DeepSafe: A Data-driven Approach for Assessing Robustness of Neural Networks

Divya Gopinath[1], Guy Katz[3], Corina S. Păsăreanu[1,2], and Clark Barrett[3]

[1] Carnegie Mellon University
[2] NASA Ames Research Center
divgml@gmail.com, corina.s.pasareanu@nasa.gov
[3] Stanford University
guyk@cs.stanford.edu, barrett@cs.stanford.edu

Abstract. Deep neural networks have achieved impressive results in many complex applications, including classification tasks for image and speech recognition, pattern analysis or perception in self-driving vehicles. However, it has been observed that even highly trained networks are very vulnerable to *adversarial perturbations*. Adding minimal changes to inputs that are correctly classified can lead to wrong predictions, raising serious security and safety concerns. Existing techniques for checking *robustness* against such perturbations only consider searching locally around a few individual inputs, providing limited guarantees. We propose DeepSafe, a novel approach for automatically assessing the overall robustness of a neural network. DeepSafe applies clustering over known labeled data and leverages off-the-shelf constraint solvers to automatically identify and check *safe regions* in which the network is robust, i.e. *all* the inputs in the region are guaranteed to be classified correctly. We also introduce the concept of *targeted robustness*, which ensures that the neural network is guaranteed not to misclassify inputs within a region to a specific target (adversarial) label. We evaluate DeepSafe on a neural network implementation of a controller for the next-generation Airborne Collision Avoidance System for unmanned aircraft (ACAS Xu) and for the well known MNIST network. For these networks, DeepSafe identified many regions which were safe, and also found adversarial perturbations of interest.

1 Introduction

Machine learning techniques such as deep neural networks (NN) are increasingly used in a variety of applications, achieving impressive results in many domains, and matching the cognitive ability of humans in complex tasks. In this paper, we study a common use of NN as classifiers that take in complex, high dimensional input, pass it through multiple layers of transformations, and finally assign to it a specific output label or class. These networks can be used to perform pattern analysis, image classification, or speech and audio recognition, and are now being integrated into the perception modules of autonomous or semi-autonomous vehicles, at major car companies such as Tesla, BMW, Ford, etc. It is expected that this trend will continue and intensify, with neural networks being increasingly used in safety critical applications which require high assurance guarantees.

However, it has been observed that state-of-the-art networks are highly vulnerable to *adversarial perturbations*: given a correctly-classified input x, it is possible to find a new input x' that is very similar to x but is assigned a different label [17]. The vulnerability of neural networks to adversarial perturbations is thus a major safety and security concern, and it is essential to explore systematic methods for evaluating and improving the robustness of neural networks against such attacks.

To date, researchers have mostly focused on efficiently finding adversarial perturbations around select individual input points. The problem is typically cast as an optimization problem: for a given network F and an input x, find an x' such that F assigns different labels to x and x' (denoted $F(x') \neq F(x)$), while minimizing the distance $\|x - x'\|$, for different distance metrics. In other words, the goal is to find an input x' as close as possible to x such that x' and x are labeled differently. Finding the optimal solution for this problem is computationally difficult, and so various approximation approaches have been proposed [17, 6, 5, 3]. There are also techniques that focus on generating *targeted attacks*: adversarial perturbations that result in the network assigning the perturbed input a specific target label [17, 6, 5].

The approaches for finding adversarial perturbations have successfully demonstrated the weakness of many state-of-the-art networks. However, using these techniques in assessing a network's robustness against adversarial perturbations is difficult, for two reasons. First, these approaches are heuristic-based, and provide no formal guarantees that they have not overlooked some adversarial perturbations; and second, these approaches only operate on individual input points, which may not be indicative of the network's robustness around other points. Approaches have also been proposed for training networks that are robust against adversarial perturbations, but these, too, provide no formal assurances [13].

Formal methods present a promising way for obtaining formal guarantees about the robustness of networks. Recent approaches tackle neural network verification [7, 10] by casting it as an SMT solving problem. Reluplex [10] can check the *local robustness* at input point x, by checking if there is another point x' within a close distance δ to x ($\|x - x'\| < \delta$) for which the network assigns a different label.The initial value δ is picked arbitrarily and is tightened iteratively until the check holds. Similarly, DLV [7] searches locally around given inputs x within a small δ distance, but unlike Reluplex, it adopts discretization of the input space to reduce the search space.

These techniques are still limited to checking local robustness around a few individual points, giving no indication about the overall robustness of the network. In principle, one can apply the local check to a set of inputs that are drawn from some random distribution thought to represent the input space. However, this would require coming up with minimally acceptable distance δ values for all these checks, which can vary greatly between different input points. Furthermore, the check will likely fail (and produce invalid adversarial examples) for the input points that are close to the legitimate boundaries between different labels.

The concept of *global robustness* is defined in [10, 11], which could be checked by Reluplex (we give the formal definition in Section 6). Whereas local robustness is measured for a specific input point, global robustness applies to all inputs simultaneously. The check requires encoding two side-by-side copies of the NN in question, operating

on separate input variables, and checking that the outputs are similar. Global adversarial robustness is harder to prove than local robustness for two reasons: (1) encoding two copies of the network results in twice as many network nodes to analyze and (2) the problem is not restricted to a small domain, and therefore can not take advantage of Reluplex's heuristics, which work best when the check is restricted to small neighborhoods. Furthermore, it requires more manual tuning, as the user needs to provide minimally acceptable values for two parameters (δ and ϵ). As a result this global check could not be applied in practice to any realistic network [10].

Thus the problem of assessing the overall robustness of a network remains open.

DeepSafe. We propose DeepSafe, an automatic, data-driven approach for assessing the *overall robustness* of a neural network. The key idea is to effectively leverage the inputs with known correct labels to automatically identify *regions* in the input space which have a high chance of being labelled consistently or uniformly, thereby making global robustness checks achievable.

Specifically, DeepSafe applies a clustering algorithm over the labeled inputs to group them into *clusters*, which are sets of inputs that are close to each other (with respect to some given distance metric) and share the same label. The labeled inputs can be drawn from the training set or can be generated directly from executing the network on some random inputs; in the latter case the user needs to validate that the labels are correct. We use k-means clustering [9], which we modified to be guided by the labels of known inputs, but other clustering algorithms could be employed as well.

Each cluster defines a *region* which is a hypersphere in the input space, where the centroid is automatically computed by the clustering algorithm and the radius is defined as the average distance of any element in the cluster from the centroid. Our hypothesis it that the NN should assign the same label to *all* the inputs in the region, not just to the elements (inputs with known labels) that are used to form the cluster. The rationale is that even highly non-linear networks should display consistent behavior in the neighborhoods of *groups of similar inputs* known to share the same true label. We check this hypothesis by formulating it as a query for an off-the-shelf constraint solver. For our experiments we use the state-of-the-art tool Reluplex, but other solvers can be used. If the solver finds no solutions, it means that the region is *safe* (all inputs are labeled the same). If a solution is found, this indicates a potential adversarial input. We use the adversarial example as feedback to iteratively reduce the radius of the region, and repeat the check until the region is proved to be safe. If a time-out occurs during this process, we can not provide any guarantee for that region (although the likely answer is that it is safe). The result of the analysis is a set of well-defined input regions in which the NN is guaranteed to be robust and s set of examples of adversarial inputs that may be of interest to the developers of the NN, and can be used in retraining.

Thus, DeepSafe decomposes the global robustness requirements into a set of local robustness checks, one for each region, which can be solved efficiently with a tool like Reluplex. These checks do not require two network copies and are restricted to small input domains, as defined by the regions. The distance used for the local checks is not picked arbitrarily as in previous approaches, but instead it is the radius computed for each regions, backed up by the labeled data. Furthermore regions define natural

decision boundaries in the input domain, thereby increasing the chances of producing valid proofs or of finding valid adversarial examples.

We introduce the concept *targeted robustness* in line with *targeted attacks* [17, 6, 5]. Targeted robustness ensures that there are no inputs in a region that are mapped by the NN to a specific incorrect label. Therefore, even if in that region the NN is not completely robust against adversarial perturbations, we can give guarantees that it is robust against specific targeted attacks. As a simple example, consider a NN used for perception in an autonomous car that classifies the images of a traffic light as red, green or yellow. Even if it is not the case that the NN never misclassifies a red light, we may be willing to settle for a guarantee that it never misclassifies a red light as a green light — leaving us with the more tolerable case in which a red light is misclassified as yellow.

We implemented a prototype of DeepSafe and evaluated it on a neural network implementation of a controller for the next-generation Airborne Collision Avoidance System for unmanned aircraft (ACAS Xu) and on the well known MNIST dataset. For these networks, our approach identified regions which were completely safe, regions which were safe with respect to some target labels, and also adversarial perturbations that were of interest to the network's developers.

2 Background

2.1 Neural Networks

Neural networks and deep belief networks have been used in a variety of applications, including pattern analysis, image classification, speech and audio recognition, and perception modules in self-driving cars. Typically, the objects in such domains are high dimensional and the number of classes that the objects need to be classified to is also high — and so the classification functions tend to be highly non-linear over the input space. Deep learning operates with the underlying rationale that groups of input parameters can be merged to derive higher-level abstract features, which enable the discovery of a more linear and continuous classification function. Neural networks are often used as *classifiers*, meaning they assign to each input an output label/class. Such a neural network F can thus be regarded as a function that assigns to input x an output label y, denoted as $F(x) = y$.

Internally, a neural network is comprised of multiple layers of nodes, called neurons. Each node refines and extracts information from values computed by nodes in the previous layer. The structure of a typical 3 layer neural network would be as follows: the first layer is the *input* layer, which takes in the input variables (also called features) $x_1, x_2, \ldots, x_n$. The second layer is a *hidden* layer: each of its neurons computes a weighted sum of the input variables according to a unique weight vector and a bias value, and then applies a non-linear *activation function* to the result. The sigmoid function ($f(x) = 1/(1 + e^{-x})$ is a widely used activation function. Most recent networks use rectified linear units (ReLUs) activation functions. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise; $f(x) = max(x, 0)$. The last layer uses a softmax function to assign an the output class is the input. The softmax function squashes the outputs of each node of the previous layer to be between 0 and 1,

equivalent to a categorical probability distribution. The number of nodes in this layer is equal to the number of output classes and their respective outputs gives the probability of the input being classified to that class.

2.2 Neural Network Verification

Traditional verification techniques often cannot directly be applied to neural networks, and this has sparked a line of work focused on transforming the problem into a format more amenable to existing tools, such as LP and SMT solvers [4, 7, 15, 16]. DeepSafe, while is not restricted to, currently uses the recently-proposed Reluplex tool, which has been shown to perform better than other solvers, such as Z3, CVC4, Yices or MathSat [10]. Reluplex is a sound and complete simplex-based verification procedure, specifically tailored to achieve scalability on deep neural networks. It is designed to operate on networks with piecewise linear activation functions, such as ReLU. Intuitively, the algorithm operates by eagerly solving the linear constraints posed by the neural network's weighted sums, while attempting to satisfy the non-linear constraints posed by its activation functions in a lazy manner. This often allows Reluplex to safely disregard many of these non-linear constraints, which is where the bulk of the problem's complexity originates.

Reluplex has been used in evaluating techniques for finding and defending against adversarial perturbations [2], and it has also been successfully applied to a real-world family of deep neural networks, designed to operate as controllers in the next-generation Airborne Collision Avoidance System for unmanned aircraft (ACAS Xu) [10]. However, as discussed, Reluplex could only be used to check local robustness around a few individual points, giving limited guarantees about the *overall* robustness of these networks.

2.3 Clustering

Clustering is an approach used to divide a population of data-points into sets called clusters, such that the data-points in each cluster are more similar (with respect to some metric) to other points in the same cluster than to the rest of the data-points. Here we focus on a particularly popular clustering algorithm called *kMeans* [9] (although our approach could be implemented using different clustering algorithms as well). Given a set of n data-points $\{x_1, \ldots, x_n\}$ and k as the desired number of clusters, the algorithm partitions the points into k clusters, such that the variance (also referred to as "within cluster sum of squares") within each cluster is minimal. The metric used to calculate the distance between points is customizable, and is typically the Euclidean distance (L_2 norm) or the Manhattan distance (L_1 norm). For points $x_1 = \langle x_1^1, \ldots, x_n^1 \rangle$ and $x_2 = \langle x_1^2, \ldots, x_n^2 \rangle$ these are defined as:

$$\|x_1 - x_2\|_{L_1} = \sum_{i=1}^{n} |x_i^1 - x_i^2|, \qquad \|x_1 - x_2\|_{L_2} = \sqrt{\sum_{i=1}^{n} (x_i^1 - x_i^2)^2} \tag{1}$$

The kMeans clustering algorithm is an iterative refinement algorithm, which starts with k random points considered as the means (the *centroids*) of k clusters. Each iteration is then comprised of two main steps: (i) assign each data-point to the cluster whose

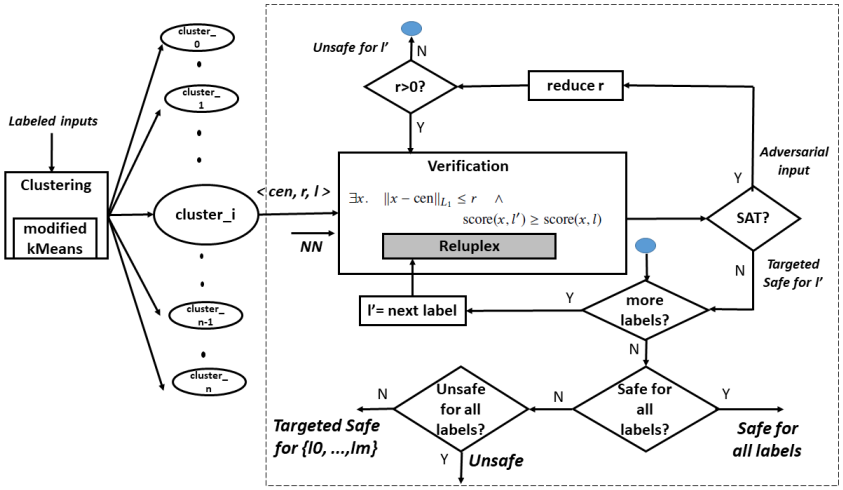

Fig. 1. The DeepSafe Approach

centroid is closest to it with respect to the chosen distance metric; and (ii) re-calculate the new means of the clusters, which will serve as the new centroids. The iterations continue until the assignment of data-points to clusters does not change. This indicates that the clusters satisfy the constraints that the variance within each cluster is minimal and that the data-points within each cluster are closer to the centroid of that cluster than to the centroid of any other cluster.

3 The DeepSafe Approach

The *DeepSafe* approach is illustrated in Figure 1. DeepSafe has two main components: clustering and verification. The inputs with known labels are fed into a clustering module which implements a modified version of the kMeans algorithm (as described in Section 3.1). The module generates clusters of similar inputs (wrt some distance metric) known to have the same label. Every such cluster defines a *region* characterized by a centroid (*cen*), radius (*r*) and label (*l*), which corresponds to the label of the inputs used to form the cluster. Thus, a cluster is a subset of the inputs while a region is the geometrical hypersphere defined by the cluster; for the rest of the paper we sometimes use regions and clusters interchangeably, when the meaning is clear from the context.

For every region, the verification module is invoked (as described in Section 3.2). This module uses Reluplex to check if there exists any input within the region for which the neural network assigns a different label than *l* (the *score* in the figure will be explained below). This check is done separately for each label *l'* other than *l*. If such an input is found (formula is SAT), then this is a potential adversarial example that is reported to the user. The radius is then reduced to exclude the adversarial input and the check is repeated. When no such adversarial example is found, the network is declared to be robust with respect to target *l'*, and correspondingly the region is declared *targeted safe* for that label *l'*. If for a particular *l'*, the solver keeps finding adversarial examples until *r* gets reduced to 0, the region is considered unsafe w.r.t. that label. If no adversar-

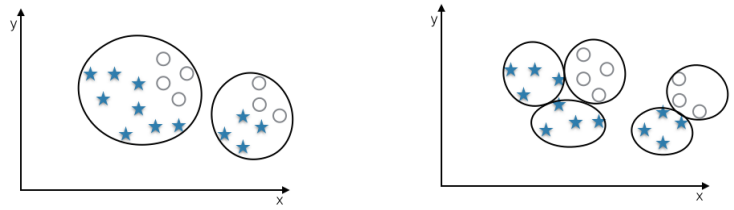

Fig. 2. Original clusters with k=2 (left), Clusters with modified kMeans (right)

ial examples are found for all the checks, the region is *completely safe*. If adversarial cases are found for all labels, it is *unsafe*.

3.1 Labeled-Guided Clustering

We employ a modified version of the kMeans clustering algorithm to perform clustering over the inputs with known correct labels. These inputs can be drawn from the training data or can be generated randomly and labeled according to the outputs given by a trained network. The kMeans approach is typically an unsupervised technique, meaning that the clustering is based purely on the similarity of the data-points themselves. Here, however, we use the labels to guide the clustering algorithm into generating clusters that have consistent labels (in addition to containing points that are similar to each other). The number of clusters, which is an input parameter of the kMeans algorithm, is often chosen arbitrarily but in our case the algorithm starts by setting the number of clusters, k, to be equal to the number of unique labels. Once the clusters are obtained, we check whether each cluster contains only inputs with the same label. kMeans is then applied again on each cluster that is found to contain multiple labels, with k set to the number of unique labels within that cluster. This effectively breaks the "problematic" cluster into multiple sub-clusters. The process is repeated until all the clusters contain inputs which share a single label.

To illustrate the clustering, let us consider a small example with training data labeled as either stars or circles. Each training data point is characterized by two dimensions/attributes (x,y). The original kMeans algorithm with $k = 2$, will partition the training inputs into 2 groups, purely based on proximity w.r.t. the 2 attributes (Fig. 2a). However, this simple approach would group stars and circles together. Our modified algorithm creates the same partitions in its first iteration, but because each cluster contains training inputs with multiple labels, it then proceeds to iteratively divide each cluster into sub-clusters. This finally creates 5 clusters as shown in (Fig. 2b): three with label star and two with label circle. This example is typical for domains such as image classification, where even a small change in some attribute value for certain inputs could change their label.

Distance Metrics. We note that the similarity of the inputs within a cluster is determined by the distance metric used for calculating the proximity of the inputs. Therefore, it is important to choose a distance metric that generates acceptable levels of similarity for the domain under consideration. We assume that inputs are representable as vectors

of numeric attributes and hence can be considered as points in Euclidean space. The Euclidean distance (Eq. 1) is a commonly used metric for measuring the proximity of points in Euclidean space. However, recent studies indicate that the usefulness of Euclidean distance in determining the proximity between points diminishes as the dimensionality increases [1]. The Manhattan distance (Eq. 1) has been found to capture proximity more accurately in high dimensions. Therefore, in our experiments, we set the distance metric depending on the dimensionality of the input space (Section 4).

The clustering algorithm aims to produce neighborhoods of consistently-labeled inputs. The underlying assumption of our approach is that each such cluster will define a *safe region*, in which all inputs (and not just the inputs used to form the cluster) should be labeled consistently. We define the regions to have the centroid cen computed by kMeans and the radius to be the *average distance* r of any instance from the centroid. Note that we use the average instead of maximum distance. The reason is that kMeans typically generates clusters that are convex, compact and spherically shaped. However, the ideal boundaries of regions encompassing consistently labeled points need not conform to this. Further, while inputs deep within a region are expected to be labeled consistently, the points that are close to the boundaries may have different labels. We therefore *shrink* the radius to increase the chances of obtaining a region that is indeed safe.

3.2 Region Verification

In this step, we check if the regions defined by the clusters formed in the previous step are *safe* for a given NN. The main hypothesis behind our approach is as follows:

Hypothesis 1 *For a given region R, with centroid cen and radius r, any input x within distance r from cen has the same true label l as that of the region:*

$$\|x - cen\| \leq r \quad \Rightarrow \quad label(x) = l$$

If this hypothesis holds, it follows that any point x' in the region which is assigned a different label $F(x') \neq l$ by the network constitutes an adversarial perturbation. To illustrate this on our example; a NN, may represent an input which is close to other stars in the input layer as a point further away from them in the inner layers. Therefore, it may incorrectly classify it as a circle.

We use Reluplex [10] to check the satisfiability of a formula representing the negation of Hypothesis 1 for every cluster for every label. The encoding is as follows://shown in Eq. 2:

$$\exists x. \quad \|x - \text{cen}\|_{L_1} \leq r \quad \wedge \quad \text{score}(x, l') \geq \text{score}(x, l) \tag{2}$$

Here, x represents an input point, and cen, r and l represent the centroid, radius and label of the region, respectively. l' represents a specific label, different from l. Reluplex models the network without the final softmax layer, and so the outputs of the model correspond to the outputs of the second last layer of the NN [REF]. This layer consists of as many nodes as the number of labels and the output value for each node corresponds

to the level of *confidence* that the network assigns to that label for the given input. We use score(x, y) to denote the level of confidence assigned to label y at point x.

The above formula holds for a given l' if and only if there exists a point x within distance at most r from cen, for which l' is assigned higher confidence than l. Consequently, if the property does *not* hold, then for every x within the cluster l, its score is higher than l'. This ensures *targeted robustness* of the network for label l': the network is guaranteed to never map any input within the region to the target label l'. The formula in Eq. 2 is checked for every possible $l' \in L - \{l\}$, where L denotes the set of all possible labels. If the query is unsatisfiable for all l', it ensures *complete robustness of the network for the region*; i.e., the network is guaranteed to map all the inputs in the region to the same label as the centroid. This can be expressed formally as follows:

$$\forall x.\ \|x - \text{cen}\|_{L_1} \leq r \quad \Rightarrow \quad \forall l \in L - \{l\}.\quad \text{score}(x, l) \geq \text{score}(x, l') \tag{3}$$

from which it follows that Hypothesis 1 holds, i.e. that:

$$\forall x.\ \|x - \text{cen}\|_{L_1} \leq r \quad \Rightarrow \quad label(x) = l \tag{4}$$

Note that as is the case with many SMT-based solvers, Reluplex typically solves satisfiable queries more quickly than unsatisfiable ones. Therefore, in order to optimize performance, we test the possible target labels l' in descending order based on the scores that they are assigned at the centroid, score(cen, l'). Intuitively, this is because labels with higher scores are more likely to yield a satisfiable query.

Encoding Distance Metrics in Reluplex. Reluplex takes as input a conjunction of linear equations and certain piecewise-linear constraints. Consequently, it is straightforward to model the neural network itself and the query in Eq. 2. Our ability to encode the distance constraint from the equation, $\|x - \text{cen}\| \leq r$, depends on the distance metric being used. While L_1 is piecewise linear and can be encoded, L_2 unfortunately cannot. When dealing with domains where L_2 distance is a better measure of proximity, we thus use the following approximation. We perform the clustering phase using the L_2 distance metric as described before and for each cluster obtain the radius r. When verifying the property in Eq. 2, however, we use the L_1 norm. Because $\|x - \text{cen}\|_{L_1} \leq \|x - \text{cen}\|_{L_2}$, it is guaranteed that any adversarial perturbation discovered would have also been discovered using the L_2 norm. If no such adversarial perturbation is discovered, however, we can only conclude that the portion of the corresponding region that was checked is safe. This limitation could be overcome by using a tool that directly supports L_2 (however no such tools currently exist), or by enhancing Reluplex to support it.

Safe Regions and Scalability. The main source of computational complexity in neural network verification is the presence of non-linear, non-convex activation functions. However, when restricted to a small domain of the input space, these functions may exhibit linear behavior — in which case they can be disregarded and replaced with a linear constraint, which greatly simplifies the problem. Consequently, performing verification within the small regions discovered by DeepSafe is beneficial, as many activation functions can often be disregarded.

The search space within a region is further reduced by restricting the range of values for each input attribute (input variable to the NN model). We calculate the minimum and maximum values for every attribute based on the instances with known labels encompassed within a cluster. Reluplex has built-in *bound tightening* [10] functionality. We leverage this by setting the lower and upper bounds for each of the input variables within the cluster based on the respective minimum and maximum values.

Our approach lends itself to more scalable verification also through parallelization. Because each region involves stand-alone verification queries, their verification can be performed in parallel. Also, because Eq. 2 can be checked independently for every l', these queries can be performed in parallel — expediting the process even further.

4 Case Studies

We implemented DeepSpace using MATLAB R2017a for the clustering algorithm and Reluplex v1.0 for verification. The runs were dispatched on a 8-Core 64GB server running Ubuntu 16.0.4. We evaluated DeepSpace on two case studies. The first network is part of a real-world controller for the next-generation Airborne Collision Avoidance System for unmanned aircraft (ACAS Xu), a highly safety-critical system. The second network is a digit classifier over the popular MNIST image dataset.

Table 1. Summary of the analysis for the ACAS Xu network for 210 clusters

property	# clusters	min radius	time(hours)	#queries
safe	125	0.084	4	11.8
targeted safe	52	0.135	7.6	14.4
time out	33	NA	12	NA

4.1 ACAS Xu

ACAS X is a family of collision avoidance systems for aircraft which is currently under development by the Federal Aviation Administration (FAA) [8]. ACAS Xu is the version for unmanned aircraft control. It is intended to be airborne and receive sensor information regarding the drone (the *ownship*) and any nearby intruder drones, and then issue horizontal turning advisories aimed at preventing collisions. The input sensor data includes: (i) ρ: distance from ownship to intruder; (ii) θ: angle of intruder relative to ownship heading direction; (iii) ψ: heading angle of intruder relative to ownship heading direction; (iv) v_{own}: speed of ownship; (v) v_{int}: speed of intruder; (vi) τ: time until loss of vertical separation; and (vii) a_{prev}: previous advisory. The five possible output actions are as follows: Clear-of-Conflict (COC), Weak Right, Weak Left, Strong Right, and Strong Left. Each advisory is assigned a score, with the lowest score corresponding to the best action. The FAA is currently exploring an implementation of ACAS Xu that uses an array of 45 deep neural networks. These networks were obtained by discretizing the two parameters, τ and a_{prev}, and so each network contains five input dimensions

Table 2. Details of the analysis for some clusters for ACAS Xu

cluster#	safe for label	radius	#queries	time(min)	slice (Y/N)
5282 label:0	1	0.04	1	5.45	N
	2	0.04	1	3.91	N
	3	0.04	1	3.57	N
	4	0.04	1	4.01	N
1783 label:0	1	0.16	4	1.28	Y
	2	0.17	1	279	N
	3	0.17	1	236	N
	4	0.17	1	223	N
2072 label:1	0	0.06	1	11.51	N
	2	0.014	9	0.98	N
	3	0.011	7	0.71	N
	4	0.012	5	0.58	N
6138 label:0	1	0.089	9	103.2	N
	2	0.11	4	2.86	N

and treats τ and a_{prev} as constants. Each network has 6 hidden layers and a total of 300 hidden ReLU activation nodes.

We applied our approach to several of the ACAS XU networks. We describe here in detail the results for one network. Each input consists of 5 dimensions and is assigned one of 5 possible output labels, corresponding to the 5 possible turning advisories for the drone (0:COC, 1:Weak Right, 2:Weak Left, 3:Strong Right, and 4:Strong Left). We were supplied a set of cut-points, representing valid important values for each dimension, by the domain experts [8]. We generated 2662704 inputs (cartesian product of the values for all the dimensions). The network was executed on these inputs and the output advisories (labels) were verified. These were considered as the inputs with known labels for our experiments.

The labeled-guided clustering algorithm was applied on the inputs using the L_2 distance metric. Clustering yielded 6145 clusters with more than one input and 321 single-input clusters. The clustering took 7 hours. For each cluster we computed a region, characterized by a centroid (computed by kMeans), radius (average distance of every cluster instance from the centroid), and the expected label (the label of all the cluster instances).

We first evaluated the network on all the centroids as they are considered representative of the entire cluster and should ideally have the expected label. The network assigned the expected label for the centroids of 5116 clusters (83% of total number of clusters). For the remaining 1029 clusters, we found that they contained few labeled instances spread out in large areas.Therefore, we considered these clusters were not precise and our analysis was inconclusive. For singleton clusters, we fall back to checking local robustness using previous techniques [10]. These stand-alone points serve to identify portions of the input space which require more training data, thus potentially more vulnerable to adversarial perturbations.

Amongst the remaining 5116 clusters, we picked randomly 210 clusters to illustrate our technique. These clusters contain 659315 labeled inputs (24% of the total inputs

with known labels). For each region corresponding to the respective clusters, we applied DeepSafe to check equation 2 for every label. The distance metric used was L_1 since L_2 can not be handled by Reluplex (see section 3 that explains why this is still safe). The results are presented in tables 1 and 2. The *min radius* in table 1, refers to the average minimum radius around the centroid of each region for which the safety guarantee applies (averaged over the total number of regions for that safety type). The *# queries* refers to the number of times the solver had to be invoked until an UNSAT was obtained, averaged over all the regions for that property.

DeepSafe was able to identify 125 regions which are completely *safe*, i.e. the network yields a label consistent with the neighboring labeled inputs within the region. 52 regions are *targeted safe*, the network is safe against misclassifying inputs to certain labels. For instance, the inputs within region 6138 (Table 2) with an expected label 0 (COC), were safe against misclassification only to labels 1 (weak right) and 2 (weak left). The solver timed out without returning any result for the remaining labels. The analysis timed out without returning a concrete result for any label for 33 clusters. A time out does not allow to provide a proof for the regions, although the likely answer is safe (generally, solvers take much longer when there is no solution).

The *min radius* in table 1, refers to the average minimum radius around the centroid of each region for which the safety guarantee applies (averaged over the total number of regions for that safety type).

The *# queries* refers to the number of times the solver had to be invoked until an UNSAT was obtained, averaged over all the regions for that property.

4.2 MNIST Image Dataset

The MNIST database is a large collection of handwritten digits that is commonly used for training various image processing systems [12]. The dataset has 60,000 training input images, each characterized by 784 attributes and belonging to one of 10 labels. We used a network that comprised of 3 layers, each with 10 ReLU activation nodes. Clustering was applied using the L_1 distance metric. It yielded 6654 clusters with more than one input and 5681 single-input clusters. The clustering consumed 10 hours. A separate process for verification of each cluster was spawned with a time-out of 12 hours.

Table 3. Summary of the analysis for MNIST network for 80 clusters

property	# clusters	min radius	time(hours)	# queries
safe	7	2.46	11.27	2.85
targeted safe	63	5.19	11.02	4.87
time out	10	NA	12	NA

For the singleton clusters, as is the case with ACAS Xu, we performed local robustness checking as in previous approaches.

Table 3 shows the summary of the results for the runs for 80 clusters that we selected for evaluation. In past studies, the MNIST network has been shown to be extremely

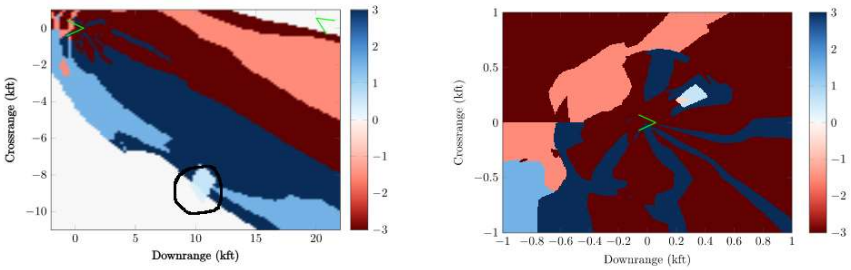

Fig. 3. Inputs highlighted in light blue are mis-classified as Strong Right instead of COC (left). Inputs highlighted in light blue are mis-classified as Strong Right instead of Strong Left(right).

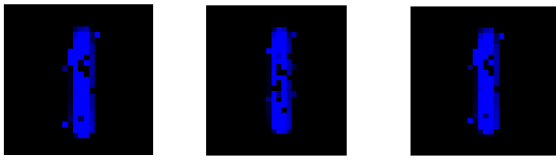

Fig. 4. Images of 1 misclassified as 8, 4, and 3

vulnerable to misclassification on adversarial perturbations even with state-of-the art networks [3]. Therefore, as expected, it is easy to determine SAT solutions and they were discovered very fast (within a minute). However, it is very time consuming to prove safety; the verification time is much higher than that of the ACAS Xu application as it is mainly impacted by the large number of input variables (784 attributes). We would like to highlight that our work is the first to successfully identify safety regions for MNIST even on a fairly vulnerable network.

For 7 clusters, the solver returned UNSAT for all labels within 12 hours. For 30 clusters, the solver returned UNSAT only for few labels but timed out before returning any solution for the other labels. These have been included in the *targeted safe* property in the table. Additionally, based on the nature of this domain, we can consider it safe to assume that if for any label the solver does not return a SAT solution within 10 hours, then it is safe w.r.t. that label even if it does not prove unsatisfiability within this time. This happened to be the case for 33 clusters, where the solver could not find a solution for a specific target label despite executing for more than 10 hours. These have been included in the *targeted safe* type as well. For 10 of the remaining clusters, the solver kept finding adversarial examples despite iterative reductions of the radius and the time-out occurred before the radius reduced to 0. These have been included as time out in the table, since we cannot determine for sure if the region should be marked unsafe for the specific labels.

5 Discussion

We compared DeepSafe with a method of randomly choosing inputs with known labels and checking for local adversarial robustness using previous work [10]. This technique

searches for inputs around the given fixed input, by varying each input variable (dimension) in the range of $[fixedvalue - \epsilon, fixedvalue + \epsilon]$ (L_∞ distance metric). It checks if there exists any input in this range, for which the network assigns a higher score to any other label than that of the fixed input. The algorithm starts with a standard epsilon value of 0.1 and iteratively reduces the value until UNSAT is obtained or the value reduces to 0.01. We chose 210 random points for ACAS Xu and 80 random points for MNIST respectively, in line with the number of regions that we checked with DeepSafe.

We found that for MNIST, local robustness checking found no safe regions around any of the 80 points, whereas DeepSafe found 7 safe regions and 63 targeted safe regions. For ACAS Xu, this technique yielded only 62 safe regions which are completely safe compared to 125 safe regions that were found using DeepSafe. This experiment shows that the choice of input points and the delta around these points play an important role in effective adversarial robustness checking. We also looked at the validity of the adversarial examples generated by DeepSafe. If an adversarial example is invalid or spurious, it indicates that the expected label is incorrect for that input and that the label generated by the network is in fact correct. During our analysis for ACAS Xu we found adversarial examples, which were validated by domain experts. The adversarial cases were found to be valid, albeit not of high criticality. The adversarial examples for MNIST were converted to images and manually verified to be valid (see Fig. 4).

There could be scenarios where both the region and the network agree on the labels for all inputs, and still this could not be the desired behavior. This would impact the *validity of the safety guarantees* provided by DeepCheck. We addressed this issue by validating the safety regions for ACAS Xu with the domain experts. The mismatch of labels for the centroid of a region does potentially indicate an imprecise oracle. However, we found that the number of such regions is not high (1029 out of 6145 clusters for ACAS Xu).

6 Related Work

The vulnerability of neural networks to adversarial perturbations was first discovered by Szegedy et al. in 2013 [17]. They model the problem of finding the adversarial example as a constrained minimization problem. Goodfellow et al. [6] introduced the Fast Gradient Sign Method for crafting adversarial perturbations using the derivative of the model's loss function with respect to the input feature vector. Jacobian-based Saliency Map Attack (JSMA) [14] proposed a method for targeted misclassification by exploiting the forward derivative of an NN to find an adversarial perturbation that will force the model to misclassify into a specific target class.

Carlini and Wagner [3] recently proposed an approach that could not be resisted by state-of-the-art networks such as those using defensive distillation. Their optimization algorithm uses better loss functions and parameters (empirically determined) and uses three different distance metrics. Deep Learning Verification (DLV) [7] is an approach that defines a region of safety around a known input and applies SMT solving for checking robustness. They search for possibly-adversarial inputs by manipulating the given valid input in a discretized input space. They can only guarantee freedom of adversarial

perturbations within the discrete points that are explored. Our clustering approach can potentially improve the technique by constraining the discrete search within regions.

Recent work [11] using Reluplex discusses in depth refined versions of global and local robustness, which take into account the *confidence* (C) that the network is placing on its predictions. For instance, the local robustness at input x_0 is defined as $\forall x.\ \|x - x_0\| < \delta \implies \forall l.\|C(F, x, l) - C(F, x_0, l)\| < \epsilon$. Similarly, the global robustness, informally introduced in [10], is defined as $\forall x, x'.\ \|x - x'\| < \delta \implies \forall l.\|C(F, x, l) - C(F, x, l)\| < \epsilon$. However, this check is expensive and also requires user input for acceptable values for both δ and ϵ. The motivation for taking into account the confidence is to better handle the inputs that lay on *boundaries* between labels, in the sense that there should be no spikes greater than ϵ in the levels of confidence that the network assigns to each labels for these points. However, the value of δ and $epsilon$ need not be the same for all inputs and all labels respectively. For instance, points embedded deep inside consistently labeled regions, the δ should be large while for points on the boundaries between labels only a small δ could be tolerable. Nevertheless, we believe that DeepSafe can be used beneficially with the above approach, by automatically finding regions that can then be checked using the more refined local check.

7 Conclusion

This paper presents a data-guided technique for assessing the adversarial robustness of neural networks. The technique can find adversarial perturbations or prove they cannot occur within well-defined geometric regions in the input space that correspond to clusters of similar inputs known to share the same label. In doing so, the approach identifies and provides proofs for regions of safety in the input space within which the network is robust with respect to target labels. Experiments with the ACAS Xu and MNIST networks highlight the potential of the approach in providing formal guarantees about the robustness of neural networks in a scalable manner. Checking robustness for deep neural networks is an active area of research. As part of future work, we plan to integrate our approach with other solvers that will broaden the application of DeepSpace to other kinds of neural networks and also investigate testing, guided by the computed regions, as an alternative to verification for increased scalability.

Acknowledgements. This work was partially supported by grants from NASA, NSF, FAA and Intel.

References

1. Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc. 8th Int. Conf. on Database Theory (ICDT)*, pages 420–434, 2001.
2. N. Carlini, G. Katz, C. Barrett, and D. Dill. Ground-Truth Adversarial Examples, 2017. Technical Report. `http://arxiv.org/abs/1709.10207`.
3. Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proc. 38th IEEE Symposium on Security and Privacy*, 2017.

4. R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, 2017.
5. Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. Detecting adversarial samples from artifacts, 2017. Technical Report. `http://arxiv.org/abs/1703.00410`.
6. Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014. Technical Report. `http://arxiv.org/abs/1412.6572`.
7. Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.
8. K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Proc. 35th Digital Avionics System Conf. (DASC)*, pages 1–10, 2016.
9. Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.
10. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.
11. G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Towards proving the adversarial robustness of deep neural networks. In *Proc. 1st Workshop on Formal Verification of Autonomous Vehicles (FVAV)*, pages 19–26, 2017.
12. Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.
13. Nicolas Papernot and Patrick D. McDaniel. On the effectiveness of defensive distillation, 2016. Technical Report. `http://arxiv.org/abs/1607.05113`.
14. Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. 1st IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2016.
15. L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Proc. 22nd Int. Conf. on Computer Aided Verification (CAV)*, pages 243–257, 2010.
16. L. Pulina and A. Tacchella. Challenging SMT solvers to verify neural networks. *AI Communications*, 25(2):117–135, 2012.
17. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2013. Technical Report. `http://arxiv.org/abs/1312.6199`.