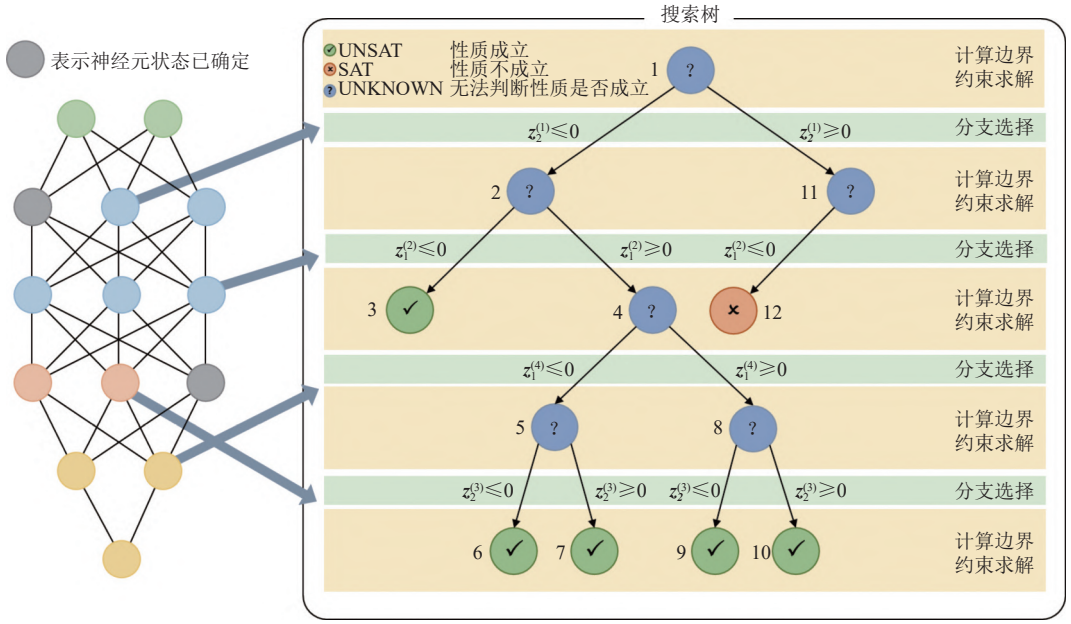


图 5 验证过程示例

图 5 中对原始网络进行边界计算与约束求解后仍然无法判断性质是否成立, 所以需要进行分支选择过程. 在分支选择时, 我们启发式的选择节点, 枚举其激活状态. 这里节点旁边的数字表示探索分支的顺序. 我们首先选择 $\text{node}_1^{(2)}$ 进行分支, 可以得到激活 ($z_2^{(1)} \leqslant 0$) 与非激活 ($z_2^{(1)} \geqslant 0$) 两个分支. 进入左边的分支后 (节点 2), 相比分支之前 (节点 1), 我们可以得到新的约束信息 ($z_2^{(1)} \leqslant 0$), 通过添加该约束, 可以消除一定的不确定性, 此时重新计算边界可能得到更紧的边界. 图 5 中仅添加约束 $z_2^{(1)} \leqslant 0$ 无法判断性质是否成立, 所以我们选择 $\text{node}_1^{(2)}$ 进行分支选择. 在探索 $\text{node}_1^{(2)}$ 的非激活分支时, 可以判断性质成立, 于是进行回溯, 探索其他分支.

通过不断执行分支限界过程, 我们判断出在 $\text{node}_2^{(1)}$ 的非激活分支中性质总是成立, 所以我们进入 $\text{node}_2^{(1)}$ 的激活分支进行探索. 在探索 $\text{node}_1^{(2)}$ 的非激活分支时发现反例, 所以可以直接判定性质不成立, 算法直接终止. 如果在整个搜索树中所有叶子节点中性质都成立, 那么我们可以判断神经网络 f 具有该性质 $\mathcal{P}$.

3.6 通用验证框架

我们在算法 1 中总结出使用边界计算、约束求解与分支限界方法进行神经网络验证的通用框架.

算法 1. 神经网络验证通用验证框架.

输入: 验证问题 $\langle f,C,\mathcal{P}\rangle$;
输出: 如果网络 f 在给定输入约束 C 下满足性质 $\mathcal{P}$ 返回 UNSAT; 否则, 返回 SAT.

```
1. function Verify( f,C,P )
2.    deque.put( f,C,P )
3.    while ¬ deque.empty() [parallel] do
4.       ⟨ f,C′,P ⟩ ← deque.get(direction)        ▷ direction ∈ {front, back, heuristic}
5.       bounds ← CalculateBounds( f,C′,P )
6.       if PropertyHolds( f, bounds, P ) = True then
7.          BackTrack()
8.          continue
9.       end if
10.      if ConstraintSolve( f, bounds, ¬P ) = UNSAT then
11.         BackTrack()
12.      else if ConstraintSolve( f, bounds, ¬P ) = SAT then
13.         return SAT
14.      else              ▷ ConstraintSolve( f, bounds, ¬P ) = UNKNOWN
15.         subproblems ← Branch( f, bounds, P )
16.         for ⟨ f, C″, P ⟩ in subproblems do
17.            deque.put( f, C″, P )
18.         end for
19.      end if
20.   end while
21.   return UNSAT
22. end function
```

与文献 [57] 中以全局上下界计算为核心的框架不同, 我们的框架将约束求解过程纳入其中, 从而能够概括更多的验证方法. 开始验证时, 我们将原始的验证问题入队. 随后, 执行循环直到队列中的元素为空. 在循环中, 取出队列中的验证问题并计算边界. 如果可以证明该问题性质成立则进行回溯并进入下一次循环. 如果无法证明, 则进行约束求解. 若约束求解成功证明性质成立, 则进行回溯并进入下一次循环, 如果证明出性质不成立则返回 SAT, 如果无法判断是否成立, 则进行分支, 并将子问题入队. 我们对其中较为关键的函数进行说明.

(1) *deque.get* 函数 (第 2 行). 在非并行时, 可以选择从队首或队尾获取待验证问题. 选择从队首获取待验证问题则是采用与图 5 相似的深度优先搜索过程; 而选择从队尾获取待验证的问题则是广度优先搜索的过程. 在并行的情况下 *deque* 被当作消息队列进行使用. 当然, 无论是并行还是非并行, 都可以使用启发式的方法, 优先对某些分支 (子问题) 进行求解.

(2) *CalculateBounds* 函数 (第 5 行). 计算边界可以使用 CROWN 等非完备的方法, 通常而言, 计算得到的边界越紧越有利于加速后续的约束求解步骤. 同时, 由于添加了约束, 在计算边界时可能会额外确定某些节点的状态, 导致可以得到更紧一些的边界. 所以, 一种常见的做法是, 如果有新的节点状态被确定, 则重新计算边界, 直到没有新的节点状态能够被确定.

(3) *PropertyHolds* 函数 (第 6 行). 如果利用边界信息可以直接判断性质成立, 那么该子问题验证结束, 直接进

行回溯.

(4) *BackTrack* 函数 (第 7, 11 行). 在队列中, 体现为去除队列的一些队列中的子问题. 在搜索树上, 则体现为回到搜索树某个更靠近根的节点, 并探索其他分支.

(5) *ConstraintSolve* 函数 (第 14 行). 利用约束求解器进行求解. 如果使用 MILP, 则通常返回结果只有 SAT 或者 UNSAT 两种情况; 如果利用带松弛的 LP, 则可能出现 UNKNWON 的结果.

(6) *Branch* 函数 (第 15 行). 启发式的选择某个 ReLU 节点并枚举其状态 (激活或非激活) 或者对输入域进行分支. 不同的分支顺序对验证速度的影响显著, 所以有很多工具都对分支方式进行了探索.

在本节中, 我们首先介绍了 ReLU 激活函数的多种抽象方法以及拉格朗日乘子法. 随后, 我们介绍了目前大多数工具中边界计算方法的基础符号区间传播方法. 接下来, 我们展示了如何使用线性规划以及混合整数线性规划对神经网络验证问题进行编码. 最后, 我们介绍了使用分支限界方法进行完备神经网络验证的过程, 并提出完备神经网络验证的通用框架. 我们指出完备神经网络验证框架中的核心部分: 边界计算、约束求解与分支选择, 并结合框架介绍了各个关键步骤的常见做法. 在第 4 节, 我们将对目前最先进的工具在这 3 方面提出的加速算法进行详细介绍.

4 神经网络验证加速技术

在本节中, 我们详细调研了目前最先进的完备验证工具针对边界计算、分支选择和约束求解这 3 个完备验证算法的核心步骤. 表 1 对我们将要介绍的工具及其所使用的加速方法进行了简单总结. 我们将选择其中较为关键或者新颖的加速方法进行详细介绍, 并给出便于理解的例子来说明加速方法的关键步骤.

表 1 工具及其优化技术

工具	边界计算	约束求解	分支选择	其他优化
α, β-CROWN	将三角形松弛下界的斜率替换为变量, 使用梯度方法进行优化 (α-CROWN[58]); 将额外约束利用拉格朗日乘子法或对偶方法加入边界的约束中, (GCP-CROWN[59], β-CROWN[60])	利用拉格朗日乘子法, 将约束是否可以被满足的判定加入到边界计算过程中, GCP-CROWN使用Cplex计算切割平面 (cutting plane)	利用GPU的并行能力, 同时计算 n 组验证子问题边界; 将FSB与BaBSR方法拓展到并行形式	实现投影梯度下降(projected gradient descent, PGD)攻击; 在GCP-CROWN中启动多个单独的线程利用MILP对原始问题进行求解, 并将产生的额外约束进行保存. 主线程中读取额外约束, 并将其添加到边界计算过程中以获得更紧的边界
MN-BaB[61]	提出PRIMA框架[27]用于计算多神经元约束. 并将多神经元约束、分支选择状态等利用拉格朗日乘子法添加到边界计算的约束中	利用拉格朗日乘子法判断约束是否能被满足; 使用Gurobi计算凸包, 产生多神经元约束	提出激活约束评分分支策略, 以及成本调整分支策略	实现PGD攻击
Verinet[62]	提出基于误差的符号区间传播方法, 每个神经元只维护下界的线性方程, 并计算所有节点都使用下界的线性方程而引入的误差	使用Xpress求解器求解验证问题	提出影响分数的概念, 用来评估节点分支后对结果的影响程度. 分支时, 选择影响分数最高的节点进行分支	实现PGD攻击
nnenum[54]	使用星集来表示每个神经元的输入输出的上近似	使用Gurobi求解器优化节点的上下界	—	提出单循环LP方法优化节点的上下界. 此外提出了多轮抽象与每轮内进行多重抽象两种方法确定对神经元抽象方式, 以及执行引导抽象方法来启发式地对神经元进行抽象

表 1 工具及其优化技术 (续)

工具	边界计算	约束求解	分支选择	其他优化
Planet[8]	对每个节点抽象后, 使用LP求解器求出每个节点的上下界	使用线性规划工具集GLPK求解线性规划问题, 使用Minisat求解SAT问题	将LP与SAT框架相结合. 提出了3种子句学习方式, 利用弹性滤波进行冲突子句学习, 保存可行节点状态组合以及节点状态推断	—
PEREGRiNN[63]	使用Dutta等人[64]工作中的边界计算方法	使用Gurobi求解器求解验证问题	使用松弛变量之和作为最优化目标, 并做了进一步修改, 使得分支选择时总倾向于选择更靠近输入且松弛最小的神经元	实现简单的攻击方法, 在输入约束范围内均匀采样, 判断采样点是否为反例
Marabou[25]	使用DeepPoly进行边界计算	使用单纯形法的拓展Reluplex, 将约束分为线性约束与非线性约束. 使用单纯形法处理线性约束, 当线性约束满足时再调整赋值以满足非线性约束	使用马尔科夫链蒙特卡洛搜索来找到使得不可满足和函数全局最小的节点状态组合	计算边界过程中如果可以额外推导出其他节点的状态, 则重新计算边界, 直到无法确定新的节点状态
Venus2[65]	使用符号区间分析[66]	使用Gurobi求解器求解验证问题	计算神经网络的层内依赖与层间依赖, 并根据依赖关系创建依赖图以减去更多分支	实现PGD攻击
LayerSAR[67]	使用符号区间传播技术	利用Lpsolve求解器求解规划问题	LayerSAR 优先选择最接近输入并且存在未确定节点的层, 随后, 按照节点的数值上下界的间隔选择节点进行分支	—
CEGAR方法[34]	—	使用Marabou作为求解基础	在神经元合并时选择输入边权重最小的节点合并	对节点进行分类, 根据类别合并节点进行抽象, 如果验证失败则进行精化, 即拆分合并的节点
DeepInc[43]	—	使用Marabou作为求解基础	—	保存第1次验证的搜索树以及叶子节点的验证格局, 增量验证时利用第一次的格局加速证明
IVAN[44]	—	使用Gurobi求解规划问题	—	重建更加紧凑的搜索树, 同时重新规划分支顺序并删除一些效果较差的分支决策, 然后按照全新的搜索树进行搜索

4.1 边界计算

边界计算是验证中关键的一环, 通常越紧的边界越有利于性质的证明. α,β-CROWN, MN-BaB 将节点约束编码到边界计算的最优化目标中, 以此获得满足条件且相对较紧的边界; MN-BaB 利用层内神经元之间的约束关系来得更精确的边界信息. 这两种边界计算方法都可以在 GPU 上并行执行; Verinet 提出了基于误差的区间传播方法; 工具 nnenum 使用星集表示网络输入输出的边界. 接下来, 我们将对上述边界计算方法展开详细的说明.

4.1.1 基于线性松弛的微扰分析

工具 α-CROWN[58]中提出了基于线性松弛的微扰分析 (linear relaxation based perturbation analysis, LiRPA) 方法. LiRPA 方法与 CROWN 基本一致, 不同的是在 $\boldsymbol{l}_j^{(i)} < 0 < \boldsymbol{u}_j^{(i)}$ 时, CROWN 使用图 2(b) 或图 2(c) 中的三角形约束, 而 LiRPA 则使用图 2(d) 所示的三角形约束, 即将下界设置为过原点且斜率为 $\underline{\boldsymbol{\alpha}}_j^{(i)}$ 的直线, 也就是将公式 (3) 中的 $\underline{\boldsymbol{\alpha}}_j^{(i)}$ 视为变量.

在假设 1 下, 当采用与 CROWN 相同的方式的反向传播时, 第 i 层每个不能确定状态的节点 j 都会有一个对应的 $\underline{\boldsymbol{\alpha}}_j^{(i)}$. 令 $\boldsymbol{\alpha}$ 表示所有不确定节点的斜率变量 $\underline{\boldsymbol{\alpha}}_j^{(i)}$, 那么对于输出的松弛上下界可以表示为关于 $\boldsymbol{\alpha}$ 的函数 $\underline{f}_{\boldsymbol{\alpha}}^{(L)}(\boldsymbol{x},\boldsymbol{\alpha})$, $\overline{f}_{\boldsymbol{\alpha}}^{(L)}(\boldsymbol{x},\boldsymbol{\alpha})$.

固定 $\boldsymbol{\alpha}$ 的值时得到的边界并不理想. 这里作者使用投影梯度下降方法来优化参数 $\boldsymbol{\alpha}$ 进而算出对于神经网络输出的更紧的上下界, 即:

$$\boldsymbol{l}^{(L)} = \min_{\boldsymbol{\alpha}} \min_{\boldsymbol{x}\in C} \underline{f}_{\boldsymbol{\alpha}}^{(L)}(\boldsymbol{x},\boldsymbol{\alpha}),\ \boldsymbol{u}^{(L)} = \max_{\boldsymbol{\alpha}} \max_{\boldsymbol{x}\in C} \overline{f}_{\boldsymbol{\alpha}}^{(L)}(\boldsymbol{x},\boldsymbol{\alpha}),\ \boldsymbol{\alpha}_j \in [0,1] \tag{23}$$

由于梯度相关的计算和优化可以在 GPU 上运行, 所以使用这种方式计算边界往往比 LP 求解器更快. 此外 LiRPA 方法可以同时优化隐藏层节点的边界, 所以得到的边界往往比使用 LP 求解器更紧.

由于 LiRPA 可能无法判断约束是否可以满足, 所以仍然需要调用 LP 求解器来确保验证的完备性. 作者在 LiRPA 无法进一步改善边界或者所有不确定状态的神经元都分支之后再调用 LP 求解器进行约束求解在保证验证的完备性的同时提高验证效率.

4.1.2 拉格朗日乘子法

CROWN 或 α-CROWN 虽然可以传播线性或者二次边界, 但是其传播方式无法有效利用分支限界方法中由于分支选择而产生的约束, 即使分支限界过程中产生了无法满足的约束, 传统的边界传播方式可能依然无法判断出来. 为了解决这个问题, β-CROWN [60]使用拉格朗日乘子法来引入对约束的编码.

在假设 1 下, 只要证明输出的下界大于 0, 即 $\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) > 0$ 那么就可以直接证明性质成立. 作者采用下面的方式来对神经网络输出的下界进行估计.

对于神经网络的最后一层, 根据仿射关系有:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) = \min_{\boldsymbol{x}\in C} \underbrace{\mathbf{W}^{(L)}}_{\mathbf{A}^{(L-1)}} \hat{\boldsymbol{z}}^{(L-1)} + \boldsymbol{b}^{(L)} \tag{24}$$

随后采用与方程 (3) 相同的方式对第 $\hat{\boldsymbol{z}}^{(L-1)}$ 进行松弛. 为了方便描述, 我们使用 C 来代表常数项:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) \geqslant \min_{\boldsymbol{x}\in C} \mathbf{W}^{(L)} \underline{\mathbf{D}}^{(L-1)} \boldsymbol{z}^{(L-1)} + C \tag{25}$$

为了编码第 i 层的约束, 首先定义对角矩阵 $\mathbf{S}^{(i)} \in \mathbb{R}^{d_i \times d_i}$, 若第 j 个神经元激活, 则矩阵对角线第 j 个元素为–1; 如果神经元非激活, 则矩阵对角线第 j 个元素为 1, 否则, 该对角线元素为 0:

$$\mathbf{S}_{j,j}^{(i)} = \begin{cases} -1, & \text{if } \boldsymbol{z}_j^{(i)} \geqslant 0 \\ +1, & \text{if } \boldsymbol{z}_j^{(i)} < 0 \\ 0, & \text{others} \end{cases} \tag{26}$$

随后使用 $\boldsymbol{\beta}^{(L-1)\mathrm{T}} \mathbf{S}^{(L-1)} \boldsymbol{z}^{(L-1)}$ 作为拉格朗日函数的乘子项, 这里 $\boldsymbol{\beta}^{(L-1)} \in \mathbb{R}^{n_{L-1}}$ 相当于方程 (1) 中的乘子向量 $\boldsymbol{\lambda}$, 并且其中每个元素均非负. 根据 $\mathbf{S}^{(L-1)}$ 的构造方式可知, 当添加节点激活约束, 即 $\boldsymbol{z}_j^{(i)} \geqslant 0$ 时, 乘子项中 $\boldsymbol{z}_j^{(i)}$ 前的系数为 $-\boldsymbol{\beta}_j^{(L-1)} \leqslant 0$; 反之, 当节点状态约束为 $\boldsymbol{z}_j^{(i)} < 0$ 时, 乘子项中 $\boldsymbol{z}_j^{(i)}$ 前的系数为 $\boldsymbol{\beta}_j^{(L-1)} \geqslant 0$.

在方程 (25) 中引入乘子项, 可以得到:

$$\begin{aligned} \min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) &\geqslant \min_{\boldsymbol{x}\in C} \max_{\boldsymbol{\beta}^{(L-1)} \geqslant \mathbf{0}} \mathbf{W}^{(L)} \underline{\mathbf{D}}^{(L-1)} \boldsymbol{z}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\mathrm{T}} \mathbf{S}^{(L-1)} \boldsymbol{z}^{(L-1)} + C \\ &\geqslant \max_{\boldsymbol{\beta}^{(L-1)} \geqslant \mathbf{0}} \min_{\boldsymbol{x}\in C} \left(\mathbf{W}^{(L)} \underline{\mathbf{D}}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\mathrm{T}} \mathbf{S}^{(L-1)} \right) \boldsymbol{z}^{(L-1)} + C \end{aligned} \tag{27}$$

其中, 第 1 个等式是根据拉格朗日函数的定义产生而来, 将节点状态约束转换为 $\boldsymbol{\beta}^{(L-1)\mathrm{T}} \mathbf{S}^{(L-1)} \boldsymbol{z}^{(L-1)}$. 第 2 个不等式则是由于该优化问题的弱对偶性得出, 执行这种转化可以让最里层满足拉格朗日乘子法的形式, 从而使得反向传播过程得以继续.

继续执行反向传播过程, 将 $\boldsymbol{z}^{(L-1)}$ 替换为上一层 $\hat{\boldsymbol{z}}^{(L-2)}$ 的线性组合:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) \geqslant \max_{\boldsymbol{\beta}^{(L-1)} \geqslant \mathbf{0}} \min_{\boldsymbol{x}\in C} \underbrace{\left(\mathbf{W}^{(L)} \underline{\mathbf{D}}^{(L-1)} + \boldsymbol{\beta}^{(L-1)\mathrm{T}} \mathbf{S}^{(L-1)} \right) \mathbf{W}^{(L-1)}}_{\mathbf{A}^{(L-2)}} \hat{\boldsymbol{z}}^{(L-2)} + C \tag{28}$$

为了表示方便, 定义矩阵 $\mathbf{A}^{(i)}$ 表示神经网络输出与 $\hat{\boldsymbol{z}}^i$ 的之间的关系. 通过不断的迭代, 最终, 我们有:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) \geqslant \max_{\beta\geqslant \mathbf{0}} \min_{\boldsymbol{x}\in C} \mathbf{A}^{(0)}\boldsymbol{x} + C \tag{29}$$

其中, $\boldsymbol{\beta} := [\boldsymbol{\beta}^{(1)\mathrm{T}}\boldsymbol{\beta}^{(2)\mathrm{T}}\ldots\boldsymbol{\beta}^{(L-1)\mathrm{T}}]^{\mathrm{T}}$ 即将所有 $\boldsymbol{\beta}^{(i)}$ 拼接起来.

举例来说, 图 4 中所示网络最后一层有如下关系:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) = \min_{\boldsymbol{x}\in C} \begin{bmatrix} 1 & 1 \end{bmatrix} \hat{\boldsymbol{z}}^{(4)} \geqslant \begin{bmatrix} 1 & 1 \end{bmatrix} \underline{\mathbf{D}}^{(4)} \boldsymbol{z}^{(4)} + C.$$

如果在分支限界过程中选择了 $\mathrm{node}_1^{(4)}$ 非激活分支, 则在反向传播过程中有:

$$\min_{\boldsymbol{x}\in C} f(\boldsymbol{x}) = \max_{\boldsymbol{\beta}^{(4)}\geqslant \mathbf{0}} \min_{\boldsymbol{x}\in C} \begin{bmatrix} 1 & 1 \end{bmatrix} \underline{\mathbf{D}}^{(4)} \boldsymbol{z}^{(4)} + \boldsymbol{\beta}^{(4)} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + C.$$

随后继续执行反向传播过程即可.

上面的最优化问题可以使用梯度方法来求解. 由于每个 $\boldsymbol{\beta}\geqslant\mathbf{0}$ 都会得到一个 $\min_{x\in C} f(\boldsymbol{x})$ 的有效的下界, 所以求解该最优化问题时是否收敛不会影响算法的可靠性. 当 $\boldsymbol{\beta}$ 固定时, 它具有与 CROWN 相同的渐近复杂度. 当 $\boldsymbol{\beta}=0$ 时, β-CROWN 产生与 CROWN 相同的结果; 然而, 由于添加了额外的约束, β-CROWN 可能会得到比 CROWN 更紧的下界.

在实现时, 中间隐藏层节点的上下界同样通过 β-CROWN 方法计算. 即将该节点作为输出节点, 执行上面的方法. 由于拉格朗日函数的引入, 当添加的节点约束无法被满足时, 最优化目标的值会变为正无穷, β-CROWN 可以通过该现象判断约束是否能被满足. 所以, β-CROWN 结合分支限界法后, 可以实现完备的验证工具, 并不需要引入额外的约束求解器.

4.1.3 切割平面约束

β-CROWN 方法添加的约束都局限于单层神经元. 而 GCP-CROWN 将可以添加的约束推广到了一般情形, 允许添加任意的切割平面, 即任何层神经元激活前后的值与编码神经元状态的整数变量共同组成的约束. 可以证明 β-CROWN 是 GCP-CROWN 的一个特例[59].

具体来说, 对于任意层 i 的预激活节点 $\boldsymbol{z}^{(i)}$, 激活节点 $\hat{\boldsymbol{z}}^{(i)}$, ReLU 状态编码变量 $\boldsymbol{\delta}^{(i)}$, 他们组成的切割平面为:

$$\sum_{i=1}^{(L-1)} \left(\boldsymbol{h}^{(i)\mathrm{T}} \boldsymbol{z}^{(i)} + \boldsymbol{g}^{(i)\mathrm{T}} \hat{\boldsymbol{z}}^{(i)} + \boldsymbol{q}^{(i)\mathrm{T}} \boldsymbol{\delta}^{(i)} \right) \leqslant \boldsymbol{d}_i \tag{30}$$

这里的 $\boldsymbol{h}^{(i)}$, $\boldsymbol{g}^{(i)}$ 和 $\boldsymbol{q}^{(i)}$ 为约束的系数. 一个有效切割平面不应从解空间中删除任何有效整数解. 由于添加了额外的约束, GCP-CROWN 方法得到的边界的至少不会比不添加约束差, 更有可能接近网络真实的下界. 作者由上面线性约束的对偶问题推导出加入切割平面后的目标方程 $g(\boldsymbol{\alpha},\boldsymbol{\beta})$. 与 β-CROWN 类似, 可以通过梯度方法来优化 $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ 来优化该问题的下界, 且任意一组 $0\leqslant\boldsymbol{\alpha}_j^{(i)}\leqslant 1$ 和 $\boldsymbol{\beta}\geqslant\mathbf{0}$ 都会得到一组有效的下界.

4.1.4 多神经元约束

大部分验证工具只考虑对单个神经元进行松弛或者编码, 但忽略了层内多个神经元之间的约束关系. MN-BaB[61] 对层内的多神经元约束进行了探索, 在边界计算时添加多神经元松弛来维护更紧的边界, 以此来减少分支限界过程中生成的子问题的数量. 为找出多神经元约束, MN-BaB 使用 PRIMA 框架[27]计算输入输出的凸多面体来对输入输出范围进行上近似. 给定一个 n_i 维凸多面体 $\mathcal{S}$ 作为第 i 层输入约束的上近似, PRIMA 首先进行分组, 即在第 i 层的 n_i 个神经元的幂集中选择多个大小为 k 的子集, 被选择的某个子集至少与一个其他被选择的子集存在交集. 在验证前馈式神经网络时, 通常会选择 $\binom{n_i}{k}$ 个子集. 随后将表示上一层输出的凸多面体 $\mathcal{S}$ 投影至第 j 个子集的输入空间上, 并计算其凸八面体上近似 $\mathcal{P}^j$, 再结合拆分限界升维方法 (split-bound-lift method, SBLM) 与部分双重描述法 (partial double description method, PDDM) 对于每个多面体 $\mathcal{P}^j$, 计算其输出的上近似 $\mathcal{K}^j$ 计算子集的输出多面体, 最后计算所有输出约束的交集 $\mathcal{K}=\bigcap_i \mathcal{K}^j$, 来维护该层输出的上近似多面体.

- 拆分限界升维方法. 我们以计算某一层的神经元子集 ($k=2$) 为例说明 SBLM 方法的过程. 如图 6 所示 (图中忽略了多面体 $\mathcal{P}^j$ 的上标), 假设输入多面体为 $\mathcal{P}^j$, SBLM 方法按照下面的方式进行.

(1) 节点分支. 首先, 将输入多面体 $\mathcal{P}^j$ 按照象限进行分支. 根据 $z_1 \leqslant 0$ 与 $z_1 \geqslant 0$ 可以分为 $\mathcal{P}_1$ 与 $\mathcal{P}_2$ (第 2 列). 在此基础上, 对 z_2 进行分支, 可以得到 4 个子多面体 (第 3 列).

(2) 升维. 将坐标系从 (z_1, z_2) 空间升维到 $(\hat{z}_2, z_1, z_2)$ 空间 (第 4 列), 注意这里先升 $\hat{z}_1$ 还是 $\hat{z}_2$ 是随机的.

(3) 限定边界. 升维后, 需要结合节点状态对新升的一维的边界进行限制. 对于 $\mathcal{P}_{2,1}$, 由于 $z_2 \geqslant 0$, 节点 2 处于激活状态, 所以 $\hat{z}_2 \leqslant z_2$ 并且 $\hat{z}_2 \geqslant z_2$, 得到 $\mathcal{K}'_{2,1}$. 对于 $\mathcal{P}_{2,1}$, 由于 $z_2 \leqslant 0$, 节点 2 处于非激活状态, 所以 $\hat{z}_2 \leqslant 0$ 并且 $\hat{z}_2 \geqslant 0$, 得到 $\mathcal{K}'_{2,1}$. 同理, 可以计算出其他几个边界 (第 5 列).

(4) 计算近似凸多面体. 随后使用 PDDM 算法分别计算 $\mathcal{K}'_{1,1}$ 和 $\mathcal{K}'_{1,1}$ 与 $\mathcal{K}'_{2,2}$ 和 $\mathcal{K}'_{2,2}$ 的凸多面体 $\mathcal{K}_1, \mathcal{K}_2$. 重复升维、限定边界与计算近似凸多面体过程, 直到处理完所有的维度.

图 6 中, 将 $\mathcal{K}_1, \mathcal{K}_2$ 从 $(\hat{z}_1, z_1, z_2)$ 空间升维到 $(\hat{z}_1, \hat{z}_2, z_1, z_2)$ 空间并限定边界后分别得到 $\mathcal{K}_{1'}, \mathcal{K}_{2'}$. 随后计算 $\mathcal{K}_{1'}, \mathcal{K}_{2'}$ 的凸多面体上近似得到最终结果. 由于四维平面无法作图表示, 我们分别展示了将凸多面体分别投影到 $(\hat{z}_1, z_1, z_2)$ 与 $(\hat{z}_2, z_1, z_2)$ 平面的结果.

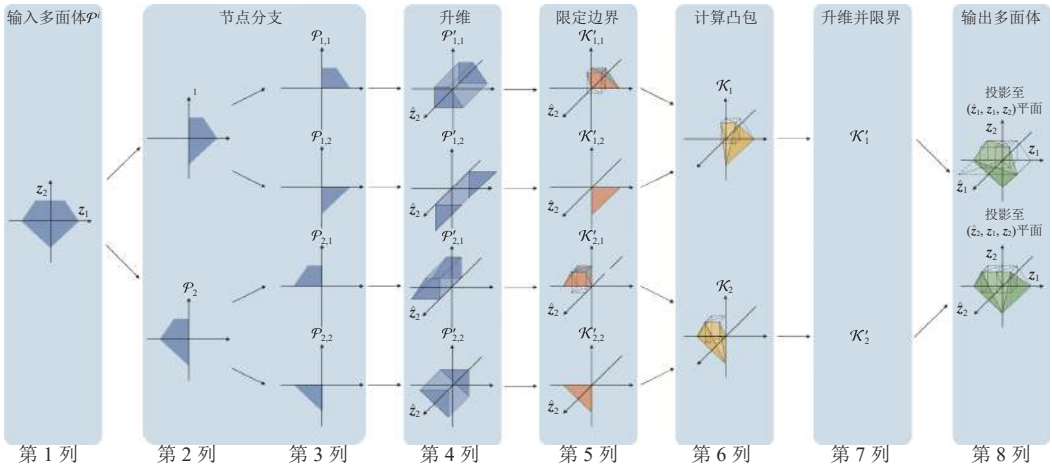


图 6　拆分限界升维方法[27]

● 部分双重描述法. 部分描述法用来计算近似计算两个凸包的交集, 图 6 中计算凸包的部分均通过部分双重描述法进行. 我们以图 7 为例, 来说明 PDDM 方法的流程.

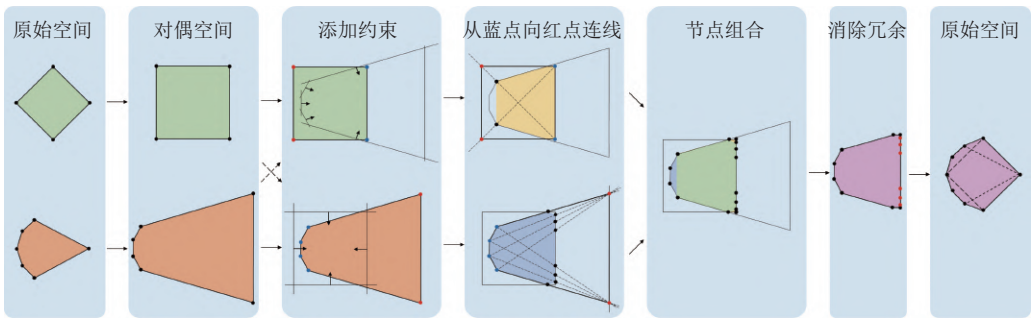


图 7　部分双重描述法[27]

(1) 空间转换. 将输入的多面体 (第 1 列) 从原始空间转换至对偶空间 (第 2 列).

(2) 添加约束. 在对偶空间中, 互相将自己所表示的约束添加得到另一个约束中 (第 3 列). 根据是否在对偶表示中存在交集, 对当前约束的顶点进行分类, 在交集中的为蓝色, 不在交集中的为红色.

(3) 连接每个蓝点与每个红点. 找出红蓝点之间的连线与交集之间的交点, 标记为黑色 (第 4 列).

(4) 将上面步骤找出来的所有点进行合并 (第 5 列), 并去除冗余点 (第 6 列). 最后将对偶表示转化为原始表示 (第 7 列), 得到两个多面体的上近似多面体.

通过 PRIMA 方法计算出的多神经元约束可以通过拉格朗日函数引入到边界计算过程中, 以此得到更紧的边界.

4.1.5 基于误差的符号区间传播

Verinet[62]中提出了基于误差的符号区间传播 (error-based symbolic interval propagation, ESIP) 技术. 与 CROWN, DeepPoly 方法不同, ESIP 对于每个节点只维护一个线性边界. 对于第 i 层的第 k 个节点, 作者用 $q^i(\boldsymbol{x})_k$ 来表示该节点所有的输入节点都处于其符号下界时, 该节点的输出关于输入 $\boldsymbol{x}$ 的表达式.

为了获得节点的符号上界, 作者计算使用符号下界时可能引入的误差值. 第 i 层的误差通过一个误差矩阵 $\mathbf{E}^{(i)} \in \mathbb{R}^{n_i \times n_i'}$ 表示, 其中 n_i 是第 i 层的节点数, $n_i' = \sum_{j\in[i-1]} n_j$ 是所有之前层节点数目的总和, 元素 $\mathbf{E}_{k,h}^{(i)}$ 表示使用节点 h 的数值上界而不是数值下界来传播时, 方程 $q_k^{(i)}(x)$ 的最大改变量. 在 ESIP 中, 对于第 i 层第 k 个神经元, 其与激活节点符号上下界为:

$$\bar{f}_k^{(i)}(\boldsymbol{x}) = q_k^{(i)}(\boldsymbol{x}) + \sum_{h|\mathbf{E}_{k,h}^{(i)}>0} \mathbf{E}_{k,h}^{(i)},\ \underline{f}_k^{(i)}(\boldsymbol{x}) = q_k^{(i)}(\boldsymbol{x}) + \sum_{h|\mathbf{E}_{k,h}^{(i)}<0} \mathbf{E}_{k,h}^{(i)} \tag{31}$$

即若某个节点 h 使用上界传播时对 $q_k^{(i)}(\boldsymbol{x})$ 的影响为正, 则将其加到上界中, 否则加到下界中. ESIP 在预激活节点的上下界具体化后采用三角形松弛获得激活后节点的上下界, 并由此计算出该节点引入的误差 $\max_{z_k^{(i)}\in[\boldsymbol{u}_k^{(i)},\boldsymbol{l}_k^{(i)}]}(\bar{f}_k^{(i)}(\boldsymbol{z}_k^{(i)}) - \underline{f}_k^{(i)}(\boldsymbol{z}_k^{(i)}))$. 下一层的误差矩阵则是当前层的误差矩阵根据松弛下界的系数进行放缩再拼接上当前层的计算出的误差后按照系数矩阵进行仿射变换得到.

4.1.6 星 集

星集 (star set) 是一种表示欧几里得空间 $\mathbb{R}^n$ 中状态集合的方法. 星集在计算集合的仿射变换与添加约束时表现高效, 并且方便使用线性规划进行求解, 而这些也都是使用抽象解释方法进行神经网络验证所需要的操作. 这里我们使用文献 [68] 中对星集的形式化定义.

定义 2 (星集). 星集 Θ 是一个元组 $\langle c, V, P\rangle$, 其中 $c \in \mathbb{R}^m$ 是星集的中心, 基向量集合 $V = \{v_1, v_2, \ldots, v_m\}$ 是 m 个在 $\mathbb{R}^n$ 空间中的基向量的集合, $P : \mathbb{R}^m \to \{\top, \bot\}$ 是一个谓词. 星集所表示的状态集合可以表示为:

$$[\![\Theta]\!] = \left\{ \boldsymbol{x} = \boldsymbol{c} + \sum_{i=1}^{m} (\boldsymbol{\alpha}_i \boldsymbol{v}_i) \mid P(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m) = \top \right\} \tag{32}$$

在神经网络验证问题中, 通常会将谓词限定为线性约束的合取, 即 $P(\boldsymbol{\alpha}) \triangleq \mathbf{C}\boldsymbol{\alpha} \leqslant \boldsymbol{d}$, 如果有 p 个线性约束, 则 $\mathbf{C} \in \mathbb{R}^{p\times m}$, $\boldsymbol{\alpha} \in \mathbb{R}^m$ 是包含 m 个变量的向量, 即 $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m]^{\mathrm{T}}$, 并且 $\boldsymbol{d} \in \mathbb{R}^{p\times 1}$. 星集 Θ 是空的, 当且仅当 $P(\alpha)$ 是空的.

对于星集 $\Theta = \langle \boldsymbol{c}, V, P\rangle$, 可以快速地进行仿射变换与添加新的约束. 给定仿射变换矩阵 $\mathbf{W}$ 和偏置向量 $\boldsymbol{b}$, 集合 $\{\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \mid \boldsymbol{x} \in \Theta\}$ 可以由星集 $\bar{\Theta} = \mathbf{W}\Theta + \boldsymbol{b} = \langle \bar{\boldsymbol{c}} = \mathbf{W}\boldsymbol{c} + \boldsymbol{b}, \bar{V} = \{\mathbf{W}\boldsymbol{v}_1, \mathbf{W}\boldsymbol{v}_2, \ldots, \mathbf{W}\boldsymbol{v}_m\}, \bar{P} = P\rangle$ 表示. 当添加一组新的约束 $\Theta \cap \{\boldsymbol{x} \mid \mathbf{H}\boldsymbol{x} \leqslant \boldsymbol{g}\}$ 时, 其结果也可以用一个新的星集 $\bar{\Theta} = \langle \bar{\boldsymbol{c}} = \boldsymbol{c}, \bar{V} = V, \bar{P} = P \wedge P'\rangle$ 表示, 其中 $P'(\boldsymbol{\alpha}) \triangleq \mathbf{H}\mathbf{V}_m\boldsymbol{\alpha} \leqslant \boldsymbol{g} - \mathbf{H}\boldsymbol{c}$, $\mathbf{V}_m = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m]$.

星集可以方便地表达多种抽象域, 常见的三角形松弛与 Zonotope 松弛均可以使用星集表示. 例如令 $\boldsymbol{c} = [0 \quad 0]^{\mathrm{T}}$, $\boldsymbol{v}_1 = [1 \quad 0]^{\mathrm{T}}$, $\boldsymbol{v}_2 = [0 \quad 1]^{\mathrm{T}}$, 且 $P(\boldsymbol{\alpha}) \triangleq \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{u}_j^{(i)} & -\boldsymbol{l}_j^{(i)} & \boldsymbol{u}_j^{(i)} & 0 \end{bmatrix}^{\mathrm{T}}$, 则该星集表示的松弛即为图 3(a) 所示的 Zonotope 松弛.

最开始使用星集进行神经网络验证的原始算法[68]对神经网络进行逐层处理. 假设第 i 层的输入星集集合, 也就是第 $i-1$ 层的输出星集集合为 $\mathcal{I}_i = \{\Theta_1, \Theta_2, \ldots, \Theta_N\}$. 对于每个 $\Theta_j \in \mathcal{I}_i$, 计算 Θ_j 代表的输入约束经过第 i 层节点可能到达的输出星集集合 $\mathcal{O}_j$, 最后 $\mathcal{R}_i = \bigcup_j \mathcal{O}_j$ 即为第 i 层的可达集.

计算 Θ_j 对应的输出星集集合 $\mathcal{O}_j$ 时, 按照下面的方式进行计算.

(1) 仿射变换. 将上一层的输入进行仿射变换, 即 $In = \mathbf{W}^{(i)}\Theta_j + \boldsymbol{b}^{(i)}$.

(2) 初始化. 计算在 In 表示的约束下每个节点 $\text{node}_k^{(i)}$ 的上下界 $\boldsymbol{l}_k^{(i)}$, $\boldsymbol{u}_k^{(i)}$. 初始化集合 $O_j=\{In\}$. 令 $\mathbf{M}=[\boldsymbol{e}_1\ \ldots\ \boldsymbol{e}_{k-1}\ 0\ \boldsymbol{e}_{k+1}\ \ldots\ \boldsymbol{e}_{n_i}]\in\mathbb{R}^{n_i}\times\mathbb{R}^{n_i}$, 即将单位矩阵的第 k 个元素置为 0.

(3) 计算输出星集. 对于该层第 k 个神经元, 初始化 $\mathcal{T}=\varnothing$. 遍历 O_j, 对于其中的元素 Θ', 如果 $\boldsymbol{l}_k^{(i)}\geqslant 0.$, 则 $\mathcal{T}=\mathcal{T}\cup\Theta'$; 如果 $\boldsymbol{u}_k^{(i)}\leqslant 0$, 则 $\mathcal{T}=\mathcal{T}\cup\mathbf{M}\Theta'$, 其中 $\mathbf{M}\Theta'$ 相当于多面体的第 k 维始终为 0; 如果 $\boldsymbol{l}_k^{(i)}<0<\boldsymbol{u}_k^{(i)}$, 则有 $\mathcal{T}=\mathcal{T}\cup(\Theta'\wedge\boldsymbol{x}[k]\geqslant 0)\cup\mathbf{M}(\Theta'\wedge\boldsymbol{x}[k]<0)$, 其中 $\Theta'\wedge\boldsymbol{x}[k]<0$ 表示对星集 Θ' 表示的多面体添加第 k 维小于 0 的约束, $\Theta'\wedge\boldsymbol{x}[k]\geqslant 0$ 同理, 随后令 $O_j=\mathcal{T}$, 继续处理下一个节点. 当所有节点均处理完成后, O_j 即为输入 Θ_j 对应的输出星集.

图 8 展示的是使用星集计算图 1(b) 中网络第 1 层输出星集的过程. 这里用 $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ 分别对应 $\text{node}_1^{(1)}$, $\text{node}_2^{(1)}$, $\text{node}_3^{(1)}$ 的取值. 首先, 将表示输入的星集通过仿射变换转换到三维, 作为第 1 层节点的输入星集. 随后处理 $\text{node}_1^{(1)}$, 由于 $\text{node}_1^{(1)}$ 状态无法确定, 所以需要对其状态进行分支, 体现在图中则是对将原始多面体分为 $\boldsymbol{x}_1\geqslant 0$ ($\text{node}_1^{(1)}$ 激活) 与 $\boldsymbol{x}_1\leqslant 0$ ($\text{node}_1^{(1)}$ 非激活) 两个部分, 随后根据节点的激活状态进行仿射变换 (乘以 $\mathbf{M}$) 形成两个星集. 继续对该层其他神经元进行讨论, $\text{node}_2^{(1)}$ 的状态同样无法确定, 所以同样对 $\boldsymbol{x}_2$ 大于 0 或小于 0 进行讨论, 形成 4 个星集. 最后, $\text{node}_3^{(1)}$ 的上界小于 0, 处于非激活状态, 不需要进行分支. 最终得到的 4 个星集即为该层的输出星集.

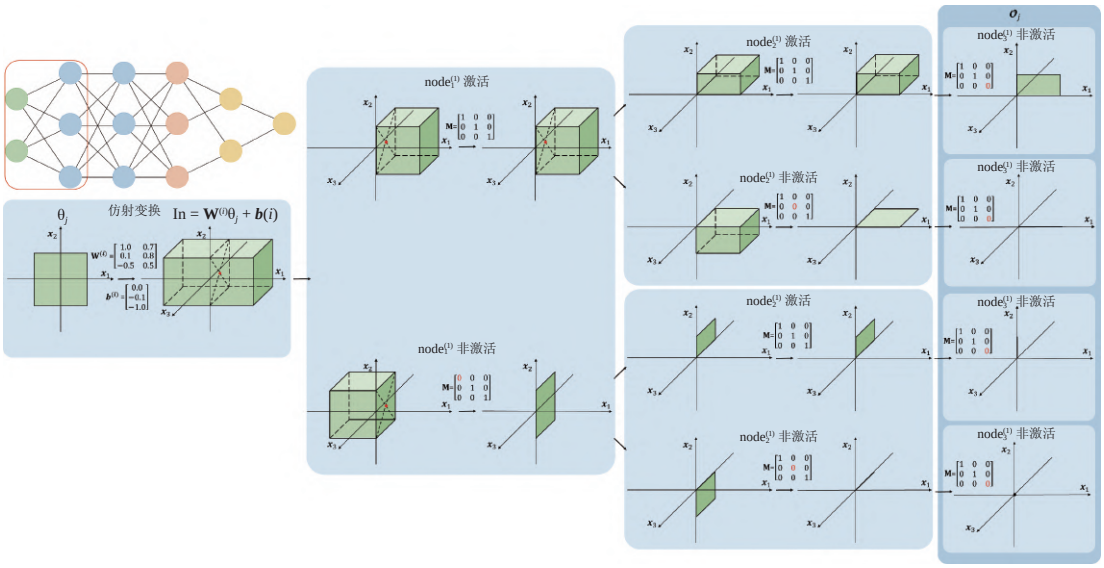


图 8　星集计算

实际上, 该算法同样是将每个无法确定状态的神经元进行分支, 分别计算激活和非激活两种情况下的星集. 最坏情况下对于一组输入星集, 可能会产生指数级别的输出星集. 为了避免这种指数爆炸的情况, 作者将 $\boldsymbol{x}[k]<0$ 与 $\boldsymbol{x}[k]\geqslant 0$ 两种情况的约束转换为三角形松弛约束. 通过这种方式, 每层可以用一个星集表示, 从而减少了星集的数量. 但是由于使用了松弛, 这种验证并不是完备的, 需要引入分支限界过程来保证验证的完备性.

4.2　约束求解

在约束求解方面, α,β-CROWN, MN-BaB 直接使用拉格朗日乘子法, 将约束编码到边界计算时的目标方程中. Marabou 采用基于 SMT 的求解算法 Reluplex 求解约束问题. 其他的工具大都使用混合整数线性规划或者使用带松弛的线性规划来进行约束求解. 下面, 我们将详细介绍上述约束求解方法.

4.2.1　拉格朗日乘子法

GCP-CROWN 与 β-CROWN 均是将边界约束利用拉格朗日乘子法或者对偶方法转化为优化问题, 在优化边界的同时可以判断出约束是否成立. 此外, GCP-CROWN 中会启动额外的线程对原始问题使用 MILP 方法进行求解.

MN-BaB 中由 PRIMA 方法产生的第 i 层约束可以写成下面的形式:

$$\mathbf{H}^{(i)}\boldsymbol{z}^{(i)}+\mathbf{G}^{(i)}\hat{\boldsymbol{z}}^{(i)}-\boldsymbol{d}\leqslant\mathbf{0} \tag{33}$$

然后使用拉格朗日乘子法引入到边界计算过程中. 本质上 MN-BaB 使用多神经元约束的方式相当于 β-CROWN 的基础上, 将 PRIMA 产生的多神经元约束利用拉格朗日乘子法添加到反向传播的过程中, 最后利用 GPU 来加速优化和求解过程.

4.2.2 Reluplex

Reluplex 是基于 SMT 的针对神经网络的求解算法. 该算法将所有 ReLU 拆分为预激活节点与激活节点. 将节点间的关系分为线性关系与非线性关系 (ReLU 约束), 首先利用单纯形法 (simplex) 处理线性关系, 当线性关系满足时, 再逐步调整非线性关系. 算法在找到反例或者成功验证网络性质成立时终止.

在单纯形法中, 原子命题 (atom) 为 $\sum_{x_i\in\mathcal{X}}c_ix_i\ \square d$ 的形式, 其中 $\square\in\{=,\leqslant,\geqslant\}$, $\mathcal{X}$ 为所有变量的集合. 对每个原子命题引入一个松弛变量 $b=\sum_{x_i\in\mathcal{X}}c_ix_i$, d 作为变量 b 的一个界. 这样就把问题转换成了一组等式约束以及一组上下界约束.

单纯形法中的变量分为基础变量和非基础变量, 并且每个变量都维护一个满足等式约束的赋值. 其中, 基础变量的赋值不能直接更改, 非基础变量的赋值可以在上下界范围内随意更改. 如果要对基础变量的值进行更改, 则需要通过移项的方法将基础变量和非基础变量进行交换 (pivot). 在单纯形法开始时初始化所有变量的赋值为 0, 以保证等式约束被满足; 随后通过不断调整非基础变量的赋值, 来使得变量满足上下界的约束.

Reluplex 是在单纯形法的基础上增加 ReLU 原子, 即预激活值 $z_j^{(i)}$ 与激活后的值 $\hat{z}_j^{(i)}$ 组成的元组 $\langle z_j^{(i)},\hat{z}_j^{(i)}\rangle$, 并要求满足 $\hat{z}_j^{(i)}=\max(0,z_j^{(i)})$; 若不满足, 则先尝试对其进行修改以强制满足 ReLU 约束. 如果发现某个 ReLU 节点多次不满足, 那么对这个节点进行拆分 (split), 猜测他是 (下界为 0) 否 (上界为 0) 被激活, 类似分支限界中的分支选择. 如果找到一组赋值, 使得性质不成立, 则我们找到了原性质的一组反例; 否则, 我们可以说明性质成立.

我们取图 4 中第 1 层与第 2 层的第 2 个节点构成的网络来说明 Marabou 的验证过程. 如图 9 所示, 假设我们要验证在 $\boldsymbol{x}_1,\boldsymbol{x}_2\in[-1,1]$ 时, $z_2^{(2)}<0.3$. Marabou 首先将问题转换为寻找反例, 即尝试找到 $z_2^{(2)}\geqslant 0.3$ 的一组赋值. 随后将所有 ReLU 节点拆为两个节点并计算节点的边界信息. 根据边界信息可以得出 $\text{node}_3^{(1)}$ 处于非激活状态, 所以在计算时不需要考虑该节点. 在此基础上, Marabou 开始执行 Reluplex 算法.

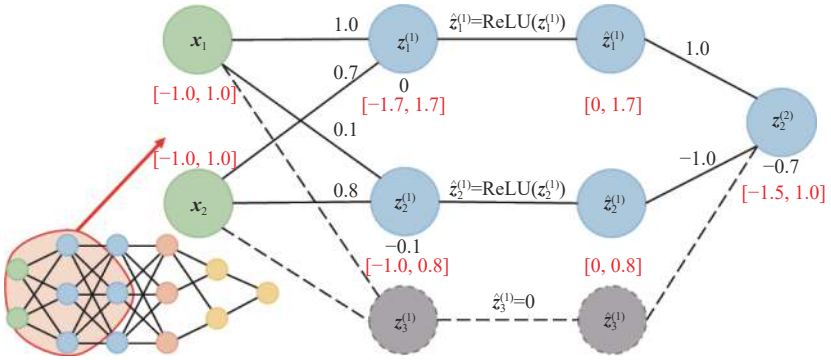


图 9 待验证网络 f'

首先, 将神经网络中的约束添加松弛变量, 将所有约束转换为等式约束, 有:

$$b_1=z_1^{(1)}-\boldsymbol{x}_1-0.7\boldsymbol{x}_2,\ b_2=z_2^{(1)}-0.1\boldsymbol{x}_1-0.8\boldsymbol{x}_2,\ b_3=z_2^{(2)}-\hat{z}_1^{(1)}+\hat{z}_2^{(1)},$$

其中, 初始化所有变量的赋值为 0, 以保证等式约束成立, 上下界信息根据边界计算获取, 并且我们默认等式左侧为基础变量. 初始化时变量的上下界信息如表 2 所示.

可以发现变量 b_2, $z_2^{(2)}$ 与 b_3 的赋值不满足上下界. 首先, $z_2^{(2)}$ 为非基础变量, 可以直接调整其赋值, 令 $z_2^{(2)}=0.3$ 以满足上下界约束. 此时 $b_2=0$, $b_3=0.3$ 仍然不满足上下界约束, 并且由于二者均为基础变量无法调整其赋值, 所

以需要通过移项来将其变为非基础变量. 我们将 b_2 与 $z_2^{(1)}$ 进行交换, 得到 $z_2^{(1)} = 0.1\boldsymbol{x}_1 + 0.8\boldsymbol{x}_2 + b_2$, 并修改 b_2 的值为 -0.1 以满足上下界条件. 同样, 我们将 b_3 与 $\hat{z}_1^{(1)}$ 进行交换并令 $b_3 = -0.7$. 此时基础变量及等式约束为:

$$b_1 = z_1^{(1)} - \boldsymbol{x}_1 - 0.7\boldsymbol{x}_2,\ \ z_2^{(1)} = 0.1\boldsymbol{x}_1 + 0.8\boldsymbol{x}_2 + b_2,\ \ \hat{z}_1^{(1)} = z_2^{(2)} + \hat{z}_2^{(1)} - b_3.$$

调整后变量赋值情况如表 3 所示.

表 2　初始变量赋值

变量	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$z_1^{(1)}$	$z_2^{(1)}$	$\hat{z}_1^{(1)}$	$\hat{z}_2^{(1)}$	$z_2^{(2)}$	b_1	b_2	b_3
上界	1	1	1.7	0.8	1.7	0.8	1	0	−0.1	−0.7
赋值	0	0	0	0	0	0	**0**	0	**0**	**0**
下界	−1	−1	−1.7	−1	0	0	0.3	0	−0.1	−0.7

表 3　修改变量赋值

变量	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$z_1^{(1)}$	$z_2^{(1)}$	$\hat{z}_1^{(1)}$	$\hat{z}_2^{(1)}$	$z_2^{(2)}$	b_1	b_2	b_3
上界	1	1	1.7	0.8	1.7	0.8	1	0	−0.1	−0.7
赋值	0	0	**0**	−0.1	**1**	0	0.3	0	−0.1	−0.7
下界	−1	−1	−1.7	−1	0	0	0.3	0	−0.1	−0.7

可以发现, 此时所有变量的赋值都满足上下界与等式约束. 随后 Marabou 检查所有 ReLU 节点拆成的变量之间是否满足 ReLU 约束, 这里可以发现 $\hat{z}_1^{(1)} \neq \max(z_1^{(1)}, 0)$, 且此刻 $\hat{z}_1^{(1)}$ 为基础变量无法进行修改, 于是令 $z_1^{(1)} = 1.0$ 以满足 ReLU 约束, b_1 的值随之变为 1.0, 下界发生冲突, 且由于 b_1 为基础变量, 无法直接对 b_1 的赋值进行修改, 所以我们将 b_1 与 $\boldsymbol{x}_1$ 交换后修改其赋值为 1. 此时我们的基础变量及等式约束为:

$$\boldsymbol{x}_1 = z_1^{(1)} - 0.7\boldsymbol{x}_2 - b_1,\ \ z_2^{(1)} = 0.1\boldsymbol{x}_1 + 0.8\boldsymbol{x}_2 + b_2,\ \ \hat{z}_1^{(1)} = z_2^{(2)} + \hat{z}_2^{(1)} - b_3.$$

并且对应的赋值如表 4 所示. 可以看到此时所有变量都满足等式与上下界约束, 并且所有的 ReLU 约束也都被满足. 所以, 我们找到了一组违反性质的赋值, 即 $\boldsymbol{x}_1 = 1$, $\boldsymbol{x}_0 = 0$, 所以 $\boldsymbol{x}_1, \boldsymbol{x}_2 \in [-1, 1]$ 时, 性质 $z_2^{(2)} < 0.3$ 不成立.

表 4　最终变量赋值

变量	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$z_1^{(1)}$	$z_2^{(1)}$	$\hat{z}_1^{(1)}$	$\hat{z}_2^{(1)}$	$z_2^{(2)}$	b_1	b_2	b_3
上界	1	1	1.7	0.8	1.7	0.8	1	0	−0.1	−0.7
赋值	1	0	1.0	0	1.0	0	0.3	0	−0.1	−0.7
下界	−1	−1	−1.7	−1	0	0	0.3	0	−0.1	−0.7

4.3　分支选择

多数工具都提出了各自的启发式方法来进行分支选择. α,β-CROWN 使用 FSB[69]与 BaBSR[61]的并行版本选择待分支的节点; MN-BaB 提出激活约束评分分支策略与成本分支调整策略启发式的选择分支节点; Vernet 则提出了影响分数的概念, 并选择具有最大影响的节点进行分支; Marabou 基于马尔科夫链蒙特卡洛搜索来找到使得不可满足和函数全局最小的节点状态组合, PEREGRiNN 使用带松弛的 LP 编码网络, 并优先选择层数考前且松弛最大的节点进行分裂, LayerSAR 也采用逐层确定节点的分支顺序; Planet 着重分析冲突产生的原因, 并利用 SAT 框架进行剪枝; Vennus 则利用将节点依赖关系编码至 MILP 求解器进行剪枝. 以下是对上述部分分支选择方法的介绍.

4.3.1　激活约束评分策略与成本调整分支策略

MN-BaB 中, 作者提出了两种新的启发式策略: 基于拉格朗日乘子法中激活约束的激活约束评分分支策略 (active constraint score branching, ACSB) 以及侧重于分支后所需计算成本的成本调整分支策略 (cost adjusted branching, CAB).

● 激活约束评分分支策略. 假设 PRIMA 框架生成的第 i 层约束为 $g(\boldsymbol{z}^{(i)}, \hat{\boldsymbol{z}}^{(i)}) = \mathbf{H}^{(i)}\boldsymbol{z}^{(i)} + \mathbf{G}^{(i)}\hat{\boldsymbol{z}}^{(i)} - d \leqslant 0$, 这些约束

在反向传播中乘上 $\boldsymbol{\gamma}^{(i)\top}$ 作为拉格朗日乘子项. 当 $\boldsymbol{\gamma}_j^{(i)}>0$ 时, 表示第 j 条约束被激活, 并且有 $g(\boldsymbol{z}^{(i)},\hat{\boldsymbol{z}}^{(i)})_j=0$. 对于第 i 层的第 k 个节点, 作者将该节点相关的拉格朗日乘子系数相加, 即 $\text{score}_k^{(i)}=|\boldsymbol{\gamma}^{(i)\mathrm{T}}\mathbf{H}^{(i)}|_k+|\boldsymbol{\gamma}^{(i)\mathrm{T}}\mathbf{G}^{(i)}|_k$. 这实际上相当于计算 $|\partial_{z^{(i)}}\boldsymbol{\gamma}^{(i)\mathrm{T}}g(\boldsymbol{z}^{(i)},\hat{\boldsymbol{z}}^{(i)})|_k+|\partial_{\hat{z}^{(i)}}\boldsymbol{\gamma}^{(i)\mathrm{T}}g(\boldsymbol{z}^{(i)},\hat{\boldsymbol{z}}^{(i)})|_k$, 直观上这代表了修改节点 $\text{node}_k^{(i)}$ 会对这组约束造成的影响, $\text{score}_k^{(i)}$ 越大则表示这组约束对节点 $\text{node}_k^{(i)}$ 的变化越敏感, 对节点 $\text{node}_k^{(i)}$ 状态进行分支更有可能导致约束 $g(\boldsymbol{z}^{(i)},\hat{\boldsymbol{z}}^{(i)})\leqslant 0$ 被违反, 从而加速证明过程或者收紧这组约束.

● 成本调整分支策略. 现有的分支选择方法在分支时, 往往只考虑分支后对边界的改进, 而忽略了不同的分支方法在计算成本方面有时会有相当大的差异. 成本调整分支策略则将预期的边界改进 (用分支得分近似) 与拆分所需要的期望成本的倒数相乘, 然后选择单位成本中对边界改进最高的分支. 拆分的真实成本包括下一步分支的直接成本和所有后续步骤的累积成本, 但通常只考虑前者就是一个不错的近似. 在 MN-BaB 中, 作者使用计算新的边界所需的浮点操作数量来近似这个直接成本.

4.3.2 不满足和函数

在 Marabou 的基础上, Wu 等人[70]提出一种基于不可满足和 (sum-of-infeasibilities, SoI) 函数的方法来引导约束求解器尽可能满足非线性约束, 从而减少分支选择的次数.

具体来说, 该方法将最优化目标改为方程 (34):

$$\min f_{\text{soi}}=\min\sum_{i\in[L],j\in[n_i]}E(\text{node}_j^{(i)}) \tag{34}$$

其中, $E(\text{node}_j^{(i)})$ 如方程 (35) 所示:

$$E(\text{node}_j^{(i)})=\min\{\hat{\boldsymbol{z}}_j^{(i)}-\boldsymbol{z}_j^{(i)},\hat{\boldsymbol{z}}_j^{(i)}\} \tag{35}$$

这里 $\hat{\boldsymbol{z}}_j^{(i)}-\boldsymbol{z}_j^{(i)}$, $\hat{\boldsymbol{z}}_j^{(i)}$ 两个参数分别对应 ReLU 节点处于激活与非激活状态的误差. 当且仅当误差为 0 时, $E(\text{node}_j^{(i)})=0$. 为了求解方程 (34), 作者利用马尔可夫链蒙特卡罗 (MCMC) 搜索来找到使得 SoI 全局最小的节点状态组合.

4.3.3 影响分数

Verinet 基于影响分数或分割分数进行分支选择, 并且不将性质编码到神经网络. 假设要验证的性质为: 在给定约束下, 神经网络的分类始终是 c. 分支选择时, Verinet 启发式地选择对于原神经网络中第 c 个输出下界 $\underline{f}_c^{(L)}$ 和第 t 个输出上界 $\bar{f}_t^{(L)}$ 影响最大的节点. 为了更好的评估隐藏层节点对输出的影响, 作者提出了影响分数 (impact-score) 的概念. 影响分数定义为下:

$$impact(h)=|\mathcal{T}|\cdot\left|\min\left(\mathbf{E}_{c,h}^{(m)},0\right)\right|+\sum_{t\in\mathcal{T}}\max\left(\mathbf{E}_{t,h}^{(m)},0\right) \tag{36}$$

其中, $|\mathcal{T}|$ 是集合 $\mathcal{T}$ 的势 (元素个数), $\mathcal{T}$ 是可能存在反例的分类的集合, 即:

$$\mathcal{T}=\left\{t\in\{1,\ldots,m\}\mid t\neq c,\left(\min\nolimits_{\boldsymbol{x}\in C}\left(\underline{f}_c^{(L)}(\boldsymbol{x})\right)<\max\nolimits_{\boldsymbol{x}\in C}\left(\bar{f}_t^{(L)}(\boldsymbol{x})\right)\right)\right\}.$$

在 ESIP 中负值 $\mathbf{E}_{c,h}^{(m)}$ 被添加到 $\underline{f}_c^{(L)}(\boldsymbol{x})$, 正值 $\mathbf{E}_{t,h}^{(m)}$ 被添加到 $\bar{f}_t^{(L)}(\boldsymbol{x})$. 所以影响分数可以表示相关的边界在分支后的改变量. 由于改善 $\underline{f}_c^{(L)}(\boldsymbol{x})$ 的界对于所有 $t\neq c$, 都有助于证明 $\bar{f}_t^{(L)}(\boldsymbol{x})<\underline{f}_c^{(L)}(\boldsymbol{x})$ 的成立, 所以在其前面加上权重因子 $|\mathcal{T}|$.

Henriksen 等人[71]在影响分数的基础上提出了分割分数 (split score) 且分割分数 $split=split_{\text{dir}}+split_{\text{indir}}$ 同时考虑对一个节点分支造成的直接影响 (对输出层直接造成影响) $split_{\text{dir}}$ 与间接影响 (通过改变隐藏层节点的上下界间接影响输出) $split_{\text{indir}}$. 作者使用某个节点分支后减少的误差量近似直接和间接影响. 具体来说, 最后一层节点会直接改变输出, 所以最后一层节点引入的误差即为其直接影响. 当评估某个隐藏层节点对输出的间接影响时, 作者通过该节点引入的误差与其直接后继上界或下界的比值来估计分支会对后继的分割分数造成多大的影响, 并逐层传播直到输出层.

4.3.4 选择靠近输入的节点

PEREGRiNN[69]使用松弛变量的和作为最优化目标. 即将方程 (15) 中最优化目标改为:

$$\min_{\boldsymbol{x},\boldsymbol{z},\hat{\boldsymbol{z}},\boldsymbol{\alpha}}\sum_{i\in[L-1]}\sum_{j\in[n_i]}\boldsymbol{\alpha}_j^{(i)} \tag{37}$$

作者认为松弛变量之和有效地激励凸优化器在解决方案中尽可能少地使用未确定状态神经元, 并尽可能多地固定神经元的激活状态.

在分支时, 作者优先选择具有最大松弛的节点. PEREGRiNN 的搜索过程按从输入层到输出层逐层进行的方式. 当执行分支选择时, PEREGRiNN 总是选择输入最近的层中松弛变量最大的状态未确定的神经元. 作者认为松弛变量越大, 其对应的神经元对验证结果有更大的影响.

为了保证最大的松弛集中在最靠近输入的层中, 作者对最优化目标做了进一步的修改. 参考 Shoukry 等人[72]提出的方法, 找到一组合适的权重 $q_1,q_2,\ldots,q_L$, 使得求解最优化问题

$$\min_{x,z,\hat{z},\alpha}\sum_{i\in[L-1]}\sum_{j\in[n_i]}q_i\alpha_j^{(i)} \tag{38}$$

时, 可以使得距离输入层最近的层在所有层中拥有最大的总松弛.

与 PEREGRiNN 类似, LayerSAR[67]中也按照层来进行分支选择. LayerSAR 优先选择最接近输入并且存在未确定节点的层, 随后, 按照节点的数值上下界的间隔选择节点进行分支. 直观上, 靠近输入层的节点对神经网络输出的影响更大, 因为越靠近输入的节点上下界变更可能导致几乎所有隐藏层节点的上下界变化, 从而对输出产生较大的影响. 此外, 如果将间隔视为不确定性, 选择上下界间隔更大的节点进行分支可能会减少更多的不确定性. 最后, 当靠前的层中所有节点激活状态都被确定后, 网络的行为也可以被极大程度的限定住, 从而加速证明.

4.3.5 冲突子句学习

Planet[8]将 SAT 框架与线性规划相结合. 利用 SAT 求解器来枚举节点状态, 并利用冲突子句学习、单元传播等 SAT 求解技术来减少搜索空间. 对 ReLU 节点状态的枚举可以看作是一组约束集合. 对于一组约束, 可能真正导致冲突的原因只有少数几个约束, 即这组约束集合中可能存在冗余约束. Planet 利用弹性滤波 (elastic filtering) 方法, 从约束集合中去除冗余的约束, 保留真正导致冲突的节点状态组合, 并将其编码为冲突子句. 通过这种方式, 可以在随后的搜索中避免搜索到同样的状态组合.

为了找到更小的导致冲突的约束组合, 作者采用了弹性滤波方法. 例如对于约束 $z_2^{(1)}\leqslant 0\wedge\hat{z}_2^{(1)}\leqslant 0$ 添加松弛变量 a_1, 使得约束变为 $z_2^{(1)}\leqslant a_1\wedge\hat{z}_2^{(1)}\leqslant a_1$, 对于其他约束同理. 随后, 将松弛后的约束与原始约束合取, 并将所有松弛变量的和作为最优化目标, 最后选择最优解中最大的松弛变量并将其置为 0. 重复上述过程, 直到约束不可被满足, 这时所有松弛变量为 0 的约束构成了冲突子句. 直观来讲, 松弛变量越大表示其对应的约束需要更多的松弛才能被满足, 这意味着该约束更不容易被满足, 很有可能会导致冲突.

通过弹性滤波方法找到冲突之后, 可以利用 SAT 求解器框架进行剪枝. 我们假设 $\text{node}_2^{(1)}$ 非激活与 $\text{node}_2^{(3)}$ 非激活会导致冲突 (图 10 红色背景的约束), $\text{node}_1^{(2)}$ 激活与 $\text{node}_2^{(3)}$ 激活会导致冲突 (图 10 灰色背景约束). 当在节点 6 证明路径 $1\to 2\to 4\to 5\to 6$ 无法满足后, 利用弹性滤波方法发现 $\text{node}_2^{(1)}$ 非激活与 $\text{node}_2^{(3)}$ 非激活无法同时成立, 也就是说, 如果 $\text{node}_2^{(1)}$ 非激活则 $\text{node}_2^{(3)}$ 一定处于激活状态. 所以, 我们可以直接回溯到第 1 层, 并且直接可以断言在当前条件下 $\text{node}_2^{(3)}$ 一定激活. 随后再执行分支限界过程. 这里仍然选择 $\text{node}_1^{(2)}$ 激活分支, 由于 $\text{node}_1^{(2)}$ 激活与 $\text{node}_2^{(3)}$ 激活冲突, 所以此时可以直接完成节点 1 左子树的证明. 相比于原来的搜索树, 查找冲突约束可以减少大量的分支加速验证过程.

除此之外, 作者还额外提出了两种子句学习的方法. 一种方法是, 保存可行状态集合. 作者将松弛之和作为最优化目标, 当求解器判断出某种分支组合不会导致冲突时, 作者将这组分支所有确定的所有节点状态组合作为子句保存, 以减少可能的重复计算, 同时如果通过边界计算发现某种分支状态组合可以使得其他节点的状态被确定, 则也将这种情况作为子句保存. 另一种方法则是节点状态推断. 对某个节点, 人为枚举一些该节点输入节点的激活状态, 看是否能推导出在这种组合下当前节点的状态, 如果能确定当前节点的状态, 则将这种输入节点的组合作为子句.

4.3.6 节点依赖关系

Botoeva 等人[73]实现了 Venus 验证工具. 作者发现神经网络中节点之间具有相互依赖的关系, 即某个节点的

激活或者非激活可以导致另一个节点激活或者非激活. 作者将依赖关系分为层内依赖关系和层间依赖关系, 提出了计算这两种依赖关系的方法, 并基于这种依赖关系进行剪枝以加速证明.

给定同一层的一对节点 $\text{node}_q^{(i)}, \text{node}_r^{(i)}$, 我们定义 $\mathcal{V}_{q,r=0}^{(i)}$ 为 $\text{node}_r^{(i)}$ 的输入 (即第 $i-1$ 层的神经元) 的线性组合为 0 时, $\text{node}_q^{(i)}$ 可能的取值, 即 $\mathcal{V}_{q,r=0}^{(i)} \triangleq \{\mathbf{W}_{q:}^{(i)} \cdot \hat{\boldsymbol{z}}^{(i-1)} + \boldsymbol{b}_q^{(i)} \mid \mathbf{W}_{r:}^{(i)} \hat{\boldsymbol{z}}^{(i-1)} + \boldsymbol{b}_r^{(i)} = 0\}$. 根据之前的区间算数得到的 $i-1$ 层的上下界, 可以得到 $\mathcal{V}_{q,r=0}^{(i)}$ 与 $\mathcal{V}_{r,q=0}^{(i)}$ 的上下界, 这里我们分别用 $\boldsymbol{l}_{q,r=0}^{(i)}$, $\boldsymbol{u}_{q,r=0}^{(i)}$ 与 $\boldsymbol{l}_{r,q=0}^{(i)}$, $\boldsymbol{u}_{r,q=0}^{(i)}$ 表示. 将下界和上界分别相连可以得到图 11, 可以据此判断层内依赖类型. 在图 12(a) 中, 对于 $\text{node}_2^{(2)}$ 与 $\text{node}_3^{(2)}$, 有:

$$\begin{cases} \mathcal{V}_{2,3=0}^{(2)} = \{\hat{z}_1^{(1)} - \hat{z}_2^{(1)} - 0.7 \mid -\hat{z}_1^{(1)} + 0.7\hat{z}_2^{(1)} - 0.1 = 0\} \in [-1.04, -0.8] \\ \mathcal{V}_{3,2=0}^{(2)} = \{-\hat{z}_1^{(1)} + 0.7\hat{z}_2^{(1)} - 0.1 \mid \hat{z}_1^{(1)} - \hat{z}_2^{(1)} - 0.7 = 0\} \in [-1.04, -0.8]. \end{cases}$$

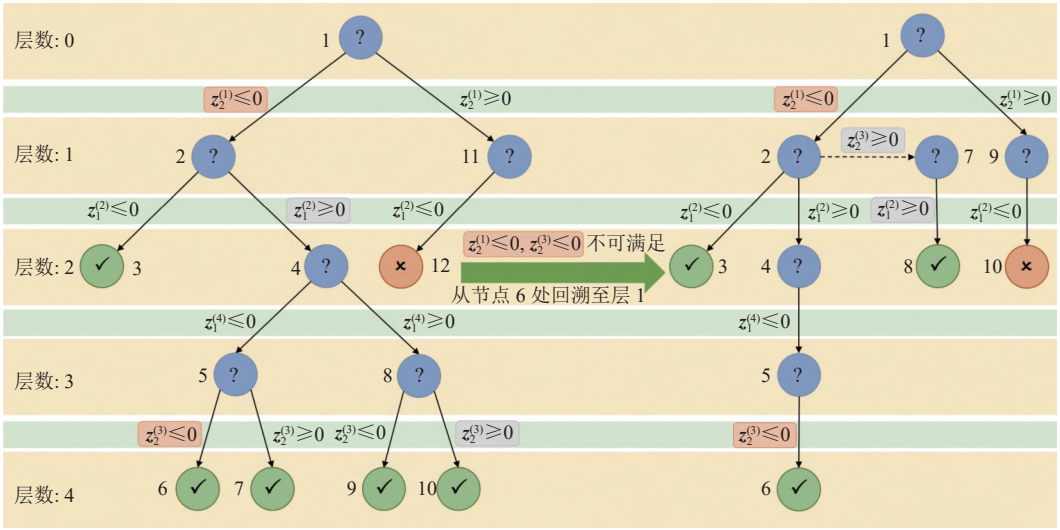


图 10 冲突子句引导的回溯

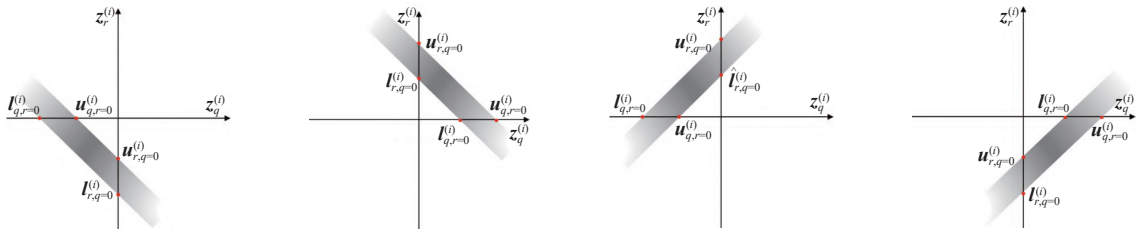


图 11 层内依赖类型[73]

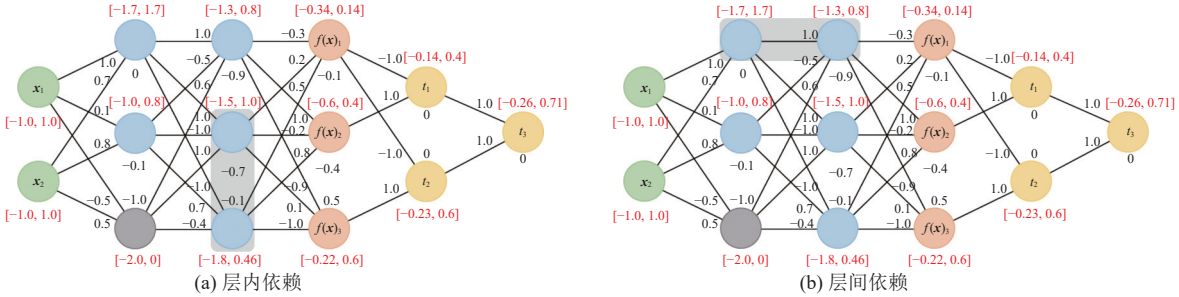


图 12 节点依赖

根据图 11(a) 可以得到, 如果 $\text{node}_2^{(2)}$ 激活, 则 $\text{node}_3^{(2)}$ 一定非激活.

层间依赖关系的计算则较为简单, 对于 $j=i+1$, 如果 $\boldsymbol{u}_r^{(j)}-\mathbf{W}_{r,q}^{(j)}\cdot\hat{\boldsymbol{u}}_q^{(i)}\leqslant 0$ 则若 $\text{node}_q^{(i)}$ 激活, $\text{node}_r^{(i)}$ 一定非激活; 如果 $\boldsymbol{l}_r^{(j)}-\mathbf{W}_{r,q}^{(j)}\cdot\hat{\boldsymbol{u}}_q^{(i)}\geqslant 0$ 则若 $\text{node}_q^{(i)}$ 激活, $\text{node}_r^{(i)}$ 一定激活. 对于图 12(b) 中的节点 $\text{node}_1^{(2)}$, 有 $\boldsymbol{u}_1^{(2)}-\mathbf{W}_{1,1}^{(2)}\hat{\boldsymbol{u}}_1^{(1)}=0.8-1.7=-0.9<0$, 所以我们找到了依赖关系: 若 $\text{node}_2^{(1)}$ 非激活则 $\text{node}_1^{(2)}$ 非激活.

Venus 是预先计算层间依赖和层内依赖, 并将这种依赖关系编码到 MILP 求解器中. 随后对输入以及选定的 ReLU 节点进行分割到一定程度之后, 再直接调用 MILP 进行求解. 在 Venus 的基础上, Venus2[65]提出了依赖图的概念. 依赖图是将依赖关系当作有向边进行建图. 在依赖图的基础上, 作者定义了依赖度, 所谓的依赖度是指如果当前节点状态确定, 可以推导出多少其他节点的状态. 选择依赖度最大的节点可以最大化地减少输入搜索空间. 但由于依赖图计算复杂度过高, 所以作者在一副依赖图上执行多次节点分支后再重新构建依赖图.

4.4 其他优化

除了上述 3 个核心部分外, 也存在一些其他的加速方法. 并行处理是加速验证的重要手段. α,β-CROWN 利用 GPU 的并行能力, 一次并行处理一批节点的边界; 其他工具如 Marabou, Venus2 等则是将分支后的信息以消息队列的形式进行保存, 随后并行处理每个子问题. 在神经网络攻击上, Verinet 工具使用基于误差的损失函数, 并使用梯度方法寻找反例. PEREGRiNN 针对星集方法做出了诸多优化. Elboher 等使用反例引导的抽象精化方法, 对神经网络整体进行抽象, 如果证明失败, 再尝试对网络进行精化. 最后, DeepInc 等工作更关注神经网络的增量验证. 在神经网络受到微扰后, 利用验证原始网络的信息加速第 2 次进行验证的速度. 下面, 我们将对上述加速方法进行详细的介绍.

4.4.1 并行优化

在并行上, 多数工具都支持将问题划分为多个子问题分别进行求解. 在此基础上 CROWN 系列还可以充分利用 GPU 的并行能力, 大大加速了边界运算. 除此之外, 还有一种类型是是并行启动一些进程产生一些辅助信息. 如 GCP-CROWN 在实现时, 当读取完成神经网络后, 将初始状态下的神经网络验证问题编码为 MILP 问题, 启动多个 MILP 求解器 (Cplex), 每个求解器都用来验证一个单独的输出约束并在单独的线程中运行. 同时, 主进程执行分支限界, 而无需等待 MILP 求解器进程. 在每次分支限界迭代中, 查询 MILP 解决进程并获取任何新生成的切割平面. 如果生成任何切割平面, 则将其添加到边界计算中优化后续分支和限界的边界. 如果未生成任何切割平面, 则 GCP-CROWN 将退化为 β-CROWN.

4.4.2 梯度方法寻找反例

Verinet 使用梯度方法寻找反例. 假设想要验证的性质是 $\boldsymbol{x}\in C$ 时, 神经网络的分类是 c. 基于 ESIP 方法可知, 如果有:

$$\underline{f}_c^{(L)}(\boldsymbol{x})-\tilde{f}_t^{(L)}(\boldsymbol{x})=\left(q_t^{(L)}(\boldsymbol{x})+\sum_{h|\mathbf{E}_{t,h}^{(L)}>0}\mathbf{E}_{t,h}^{(L)}\right)-\left(q_c^{(L)}(\boldsymbol{x})+\sum_{h|\mathbf{E}_{c,h}^{(L)}<0}\mathbf{E}_{c,h}^{(L)}\right)>0.$$

对于所有 $t\neq c$ 成立, 那么神经网络在给定输入内不会分类到除 c 外的其他类中, 即神经网络是安全的. 为了加速反例的寻找, Verinet 首先使用符号区间传播, 并使用 LP 求解器检查上述的方程的可满足性. 如果可满足 (意味着无法判断性质是否成立), 则将 $Loss(\boldsymbol{x})=f_c^{(L)}(\boldsymbol{x})-f_t^{(L)}(\boldsymbol{x})$ 作为损失函数, 使用梯度下降法来寻找反例; 如果找到了反例, 可以直接判断性质不成立; 否则, 当损失函数小于给定值或达到给定次数时停止.

4.4.3 星集优化

如果某个神经元具有 n 个输入, 则需要 $2n$ 次 LP 计算其对应每个输入的上下界, 这会消耗大量的时间. 为了解决这个问题, 作者提出了单循环 LP 方法快速筛选出需要进行 LP 的神经元变量. 首先, 算法将所有表示神经元的变量的和最小化, 以找到所有下界发生变化的变量. 如果有变量的下界发生变化, 那么将所有发生变化的变量之和作为新的最优化目标, 重新进行计算. 当没有新变量的下界发生变化时, 算法停止, 更新变量的下界. 随后, 对于上界采用类似的方式进行计算. 以图 13(a) 为例, 假设横坐标为 x, 纵坐标为 y, 原始约束为图中的矩形, 此时添加一

个新的约束, 深色部分是上近似的部分. 为了找出需要 LP 的变量, 首先最小化 $x+y$, 这里可以找到点 p_1, 与原始值相同, 这说明 x 和 y 的下界都没有变化. 随后, 最大化 $x+y$, 可以得到 p_2. 这里发现 x 的上界没有变化, 而 y 的上界变小了, 所以继续最大化 y, 可以得到 p_3, 此时即为 y 的新上界. 在这个例子中, LP 的次数从 4 次减到了 3 次. 作者认为在多维的情况下, 新增一个约束后上下界往往不发生变化, 如图 13(b) 所示, 所以这就可以把 $2n$ 次 LP 减少到了 2 次, 从而大大加快了验证速度.

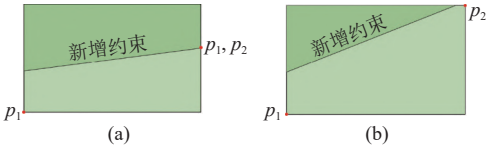


图 13 区间优化方法[54]

在抽象方式上, 作者提出了多轮抽象与每轮内进行多重抽象两类方法. 多轮分析的过程是先尝试一种抽象, 然后再尝试另一种抽象, 最后再进行分支每轮内进行多重抽象则是, 第 1 轮中, 可以仅尝试使用最佳面积 Zonotope 抽象来证明性质, 如果失败, 则第 2 轮可以同时使用所有 3 种 Zonotopes. 为了防止误差累积, 作者提出了执行引导抽象 (execution-guided overapproximation, EGO) 方法. 首先像进行精确分析一样尽可能地进行分支. 在成功验证了搜索树上的一条路径之后, 该方法像普通深度优先搜索一样回溯, 回溯后开始尝试抽象分析, 如果抽象分析成功, 则继续回溯直到抽象分析不再能成功证明属性为止, 随后再执行精确分析并重复.

此外, 作者还提出了 3 种其他启发式策略. 一种是分支限制 (split limit) 策略, 如果一条路径验证成功, 记这条路径上的分支次数为 m, 如果其他路径的分支次数大于 $k\cdot m$ (k 为一个常数), 那么就不再进行抽象分析, 直接将剩余的未确定节点进行分支. 第 2 种策略是抽象超时策略 (abs timeout), 即对抽象分析的运行时间进行限制, 如果超过则停止运行. 最后一种策略则是最少的分割次数 (split min) 策略, 由于分支限制策略可能导致能够抽象分析的部分很少, 所以可以强制一些分割次数少于一定阈值的抽象域始终采用星集分析.

4.4.4 反例引导的抽象精化方法

在神经网络验证领域, CEGAR 框架是一种用于验证复杂的神经网络模型的方法. CEGAR 框架结合了抽象和精化的思想, 通过迭代地进行抽象和具体化来逐步验证神经网络. 在 CEGAR 框架中, 首先对神经网络进行抽象, 将其转化为一个更简单的模型. 如 Prabhakar 等人[74]将同一隐藏层的两个神经元合并到一起, 与这两个神经元有关的权重和偏差将拓展为区间的形式以达到抽象的目的; DeepAbstract[33]使用 k 均值方法对同一层中输入输出上表现相似的神经元进行合并, 从而产生一个更小的网络; Elboher 等人[34]对神经元进行分类, 并根据类别合并网络从而简化网络结构. 得到抽象模型后, CEGAR 框架使用形式化验证技术 (如模型检测或定理证明) 对抽象模型进行验证. 如果抽象模型被证明是满足性质的, 则可以认为原始神经网络也满足该性质, 验证过程结束. 然而, 如果抽象模型验证失败, 即发现了反例, 则需要进行具体化步骤. 在具体化步骤中, 根据反例信息, 将抽象模型细化为一个更精确的模型, 以更好地捕捉原始神经网络的细节和行为. 重复进行抽象和具体化的迭代过程, 直到验证成功或无法再进一步细化为止. CEGAR 框架的核心思想是通过反例引导抽象的过程, 从而逐步优化验证过程的效率和准确性以此解决复杂网络的验证难题.

这里我们以文献 [34] 提出的抽象精化方法来说明 CEGAR 框架. 对于验证问题 $\langle f,C,\mathcal{P}\rangle$, 基于反例引导的抽象精化框架首先将性质 $\mathcal{P}$ 编码到神经网络 f 中, 得到网络 $f_{\mathcal{P}}$. 随后对神经网络 $f_{\mathcal{P}}$ 进行抽象, 得到抽象网络 $\overline{f}_{\mathcal{P}}$, 并保证对于所有的输入 $\boldsymbol{x}$ 都有 $f_{\mathcal{P}}(\boldsymbol{x})\geqslant\overline{f}_{\mathcal{P}}(\boldsymbol{x})$.

如果证明出 $\forall\boldsymbol{x}\in C.\overline{f}_{\mathcal{P}}(\boldsymbol{x})>0$, 那么 $\forall\boldsymbol{x}\in C.f_{\mathcal{P}}(\boldsymbol{x})>0$ 也一定成立. 否则, 验证工具会给出一个反例. 如果该反例是真反例, 则性质不成立; 否则, 需要对抽象网络 $f_{\mathcal{P}}$ 进行精化, 即寻找一个网络 $\overline{f}_{\mathcal{P}}{}'$ 使得对于任意输入 $\boldsymbol{x}$ 都有 $f_{\mathcal{P}}(\boldsymbol{x})\geqslant\overline{f}_{\mathcal{P}}{}'(\boldsymbol{x})\geqslant\overline{f}_{\mathcal{P}}(\boldsymbol{x})$, 并在新产生的精化网络上重新进行验证. 该框架重复精化与验证过程, 直到成功判断出性质是否成立. 该框架最坏情况下相当于直接在原网络上使用验证工具进行求解, 因此采用该框架的方法是否完备取决于框

架采用的求解工具. 文献 [34] 实现时使用完备验证工具 Marabou 作为求解器, 所以该实现版本理论上属于完备验证. 我们以图 14 与图 15 例说明该框架抽象和精化的大致流程. 为方便在图上表示, 用 $z_j^{(i)}$ 来代表第 i 层第 j 个神经元.

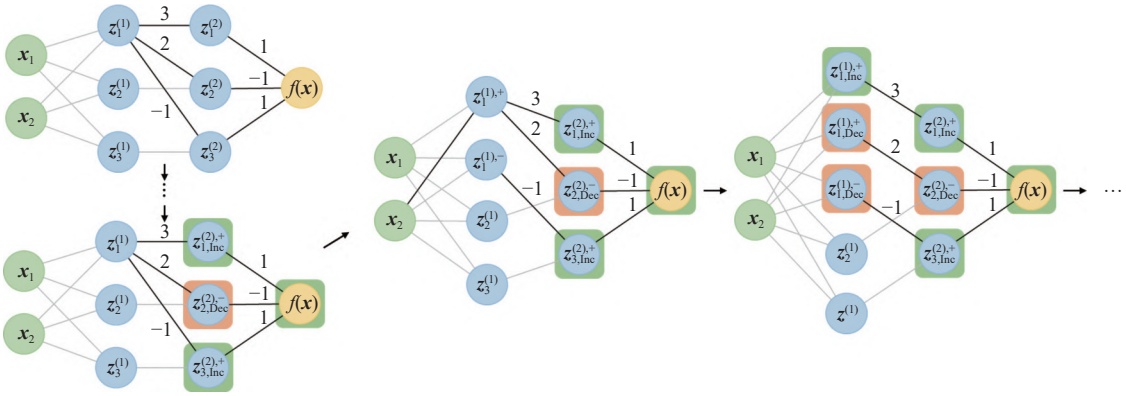


图 14　步骤 1: 网络等价变化并进行节点分类

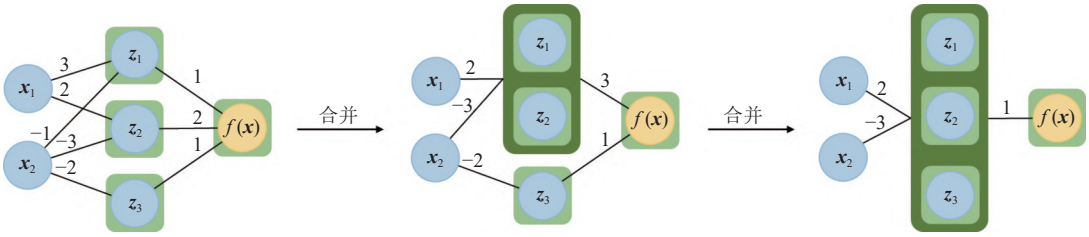


图 15　步骤 2: 合并同类神经元

● 抽象. 对于隐藏层神经元 $z_j^{(i)}$ 作者最多将其拆分为 4 个节点, $z_{j,\mathrm{Inc}}^{(i),+}$, $z_{j,\mathrm{Dec}}^{(i),+}$, $z_{j,\mathrm{Inc}}^{(i),-}$, $z_{j,\mathrm{Dec}}^{(i),-}$. 其中, 有“+, −”上标的节点分别只保留原节点中权值为正数还是负数的出边. “Dec”与“Inc”分别表示表示增大当前节点的值会使得神经网络的输出增大还是减小. 图 14 展示了第 2 层隐藏层的节点转换和分类已经完成时, 节点 $z_1^{(1)}$ 进行等价变换以及分类的过程. 其中绿色背景表示该节点分类为 Inc, 红色背景表示该节点分类为 Dec.

网络变换并分类后, 可以证明其与原始网络输出一致, 网络神经元数最多变为原来的 4 倍. 并且, 每个节点都属于 $\langle +,\mathrm{Inc}\rangle$, $\langle -,\mathrm{Inc}\rangle$, $\langle -,\mathrm{Inc}\rangle$, $\langle -,\mathrm{Dec}\rangle$ 这 4 类之一. 随后, 对所有的同类型节点进行合并. 图 15 以合并 $\langle +,\mathrm{Inc}\rangle$ 类节点为例展示合并过程. 合并时, 每次选择两个节点替换为一个新的节点. 为了满足 $f_{\mathcal{P}}(z)\geqslant \overline{f}_{\mathcal{P}}(z)$, 新节点相对于前一层节点的权值为两个节点相对于前一层节点权值中的较小值. 同时为了尽可能减少误差, 新节点相对于后一层节点的权值为两个节点相对于后一层节点的权值之和. 如图 15 中第 1 次合并时, x_1 与 z_1 有一条权值为 3 的边, x_1 与 z_2 有一条权值为 2 的边, 所以 x_1 对于合并 z_1, z_2 后的节点权值为 $\min(2,3)=2$. 对其他的节点采用类似的规则进行合并, 最终可以得到神经元数远小于 $f_{\mathcal{P}}$ 的抽象网络 $\overline{f}_{\mathcal{P}}$. 在选择节点合并时, 可以启发式地进行选择, 例如选择对输出改变最小的两个节点进行合并.

● 精化. 精化则是抽象的逆过程, 在该框架下, 则是将合并后的节点进行拆分. 拆分时也可以启发式地选择节点, 如选择拆分后对输出改变最大的节点进行精化. 最坏情况下, 精化过程会不断进行直到拆分为与 $f_{\mathcal{P}}$ 等价的网络, 如果调用完备的求解器, 则该方法也完备.

4.4.5　增量求解优化

近年来, 增量验证 (incremental verification) 问题已经成为神经网络验证领域的一个新兴研究方向. 部署的深度神经网络经常需要进行修改, 如微调[75]、模型修复[76]等. 每次经过类似的修改后, 都需要重新执行完备的验证

来检查新的神经网络是否具备某些性质. 传统的验证方法通常需要对新的网络重新进行重新验证, 这一过程在大型复杂的神经网络系统中可能非常耗时, 因此需要更高效的验证方法. 增量验证在第 1 次验证时保存有助于验证的关键信息, 从而大幅度提升第 2 次验证的效率.

DeepInc[43]在 Marabou 工具的基础上实现了增量求解的框架. 该框架保存第 1 次验证过程中的由于分支选择而形成的搜索树信息, 并且保留叶子节点 (UNSAT 节点或 SAT 节点) 的证明格局, 即基础变量与非基础变量的分布. 作者认为修复过程往往只会对网络结构或网络权值进行轻微的修改, 所以原始问题中验证出 SAT/UNSAT 节点在权值轻微变化后仍然很可能是 SAT/UNSAT. 所以在增量验证时, 可以直接定位到叶子节点, 采用相同的格局尝试证明, 如果证明成功则处理下一个叶子节点; 否则, 则在当前的状态下继续执行分支限界. 由于省去了第 1 次搜索树构建过程中的启发式分支选择以及单纯形法的过程, 该方法可以对第 2 次验证起到一定的加速效果.

Ugare 等人[44]提出增量求解框架 IVAN. 与 DeepInc 保存证明格局加速证明的角度不同, IVAN 则更侧重于对分支决策的优化. IVAN 设计了新的算法来重建一个更加紧凑的搜索树, 同时重新规划分支顺序并删除一些效果较差的分支决策, 然后按照全新的搜索树进行搜索. 通过这种方式可以充分利用重要的节点, 并避免做出较差的分支决策而拖慢证明效率.

5 实　验

神经网络验证竞赛 VNN-COMP[49,50]对多个工具的性能进行了比较. 然而, 由于比赛中不同工具可能使用不同的硬件设备, 有些工具使用 GPU 进行加速, 而另一些则仅使用 CPU, 这使得评判算法的优劣变得困难. 更重要的是, 该比赛仅关注各种工具的运行时间, 却没有深入探究不同工具所使用的各种加速方法真正起到的作用. 因此, 我们选择目前最先进并且开源的工具, 以及大 (CIFAR10_ResNet 数据集[49])、中 (MNIST FC 数据集[49])、小 (ACAS Xu 数据集[25]) 这 3 类开源数据集重新设计了一系列实验.

首先, 我们在统一设备上进行实验, 以公平地比较各个工具的性能. 其次, 我们将原本使用 GPU 加速的程序转换为使用 CPU 运行, 以比较在没有 GPU 加速的情况下核心算法的性能. 最后, 我们针对并行计算、PGD 攻击等常用的加速方法以及工具中提到的核心加速方式进行了一系列实验以探究这些加速方法的效果, 并尝试找出值得付出努力优化的方向. 限于篇幅, 我们将实验配置以及具体的实验结果和分析于链接 (https://docs.qq.com/doc/DSkROcFJVREJPdGlE?_t=1703775337829&u=17d25d05211c448cb4e6ee6f87d5ac58) 公开.

根据实验结果我们发现: 在性能方面, 不同工具擅长处理不同规模的网络, 不存在工具在所有规模的网络上的都做到顶尖的性能. 如在单线程情况下, VeriNet 在 ACAS Xu 上远超其他工具, Venus2 则在 MNIST FC 中表现最好, 在 Cifar10_resnet 数据集中则是 MN-BaB 表现最为突出; 而在开启所有加速方法时, nnenum 在 ACAS Xu 上表现最好, α,β-CROWN 则在其他两个数据集上有最好的表现. 在加速方法方面, 进行边界计算与约束求解时, 添加高质量的约束有助于加速验证; 例如 GCP-CROWN 添加额外的约束可以让平均求解速度上升 32.9%, 而是否添加多神经元约束对 MN-BaB 的效率影响不大. 分支选择顺序对于验证效率的影响较大, 合理的分支顺序可以加快求解效率; 例如 Verinet 使用随机分支选择时求解数目下降到原来的一半, 而 Marabou 关闭 SoI 方法反而可以多求解出几个验证问题. 并行方法对于小规模的网络加速明显, 但在中大型网络中, 并行的效果并不显著; 最后, 攻击方法多数情况下可以帮助工具迅速找出违反性质的反例.

6 结　语

完备神经网络验证领域发展迅速, 各种验证工具层出不穷. 本文总结了目前先进验证工具的通用框架, 并指出其中的核心部分: 边界计算、分支选择与约束求解. 我们对目前最先进工具在这 3 个核心部分中做出的探索进行了详细调研, 并给出易于理解的例子进行说明. 为了找出最有效的优化方向, 我们针对各种优化方法进行了实验.

从实验结果中我们发现, 在边界计算与约束求解时, 添加额外的约束确实能起到一定程度的加速, 但添加约束的质量更加关键. 在 MN-BaB 中, 是否使用多神经元约束对于验证效果来说几乎没有差别, 但是在 GCP-CROWN

中, 添加切割平面约束后可以减少 32.9% 的平均验证时间; 在 Venus2 中, 添加依赖关系后求解能力以及验证速度也都有一定程度的上升. 如何产生更加有效的约束以及如何更加高效的利用这些约束可能是一个可行的优化方向.

约束求解是目前大多数研究所欠缺的地方. 大多数工具直接使用现成的数学规划工具与理论, 只有 Reluplex 提出了自己的理论, 但在效率上却有所欠缺. 因此, 有必要进一步研究约束求解的理论创新以及设计针对神经网络验证的数学规划方法, 这可能是一个困难但充满潜力的方向.

另外, 分支选择的顺序对求解的效率有着较大的影响, 比较明显的是 Verinet 工具, 随机进行分支选择可能会造成超时, 但使用启发式选择却可以很快进行验证; 其他的工具如 PEREGRiNN 使用启发式的分支选择时也有着一定幅度的加速. 如何设计高效的分支选择策略是一个值得探索的问题.

其他加速技术中, 并行方法对于小型网络 (ACAS Xu) 的验证效果较为显著, 对于中型 (MNIST FC) 或者大型网络 (CIFAR10_ResNet) 的验证虽然有一定程度的加速, 但对工具的求解能力并没有本质上的提升, 除非极大的提升并行数量, 但这对计算设备的要求较高. PGD 攻击方法对于解决一些本身性质不成立的验证问题有很大的帮助, 但仍然需要进行小心的实现并设置合理的迭代次数, 否则可能反而拖慢求解速度, 甚至出现错误.

此外, 各个工具在易用性以及实现细节上仍然需要提升, α,β-CROWN、MN-BaB 虽然性能较好但配置繁琐; Verinet 有严重的内存泄露问题; Venus2 与 Marabou 的使用虽然简便, 但总是容易出现验证错误; PEREGRiNN, nnenum 同样使用简便, 但支持的网络类型较少.

最后, 值得注意的是, 网络规模仍然是横亘在验证工具发展之路上的一座大山, 即使是目前最先进的工具在验证稍大型的网络的性质时仍有无法在规定时间内完成的情况, 需要对完备神经网络加速技术进行进一步的研究. 完备神经网络验证领域的发展任重道远, 但我们相信通过不断的努力和创新, 该领域将取得更大的突破.

References:

[1] Lu D, Weng Q. A survey of image classification methods and techniques for improving classification performance. Int'l Journal of Remote Sensing, 2007, 28(5): 823–870. [doi: 10.1080/01431160600746456]

[2] Sheikhtaheri A, Sadoughi F, Hashemi Dehaghi Z. Developing and using expert systems and neural networks in medicine: A review on benefits and challenges. Journal of Medical Systems, 2014, 38(9): 110. [doi: 10.1007/s10916-014-0110-5]

[3] Julian KD, Kochenderfer MJ, Owen MP. Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics, 2019, 42(3): 598–608. [doi: 10.2514/1.G003724]

[4] Urmson C, Whittaker W. Self-driving cars and the urban challenge. IEEE Intelligent Systems, 2008, 23(2): 66–68. [doi: 10.1109/MIS.2008.34]

[5] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R. Intriguing properties of neural networks. arXiv:1312.6199, 2013.

[6] Pulina L, Tacchella A. An abstraction-refinement approach to verification of artificial neural networks. In: Proc. of the 22nd Int'l Conf. on Computer Aided Verification. Edinburgh: Springer, 2010. 243–257. [doi: 10.1007/978-3-642-14295-6_24]

[7] Katz G, Barrett C, Dill DL, Julian K, Kochenderfer MJ. Reluplex: An efficient SMT solver for verifying deep neural networks. In: Proc. of the 29th Int'l Conf. on Computer Aided Verification. Heidelberg: Springer, 2017. 97–117. [doi: 10.1007/978-3-319-63387-9_5]

[8] Ehlers R. Formal verification of piece-wise linear feed-forward neural networks. In: Proc. of the 15th Int'l Symp. on Automated Technology for Verification and Analysis. Pune: Springer, 2017. 269–286. [doi: 10.1007/978-3-319-68167-2_19]

[9] Huang XW, Kwiatkowska M, Wang S, Wu M. Safety verification of deep neural networks. In: Proc. of the 29th Int'l Conf. on Computer Aided Verification. Heidelberg: Springer, 2017. 3–29. [doi: 10.1007/978-3-319-63387-9_1]

[10] de Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Budapest: Springer, 2008. 337–340. [doi: 10.1007/978-3-540-78800-3_24]

[11] Gehr T, Mirman M, Drachsler-Cohen D, Tsankov P, Chaudhuri S, Vechev M. AI2: Safety and robustness certification of neural networks with abstract interpretation. In: Proc. of the 2018 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2018. 3–18. [doi: 10.1109/SP.2018.00058]

[12] Mirman M, Gehr T, Vechev MT. Differentiable abstract interpretation for provably robust neural networks. In: Proc. of the 35th Int'l

Conf. on Machine Learning. Stockholm: PMLR, 2018. 3575–3583.

[13] Singh G, Gehr T, Mirman M, Püschel M, Vechev M. Fast and effective robustness certification. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 10825–10836.

[14] Singh G, Gehr T, Püschel M, Vechev M. Boosting robustness certification of neural networks. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[15] Singh G, Gehr T, Püschel M, Vechev M. An abstract domain for certifying neural networks. Proc. of the ACM on Programming Languages, 2019, 3(POPL): 41. [doi: 10.1145/3290354]

[16] Singh G, Ganvir R, Püschel M, Vechev M. Beyond the single neuron convex barrier for neural network certification. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 1352.

[17] Wang SQ, Pei KX, Whitehouse J, Yang JF, Jana S. Formal security analysis of neural networks using symbolic intervals. In: Proc. of the 27th USENIX Conf. on Security Symp. Baltimore: USENIX Association, 2018. 1599–1614.

[18] Weng L, Zhang H, Chen HG, Song Z, Hsieh CJ, Daniel L, Boning D, Dhillon I. Towards fast computation of certified robustness for ReLU networks. In: Proc. of the 35th Int'l Conf. on Machine Learning. Stockholmsmässan: PMLR, 2018. 5276–5285.

[19] Wang SQ, Pei KX, Whitehouse J, Yang JF, Jana S. Efficient formal safety analysis of neural networks. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems (NIPS 2018). Montréal, 2018. 6369–6379.

[20] Gowal S, Dvijotham K, Stanforth R, Bunel R, Qin CL, Uesato J, Arandjelovic R, Mann TA, Kohli P. Scalable verified training for provably robust image classification. In: Proc. of the 2019 IEEE/CVF Int'l Conf. on Computer Vision. Seoul: IEEE, 2019. 4842–4851. [doi: 10.1109/ICCV.2019.00494]

[21] Boopathy A, Weng TW, Chen PY, Liu SJ, Daniel L. CNN-Cert: An efficient framework for certifying robustness of convolutional neural networks. In: Proc. of the 33rd AAAI Conf. on Artificial Intelligence. Honolulu: AAAI Press, 2019. 3240–3247. [doi: 10.1609/aaai.v33i01.33013240]

[22] Li JL, Liu JC, Yang PF, Chen LQ, Huang XW, Zhang LJ. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: Proc. of the 26th Int'l Static Analysis Symp. Porto: Springer, 2019. 296–319. [doi: 10.1007/978-3-030-32304-2_15]

[23] Zhang H, Chen HG, Xiao CW, Gowal S, Stanforth R, Li B, Boning D, Hsieh CJ. Towards stable and efficient training of verifiably robust neural networks. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020.

[24] Gopinath D, Katz G, Păsăreanu CS, Barrett C. DeepSafe: A data-driven approach for assessing robustness of neural networks. In: Proc. of the 16th Int'l Symp. on Automated Technology for Verification and Analysis. Los Angeles: Springer, 2018. 3–19. [doi: 10.1007/978-3-030-01090-4_1]

[25] Katz G, Huang DA, Ibeling D, Julian K, Lazarus C, Lim R, Shah P, Thakoor S, Wu HZ, Zeljić A, Dill DL, Kochenderfer MJ, Barrett C. The Marabou framework for verification and analysis of deep neural networks. In: Proc. of the 31st Int'l Conf. on Computer Aided Verification. New York City: Springer, 2019. 443–452. [doi: 10.1007/978-3-030-25540-4_26]

[26] Bastani O, Ioannou Y, Lampropoulos L, Vytiniotis D, Nori AV, Criminisi A. Measuring neural net robustness with constraints. In: Proc. of the 30th Int'l Conf. on Neural Information Processing Systems. Barcelona: Curran Associates Inc., 2016. 2621-2629.

[27] Müller MN, Makarchuk G, Singh G, Püschel M, Vechev M. PRIMA: General and precise neural network certification via scalable convex hull approximations. Proc. of the ACM on Programming Languages, 2022, 6(POPL): 43. [doi: 10.1145/3498704]

[28] Anderson G, Pailoor S, Dillig I, Chaudhuri S. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In: Proc. of the 40th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Phoenix: ACM, 2019. 731–744. [doi: 10.1145/3314221.3314614]

[29] Bertsimas D, Tsitsiklis JN. Introduction to Linear Optimization. Belmont: Athena Scientific, 1997.

[30] Tran HD, Yang XD, Manzanas Lopez D, Musau P, Nguyen LV, Xiang WM, Bak S, Johnson TT. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Proc. of the 32nd Int'l Conf. on Computer Aided Verification. Los Angeles: Springer, 2020. 3–17. [doi: 10.1007/978-3-030-53288-8_1]

[31] Tjeng V, Xiao KY, Tedrake R. Evaluating robustness of neural networks with mixed integer programming. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[32] Yang PF, Li RJ, Li JL, Huang CC, Wang JY, Sun J, Xue B, Zhang LJ. Improving neural network verification through spurious region guided refinement. In: Proc. of the 27th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Luxembourg City: Springer, 2021. 389–408. [doi: 10.1007/978-3-030-72016-2_21]

[33] Ashok P, Hashemi V, Křetínský J, Mohr S. DeepAbstract: Neural network abstraction for accelerating verification. In: Proc. of the 18th

Int'l Symp. on Automated Technology for Verification and Analysis. Hanoi: Springer, 2020. 92–107. [doi: 10.1007/978-3-030-59152-6_5]

[34] Elboher YY, Gottschlich J, Katz G. An abstraction-based framework for neural network verification. In: Proc. of the 32nd Int'l Conf. on Computer Aided Verification. Los Angeles: Springer, 2020. 43–65. [doi: 10.1007/978-3-030-53288-8_3]

[35] Hein M, Andriushchenko M. Formal guarantees on the robustness of a classifier against adversarial manipulation. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems (NIPS 2017). Long Beach, 2017. 2263–2273.

[36] Weng TW, Zhang H, Chen PY, Yi JF, Su D, Gao YP, Hsieh CJ, Daniel L. Evaluating the robustness of neural networks: An extreme value theory approach. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018.

[37] Bunel R, Lu JY, Turkaslan I, Torr PHS, Kohli P, Kumar MP. Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research, 2020, 21(42): 1–39.

[38] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2014.

[39] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018.

[40] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. In: Proc. of the 2016 IEEE European Symp. on Security and Privacy. Saarbruecken: IEEE, 2016. 372–387. [doi: 10.1109/EuroSP.2016.36]

[41] Moosavi-Dezfooli SM, Fawzi A, Frossard P. DeepFool: A simple and accurate method to fool deep neural networks. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 2574–2582. [doi: 10.1109/CVPR.2016.282]

[42] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proc. of the 2017 IEEE Symp. on Security and Privacy. San Jose: IEEE, 2017. 39–57. [doi: 10.1109/SP.2017.49]

[43] Yang PF, Chi ZM, Liu ZX, Zhao MY, Huang CC, Cai SW, Zhang LJ. Incremental satisfiability modulo theory for verification of deep neural networks. arXiv:2302.06455, 2023.

[44] Ugare S, Banerjee D, Misailovic S, Singh G. Incremental verification of neural networks. Proc. of the ACM on Programming Languages, 2023, 7(PLDI): 185. [doi: 10.1145/3591299]

[45] Urban C, Min'e A. A review of formal methods applied to machine learning. arXiv:2104.02466, 2021.

[46] Huang XW, Kroening D, Ruan WJ, Sharp J, Sun YC, Thamo E, Wu M, Yi XP. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review, 2020, 37: 100270. [doi: 10.1016/j.cosrev.2020.100270]

[47] Liu Y, Yang PF, Zhang LJ, Wu ZL, Feng Y. Survey on robustness verification of feedforward neural networks and recurrent neural networks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(7): 3134–3166 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6863.htm [doi: 10.13328/j.cnki.jos.006863]

[48] Ji SL, Du TY, Deng SG, Cheng P, Shi J, Yang M, Li B. Robustness certification research on deep learning models: A survey. Chinese Journal of Computers, 2022, 45(1): 190–206 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.00190]

[49] Bak S, Liu CL, Johnson T. The second international verification of neural networks competition (VNN-COMP 2021): Summary and results. arXiv:2109.00498, 2021.

[50] Müller MN, Brix C, Bak S, Liu CL, Johnson TT. The third international verification of neural networks competition (VNN-COMP 2022): Summary and results. arXiv:2212.10376, 2022.

[51] Ruan WJ, Huang XW, Kwiatkowska M. Reachability analysis of deep neural networks with provable guarantees. In: Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence. Stockholm: AAAI Press, 2018. 2651–2659.

[52] Ghorbal K, Goubault E, Putot S. The zonotope abstract domain taylor1+. In: Proc. of the 21st Int'l Conf. on Computer Aided Verification. Grenoble: Springer, 2009. 627–633. [doi: 10.1007/978-3-642-02658-4_47]

[53] Wu YT, Zhang M. Tightening robustness verification of convolutional neural networks with fine-grained linear approximation. In: Proc. of the 35th AAAI Conf. on Artificial Intelligence. AAAI Press, 2021. 11674–11681.

[54] Bak S. nnenum: Verification of ReLU neural networks with optimized abstraction refinement. In: Proc. of the 13th NASA Formal Methods Symp. Springer, 2021. 19–36. [doi: 10.1007/978-3-030-76384-8_2]

[55] Everett III H. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. Operations Research, 1963, 11(3): 399–417. [doi: 10.1287/opre.11.3.399]

[56] Gowal S, Dvijotham K, Stanforth R, Bunel R, Qin CL, Uesato J, Arandjelović R, Mann T, Kohli P. On the effectiveness of interval bound propagation for training verifiably robust models. arXiv:1810.12715, 2019.

[57] Bunel R, Turkaslan I, Torr PHS, Kohli P, Mudigonda PK. A unified view of piecewise linear neural network verification. In: Proc. of the

31st Int'l Conf. on Neural Information Processing Systems. Montréal, 2018. 4795–4804.

[58] Xu KD, Zhang H, Wang SQ, Wang YH, Jana S, Lin X, Hsieh CJ. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[59] Zhang H, Wang SQ, Xu KD, Li LY, Li B, Jana S, Hsieh CJ, Kolter JZ. General cutting planes for bound-propagation-based neural network verification. In: Proc. of the 35th Int'l Conf. on Neural Information Processing Systems. New Orleans, 2022. 1656–1670.

[60] Wang SQ, Zhang H, Xu KD, Lin X, Jana S, Hsieh CJ, Kolter JZ. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. 2021. 29909–29921.

[61] Ferrari C, Müller MN, Jovanović N, Vechev MT. Complete verification via multi-neuron relaxation guided branch-and-bound. In: Proc. of the 10th Int'l Conf. on Learning Representations. OpenReview.net, 2022.

[62] Henriksen P, Lomuscio A. Efficient neural network verification via adaptive refinement and adversarial search. In: Proc. of the 24th European Conf. on Artificial Intelligence. Santiago de Compostela, 2020. 2513–2520.

[63] Khedr H, Ferlez J, Shoukry Y. PEREGRiNN: Penalized-relaxation greedy neural network verifier. In: Proc. of the 33rd Int'l Conf. on Computer Aided Verification. Springer, 2020. 287–300. [doi: 10.1007/978-3-030-81685-8_13]

[64] Dutta S, Jha S, Sankaranarayanan S, Tiwari A. Output range analysis for deep feedforward neural networks. In: Proc. of the 10th NASA Formal Methods Symp. Newport News: Springer, 2018. 121–138. [doi: 10.1007/978-3-319-77935-5_9]

[65] Kouvaros P, Lomuscio A. Towards scalable complete verification of ReLU neural networks via dependency-based branching. In: Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence. IJCAI.org, 2021. 2643–2650. [doi: 10.24963/ijcai.2021/364]

[66] Zhang H, Weng TW, Chen PY, Hsieh CJ, Daniel L. Efficient neural network robustness certification with general activation functions. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Montréal, 2018. 4944–4953.

[67] Yin BH, Chen LQ, Liu JC, Wang J. Efficient complete verification of neural networks via layerwised splitting and refinement. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 2022, 41(11): 3898–3909. [doi: 10.1109/TCAD.2022.3197534]

[68] Tran HD, Manzanas Lopez D, Musau P, Yang XD, Nguyen LV, Xiang WM, Johnson TT. Star-based reachability analysis of deep neural networks. In: Proc. of the 3rd Int'l Symp. on Formal Methods. Porto: Springer. 2019. 670–686. [doi: 10.1007/978-3-030-30942-8_39]

[69] De Palma A, Behl HS, Bunel R, Torr PHS, Kumar MP. Scaling the convex barrier with active sets. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[70] Wu HZ, Zeljić A, Katz G, Barrett C. Efficient neural network analysis with sum-of-infeasibilities. In: Proc. of the 28th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Munich: Springer, 2022. 143–163. [doi: 10.1007/978-3-030-99524-9_8]

[71] Henriksen P, Lomuscio A. DEEPSPLIT: An efficient splitting method for neural network verification via indirect effect analysis. In: Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence. Montreal: IJCAI.org, 2021. 2549–2555. [doi: 10.24963/ijcai.2021/351]

[72] Shoukry Y, Nuzzo P, Sangiovanni-Vincentelli AL, Seshia SA, Pappas GJ, Tabuada P. SMC: Satisfiability modulo convex programming. Proc. of the IEEE, 2018, 106(9): 1655–1679. [doi: 10.1109/JPROC.2018.2849003]

[73] Botoeva E, Kouvaros P, Kronqvist J, Lomuscio A, Misener R. Efficient verification of ReLU-based neural networks via dependency analysis. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI Press, 2020. 3291–3299.

[74] Prabhakar P, Rahimi Afzal Z. Abstraction based output range analysis for neural networks. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 1414.

[75] Tajbakhsh N, Shin JY, Gurudu SR, Hurst RT, Kendall CB, Gotway MB, Liang JM. Convolutional neural networks for medical image analysis: Full training or fine tuning? IEEE Trans. on Medical Imaging, 2016, 35(5): 1299–1312. [doi: 10.1109/TMI.2016.2535302]

[76] Yu B, Qi H, Guo Q, Juefei-Xu F, Xie XF, Ma L, Zhao JJ. DeepRepair: Style-guided repairing for deep neural networks in the real-world operational environment. IEEE Trans. on Reliability, 2022, 71(4): 1401–1416. [doi: 10.1109/TR.2021.3096332]

附中文参考文献:

[47] 刘颖, 杨鹏飞, 张立军, 吴志林, 冯元. 前馈神经网络和循环神经网络的鲁棒性验证综述. 软件学报, 2023, 34(7): 3134–3166. http://www.jos.org.cn/1000-9825/6863.htm [doi: 10.13328/j.cnki.jos.006863]

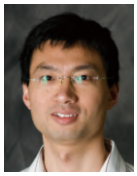
[48] 纪守领, 杜天宇, 邓水光, 程鹏, 时杰, 杨珉, 李博. 深度学习模型鲁棒性研究综述. 计算机学报, 2022, 45(1): 190–206. [doi: 10.11897/SP.J.1016.2022.00190]

刘宗鑫(1999－), 男, 博士生, CCF 学生会员,主要研究领域为形式化方法, 神经网络验证.

杨鹏飞(1993－), 男, 博士, CCF 专业会员, 主要研究领域为人工智能安全, 概率模型检验.

张立军(1979－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为概率模型检测, 协议验证, 学习算法, 自动驾驶系统验证.

吴志林(1980－), 男, 博士, 研究员, CCF 高级会员, 主要研究领域为计算逻辑, 自动机理论, 计算机软硬件基础设施的形式化验证.



黄小炜(1979－), 男, 博士, 教授, 博士生导师, 主要研究领域为人工智能安全与验证, AI 可解释性, 形式化方法.