DeepAbstract: Neural Network Abstraction for Accelerating Verification

Pranav Ashok[1], Vahid Hashemi[2], Jan Křetínský[1], and Stefanie Mohr[1]

[1] Technical University of Munich, Germany
[2] Audi AG, Germany

Abstract. While abstraction is a classic tool of verification to scale it up, it is not used very often for verifying neural networks. However, it can help with the still open task of scaling existing algorithms to state-of-the-art network architectures. We introduce an abstraction framework applicable to fully-connected feed-forward neural networks based on clustering of neurons that behave similarly on *some* inputs. For the particular case of ReLU, we additionally provide error bounds incurred by the abstraction. We show how the abstraction reduces the size of the network, while preserving its accuracy, and how verification results on the abstract network can be transferred back to the original network.

虽然抽象是扩展验证的经典工具，但它并不经常用于验证神经网络。然而，它可以帮助解决将现有算法扩展到最先进的网络架构的仍然开放的任务。我们引入了一个抽象框架，适用于全连接的前馈神经网络，基于对某些输入上行为类似的神经元的聚类。对于ReLU的特殊情况，我们还提供了抽象所产生的错误边界。我们展示了抽象如何在保持网络准确性的同时减小网络的大小，以及如何将抽象网络上的验证结果转移回原始网络。

1 Introduction

Neural networks (NN) are successfully used to solve many hard problems reasonably well in practice. However, there is an increasing desire to use them also in safety-critical settings, such as perception in autonomous cars [Che+17a], where reliability has to be on a very high level and that level has to be guaranteed, preferably by a rigorous proof. This is a great challenge, in particular, since NN are naturally very susceptible to adversarial attacks, as many works have demonstrated in the recent years [Pap+16; AM18; Don+18; SVS19].Consequently, various verification techniques for NN are being developed these days. Most verification techniques focus on proving robustness of the neural networks [CNR17; Ehl17; Hua+17; Kat+17; Geh+18; Sin+19b], i.e. for a classification task, when the input is perturbed by a small ε, the resulting output should be labeled the same as the output of the original input. Reliable analysis of robustness is computationally extremely expensive and verification tools struggle to scale when faced with real-world neural networks [Dvi+18].

神经网络(NN)在实践中被成功地解决了许多难题。然而，越来越多的人也希望在安全关键环境中使用它们，比如自动驾驶汽车的感知[Che+17a]，其中可靠性必须在非常高的水平上，该水平必须得到保证，最好是通过严格的证明。
这是一个巨大的挑战，特别是由于神经网络本身非常容易受到对抗性攻击，正如近年来许多工作所证明的那样[Pap+16；AM18；Don+18；SVS19]。因此，目前正在开发各种针对神经网络的验证技术。大多数验证技术侧重于证明神经网络的鲁棒性[CNR17；Ehl17；Hua+17；Kat+17；Geh+18；Sin+19b]，即对于分类任务，当输入受到小ε的干扰时，结果输出应该与原始输入的输出相同。可靠的鲁棒性分析在计算上极其昂贵，当面对现实世界的神经网络时，验证工具难以扩展[Dvi+18]。

Abstraction [CGL94; Cla+00] is one of the very classic techniques used in formal methods to obtain more understanding of a system as well as more efficient analysis. Disregarding details irrelevant to the checked property allows for constructing a smaller system with a similar behaviour. Although abstraction-based techniques are ubiquitous in verification, improving its scalability, such ideas have not been really applied to the verification of NN, except for a handful of works discussed later.

忽略与检查属性无关的细节，可以构建具有类似行为的更小的系统。虽然基于抽象的技术在验证中无处不在，提高了其可伸扩展性，但除了后来讨论的一些工作之外，这些想法并没有真正应用于神经网络的验证。

In this paper, we introduce an abstraction framework for NN. In contrast to syntactic similarities, such as having similar weights on the edges from the previous layer [ZYZ18], our aim is to provide a behavioural, semantic notion of similarity, such as those delivered by predicate abstraction, since such notions are more general and thus more powerful. Surprisingly, this direction has not been explored for NN. One of the reasons is that the neurons do not have an explicit structure like states of a program that are determined by valuations of given variables. What are actually the values determining neurons in the network?

本文介绍了一个神经网络的抽象框架。与语法相似性，如在前一层[ZYZ18]的边缘有相似的权重相比，我们的目标是提供一个行为、语义相似概念，如谓词抽象传递的概念，因为这些概念更普遍，因此更强大。令人惊讶的是，这个方向还没有被NN探索。其中一个原因是，神经元没有一个像由给定变量的评估决定的程序状态那样的显式结构。决定网络中神经元的实际值是什么？

Note that in many cases, such as recognition of traffic signs or numbers, there are finitely many (say k) interesting data points on which and on whose neighbourhood the network should work well. Intuitively, these are the key points that determine our focus, our scope of interest. Consequently, we propose the following equivalence on neurons. We evaluate the k inputs, yielding for each neuron a k-tuple of its activation values. This can be seen as a vector in $\mathbb{R}^k$. We stipulate that two neurons are similar if they have similar vectors, i.e, very close to each other. To determine reasonable equivalence classes over the vectors, we use the machine-learning technique of k-means clustering [HTF09]. While other techniques, e.g. principal component analysis [Bis06], might also be useful, simple clustering is computationally cheap and returns reasonable results. To summarize in other words, in the lack of structural information about the neurons, we use empirical behavioural information instead.

直观地说，这些都是决定我们的重点，我们感兴趣的范围的关键点。因此，我们提出了以下对神经元的等价性。我们评估k个输入，为每个神经元产生其激活值的k个元组。这可以看作是Rk中的一个向量。我们规定，如果两个神经元有相似的向量，那么它们是相似的，即彼此非常接近。为了确定向量上合理的等价类，我们使用了k-means聚类的机器学习[HTF09]的技术。虽然其他技术，如主成分分析[Bis06]，也可能很有用，但简单的聚类计算成本便宜，并返回合理的结果。换句话说，在缺乏关于神经元的结构信息的情况下，我们使用经验行为信息来代替。

Applications Once we have a way of determining similar neurons, we can merge each equivalence class into a single neuron and obtain a smaller, abstracted NN. There are several uses of such an NN. Firstly, since it is a smaller one, it may be preferred in practice since, generally, smaller networks are often more robust, smoother, and obviously less resource-demanding to run [Che+17b]. Note that there is a large body of work on obtaining smaller NN from larger ones, e.g. see [Che+17b; Den+20]. Secondly, and more interestingly in the safety-critical context, we can use the smaller abstract NN to obtain a guaranteed solution to the original problem (verifying robustness or even other properties) in two distinct ways:

一旦我们有一种确定相似神经元的方法，我们就可以将每个等价类合并为一个神经元，并获得一个更小的抽象神经网络。 这种神经网络有多种用途。 首先，由于它是一个较小的网络，因此在实践中可能更受欢迎，因为通常较小的网络通常更健壮、更流畅，并且运行所需的资源明显更少 [Che+17b]。 请注意，从较大的 NN 获得较小的 NN 方面有大量工作，例如 见 [Che+17b; 书房+20]。 其次，在安全关键上下文中更有趣的是，我们可以使用较小的抽象神经网络以两种不同的方式获得原始问题的有保证的解决方案（验证鲁棒性或什至其他属性）：

1. The smaller NN could replace the original one and could be easier to verify, while doing the same job (more precisely, the results can be ε-different where we can compute an upper bound on ε from the abstraction).

   较小的神经网络可以取代原来的神经网络，并且可以更容易验证，同时做相同的工作（更准确地说，结果可以是ε-不同的，我们可以从抽象计算ε的上界）。

2. We can analyze the abstract NN more easily as it is smaller and then transfer the results (proof of correctness or a counterexample) to the original one, provided the difference ε is small enough.

   由于抽象的神经网络较小，我们可以更容易地分析它，然后将结果（正确性证明或反例）转移到原始的神经网络，只要差异ε足够小。

The latter corresponds to the classic abstraction-based verification scenario. For each of these points, we provide proof-of-concept experimental evidence of the method's potential.

后者对应于经典的基于抽象的验证场景。对于每一点，我们都提供了该方法的潜力的概念证明实验证据。

Our contribution is thus the following:

- We propose to explore the framework of abstraction by clustering based on experimental data. For feed-forward NN with ReLU, we provide error bounds.
- We show that the abstraction is also usable for compression. The reduction rate grows with the size of the original network, while the abstracted NN is able to achieve almost the same accuracy as the original network.
- We demonstrate the verification potential of the approach: (i) In some cases where the large NN was not analyzable (within time-out), we verified the abstraction using existing tools; for other NN, we could reduce verification times from thousands to hundreds of seconds. (ii) We show how to transfer a proof of robustness by a verification tool DeepPoly [Sin+19a] on the abstract NN to a proof on the original network, whenever the clusters do not have too large radii.

- 我们建议通过基于实验数据的聚类来探索抽象框架。 对于带有 ReLU 的前馈神经网络，我们提供了误差界限。

- 我们表明抽象也可用于压缩。 减少率随着原始网络的大小而增长，而抽象的 NN 能够达到与原始网络几乎相同的精度。

-我们展示了该方法的验证潜力：(i) 在某些大型 NN 不可分析的情况下（在超时内），我们使用现有工具验证了抽象； 对于其他神经网络，我们可以将验证时间从数千秒减少到数百秒。 (ii) 我们展示了如何通过验证工具 DeepPoly [Sin+19a] 将抽象神经网络上的稳健性证明转移到原始网络上的证明，只要集群没有太大的半径。

Related work In contrast to compression techniques, our abstraction provides a mapping between original neurons and abstract neurons, which allows for transferring the claims of the abstract NN to the original one, and thus its verification.

与压缩技术相比，我们的抽象提供了原始神经元和抽象神经元之间的映射，这允许将抽象神经网络的声明转移到原始神经网络，从而对其进行验证。

The very recent work [YGK19] suggests an abstraction, which is based solely on the sign of the effect of increasing a value in a neuron. While we can demonstrate our technique on e.g. 784 dimension input (MNIST) and work with general networks, [YGK19] is demonstrated only on the Acas Xu [JKO18] networks which have 5 dimensional input; our approach handles thousands of nodes while the benchmark used in [YGK19] is of size 300. Besides, we support both classification and regression networks. Finally, our approach is not affected by the number of outputs, whereas the [YGK19] grows exponentially with respect to number of outputs.

最近的工作 [YGK19] 提出了一种抽象，它完全基于增加神经元中值的影响的符号。 虽然我们可以展示我们的技术，例如 784 维输入 (MNIST) 和通用网络，[YGK19] 仅在具有 5 维输入的 Acas Xu [JKO18] 网络上演示； 我们的方法处理数千个节点，而 [YGK19] 中使用的基准测试大小为 300。此外，我们支持分类和回归网络。 最后，我们的方法不受输出数量的影响，而 [YGK19] 相对于输出数量呈指数增长。

[PA19] produces so called Interval Neural Networks containing intervals instead of single weights and performs abstraction by merging these intervals. However, they do not provide a heuristic for picking the intervals to merge, but pick randomly. Further, the results are demonstrated only on the low-dimensional Acas Xu networks.

[PA19] 产生所谓的区间神经网络，它包含区间而不是单个权重，并通过合并这些区间来执行抽象。 但是，它们不提供用于选择要合并的间隔的启发式方法，而是随机选择。 此外，结果仅在低维 Acas Xu 网络上得到证明。

Further, [SB15] computes a similarity measure between incoming weights and then starts merging the most similar ones. It also features an analysis of how many neurons to remove in order to not lose too much accuracy. However, it does not use clustering on the semantic values of the activations, but only on the syntactic values of the incoming weights, which is a very local and thus less powerful criterion. Similarly, [ZYZ18] clusters based on the incoming weights only and does not bound the error. [HMD16] clusters weights in contrast to our activation values) using the k-means clustering algorithm. However, the focus is on weight-sharing and reducing memory consumption, treating neither the abstraction mapping nor verification.

此外，[SB15] 计算传入权重之间的相似性度量，然后开始合并最相似的权重。 它还分析了要移除多少神经元，以免失去太多准确性。 然而，它没有在激活的语义值上使用聚类，而只在传入权重的句法值上使用聚类，这是一个非常局部的，因此不太强大的标准。 类似地，[ZYZ18] 仅基于传入的权重进行聚类，不限制误差。 [HMD16] 使用 k-means 聚类算法聚类权重与我们的激活值形成对比。 然而，重点是权重共享和减少内存消耗，既不处理抽象映射也不处理验证。

Finally, abstracting neural networks for verification purposes was first proposed by [PT10], transforming the networks into Boolean constraints.

最后，[PT10]首先提出了抽象用于验证的神经网络，并将网络转化为布尔约束。

2 Preliminaries

We consider simple feedforward neural networks, denoted by D, consisting of one input layer, one output layer and one or more hidden layers. The layers are numbered $1, 2, \ldots, L$ with 1 being the *input layer*, L being the *output layer* and $2, \ldots, L-1$ being the *hidden layers*. Layer ℓ contains n_ℓ *neurons*. A neuron is a computation unit which takes an input $h \in \mathbb{R}$, applies an *activation function* $\phi : \mathbb{R} \to \mathbb{R}$ on it and gives as output $z = \phi(h)$. Common activation functions include tanh, sigmoid or ReLU [MHN13], however we choose to focus on ReLU for the sake of simplicity, where $\mathrm{ReLU}(x)$ is defined as $\max(0, x)$. Neurons of one layer are connected to neurons of the previous and/or next layers by means of weighted connections. Associated with every layer ℓ that is not an output layer is a *weight matrix* $W^{(\ell)} = (w_{i,j}^{(\ell)}) \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$ where $w_{i,j}^{(\ell)}$ gives the weights of the connections to the i^{th} neuron in layer $\ell+1$ from the j^{th} neuron in layer ℓ. We use the notation $W_{i,*}^{(\ell)} = [w_{i,1}^{(\ell)}, \ldots, w_{i,n_\ell}^{(\ell)}]$ to denote the incoming weights of neuron i in layer $\ell+1$ and $W_{*,j}^{(\ell)} = [w_{1,j}^{(\ell)}, \ldots, w_{n_{\ell+1},j}^{(\ell)}]^\intercal$ to denote the outgoing weights of neuron j in layer ℓ. Note that $W_{i,*}^{(\ell)}$ and $W_{*,j}^{(\ell)}$ correspond to the i^{th} row and j^{th} column of $W^{(\ell)}$ respectively. The input and output of a neuron i in layer ℓ is denoted by $h_i^{(\ell)}$ and $z_i^{(\ell)}$ respectively. We call $\mathbf{h}^\ell = [h_1^{(\ell)}, \ldots, h_{n_\ell}^{(\ell)}]^\intercal$ the vector of *pre-activations* of layer ℓ and $\mathbf{z}^\ell = [z_1^{(\ell)}, \ldots, z_{n_\ell}^{(\ell)}]^\intercal$ the vector of *activations* of layer ℓ, where $z_i^{(\ell)} = \phi^{(\ell)}(h_i^{(\ell)})$. A vector $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$ called *bias* is also associated with all hidden layers ℓ. 给出了神经网络的定义

In a feedforward neural network, information flows strictly in one direction: from layer ℓ_m to layer ℓ_n where $\ell_m < \ell_n$. For an n_1-dimensional input $\boldsymbol{x} \in \mathcal{X}$ from some input space $\mathcal{X} \subseteq \mathbb{R}^{n_1}$, the output $\boldsymbol{y} \in \mathbb{R}^{n_L}$ of the neural network D, also written as $\boldsymbol{y} = D(\boldsymbol{x})$ is iteratively computed as follows:

$$
\begin{aligned}
\mathbf{h}^{(0)} &= \boldsymbol{x} \\
\mathbf{h}^{(\ell+1)} &= W^{(\ell)} \mathbf{z}^{(\ell)} + \mathbf{b}^{(\ell+1)} \qquad (1) \\
\mathbf{z}^{(\ell+1)} &= \phi(\mathbf{h}^{(\ell+1)}) \qquad (2) \\
\boldsymbol{y} &= \mathbf{z}^{(L)}
\end{aligned}
$$

where $\phi(x)$ is the column vector obtained on applying ϕ component-wise to $\boldsymbol{x}$. We sometimes write $\mathbf{z}^{(\ell)}(\boldsymbol{x})$ to denote the output of layer ℓ when $\boldsymbol{x}$ is given as input to the network.

We define a *local robustness* query to be a tuple $Q = (D, \boldsymbol{x}, \boldsymbol{\delta})$ for some network D, input $\boldsymbol{x}$ and perturbation $\boldsymbol{\delta} \in \mathbb{R}^{|\boldsymbol{x}|}$ and call D to be robust with respect to Q if $\forall \boldsymbol{x'} \in [\boldsymbol{x} - \boldsymbol{\delta}, \boldsymbol{x} + \boldsymbol{\delta}] : D(\boldsymbol{x'}) = D(\boldsymbol{x})$. In this paper, we only deal with local robustness. 这里定义了扰动，并且声明本篇文章只研究局部鲁棒性

3 Abstraction

In classic abstraction, states that are similar with respect to a property of interest are merged for analysis. In contrast, for NN, it is not immediately clear which

与传统验证有区别，神经元没有内部结构，因为抽象的本质是丢弃原始系统中的一些内容，而缩小待验证的状态空间。在此处，识别和丢弃哪一部分神经元成为了一个挑战。我们的建议是合并在某些输入集X上计算相似的神经元。

在经典抽象中，在感兴趣的属性方面相似的状态被合并以进行分析。相比之下，对于 NN，并不清楚要合并哪些神经元以及相似性意味着什么。事实上，神经元实际上并不是系统的状态/配置； 因此，与具有变量值的状态相反，神经元没有内部结构。 因此，识别和丢弃不相关的信息（结构的一部分）变得更具挑战性。我们建议合并在某些输入集 X 上计算相似函数的神经元，即，对于网络的每个输入 x ∈ X，它们计算 ε-close 值。 我们将此称为 I/O 相似性。 此外，我们选择仅在同一层内合并神经元，以保持分析和实现简单明了。

neurons to merge and what similarity means. Indeed, neurons are not actually states/configurations of the system; as such, neurons, as opposed to states with values of variables, do not have inner structure. Consequently, identifying and dropping irrelevant information (part of the structure) becomes more challenging. We propose to merge neurons which compute a similar function *on some set X of inputs*, i.e., for each input $x \in X$ to the network, they compute ε-close values. We refer to this as I/O-similarity. Further, we choose to merge neurons only within the same layer to keep the analysis and implementation straightforward.

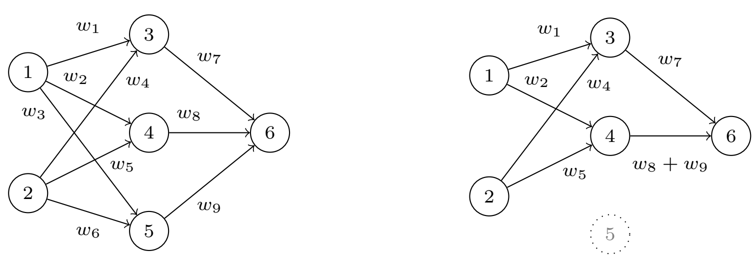
In Section 3.1, we show a straightforward way to merge neurons in a way that is sensible if they are I/O-similar. In Section 3.2, we give a heuristic for partitioning neurons into classes according to their I/O-similarity. While this abstraction idea is not limited to verification of robustness, it preserves the robustness of the original network particularly well, as seen in the experiments in Section 5.

在第 3.1 节中，我们展示了一种直接的方法，以一种明智的方式合并神经元，如果它们是 I/O 相似的。 在 3.2 节中，我们给出了一种根据 I/O 相似性将神经元划分为类的启发式方法。 虽然这种抽象思想不限于验证鲁棒性，但它特别好地保留了原始网络的鲁棒性，如第 5 节中的实验所示。

3.1 Merging I/O-similar neurons

I/O-similar neurons can be merged easily without changing the behaviour of the NN too much. First, we explain the procedure on an example.

与I/o相似的神经元可以很容易地合并，而不会过多地改变神经网络的行为。首先，我们用一个示例来解释这个过程。



(a) Original network (b) Network after merging neurons 4 and 5

Fig. 1: Before and after merge: neuron 4 is chosen as a representative of both 4 and 5. On merging, the incoming weights of neuron 5 are deleted and its outgoing weight is added to the outgoing weight of neuron 4.

合并前后：选择神经元4为4和5的代表。合并后，删除神经元5的输入权重，并将其输出权重添加到神经元4的输出权重中。

Example 1. Consider the network shown in Figure 1a. The network contains 2 input neurons and 4 ReLU neurons. For simplicity, we skip the bias term in this example network. Hence, the activations of the neurons in the middle layer are given as follows: $z_3 = ReLU(w_1 z_1 + w_4 z_2)$, $z_4 = ReLU(w_2 z_1 + w_5 z_2)$, $z_5 = ReLU(w_3 z_1 + w_6 z_2)$; and the output of neuron 6 is $z_6 = ReLU(w_7 z_3 + w_8 z_4 + w_9 z_5)$. Suppose that for all inputs in the dataset, the activations of neurons 4 and 5 are 'very' close, denoted by $z_4 \approx z_5$. Then, $z_6 = ReLU(w_7 z_3 + w_8 z_4 + w_9 z_5)$.

Since neurons 4 and 5 behave similarly, we abstract the network by merging the two neurons as shown in Figure 1b. Here, neuron 4 is chosen as a representative of the "cluster" containing neurons 4 and 5, and the outgoing weight of the

Algorithm 1 Abstract network D with given clustering K_L

procedure ABSTRACT(D, X, K_L)
  $\tilde{D} \leftarrow D$
  for $\ell \leftarrow 2, \ldots, L-1$ **do**
    $A \leftarrow \{\mathbf{a}_i^{(\ell)} \mid \mathbf{a}_i^{(\ell)} = [\tilde{z}_i^{(\ell)}(x_1), \ldots, \tilde{z}_i^{(\ell)}(x_N)]$ where $x_i \in X\}$
    $\mathcal{C} \leftarrow \text{KMEANS}(A, K_L(\ell))$
    for $C \in \mathcal{C}$ **do**
      $\tilde{W}^{(\ell)}_{*,rep(C)} \leftarrow \sum_{i \in C} W^{(\ell)}_{*,i}$
    delete $C \setminus \{rep(C)\}$ from $\tilde{D}$
  return $\tilde{D}$

representative is set to the sum of outgoing weights of all the neurons in the cluster. Note that the incoming weights of the representative do not change. In the abstracted network, the activations of the neurons in the middle layer are now given by $\tilde{z}_3 = ReLU(w_1\tilde{z}_1 + w_4\tilde{z}_2) = z_3$ and $\tilde{z}_4 = ReLU(w_2\tilde{z}_1 + w_5\tilde{z}_2) = z_4$ with neuron 5 being removed. The output of neuron 6 is therefore $\tilde{z}_6 = ReLU(w_7\tilde{z}_3 + (w_8 + w_9)\tilde{z}_4) = ReLU(w_7z_3 + (w_8 + w_9)z_4) = ReLU(w_7z_3 + w_8z_4 + w_9z_4) \approx z_6$, which illustrates that merging preserves the behaviour of the network. 这说明了合并保留了网络的行为。

Formally, the process of merging two neurons p and q belonging to the same layer ℓ works as follows. We assume, without loss of generality, that p is retained as the representative. First, the abstract network $\tilde{D}$ is set to the original network D. Next, $\tilde{W}^{(\ell-1)}$ is set to $W^{(\ell-1)}$ with the q^{th} row deleted. Further, we set the outgoing weights of the representative p to the sum of outgoing weights of p and q, $\tilde{W}^{(\ell)}_{*,p} = W^{(\ell)}_{*,p} + W^{(\ell)}_{*,q}$. This procedure is naturally extendable to merging multiple I/O-similar neurons. It can be applied repeatedly until all desired neurons are merged. For the interested reader, the correctness proof and further technical details are made available in Appendix A.1.

形式上，合并属于同一层 ? 的两个神经元 p 和 q 的过程如下。 我们假设，不失一般性，保留 p 作为代表。 首先，将抽象网络~D 设置为原始网络D。接下来，将~W(?-1) 设置为W(?-1)，删除第q 行。 此外，我们将代表 p 的输出权重设置为 p 和 q 的输出权重之和，~W?(,p ?) = W?(,p ?) +W?(,q ?)。 这个过程自然可以扩展到合并多个 I/O 类似的神经元。 它可以重复应用，直到合并所有所需的神经元。 对于感兴趣的读者，正确性证明和进一步的技术细节在附录 A.1 中提供。

完整性检查

Proposition 1 (Sanity Check). *If for neurons p and q, for all considered inputs $x \in X$ to the network D, $z_p = z_q$, then the network $\tilde{D}$ produced as described above, in which p and q are merged by removing q and letting p serve as their representative, and by setting $\tilde{W}^{(\ell)}_{*,p} = W^{(\ell)}_{*,p} + W^{(\ell)}_{*,q}$, will have the same output as D on all inputs $x \in X$. In other words, $\forall x \in X \; D(x) = \tilde{D}(x)$.*

如果对于神经元p和q，对于所有考虑的输入x∈X到网络D，zp=zq，那么按照上述方法产生的网络˜D，其中p和q通过删除q，让p作为它们的代表，并通过设置˜W?(, p?)=W?(, p?)+W?(, q?)，将在所有输入x∈X上具有与D相同的输出。换句话说，?x∈XD(x)=˜D(x)。

3.2 Clustering-based Abstraction

In the previous section, we saw that multiple I/O-similar neurons can be merged to obtain an abstract network behaving similar to the original network. However, the quality of the abstraction depends on the choice of neurons used in the merging. Moreover, it might be beneficial to have multiple groups of neurons that are merged separately. While multiple strategies can be used to identify such groups, in this section, we illustrate this on one of them — the unsupervised learning approach of *k-means clustering* [Bis06], as a proof-of-concept.

在前一节中，我们看到可以合并多个i/o相似的神经元，获得一个行为与原始网络相似的抽象网络。然而，抽象的质量取决于在合并中使用的神经元的选择。此外，有多组神经元单独合并可能是有益的。虽然可以使用多种策略来识别这些组，但在本节中，我们将用其中一种策略来说明这一点——k-means聚类的无监督学习方法[Bis06]，作为一种概念证明。

Algorithm 1 describes how the approach works in general. It takes as input the original (trained) network D, an input set X and a function K_L, which for

算法1描述了该方法的一般工作原理。它将原始的（训练过的）网络D、一个输入集X和一个函数KL作为输入，

Algorithm 2 Algorithm to identify the clusters

procedure IDENTIFY-CLUSTERS(D, X, α)
  $\tilde{D} \leftarrow D$
  for $\ell \leftarrow 2, ..., L-1$ **do** ▷ Loops through the layers
    if $accuracy(\tilde{D}) > \alpha$ **then**
      $K_L(\ell) \leftarrow$ BINARYSEARCH$(\tilde{D}, \alpha, \ell)$ ▷ Finds optimal number of clusters
      $\tilde{D} \leftarrow$ ABSTRACT$(\tilde{D}, X, K_L)$
  return K_L

算法 1 描述了该方法的一般工作原理。 它将原始（经过训练的）网络 D、一个输入集 X 和一个函数 KL 作为输入，KL 函数为每一层提供了该层中要识别的簇数。

each layer gives the number of clusters to be identified in that layer. Each $x \in X$ is input into $\tilde{D}$ and for each neuron i in layer ℓ, an $|X|$-dimensional vector of observed activations $\mathbf{a}_i^{(\ell)} = [z_i^{(\ell)}(x_1), \ldots, z_i^{(\ell)}(x_{|X|})]$ is constructed. These vectors of activations, one for each neuron, are collected in the set A. We can now use the k-means algorithm on the set A to identify $K_L(\ell)$ clusters. Intuitively, k-means aims to split the set A into $K_L(\ell)$ clusters such that the pairwise squared deviations of points in the same cluster is minimized. Once a layer is clustered, the neurons of each cluster are merged and the neuron closest to the centroid of the respective cluster, denoted by $rep(C)$ in the pseudocode, is picked as the cluster representative. As described in Section 3.1, the outgoing connections of all the neurons in a cluster are added to the representative neuron of the cluster and all neurons except the representative are deleted.

一旦一层聚集，每个簇的神经元被合并，并选择最接近各自簇质心的神经元，用伪代码中的rep(C)表示，作为簇的代表。如3.1节所述，一个簇中所有神经元的输出连接被添加到簇的代表性神经元中，除代表外的代表性神经元的所有连接都被删除。

While Algorithm 1 describes the clustering procedure, it is still a challenge to find the right K_L. In Algorithm 2, we present one heuristic to identify a good set of parameters for the clustering. It is based on the intuition that merging neurons closer to the output layer impacts the network accuracy the least, as the error due to merging is not multiplied and propagated through multiple layers. The overarching idea is to search for the best k-means parameter, $K_L(\ell)$, for each layer ℓ starting from the first hidden layer to the last hidden layer, while making sure that the merging with the said parameter (K_L) does not drop the accuracy of the network beyond a threshold α.

虽然算法1描述了聚类过程，但找到正确的KL仍然是一个挑战。在算法2中，我们提出了一个启发式方法来识别一组很好的聚类参数。它是基于直觉，合并靠近输出层的神经元对网络精度的影响最小，因为合并导致的误差不相乘并通过多层传播。总体思想是为从第一隐藏层到最后一隐藏层的每一层?搜索最佳的K-means参数KL（?），同时确保与所述参数(KL)的合并不会降低网络的精度超过阈值α。

The algorithm takes a trained network D as input along with an input set X and a parameter α, the lower bound on the accuracy of the abstract network. The first hidden layer ($\ell = 2$) is picked first and k-means clustering is attempted on it. The parameter $K_L(\ell)$ is discovered using the BINARYSEARCH procedure which searches for the lowest k such that the accuracy of the network abstracted with this parameter is the highest. We make a reasonable assumption here that a higher degree of clustering (i.e. a small k) leads to a higher drop in accuracy. Note that this might cause the BINARYSEARCH procedure to work on a monotone space and we might not exactly get the optimal. However, in our experiments, the binary search turned out to be a sufficiently good alternative to brute-force search. The algorithm ensures that merging the clusters as prescribed by K_L does not drop the accuracy of the abstracted network below α.[3] This process is

该算法将训练好的网络D作为输入，以及输入集X和参数α，作为抽象网络精度的下界。首先选择第一个隐藏层（?=2），并对其尝试k-means聚类。参数KL（?）是使用二元搜索过程发现的，该过程搜索最低的k，以便使用该参数提取的网络的精度最高。

我们在这里做了一个合理的假设，即更高程度的聚类(即一个小的k)会导致更高的精度下降。注意，这可能会导致二进制搜索过程在单调空间上工作，而我们可能不能完全得到最优的。然而，在我们的实验中，二进制搜索被证明是一个足够好的替代暴力搜索。该算法确保按照KL的规定合并集群不会将抽象网络的精度降到α以下。现在在 ~ D上从下一个隐藏层开始。最后，返回KL，准备与算法1一起使用。

[3] Naturally, the parameter α has to be less than or equal to the accuracy of D

now repeated on $\tilde{D}$ starting with the next hidden layer. Finally, K_L is returned, ready to be used with Algorithm 1.

Now we present two results which bound the error induced in the network due to abstraction. The first theorem applies to the case where we have clustered groups of I/O-similar neurons in each layer for the set X of network inputs.

现在我们给出了两个结果，它们约束了由于抽象而在网络中引起的误差。第一个定理适用于我们在网络输入的集合X的每一层中都聚集了I/O相似神经元组的情况。

Let for each neuron i, $\mathbf{a}_i = [z_i(x_1), \ldots, z_i(x_N)]$ where $x_j \in X$, and let $\tilde{D} =$ Abstract(D, X, K_L) for some given K_L. Define $\boldsymbol{\epsilon}^{(\ell)}$, the maximal distance of a neuron from the respective cluster representative, as

神经元与各自簇代表的最大距离，如

$$\boldsymbol{\epsilon}^{(\ell)} = [\epsilon_1^{(\ell)}, \ldots, \epsilon_{n_\ell}^{(\ell)}]^\intercal \qquad \text{where} \qquad \epsilon_i^{(\ell)} = \|\mathbf{a}_i - \mathbf{a}_{r_{C_i}}\| \tag{3}$$

where $\|\cdot\|$ denotes the Euclidean norm operator, C_i denotes the cluster containing i and r_{C_i} denotes the representative of cluster C_i. Further, define the absolute error due to abstraction in layer ℓ as $\boldsymbol{err}^{(\ell)} = \tilde{\mathbf{z}}^{(\ell)} - \mathbf{z}^{(\ell)}$.

Theorem 1 (Clustering-induced error). *If the accumulated absolute error in the activations of layer ℓ is given by $\boldsymbol{err}^{(\ell)}$ and $\boldsymbol{\epsilon}^{(\ell+1)}$ denotes the the maximal distance of each neuron from their cluster representative (as defined in Eqn. 3) of layer $\ell+1$, then the absolute error $\boldsymbol{err}^{(\ell+1)}$ for all inputs $\boldsymbol{x} \in X$ can be bounded by*

$$|\boldsymbol{err}^{(\ell+1)}| \leq |W^{(\ell)}\boldsymbol{err}^{(\ell)}| + \boldsymbol{\epsilon}^{(\ell+1)}$$

and hence, the absolute error in the network output is given by $\boldsymbol{err}^{(L)}$.

The second result considers the local robustness setting where we are interested in the output of the abstracted network when the input $\boldsymbol{x} \in X$ is perturbed by $\delta \in \mathbb{R}^{|x|}$.

Theorem 2. *If the inputs $\boldsymbol{x} \in X$ to the abstract network $\tilde{D}$ are perturbed by $\boldsymbol{\delta} \in \mathbb{R}^{|\boldsymbol{x}|}$, then the absolute error in the network output due to both abstraction and perturbation denoted by $\boldsymbol{err}_{total}$ is bounded for every $\boldsymbol{x} \in X$ and is given by*

$$|\boldsymbol{err}_{total}| \leq |\tilde{W}^{(L)} \ldots \tilde{W}^{(1)}\boldsymbol{\delta}| + |\boldsymbol{err}^{(L)}|$$

where $\tilde{W}^{(\ell)}$ is the matrix of weights from layer ℓ to $\ell+1$ in $\tilde{D}$, L is the number of layers in $\tilde{D}$ and $\boldsymbol{err}^{(L)}$ is the accumulated error due to abstraction as given by Theorem 1.

In other words, these theorems allow us to compute the absolute error produced due to the abstraction alone; or due to both (i) abstraction and (ii) perturbation of input. Theorem 2 gives us a direct (but naïve) procedure to perform local robustness verification by checking if there exists an output neuron i with a lower bound $(\tilde{D}_i(x) - (E_{total})_i)$ greater than the upper bound $(\tilde{D}_j(x) + (E_{total})_j)$ of all other output neurons j. The proofs of both theorems can be found in Appendix A.2.

从抽象神经网络到原始神经网络的提升保证

4 Lifting guarantees from abstract NN to original NN

In the previous section, we discussed how a large neural network could be abstracted and how the absolute error on the output could be calculated and even used for robustness verification. However, the error bounds presented in Theorem 2 might be too coarse to give any meaningful guarantees. In this section, we present a proof-of-concept approach for lifting verification results from the abstracted network to the original network. While in general the lifting depends on the verification algorithm, as a demonstrative example, we show how to perform the lifting when using the verification algorithm DeepPoly [Sin+19a] and also how it can be used in conjunction with our abstraction technique to give robustness guarantees on the original network.

在上一节中，我们讨论了如何抽象大型神经网络，以及如何计算输出的绝对误差，甚至将其用于稳健性验证。 然而，定理 2 中给出的误差界限可能太粗糙而无法提供任何有意义的保证。 在本节中，我们提出了一种概念验证方法，用于将验证结果从抽象网络提升到原始网络。 虽然一般来说提升取决于验证算法，但作为一个演示示例，我们展示了如何在使用验证算法 DeepPoly [Sin+19a] 时执行提升，以及如何将其与我们的抽象技术结合使用以提供鲁棒性 保证在原始网络上。

We now give a quick summary of DeepPoly. Assume that we need to verify that the network D labels all inputs in the δ-neighborhood of a given input $x \in X$ to the same class; in other words, check if D is locally robust for the robustness query $(D, \boldsymbol{x}, \delta)$. DeepPoly functions by propagating the interval $[\boldsymbol{x} - \boldsymbol{\delta}, \boldsymbol{x} + \boldsymbol{\delta}]$ through the network with the help of abstract interpretation, producing over-approximations (a lower and an upper bound) of activations of each neuron. The robustness query is then answered by checking if the lower bound of the neuron representing one of the labels is greater than the upper bounds of all other neurons. We refer the interested reader to [Sin+19a, Section 2] for an overview of DeepPoly. Note that the algorithm is sound but not complete.

我们现在快速总结一下 DeepPoly。 假设我们需要验证网络 D 将给定输入 x ∈ X 的 δ 邻域中的所有输入标记为同一类； 换句话说，检查 D 对于鲁棒性查询 (D, x, δ) 是否局部鲁棒。 DeepPoly 通过在抽象解释的帮助下通过网络传播区间 [x ? δ , x + δ] 来运行，从而产生每个神经元激活的过度近似（下限和上限）。 然后通过检查代表标签之一的神经元的下限是否大于所有其他神经元的上限来回答稳健性查询。 我们建议感兴趣的读者参考 [Sin+19a, 第 2 节] 以了解 DeepPoly 的概述。 请注意，该算法是合理的，但并不完整。

If DeepPoly returns the bounds $\tilde{l}$ and $\tilde{u}$ for the abstract network $\tilde{D}$, the following theorem allows us to compute $[\hat{l}, \hat{u}]$ such that $[\hat{l}, \hat{u}] \supseteq [l, u]$, where $[l, u]$ would have been the bounds returned by DeepPoly on the original network D.

如果 DeepPoly 返回抽象网络 ˜D 的边界 ˜l 和 ˜u，以下定理允许我们计算 [ˆl, uˆ] 使得 [ˆl, uˆ] ? [l, u]，其中 [l, u] 将 是 DeepPoly 在原始网络 D 上返回的边界。

Theorem 3 (Lifting guarantees). *Consider the abstraction $\tilde{D}$ obtained by applying Algorithm 1 on a ReLU feedforward network D. Let $\tilde{l}^{(\ell)}$ and $\tilde{u}^{(\ell)}$ denote the lower bound and upper bound vectors returned by DeepPoly for the layer ℓ, and let $\tilde{W}_+^{(\ell)} = \max(0, \tilde{W}^{(\ell)})$ and $\tilde{W}_-^{(\ell)} = \min(\tilde{W}^{(\ell)}, 0)$ denote the +ve and -ve entries respectively of its ℓ^{th} layer weight matrix. Let $\boldsymbol{\epsilon}^{(\ell)}$ denote the vector of maximal distances of neurons from their cluster representatives (as defined in Equation 3), and let $\boldsymbol{x}$ be the input we are trying to verify for a perturbation $[-\boldsymbol{\delta}, \boldsymbol{\delta}]$. Then for all layers $\ell < L$, we can compute*

考虑通过在 ReLU 前馈网络 D 上应用算法 1 获得的抽象 ˜D。让 ˜l (?) 和 ˜u(?) 表示 DeepPoly 为层 ? 返回的下界和上界向量，让 ˜W (?) + = max(0, ˜W(?)) 和 ˜W(?) ? = min(˜W(?), 0) 分别表示其第?层权重矩阵的+ve和-ve条目。 让 ?(?) 表示神经元与其集群代表的最大距离向量（如等式 3 中所定义），并让 x 是我们试图验证扰动 [?δ , δ] 的输入。 然后对于所有层? < L，我们可以计算

$$\hat{u}^{(\ell)} = \max\left(0, \begin{array}{l} \tilde{W}_+^{(\ell-1)}(\hat{u}^{(\ell-1)} + \boldsymbol{\epsilon}^{(\ell-1)}) \\ + \tilde{W}_-^{(\ell-1)}(\hat{l}^{(\ell-1)} - \boldsymbol{\epsilon}^{(\ell-1)}) \\ + \tilde{b}^{(\ell)} \end{array}\right) \quad \hat{l}^{(\ell)} = \max\left(0, \begin{array}{l} \tilde{W}_+^{(\ell-1)}(\hat{l}^{(\ell-1)} - \boldsymbol{\epsilon}^{(\ell-1)}) \\ + \tilde{W}_-^{(\ell-1)}(\hat{u}^{(\ell-1)} + \boldsymbol{\epsilon}^{(\ell-1)}) \\ + \tilde{b}^{(\ell)} \end{array}\right)$$

where $\hat{u}^{(1)} = \tilde{u}^{(1)} = u^{(1)} = \boldsymbol{x} + \boldsymbol{\delta}$ and $\hat{l}^{(1)} = \tilde{l}^{(1)} = l^{(1)} = \boldsymbol{x} - \boldsymbol{\delta}$ such that

$$[\hat{l}, \hat{u}] \supseteq [l, u]$$

where $[l, u]$ is the bound computed by DeepPoly on the original network.

For output layer $\ell = L$, the application of the $\max(0, \cdot)$-function is omitted, the rest remains the same.

In other words, this theorem allows us to compute an over-approximation of the bounds computed by DeepPoly on the original network D by using only the abstract network, thereby allowing a local robustness proof to be lifted from the abstraction to the original network. Note that while this procedure is sound, it is not complete since the bounds computed by Theorem 3 might still be too coarse. An empirical discussion is presented in Section 5, an example of the proof lifting can be seen in Appendix A.5, and the proof is given in Appendix A.3.

换句话说，这个定理允许我们通过只使用抽象网络来计算由DeepPoly在原始网络D上计算的边界的过近似，从而允许将局部鲁棒性证明从抽象提升到原始网络。注意，虽然这个过程是合理的，但它是不完整的，因为由定理3计算的边界可能仍然太粗糙。第5节进行了经验讨论，附录中有一个证明提升的例子A.5，证明情况见附录A.3.

5 Experiments

We now analyze the potential of our abstraction. In particular, in Section 5.1, we look at how much we can abstract while still guaranteeing a high test accuracy for the abstracted network. Moreover, we present verification results of abstracted network, suggesting a use case where it replaces the original network. In Section 5.2, we additionally consider lifting of the verification proof from the abstracted network to the original network.

我们现在分析我们的抽象的潜力。特别是，在第5.1节中，我们将研究我们可以抽象多少，同时仍然保证抽象网络的高测试精度。此外，我们给出了抽象网络的验证结果，提出了一个用它替换原始网络的用例。在第5.2节中，我们还考虑将验证证明从抽象的网络提升到原始的网络。

We ran experiments with multiple neural network architectures on the popular MNIST dataset [LeC98]. We refer to our network architectures by the shorthand $L \times n$, for example "6×100", to denote a network with L fully-connected feedforward hidden layers with n neurons each, along with a separate input and output layers whose sizes depend on the dataset — 784 neurons in the input layer and 10 in the output layer in the case of MNIST. Interested readers may find details about the implementation in Appendix A.6.

我们在流行的MNIST数据集[LeC98]上使用多种神经网络架构进行了实验。我们指我们的网络架构简称L × n，例如“ 6 × 100 ”，表示一个网络L全连接前馈隐藏层与n个神经元，以及一个单独的输入和输出层的大小依赖于数据集-输入层784神经元和输出层的10MNIST。有兴趣的读者可以在附录A.6中找到有关实现的详细信息。

Remark on Acas Xu We do not run experiments on the standard NN verification case study Acas Xu [JKO18]. The Acas Xu networks are very compact, containing only 6 layers with 50 neurons each. The training/test data for these networks are not easily available, which makes it difficult to run our data-dependent abstraction algorithm. Further, the network architecture cannot be scaled up to observe the benefits of abstraction, which, we conjecture, become evident only for large networks possibly containing redundancies. Moreover, the specifications that are commonly verified on Acas Xu are not easily encodable in DeepPoly.

我们在流行的MNIST数据集[LeC98]上使用多种神经网络架构进行了实验。我们指我们的网络架构简称L × n，例如“ 6 × 100 ”，表示一个网络L全连接前馈隐藏层与n个神经元，以及一个单独的输入和输出层的大小依赖于数据集-输入层784神经元和输出层的10MNIST。有兴趣的读者可以在附录A.6中找到有关实现的详细信息。

5.1 Abstraction results

First, we generated various NN architectures by scaling up the number of neurons per layer as well as the number of layers themselves and trained them on MNIST. More information on the training process is available in Appendix A.4. Then, we executed our clustering-based abstraction algorithm (Algorithm 1) on each trained network allowing for a drop in accuracy on a test dataset of at most 1%.

首先，我们通过扩大每层神经元的数量以及层本身的数量来生成各种神经网络结构，并在MNIST上训练它们。有关训练过程的更多信息见附录A.4。然后，我们在每个训练过的网络上执行基于聚类的抽象算法（算法1），允许测试数据集的准确率最多下降1%

Size of the abstraction Table 1 gives some information about the quality of the abstraction - the extent to which we can abstract while sacrificing accuracy of at most 1%. We can see that increasing the width of a layer (number of neurons) while keeping the depth of the network fixed increases the number of neurons

Table 1: Reduction rate of abstracted neural networks with different architectures along with the drop in accuracy (measured on an independent test set). In the top half, the number of layers (depth) is varied and in the bottom half, the number of neurons per layer (width) is increased. This table shows that the clustering-based abstraction works better with wider networks.

Network Arch.	**Accuracy Drop (%)**	**Reduction Rate (%)**
3×100	0.40	15.5
4×100	0.41	15.5
5×100	0.21	21.2
6×100	0.10	13.3
6×50	0.10	5.7
6×100	0.10	13.3
6×200	0.10	30.2
6×300	0.20	39.9
6×1000	0.01	61.7

抽象的大小 表 1 给出了一些关于抽象质量的信息——我们可以抽象的程度，同时牺牲最多 1% 的准确度。我们可以看到，在保持网络深度固定的同时增加层的宽度（神经元数量）会增加可以合并的神经元数量，即减少率增加。 我们推测，模拟原始网络的行为所需的每层神经元数量最少。 另一方面，有趣的是，如果在保持宽度不变的情况下增加网络的深度，减少率似乎徘徊在 15-20% 左右。

that can be merged, i.e. the reduction rate increases. We conjecture that there is a minimum number of neurons per layer that are needed to simulate the behavior of the original network. On the other hand, interestingly, if the depth of the network is increased while keeping the width fixed, the reduction rate seems to hover around 15-20%.

Figure 2 demonstrates the potential of the clustering-based abstraction procedure in compressing the network. Here, the abstraction is performed layer after layer from layer 1 to layer 6. We cluster as much as possible permitting the test accuracy of the network to drop by at most 1%. Unsurprisingly, we get more reduction in the later (closer to output) layers compared to the initial. We conjecture that this happens as the most necessary information is already processed and computed early on, and the later layers transmit low dimensional information. Interestingly, one may observe that in layers 4, 5 and 6, all network architectures ranging from 50 to 500 neurons/layer can be compressed to an almost equal size around 30 nodes/layer.

图2展示了基于聚类的抽象过程在压缩网络方面的潜力。这里，从一层一层到第六层执行抽象。我们尽可能多地进行集群，允许网络的测试精度最多下降1%。不出所料，与初始层相比，我们在后期（更接近输出）层中得到了更多的减少。我们推测，这是因为最必要的信息已经被早期处理和计算，而后面的层传输低维信息。有趣的是，人们可以观察到，在第4、5和第6层，所有从50到500个神经元/层的网络结构都可以被压缩到几乎相同大小的30个节点/层。

Verifying the abstraction As mentioned in the Section 1, we found that the abstraction, considered as a standalone network, is faster to verify than the original network. This opens up the possibility of verifying the abstraction and replacing the original network with it, in real-use scenarios. In Figure 3, we show the time it takes to verify the abstract network using DeepPoly against the time taken to verify the respective original network. Note that the reduction rate and accuracy drop of the corresponding networks can be found in Table 1 above. Clearly, there is a significant improvement in the run time of the verification algorithm; for the 6×1000 case, the verification algorithm timed out after 1

如在第1节中提到的，我们发现抽象，作为一个独立的网络，比原始网络更容易进行验证。这就打开了在实际使用的场景中验证抽象并用其替换原始网络的可能性。在图3中，我们展示了使用DeepPoly验证抽象网络所花费的时间和验证各自的原始网络所花费的时间。请注意，相应网络的降低率和精度下降见上表1。显然，验证算法的运行时间有显著改善；对于6×1000的情况，验证算法在1后超时

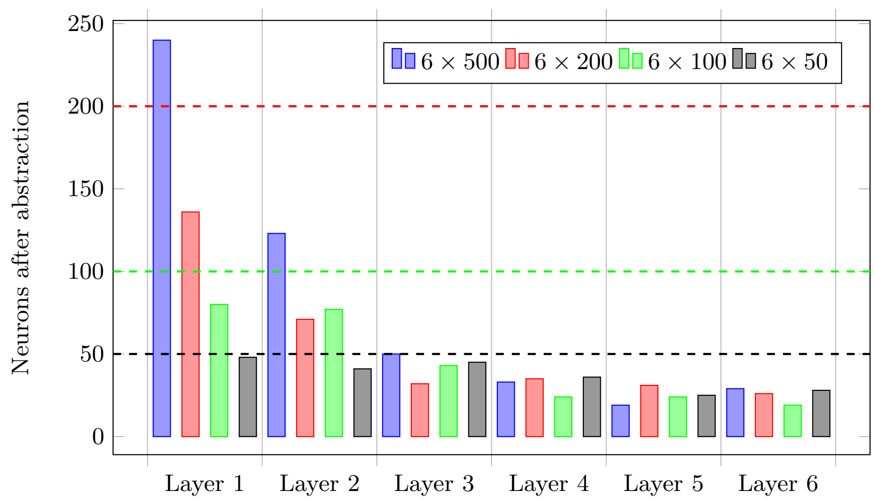


Fig. 2: Plot depicting the sizes of the abstract networks when initialized with 4 different architectures and after repetitively applying clustering-based abstraction on the layers until their accuracy on the test set is approximately 95%.

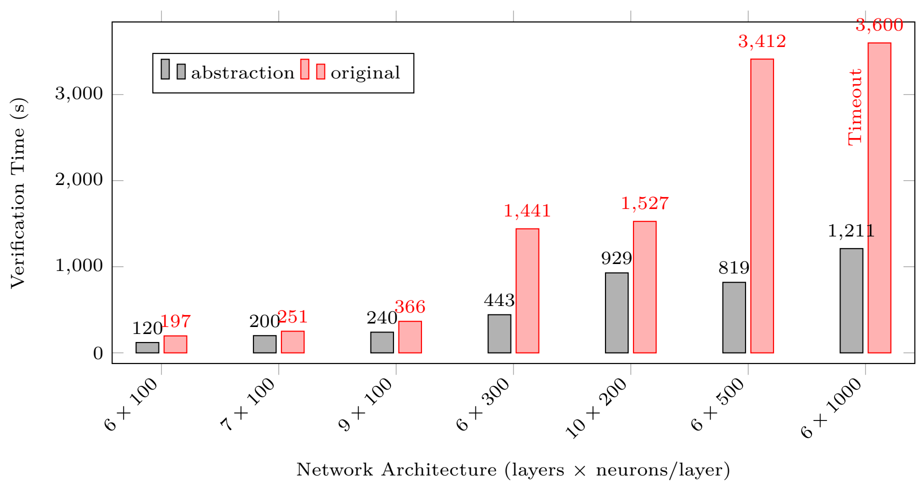


Fig. 3: Accelerated verification after abstracting compared to verification of the original. The abstracted NN are verified directly without performing proof lifting, as if they were to replace the original one in their application. The time taken for abstracting (not included in the verification time) is 14, 14, 20, 32, 37, 53, and 214s respectively.

Table 2: Results of abstraction, verification and proof lifting of a 6×300 NN on 200 images to verify. The first column gives the number of neurons removed in layers 3, 4, 5 and 6 respectively. The second column shows the reduction in the size of the abstracted network compared to the original. We also report the number of images for which the original network could be proved robust by lifting the verification proof.

Removed Neurons	**Reduction Rate (%)**	**Images Verified**	**Verification Time (*min*)**
15, 25, 100, 100	13.33	195	36
15, 50, 100, 100	14.72	195	36
25, 25, 100, 100	13.89	190	36
25, 50, 100, 100	15.28	190	36
25, 100, 100, 100	18.06	63	35
50, 100, 100, 100	19.44	0	34

hour on the original network while it finished in less than 21 minutes on the abstract network.

5.2 Results on lifting verification proof

Finally, we ran experiments to demonstrate the working of the full verification pipeline — involving clustering to identify the neurons that can be merged, performing the abstraction (Section 3.2), running DeepPoly on the abstraction and finally lifting the verification proof to answer the verification query on original network (Section 4).

最后，我们进行了实验来演示完整的验证管道的工作——包括聚类来识别可以合并的神经元，执行抽象（3.2节），在抽象上运行DeepPoly，最后提升验证证明，以回答原始网络上的验证查询（第4节）。

We were interested in two parameters: (i) the time taken to run the full pipeline; and (ii) the number of verification queries that could be satisfied (out of 200). We ran experiments on a 6×300 network that could be verified to be locally robust for 197/200 images in 48 minutes by DeepPoly. The results are shown in Table 2. In the best case, our preliminary implementation of the full pipeline was able to verify robustness for 195 images in 36 minutes — 13s for clustering and abstracting, 35 min for verification, and 5s for proof lifting. In other words, a 14.7% reduction in network size produced a 25% reduction in verification time. When we pushed the abstraction further, e.g. last row of Table 2, to obtain a reduction of 19.4% in the network size, DeepPoly could still verify robustness of the abstracted network for 196 images in just 34 minutes (29% reduction). However, in this case, the proof could not be lifted to the original network as the over-approximations we obtained were too coarse.

我们对两个参数感兴趣：(i) 运行完整管道所需的时间；(ii) 可以满足的验证查询的数量（共 200 个）。我们在 6 × 300 网络上进行了实验，DeepPoly 可以在 48 分钟内验证该网络对 197/200 图像具有本地鲁棒性。结果如表 2 所示。在最好的情况下，我们对完整管道的初步实现能够在 36 分钟内验证 195 张图像的稳健性——聚类和抽象需要 13 秒，验证需要 35 分钟，证明提升需要 5 秒。换句话说，网络规模减少 14.7%，验证时间减少 25%。当我们进一步推动抽象时，例如表 2 的最后一行，为了使网络大小减少 19.4%，DeepPoly 仍然可以在 34 分钟内验证抽象网络对 196 张图像的鲁棒性（减少 29%）。然而，在这种情况下，由于我们获得的过度近似过于粗糙，因此无法将证明提升到原始网络。

This points to the interesting fact that the time taken in clustering and proof lifting are indeed not the bottlenecks in the pipeline. Moreover, a decrease in the width of the network indeed tends to reduce the verification time. This opens the possibility of spending additional computational time exploring more powerful heuristics (e.g. principal component analysis) in place of the naïve k-means

这指出了一个有趣的事实，即聚类和证明提升所花费的时间确实不是管道中的瓶颈。 此外，网络宽度的减小确实会减少验证时间。 这开启了花费额外的计算时间来探索更强大的启发式（例如主成分分析）代替朴素的 k 均值聚类以找到更小的抽象的可能性。 此外，可以采用反例引导抽象细化（CEGAR）方法通过在必要时调整抽象来改进证明提升。

clustering in order to find smaller abstractions. Moreover, a counterexample-guided abstraction refinement (CEGAR) approach can be employed to improve the proof lifting by tuning the abstraction where necessary.

6 Conclusion

We have presented an abstraction framework for feed-forward neural networks using ReLU activation units. Rather than just syntactic information, it reflects the semantics of the neurons, via our concept of I/O-similarity on experimental values. In contrast to compression-based frameworks, the abstraction mapping between the original neurons and the abstract neurons allows for transferring verification proofs (transferring counterexamples is trivial), allowing for abstraction-based verification of neural networks.

While we have demonstrated the potential of the new abstraction approach by a proof-of-concept implementation, its practical applicability relies on several next steps. Firstly, I/O-similarity with the Euclidean distance ignores even any linear dependencies of the I/O-vectors; I/O-similarity with e.g. principal component analysis thus might yield orders of magnitude smaller abstractions, scaling to more realistic networks. Secondly, due to the correspondence between the proofs, CEGAR could be employed: one can refine those neurons where the transferred constraints in the proof become too loose. Besides, it is also desirable to extend the framework to other architectures, such as convolutional neural networks.

References

[Aba+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[AM18] Naveed Akhtar and Ajmal Mian. "Threat of adversarial attacks on deep learning in computer vision: A survey". In: *IEEE Access* 6 (2018), pp. 14410–14430.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. "Model Checking and Abstraction". In: *ACM Trans. Program. Lang. Syst.* 16.5 (1994), pp. 1512–1542.

[Che+17a] Xiaozhi Chen et al. "Multi-view 3d object detection network for autonomous driving". In: *CVPR*. 2017.

[Che+17b] Yu Cheng et al. "A Survey of Model Compression and Acceleration for Deep Neural Networks". In: *CoRR* abs/1710.09282 (2017).

[Cla+00] Edmund M. Clarke et al. "Counterexample-Guided Abstraction Refinement". In: *CAV*. 2000.

[CNR17] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. "Maximum Resilience of Artificial Neural Networks". In: *ATVA*. 2017.

[Den+20] Lei Deng et al. "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey". In: *Proceedings of the IEEE* 108.4 (2020), pp. 485–532.

[Don+18] Yinpeng Dong et al. "Boosting adversarial attacks with momentum". In: *CVPR*. 2018.

[Dvi+18] Krishnamurthy Dvijotham et al. "A Dual Approach to Scalable Verification of Deep Networks." In: *UAI*. 2018.

[Ehl17] Rüdiger Ehlers. "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks". In: *ATVA*. 2017.

[Geh+18] Timon Gehr et al. "Ai2: Safety and robustness certification of neural networks with abstract interpretation". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018.

[HMD16] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *ICLR*. 2016.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[Hua+17] Xiaowei Huang et al. "Safety Verification of Deep Neural Networks". In: *CAV (1)*. 2017.

[JKO18] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. "Deep Neural Network Compression for Aircraft Collision Avoidance Systems". In: *CoRR* abs/1810.04240 (2018).

[Kat+17] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *CAV (1)*. 2017.

[LeC98] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *ICML*. 2013.

[PA19] Pavithra Prabhakar and Zahra Rahimi Afzal. "Abstraction based Output Range Analysis for Neural Networks". In: *NeurIPS*. 2019.

[Pap+16] Nicolas Papernot et al. "The limitations of deep learning in adversarial settings". In: *EuroS&P*. IEEE. 2016.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[PT10] Luca Pulina and Armando Tacchella. "An Abstraction-Refinement Approach to Verification of Artificial Neural Networks". In: *CAV*. 2010.

[SB15] Suraj Srinivas and R. Venkatesh Babu. "Data-free Parameter Pruning for Deep Neural Networks". In: *BMVC*. 2015.

[Sin+19a] Gagandeep Singh et al. "An abstract domain for certifying neural networks". In: *Proc. ACM Program. Lang.* 3.POPL (2019), 41:1–41:30.

[Sin+19b] Gagandeep Singh et al. “Boosting Robustness Certification of Neural Networks”. In: *ICLR (Poster)*. 2019.

[SVS19] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. “One Pixel Attack for Fooling Deep Neural Networks”. In: *IEEE Trans. Evolutionary Computation* 23.5 (2019), pp. 828–841.

[YGK19] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. “An Abstraction-Based Framework for Neural Network Verification”. In: *arXiv e-prints*, arXiv:1910.14574 (2019). arXiv: `1910.14574 [cs.FL]`.

[ZYZ18] Guoqiang Zhong, Hui Yao, and Huiyu Zhou. “Merging Neurons for Structure Compression of Deep Networks”. In: *ICPR*. 2018.

A Technical Details and Proofs

A.1 Correctness of merging

Suppose for every input, the activation values of two neurons p and q of layer ℓ are equal, i.e. $z_p^{(\ell)} = z_q^{(\ell)}$ (note that for the sake of readability, we omit the input from our equations and write $z_i^{(\ell)}$ instead of $z_i^{(\ell)}(x)$), then we argue that the neurons could be merged by keeping, without loss of generality, only neuron p and setting the outgoing weights of p to the sum of outgoing weights of p and q. More formally, suppose the activation value of neuron p, $z_p^{(\ell)} = \phi(W_{p,*}^{(\ell-1)}\mathbf{z}^{(\ell-1)} + \mathbf{b}_p^{\ell})$, and that of neuron q, $z_q^{(\ell)} = \phi(W_{q,*}^{(\ell-1)}\mathbf{z}^{(\ell-1)} + \mathbf{b}_q^{\ell})$ are equal for every input to the network. Let $\tilde{D}$ be the neural network obtained after merging neurons p and q in layer ℓ. Note that D and $\tilde{D}$ are identical in all layers which follow layer ℓ. Due to the feedforward nature of the networks, it is easy to see that if for each input the vector of pre-activations of layer $\ell+1$ in D and $\tilde{D}$ are same, i.e. $\tilde{\mathbf{h}}^{(\ell+1)} = \mathbf{h}^{(\ell+1)}$, then the outputs of D and $\tilde{D}$ will also be the same.

The weight matrices of D are copied to $\tilde{D}$. $\tilde{W}^{(\ell-1)}$ is set to $W^{(\ell-1)}$ with the q^{th} row deleted. Further, we set $\tilde{W}_{*,p}^{(\ell)} = W_{*,p}^{(\ell)} + W_{*,q}^{(\ell)}$. Intuitively, this is same as deleting neuron q and moving all its outgoing edges to neuron p. Suppose the pre-activation value of neuron i of layer $\ell+1$ of D was given by

$$h_i^{(\ell+1)} = \mathbf{b}_i^{(\ell+1)} + w_{i,p}^{(\ell)} z_p^{(\ell)} + w_{i,q}^{(\ell)} z_q^{(\ell)} + \sum_{k\in\{1,\ldots,n_\ell\}\setminus\{p,q\}} w_{i,k}^{(\ell)} z_k^{(\ell)}$$

Since we assume that $z_p^{(l)} = z_q^{(l)}$, we can rewrite the RHS of the above equation as

$$h_i^{(\ell+1)} = \mathbf{b}_i^{(\ell+1)} + (w_{i,p}^{(\ell)} + w_{i,q}^{(\ell)}) z_p^{(\ell)} + \sum_{k\in\{1,\ldots,n_\ell\}\setminus\{p,q\}} w_{i,k}^{(\ell)} z_k^{(\ell)}$$

In the transformed NN $\tilde{D}$, since we have set $\tilde{W}_{*,p}^{(\ell)} = W_{*,p}^{(\ell)} + W_{*,q}^{(\ell)}$, we obtain

$$\tilde{h}_i^{(\ell+1)} = \mathbf{b}_i^{(\ell+1)} + (w_{i,p}^{(\ell)} + w_{i,q}^{(\ell)}) z_p^{(\ell)} + \sum_{k\in\{1,\ldots,n_\ell\}\setminus\{p,q\}} w_{i,k}^{(\ell)} z_k^{(\ell)}$$

A.2 Error bounds

Let n_ℓ denote the number of neurons in layer ℓ. We use the symbol $\mathbf{z}^{(\ell)} = [z_1^{(\ell)}, \ldots, z_{n_\ell}^{(\ell)}]^\intercal$ to denote the column vector of activations of layer ℓ, $W^{(\ell)} = (w_{ji}^{(\ell)})$ to denote the $n_{\ell+1} \times n_\ell$ matrix of weights $w_{ji}^{(\ell)}$ of the edge from node i in layer ℓ to node j in layer $\ell+1$, $\mathbf{h}^{(\ell)} = [h_1^{(\ell)}, \ldots, h_{n_\ell}^{(\ell)}]^\intercal$ denotes the column vector of pre-activations of layer ℓ, and $\mathbf{b}^{(\ell)} = [b_1^{(\ell)}, \ldots, b_{n_\ell}^{(\ell)}]^\intercal$ to denote the column vector of biases of layer ℓ.

In the rest of the discussion, we omit the parameter x and write $\mathbf{z}^{(\ell)}$ or $\mathbf{h}^{(\ell)}$ instead of $\mathbf{z}^{(\ell)}(x)$ or $\mathbf{h}^{(\ell)}(x)$ respectively for the sake of readability.

Lemma 1 (Single-step error). *If the activations $\mathbf{z}^{(\ell)}$ of a single layer ℓ are perturbed by $\Delta\mathbf{z}^{(\ell)}$, then the perturbation of the activations of layer $\ell+1$ is bounded according to*

$$\Delta\mathbf{z}^{(\ell+1)} \leq \phi(W^{(\ell)}\Delta\mathbf{z}^{(\ell)})$$

if the activation function ϕ is sub-additive.

Proof (of Lemma 1). Suppose that $\mathbf{z}^{(\ell)}$ was perturbed by some $\Delta\mathbf{z}^{(\ell)}$ to obtain the new activation $\tilde{\mathbf{z}}^{(\ell)} = \mathbf{z}^{(\ell)} + \Delta\mathbf{z}^{(\ell)}$, then we would define

$$\tilde{\mathbf{h}}^{(\ell+1)} = W^{(\ell)}\tilde{\mathbf{z}}^{(\ell)} + \mathbf{b}^{(\ell+1)}$$

following which, we can bound the difference between the original $\mathbf{h}^{(\ell+1)}$ and the perturbed $\tilde{\mathbf{h}}^{(\ell+1)}$:

$$\begin{aligned}\Delta\mathbf{h}^{(\ell+1)} = \tilde{\mathbf{h}}^{(\ell+1)} - \mathbf{h}^{(\ell+1)} &= W^{(\ell)}(\tilde{\mathbf{z}}^{(\ell)} - \mathbf{z}^{(\ell)}) \\ &= W^{(\ell)}\Delta\mathbf{z}^{(\ell)}\end{aligned}$$

When this error is propagated across the neurons of the $(\ell+1)^{th}$ layer, we have

$$\begin{aligned}\tilde{\mathbf{z}}^{(\ell+1)} = \phi(\tilde{\mathbf{h}}^{(\ell+1)}) &= \phi(\mathbf{h}^{(\ell+1)} + \Delta\mathbf{h}^{(\ell+1)}) \\ &= \phi(\mathbf{h}^{(\ell+1)} + W^{(\ell)}\Delta\mathbf{z}^{(\ell)})\end{aligned} \tag{4}$$

If ϕ is sub-additive, we have

$$\begin{aligned}\tilde{\mathbf{z}}^{(\ell+1)} &\leq \phi(\mathbf{h}^{(\ell+1)}) + \phi(W^{(\ell)}\Delta\mathbf{z}^{(\ell)}) \\ \tilde{\mathbf{z}}^{(\ell+1)} &\leq \mathbf{z}^{(\ell+1)} + \phi(W^{(\ell)}\Delta\mathbf{z}^{(\ell)}) \\ \Delta\mathbf{z}^{(\ell+1)} &\leq \phi(W^{(\ell)}\Delta\mathbf{z}^{(\ell)})\end{aligned}$$

□

Proof (of Theorem 1). Assume we already have clustered all layers up to layer $\ell+1$ and we know the accumulated error for layer ℓ, namely $\boldsymbol{err}^{(\ell)}$. The error in layer $\ell+1$ is defined as $\boldsymbol{err}^{(\ell+1)} = |\tilde{\tilde{\mathbf{z}}}^{(\ell+1)} - \mathbf{z}^{(\ell+1)}|$, where $\tilde{\tilde{\mathbf{z}}}$ denotes the activation values of layer $\ell+1$ after clustering it. Let $\tilde{\mathbf{z}}^{(\ell+1)}$ denote the activation values of layer $\ell+1$ when all layers before are clustered but not the layer itself, and $\mathbf{z}^{(\ell+1)}$ shall be the original activation values. We have

$$|\boldsymbol{err}^{(\ell+1)}| = |\tilde{\tilde{\mathbf{z}}}^{(\ell+1)} - \mathbf{z}^{(\ell+1)}| \tag{5}$$

$$= |\tilde{\tilde{\mathbf{z}}}^{(\ell+1)} - \tilde{\mathbf{z}}^{(\ell+1)} + \tilde{\mathbf{z}}^{(\ell+1)} - \mathbf{z}^{(\ell+1)}| \tag{6}$$

$$\leq |\tilde{\tilde{\mathbf{z}}}^{(\ell+1)} - \tilde{\mathbf{z}}^{(\ell+1)}| + |\tilde{\mathbf{z}}^{(\ell+1)} - \mathbf{z}^{(\ell+1)}| \tag{7}$$

We know from Lemma 1 how the error is propagated to the next layer. So, we know

$$|\tilde{\mathbf{z}}^{(\ell+1)} - \mathbf{z}^{(\ell+1)}| \leq |\phi(W^{((\ell))}\boldsymbol{err}^{(\ell)})| \tag{8}$$

We now have to consider the error introduced in layer $\ell+1$ by the clustering. From definition, it is $|\mathbf{z}_{r_i} - \mathbf{z}_i| \leq \boldsymbol{\epsilon}_{r_i}$ for any node i and its cluster representative r_i. Note that any node is contained in a cluster but that most of the clusters have size 1. For most nodes, we would then have $i = r_i$. However, in the general case we get

$$|\tilde{\tilde{\mathbf{z}}}^{(\ell+1)} - \tilde{\mathbf{z}}^{(\ell+1)}| \leq \boldsymbol{\epsilon}^{(\ell+1)} \tag{9}$$

Thus, equation 5 becomes

$$|\boldsymbol{err}^{(\ell+1)}| \leq |\phi(W^{((\ell))}\boldsymbol{err}^{(\ell)})| + \boldsymbol{\epsilon}^{(\ell+1)} \tag{10}$$

This can be made simpler for the ReLU-, parametric or leaky ReLU and the tanh-activation function. For all of them, it holds $|\phi(x)| \leq |x|$. Thus

$$|\boldsymbol{err}^{(\ell+1)}| \leq |W^{((\ell))}\boldsymbol{err}^{(\ell)}| + \boldsymbol{\epsilon}^{(\ell+1)} \tag{11}$$

which is what we wanted to show. □

Proof (of Theorem 2). We are interested in computing $|\boldsymbol{err}_{total}| = |\tilde{D}(\tilde{x}) - D(x)|$ which can be rewritten as $|\tilde{D}(\tilde{x}) - \tilde{D}(x)) + (\tilde{D}(x) - D(x)|$.
$|\tilde{D}(\tilde{x}) - \tilde{D}(x)| \leq |\tilde{W}^{(L)} \dots \tilde{W}^{(1)}\boldsymbol{\delta}|$ is a consequence of Lemma 1 and under the assumption that the activation function ϕ fulfills $\phi(x) \leq x$, which is true for ReLU and tanh.
$|\tilde{D}(x) - D(x)| = |\boldsymbol{err}^{(L)}|$ which is a direct consequence of Theorem 1. □

A.3 Lifting guarantees

Proof (of Theorem 3). As the verification of a specific property only considers the upper and lower bound of the output layer L, it is sufficient to show that $\hat{u}(L) \geq u(L)$ and $\hat{l}(L) \leq l(L)$, where u and l correspond to the upper- and lower-bound given by DeepPoly on the original network, and $\hat{u}(L)$ and $\hat{l}(L)$ denote the over-approximations.
We can show this inductively, where the base case is obvious. For the first layer,

$\hat{u}^{(1)} = \tilde{u}^{(1)} + \delta^u_{acc}(1) = u^{(1)} + 0$ and $\hat{l}^{(1)} = \tilde{l}^{(1)} - \delta^l_{acc}(1) = l^{(0)} - 0$.
Let's consider now some layer ℓ and start with the upper bound. We have

$$u^{(\ell)} = \max\left(0, W_+^{(\ell-1)} u^{(\ell-1)} + W_-^{(\ell-1)} l^{(\ell-1)} + b^\ell\right) \tag{12}$$

from the calculation of [Sin+19a, Section 4.4] and

$$\hat{u}^{(\ell)} = \max\left(0, \tilde{W}_+^{(\ell-1)}(\hat{u}^{(\ell-1)} + \boldsymbol{\epsilon}^{(\ell-1)}) + \tilde{W}_-^{(\ell-1)}(\hat{l}^{(\ell-1)} - \boldsymbol{\epsilon}^{(\ell-1)}) + \tilde{b}^\ell\right) \tag{13}$$

by our definition.
We want to show that $\hat{u}^{(\ell)} - u^{(\ell)} \geq 0$. We can leave out the max operation, because it is clear that $(a - b \geq 0) \Rightarrow (\max(0, a) - \max(0, b) \geq 0)$.
Let's consider only one node in layer ℓ, say node n, and omit the max-operation. For simplicity, we also omit the superscript $\ell - 1$ in the following calculation. Let I denote all nodes from layer $\ell - 1$ in the original network and $\tilde{I}$ in the abstracted one. We get

$$\begin{aligned}
\hat{u}_n^{(\ell)} - u_n^{(\ell)} = & \sum_{i \in \tilde{I}} \tilde{w}_{i,n}^+ (\hat{u}_i + \epsilon_i) \\
& + \sum_{i \in \tilde{I}} \tilde{w}_{i,n}^- (\hat{l}_i - \epsilon_i) \\
& - \left(\sum_{i \in I} w_{i,n}^+ u_i + \sum_{i \in I} w_{i,n}^- l_i\right)
\end{aligned}$$

It is $\tilde{I} \subset I$ and we can map all nodes in I to their corresponding cluster c with its cluster-representative $r \in \tilde{I}$. Thus we get

$$\begin{aligned}
\hat{u}_n^{(\ell)} - u_n^{(\ell)} = & \sum_{c \,\text{cluster}} \left(\tilde{w}_{r,n}^+ (\hat{u}_r + \epsilon_r) - \sum_{m \in c} w_{m,n}^+ u_m\right) \\
& + \sum_{c \,\text{cluster}} \left(\tilde{w}_{r,n}^- (\hat{l}_r - \epsilon_r) - \sum_{m \in c} w_{m,n}^- l_m\right)
\end{aligned}$$

For each cluster c, there are two cases: either it contains only one node or more than one node. In the case of one node per cluster, the abstracted network does not differ from the original one, so $\tilde{w}_r = w_r$. For such cluster c, we get

$$\begin{aligned}
& w_{r,n}^+ (\hat{u}_r + \epsilon_r) - w_{r,n}^+ u_r \\
= & w_{r,n}^+ (\hat{u}_r + \epsilon_r - u_r) \\
= & w_{r,n}^+ (\hat{u}_r - u_r) \\
\geq & 0
\end{aligned}$$

and

$$\begin{aligned}
&w^-_{r,n}(\hat{l}_r - \epsilon_r) - w^-_{r,n} l_r \\
=& w^-_{r,n}(\hat{l}_r - \epsilon_r - l_r) \\
=& w^-_{r,n}(\hat{l}_r - l_r) \\
\geq& 0
\end{aligned}$$

because for any un-clustered node $\epsilon_r = 0$, and by induction hypothesis $\hat{u}_r \geq u_r$ and $\hat{l}_r \leq l_r$.

Let's now consider the second case, where one cluster contains more than one node. In such cluster c, we have $\tilde{w}_r = \sum_{m \in c} w_m$, so

$$\begin{aligned}
&\tilde{w}^+_{r,n}(\hat{u}_r + \epsilon_r) - \sum_{m \in c} w^+_{m,n} u_m \\
=& \sum_{m \in c} w^+_{m,n}(\hat{u}_r + \epsilon_r) - \sum_{m \in c} w^+_{m,n} u_m \\
=& \sum_{m \in c} w^+_{m,n}(\hat{u}_r + \epsilon_r - u_m) \\
\geq& 0
\end{aligned}$$

and

$$\begin{aligned}
&\tilde{w}^-_{r,n}(\hat{l}_r - \epsilon_r) - \sum_{m \in c} w^-_{m,n} l_m) \\
=& \sum_{m \in c} w^-_{m,n}(\hat{l}_r - \epsilon_r - l_m) \\
\geq& 0
\end{aligned}$$

because by definition of ϵ_r, we have for all $m \in c$: $\hat{u}_r + \epsilon_r \geq u_m$ and similarly, $\hat{l}_r - \epsilon_r \leq l_m$. We get that $\hat{u}^{(\ell)}_n - u^{(\ell)}_n \geq 0$. The calculation for the lower bounds follows the same principle just with exchanged signs and is thus not presented here. $\square$

A.4 Details on training process

We generated various NN architectures by scaling up the number of neurons per layer as well as the number of layers themselves and trained them on MNIST. For doing so, we split the dataset into three parts: one for the training, one for validation and one for testing. The training is then performed on the training dataset by using common optimizers and loss functions. The training was stopped when the accuracy on the validation set did not increase anymore.

The NN on MNIST were trained on 60000 samples from the whole dataset. Of these, 10% are split for validation, thus there are 54000 images for the training

itself and 6000 images for validation. The optimizer used for the training process is ADAM, which is an extension to the stochastic gradient descent. To prevent getting stuck in local minima, it includes the first and second moments of the gradient. It is a common choice for the training of NN and performed reasonably well in this application. Its parameter are set to the default from TensorFlow, namely a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e - 07$.
For MNIST, the most reasonable loss function is the sparse categorical crossentropy. The training process was stopped when the loss function on the validation data did not decrease anymore. Usually, the process would stop after at most 10 epochs.

A.5 Proof lifting example

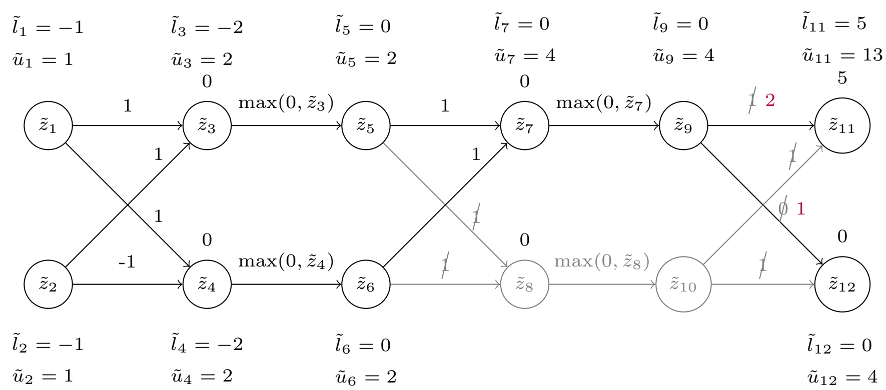


Fig. 4: Abstracted network showing the constrains returned by DeepPoly. Note that this is equivalent to having the ReLU unit identified by neurons 8 and 10 merged into the ReLU unit identified by neurons 7 and 9.

Example 2. Consider the network shown in Figure 4. The ReLU layers have already been split into two: those (i) computing the affine sum and (ii) computing $\max(0, \cdot)$. The greyed/striked out weights belong to the original network but are not present in the abstract network, in which the ReLU unit identified by neurons 8 and 10 have been merged into the ReLU unit identified by neurons 7 and 9. The two weights coloured purple (between neuron 9 and 11 as well as between 9 and 12) are a result of abstraction, as described in Section 3.1.

We apply the DeepPoly algorithm as discussed in [Sin+19a, Section 2] on the abstracted network to obtain the bounds shown in the figure. Neurons 1-7 and 9 are unaffected by the merging procedure. For neuron 11, we have $\tilde{l}_{11} = 5$ and $\tilde{u}_{11} = 13$, and for neuron 12, $\tilde{l}_{12} = 0$ and $\tilde{u}_{12} = 4$. Since $\tilde{l}_{11} > \tilde{u}_{12}$, we can conclude that the abstracted network is robust, however, we do not know if the lower bound l_{11} from the original network is indeed greater than the upper bound u_{12}.

Using Theorem 3 and the result of DeepPoly on the abstraction, we can compute the bounds $[\hat{l}, \hat{u}]$ such that it contains $[l, u]$, the bounds that would have been computed by DeepPoly for the original network.

$$\hat{l}^{(6)} \leq l^{(6)}$$

$$\text{or} \qquad \begin{bmatrix} \hat{l}_{11} \\ \hat{l}_{12} \end{bmatrix} \leq \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix}$$

and

$$\hat{u}^{(6)} \geq u^{(6)}$$

$$\text{or} \qquad \begin{bmatrix} \hat{u}_{11} \\ \hat{u}_{12} \end{bmatrix} \geq \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix}$$

Assuming the cluster diameter to be ϵ as defined in Equation 3, we get

$$\hat{u}(6) = \begin{bmatrix} 13 + 2\epsilon^{(4)} \\ 4 + \epsilon^{(4)} \end{bmatrix}$$

and

$$\hat{l}(6) = \begin{bmatrix} 5 - 2\epsilon^{(4)} \\ -\epsilon^{(4)} \end{bmatrix}$$

and therefore,

$$\begin{bmatrix} 5 - 2\epsilon^{(4)} \\ -\epsilon^{(4)} \end{bmatrix} \leq \begin{bmatrix} l_{11} \\ l_{12} \end{bmatrix} \text{ and } \begin{bmatrix} u_{11} \\ u_{12} \end{bmatrix} \leq \begin{bmatrix} 13 + 2\epsilon^{(4)} \\ 4 + \epsilon^{(4)} \end{bmatrix}$$

To determine if t $5 - 2\epsilon^{(4)} = \hat{l}_{11} > \hat{u}_{12} = 4 + \epsilon^{(4)}$ holds, we need to have a value for $\epsilon^{(4)}$. As this is only a toy example and the neurons were chosen manually and not by clustering, we do not have a value for it here. However, one can see that the proof lifting heavily depends on this value. If it was $\epsilon^{(4)} \geq \frac{1}{3}$, the property could not be lifted, even it would theoretically hold on the original network.

A.6 Implementation details

We implemented the abstraction technique described in Section 3.2 using the popular deep learning library TensorFlow [Aba+15] and the machine learning library Scikit-learn [Ped+11]. For the verification, we used the DeepPoly implementation available in the ERAN toolbox[4]

[4] Available at github.com/eth-sri/ERAN