Quality Guarantees for Autoencoders via Unsupervised Adversarial Attacks

Benedikt Böing[1], Rajarshi Roy[2], Emmanuel Müller[1], and Daniel Neider[2]

[1] University of Bonn, {boeing,mueller}@bit.uni-bonn.de
[2] Max Planck Institute for Software Systems, {rajarshi,neider}@mpi-sws.org

Abstract. Autoencoders are an essential concept in unsupervised learning. Currently, the quality of autoencoders is assessed either internally (e.g. based on mean square error) or externally (e.g. by classification performance). Yet, there is no possibility to prove that autoencoders generalize beyond the finite training data, and hence, they are not reliable for safety-critical applications that require formal guarantees also for unseen data.
To address this issue, we propose the first framework to bound the worst-case error of an autoencoder within a safety-critical region of an infinite value domain, as well as the definition of unsupervised adversarial examples that cause such worst-case errors. Technically, our framework reduces the infinite search space for a uniform error bound to checking satisfiability of logical formulas in Linear Real Arithmetic. This allows us to leverage highly-optimized SMT solvers, a strategy that is very successful in the context of deductive software verification. We demonstrate our ability to find unsupervised adversarial examples as well as formal quality guarantees both on synthetic and real-world data.

1 Introduction

Autoencoders are widely used for many unsupervised learning tasks such as cluster analysis [4], compression [14], anomaly detection [18], as well as a variety of pre-processing steps [10, 14, 17] in other machine learning pipelines. The general assumption is that data can be compressed into a lower dimensional latent space by an encoder function extracting the most relevant features of the data distribution. From this latent representation the decoder tries to reconstruct the original input. As the latent representation is an information bottleneck the autoencoders input deviates from its output. Typically the autoencoder reconstructs better in dense regions (i.e. regions with many training examples) than in regions with few training examples [18] giving rise to its application in anomaly detection. Moreover even the small errors in dense regions are a desirable property as they allow it to be used e.g. for denoising. At the same time it is necessary to control the error for all points in dense regions because otherwise the result - whether it is the latent representation or the reconstruction - is less useful. To this end current approaches to assess autoencoders either measure internally the mean square error (MSE) on the unsupervised training data or external performance

on some supervised application such as classification performance.
However, a major shortcoming of these approaches is that they cannot provide a formal guarantee in terms of the maximum deviation between input and output of the autoencoder as it is evaluated on training data only (i.e. with a finite number of inputs). We are not aware of any existing scheme to calculate the largest error of an autoencoder in an infinite input space. This lack of formal quality guarantees for autoencoders leads to a very limited applicability of such unsupervised learning schemes for safety-critical applications. For instance, it is particularly important to consider the maximum deviation when working with data containing clusters. In such situations the autoencoder should not mix up the clusters because otherwise the autoencoders results are meaningless. If the maximum deviation for the respective clusters are small enough though, the autoencoder is guaranteed to keep the clusters separated.
To address this and other shortcomings of unsupervised learning with autoencoders, we provide the first methodology to bound an autoencoder's worst-case error in a safety-critical region. As a first step towards this goal we define the notion of *unsupervised adversarial examples* which are inputs (not necessarily contained in the training data) on which the autoencoder's error exceeds a user-defined threshold. Then we define the worst-case error of an autoencoder as the largest error that can possibly manifest. Since we cannot expect to find a global maximum of the error (as there is no reason for the error itself to be bounded), we restrict our search to user-defined regions with an infinite value domain of the input space. We leave this region as a parameter to be provided by the user as it clearly depends on knowledge about the use case at hand, characteristics of the training data, or other domain-specific information.
Following a popular approach in the area of software verification, we reduce the problem of finding an unsupervised adversarial example to a satisfiability check of a formula in Real Arithmetic. This allows us to apply highly-optimized, off-the-shelf satisfiability modulo theory (SMT) solvers which can effectively reason about the infinite domains and, hence, can prove the existence or non-existence of unsupervised adversarial examples. Once we have found an unsupervised adversarial example, it serves as a lower bound for the worst-case error. Moreover, a simple binary search allows us to approximate worst-case error arbitrarily well. Note that naive approaches, such as sampling, cannot provide an upper bound on the worst-case error of an autoencoder as an exhaustive search of the input space is intractable. Moreover, our experimental evaluation shows that sampling often underestimates that worst-case error.
We demonstrate the effectiveness of our QUGA (QUality Guarantees for Autoencoders) approach and evaluate our quality guarantees for unsupervised learning on a synthetically created dataset as well as on a real dataset. In both cases we can find unsupervised adversarial examples as well as formal quality guarantees by lower and upper error bounds in safety-critical regions.

2 Related Work

Adversarials in Supervised vs. Unsupervised Learning. In the area of supervised learning, adversarial attacks have been widely studied [11, 7, 20]. While common definitions of adversarial attacks rely on the robust separation of class labels, we aim at unsupervised learning without given labels. Therefore supervised definitions do not cover the unsupervised learning case. Similarly existing approaches of adversarial attacks in unsupervised learning focus on a particular task such as clustering [6] or image retrieval [22] assuming that there is a notion of a wrong output. In contrast to these approaches we define adversarial attacks directly in terms of the intrinsic learning objective of autoencoders which is - as reflected by the loss function - approximating the identity function.
Empirical Quality Assessment vs. Formal Guarantees. Common evaluation schemes for autoencoders do an empirical quality assessment based on a given set of training data. The variety of quality measures ranges from simple average MSE to stability and robustness measures [13, 21, 15]. All of these measures have in common that they rely on the given training data. In contrast to such empirical evaluation, many safety-critical applications require formal guarantees explicitly also on unseen data. We propose such formal guarantees for trained autoencoders. Given a safety-critical data region, our method is able to either find an adversarial example or prove that such an example does not exist.
External vs. Internal Evaluation. Common external evaluation uses, for example, the classification quality of a down-stream step after the autoencoder as indirect measure of quality for the autoencoder. As such evaluation of multiple tasks is prone to the mix-up of fluctuating quality of individual tasks and dependency effects between these tasks. We belief that the modular evaluation of individual tasks is an additional requirement for safety-critical systems. Such a design-by-contract has been successfully established in modular software verification [3]. Similarly, we propose the first formal guarantee of an autoencoder (i.e., an upper bound on the maximal error on the entire data domain).
Verification of Neural Networks. Our work is related to formal methods and verification of neural networks in general (e.g., see [2, 12]). However, most of the research in this area focuses on the problem of finding adversarial examples in supervised learning tasks and lacks formal insights for unsupervised learning. In contrast, our algorithm searches for unsupervised adversarial examples. It does so by reducing the problem to a series of satisfiability checks in a Real Arithmetic and applies a highly-optimized Satisfiability Module Theories (SMT) solver as computational back-end to perform these checks. We have implemented a prototype of our algorithm on top of the Z3 SMT solver [16] which provides a convenient API and is one of the most popular tool in the domain of software verification. For extremely large, real-world scenarios, however, one would clearly use a solver that is optimized for constraints arising from feed-forward networks, such as ReluPlex [12] or Planet [8].

Apart from constraint solving, other techniques from the area of deductive software verification have been used for finding adversarial examples in supervised learning and proving robustness properties of feed-forward neural nets. The per-

haps most popular approach is abstract interpretation [9, 19]. However, abstract interpretation inherently overapproximates the behavior of the neural network and, hence, can only be used to prove safety properties. However, neither our unsupervised adversarial examples nor our worst-case error of an autoencoder can be achieved by their safety properties.

3 *QUGA*: Problem Statement

In general, an autoencoder tries to reproduce its input while propagating it through a latent space which typically has less dimensions than the input/output space. This latent space serves as an information bottleneck and, hence, introduces errors to the identity function the autoencoder is supposed to learn. However, most applications of autoencoders rely on a good approximation of the identity function, and we are naturally interested in quantifying its error. More precisely, our goal is to give formal guarantees in terms of the maximum deviation from the identity function.

As a first step towards this goal, we define the notion of *adversarial examples of autoencoders*. Intuitively, such adversarial examples are inputs on which the "distance" between the input and the output of the autoencoder is larger than a (user-defined) threshold $\varepsilon > 0$. Given the lack of definitions for adversarials in unsupervised learning (and in particular for autoencoders), we define adversarial example based on an abstract distance function *dist* which maps two data points to a non-negative real number. However, we stress that the exact distance function is not important for our definition (e.g., any L_p-norm could be used) because all autoencoders share the goal of reconstructing the input.

Definition 1 (ε-adversarial examples). *Let $f\colon \mathbb{R}^n \to \mathbb{R}^n$ be an autoencoder, $dist\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ a distance function, and $\varepsilon > 0$. An* ε-adversarial example *is a point $x \in \mathbb{R}^n$ such that*

$$dist(x, f(x)) > \varepsilon$$

(i.e., a point on which the input and output of f deviate more than ε).

Note that our definition of ε-adversarial examples is not restricted to inputs in the training or test sets but allows any input $x \in \mathbb{R}^n$. This property makes finding ε-adversarial examples a very challenging task, and in contrast to traditional internal evaluation (e.g., mean square error) on training data, searching adversarial examples is a computationally hard problem.

In the context of safety-critical systems, however, it is not enough to identify individual ε-adversarial examples, but it is necessary to know the worst-case (i.e., maximum) error an autoencoder produces. Of course, we cannot expect to find a global maximum of the error as there is no reason for the error itself to be bounded. Therefore, we restrict the region for which we want to find a bound on the error. This region depends on knowledge about the use case at hand, characteristics of the training data, or other domain-specific information. Thus, we leave it as a parameter to be provided by the user.

Definition 2 (Worst-case error of autoencoders). *Let $f\colon \mathbb{R}^n \to \mathbb{R}^n$ be an autoencoder, $dist\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ a distance function, and $A \subseteq \mathbb{R}^n$ an (infinite) safety-critical region of inputs. Then, the* worst-case error of f in A *is defined as*

$$wce(f, A) = \sup \left\{ dist\big(x, f(x)\big) \in \mathbb{R}_+ \mid x \in A \right\}$$

(i.e., the largest deviation of an input in the region A from the output).

Definition 2 serves as our novel *quality criterion* for autoencoders that reflects how good the identity function is learned in the specific region of interest. Our *wce*-definition is inspired by many areas of reliable system design, including soft- and hardware verification, as $wce(f, A)$ guarantees that a system f employed in a safety-critical region A stays within its design parameters. Furthermore, our notion of *wce* overcomes limitations of classical quality metrics that are defined on finite training data only. We actively design *wce* for typically infinite data domains of safety-critical regions. In total, this leads us to the main problem statement, which we call *QUGA: QUality Guarantees for Autoencoders*.

Problem 1 (QUGA: Quality Guarantees for Autoencoders). Given an autoencoder $f\colon \mathbb{R}^n \to \mathbb{R}^n$, a distance function $dist\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$, and a region $A \subseteq \mathbb{R}^n$, compute $wce(f, A)$.

In general, computing the worst-case error is a very challenging problem as it involves reasoning about an infinite number of inputs (not just training data) and does not make any assumption on the autoencoder, the distance function or the region. In the following section, we consider a restricted version of Problem 1 and show how a reduction to a series of constraint solving can be used to answer this restriction.

4 Solution Framework

In this section, we provide a framework for computing ε-adversarial examples and the worst-case error of autoencoders. To make these problems computationally tractable, we consider a restricted version of Problem 1. The following restrictions are designed in such a way that the solution framework remains applicable to a wide range of autoencoders used in practice:

1. We assume the the neurons of the autoencoder have linear or ReLU (Rectified Linear Units) activation functions.
2. We assume the distance function to be the L_1 or L_∞-norm.
3. We assume the safety-critical region A to be a finite union of convex compact polytopes (i.e. each polytope is an intersection of half-spaces of the $\mathbb{R}^n$).
4. We approximate the worst-case error up to a user-defined accuracy because our framework can find ε-adversarial examples for fixed ε only.

In the remainder of this section, we formally introduce autoencoders (Section 4.1) and the Satisfiability Modulo Theories (SMT) framework (Section 4.2).

In Section 4.2, we then show how the existence of an ε-adversarial example can be phrased as a satisfiability problem in Linear Real Arithmetic, one of the theories supported by the SMT framework. This allows us to use highly-optimized SMT solvers to do a symbolic search on the (potentially infinite) input space. In Section 4.4, we finally provide an effective method to approximate the worst-case error of an autoencoder by repeatedly solving the easier problem of determining the existence of ε-adversarial attacks for different values of ε.

4.1 Autoencoders

Intuitively, an *autoencoder*—like most feed-forward networks—is a collection of neurons (or nodes) arranged sequentially in layers. Each neuron (except input neurons) is connected to neurons of the previous layer by edges carrying weights (e.g., see Figure 1 on Page 9). Functionally, an autoencoder evaluates a function $f : \mathbb{R}^n \to \mathbb{R}^n$, where internally it uses the neurons to propagate information through its layers.
For an autoencoder f, we use N to denote the number of layers and $l_k \in \mathbb{N} \setminus \{0\}$ with $k \in \{1, \dots, N\}$ to represent the number of neurons in Layer k. Layer 1 is called *input layer*, Layer N is called *output layer*, and the remaining layers are called *hidden layers*. The *topology* of an autoencoder is a tuple $(l_1, l_2, \dots, l_N)$ denoting the number $l_k \in \mathbb{N} \setminus \{0\}$ of neurons in each layer $k \in \{1, \dots, N\}$. In contrast to general feed-forward networks, autoencoders have an equal number $n \in \mathbb{N} \setminus \{0\}$ of neurons in the input and output layers (i.e., $l_1 = l_N = n$). Furthermore there is a special layer that separates the autoencoder into the encoder and the decoder. The neurons in this layer span the latent space which typically has less dimensions than the input space.
For each layer $k \in \{2, \dots, N\}$, the autoencoder has a weight matrix W^k of dimension $l_{k-1} \times l_k$, containing the weights of the connections between layer $k-1$ and k. Moreover, each layer has a so-called *bias vector* $b_k \in \mathbb{R}^{l_k}$ associated with it that contains the bias for each neuron in Layer k.
The output of each neuron is calculated by taking a linear combination of the output of the previous layer and then applying an *activation function* (typically a non-linear function) on the result. We consider the linear (trivial) and the ReLU activation function, giving rise to linear and ReLU neurons. The output of Neuron j in Layer k is then $x_{k,j} = \sum_{i=1}^{l_{k-1}} x_{k-1,i} W^k_{i,j} + b^k_j$ if it is a linear neuron and $x_{k,j} = \max\{0, \sum_{i=1}^{l_{k-1}} x_{k-1,i} W^k_{i,j} + b^k_j\}$ if it is a ReLU neuron.

4.2 Satisfiability Modulo Theories (SMT)

Various problems in computer science, especially in the area of formal verification, can be solved by reducing them to constraint satisfaction problems in a suitable logic. Although propositional logic is a popular choice for many such problems, some of them require a more expressive logic: first-order logic. A formula in first-order logic is formed using constants, variables, function and predicate symbols, logical connectives, and quantifiers. In this paper, however, we require only a

specific first-order logic, namely the the *quantifier-free fragment of linear real arithmetic (LRA)*, which we introduce next.
First, let $\mathcal{X} = \{x_0, x_1, \ldots\}$ be a set of *variables* which range over values in $\mathbb{R}$. Then, we define *terms* as follows: a term is either a constant $c \in \mathbb{R}$, a variable $x \in \mathcal{X}$, or a function application $t_1 \circ t_2$, where $\circ \in \{+, \cdot\}$ and t_1, t_2 are two terms. For instance, 5, x, and $3 \cdot x + 2 \cdot y$ are terms. To reflect the usual notation, we often drop the multiplication sign.
An *atomic formula* is a predicate symbol applied to terms. In LRA, we allow the usual binary predicates $<$, $\leq$, $=$, $\geq$, and $>$. For example, $3x + 2y > 5$ is an atomic formula. Moreover, a *formula* is inductively defined as follows: a formula is either an atomic formula, the negation $\neg\varphi$ of a formula φ, or the disjunction $\varphi_1 \vee \varphi_2$ of two formulas φ_1, φ_2. We also add syntactic sugar and allow the formulas $\varphi_1 \wedge \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, and $\varphi_1 \leftrightarrow \varphi_2$, which are defined as usual.
To assign meaning to formulas, we introduce the concept of *interpretations*. An interpretation is a mapping $\mathcal{I} \colon \mathcal{X} \rightarrow \mathbb{R}$ which assigns to each variable a real value. Interpretations can easily be lifted to terms in the usual way, and we write $\mathcal{I}(t)$ for the interpretation (i.e., the value) of the term t under $\mathcal{I}$. Finally, we can define when an interpretation $\mathcal{I}$ *satisfies* a formula φ which we denote by $\mathcal{I} \models \varphi$: we have $\mathcal{I} \models t_1 \diamond t_2$ for $\diamond \in \{<, \leq, =, \geq, >\}$ if and only if $\mathcal{I}(t_1) \diamond \mathcal{I}(t_2)$ is true, $\mathcal{I} \models \neg\varphi$ if $\mathcal{I} \not\models \varphi$, and $\mathcal{I} \models \varphi_1 \vee \varphi_2$ if and only if $\mathcal{I} \models \varphi_1$ or $\mathcal{I} \models \varphi_2$. We say that a formula φ is *satisfiable* if an interpretation $\mathcal{I}$ with $\mathcal{I} \models \varphi$ exists.

Highly-optimized procedures for deciding satisfiability of formulas in LRA have been implemented in a framework called *Satisfiability Modulo Theories (SMT)* [1], which not only allows to check the satisfiability of formulas in LRA but also in many other (usually quantifier-free) fragments of first-order logic, called theories. Moreover, SMT solvers typically return an interpretation if the given formula is satisfiable. In the following, we exploit this property for our search for ε-adversarial attacks and to approximate the worst-case error of an autoencoder.

4.3 Identifying ε-Adversarial Examples

Let us now describe how to translate the problem of finding an ε-adversarial example of an autoencoder f into LRA. At its core is a formula φ^f that encodes the function computed by f in LRA. Moreover we add further constraints in the form of formulas φ^A and $\varphi_\varepsilon^{dist}$ which encode the input region A and the distance function (including the existence of an ε-adversarial example), respectively. The resulting encoding is then the conjunction $\varphi_\varepsilon^{\text{ae}} := \varphi^f \wedge \varphi^A \wedge \varphi_\varepsilon^{dist}$ which is satisfiable if and only if an ε-adversarial example exists. Moreover, a satisfying interpretation of $\varphi_\varepsilon^{\text{ae}}$ carries sufficient information to extract such an ε-adversarial example. Let us now describe these formulas in detail.

Encoding the Autoencoder: To encode the function computed by an autoencoder f in LRA, we introduce variables $x_{k,j}$ for each layer $k \in \{1, \ldots, N\}$ and each neuron $j \in \{1, \ldots, l_k\}$ in Layer k. Intuitively, each such variable captures the

output of a neuron and is used as the input for other neurons. Correspondingly, variables $x_{1,1}, \ldots, x_{1,l_1}$ represent the input to the autoencoder, while variables $x_{N,1}, \ldots, x_{N,l_N}$ represents the output of the autoencoder. To ensure that the variables $x_{k,j}$ actually have the desired meaning, we introduce constraints that describe the computation of each neuron. For a linear neuron we construct

$$\psi_{k,j} \coloneqq \Big[x_{k,j} = \big[\sum_{i=1}^{l_{k-1}} W_{i,j}^k x_{k-1,i}\big] + b_j^k\Big].$$

On the other hand, for a ReLU neuron, we construct the constraint

$$\psi_{k,j} \coloneqq \Big[\big[S_{k,j} = \sum_{i=1}^{l_{k-1}} W_{i,j}^k x_{k-1,i} + b_i^k\big] \wedge \big[x_{k,j} = \mathsf{ite}(S_{k,j} < 0, 0, S_{k,j})\big]\Big],$$

where ite (short for "if-then-else") is syntactic sugar for a conditional evaluation of terms, which is supported by virtually all SMT solvers.
Finally, we define

$$\varphi^f \coloneqq \bigwedge_{2 \leq k \leq N} \bigwedge_{1 \leq j \leq l_k} \psi_{k,j},$$

which collects the constraints for all individual neurons. By construction φ^f completely encodes the autoencoder f in the sense that $f\big(\mathcal{I}(x_{1,1}), \ldots, \mathcal{I}(x_{1,l_1})\big) = \big(\mathcal{I}(x_{N,1}), \ldots, \mathcal{I}(x_{N,l_N})\big)$ holds for all satisfying interpretations $\mathcal{I} \models \varphi^f$.

Encoding the Region: Recall that we assume that the safety-critical region A in which to search for ε-adversarial examples is provided as a finite union of compact convex polytopes. Formally, a convex polytope $\mathcal{P}$ is the finite intersection of half-spaces $\mathcal{H}_i$ of the form $\sum_{j=1}^{l_1} a_{i,j} x_j \leq c_i$ for $a_{i,j}, c_i \in \mathbb{R}$, and we write $A = \{\mathcal{P}_1, \ldots, \mathcal{P}_\ell\}$ for the sake of brevity. Thus, restricting the search space for ε-adversarial examples to a convex polytope $\mathcal{P}$ consisting of m half-spaces can simply be achieved by the formula

$$\psi_{\mathcal{P}} \coloneqq \bigwedge_{1 \leq i \leq m} \Big[\sum_{1 \leq j \leq l_1} a_{i,j} x_{1,j} \leq c_i\Big].$$

Moreover, the final formula is then the disjunction $\varphi^A \coloneqq \bigvee_{\mathcal{P} \in A} \psi_{\mathcal{P}}$ for all polytopes $\mathcal{P}$ constituting to the given region A.

Encoding the Existence of an ε-Adversarial Attack: It is left to encode the distance function *dist* as well as the existence of an ε-adversarial example. In the interest of space, however, we only show the encoding of the L_∞-norm. Encoding the L_1-norm is only slightly more complicated and can be expressed using a summation over ite-terms. In principle, it is even possible to encode L_p-norms in SMT for arbitrary $p \geq 1$, but the complexity of the underlying decision procedures for non-linear Real Arithmetic is prohibitively high.

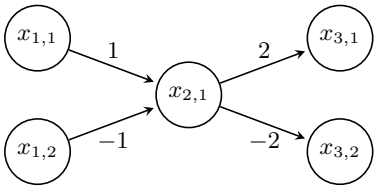

Fig. 1. A short example of an Autoencoder.

In the L_∞-norm, an input is an ε-adversarial example if there exists a dimension $i \in \{1, \dots, l_1\}$ in which the absolute value of the difference of the input output in this dimension is larger than ε. This can be expressed in LRA by

$$\varphi_\varepsilon^{dist} \coloneqq \bigvee_{1 \le i \le l_1} \Big[[x_{1,j} - x_{N,j} > \varepsilon] \vee [x_{N,j} - x_{1,j} > \varepsilon] \Big].$$

Before we continue with the final formula $\varphi_\varepsilon^{\text{ae}}$, let us briefly illustrate the constraints generated so far using an example.

Example 1. Consider the simple autoencoder (with ReLU-activation) in Figure 1, consisting of two neurons in the input layer, one neuron in the single hidden layer, and two neurons in the output layer. Moreover, assume that we are given one polytope $\mathcal{P}$ consisting of the intersection of four half-spaces $-1 \le x$, $x \le 1$, $-1 \le y$, and $y \le 1$ (i.e., a unit box around the origin). Then, the formulas φ^f, φ^A, and $\varphi_\varepsilon^{dist}$ are given by

$$\begin{aligned}
\varphi^f &\coloneqq [S_{2,1} = x_{1,1} + (-1)x_{1,2}] \wedge [x_{2,1} = \mathsf{ite}(S_{2,1} < 0, 0, S_{2,1})] \wedge \\
&\quad\ [x_{3,1} = 2x_{2,1} \wedge x_{3,2} = (-2)x_{3,2}], \\
\varphi^A &\coloneqq x_{1,1} \le 1 \wedge x_{1,1} \ge -1 \wedge x_{1,2} \le 1 \wedge x_{1,2} \ge -1, \\
\varphi_\varepsilon^{dist} &\coloneqq [x_{1,1} - x_{3,1} > \varepsilon \vee x_{3,1} - x_{1,1} > \varepsilon] \vee [x_{1,2} - x_{3,2} > \varepsilon \vee x_{3,2} - x_{1,2} > \varepsilon].
\end{aligned}$$

Finally, we combine all constraints generated so far into a single formula $\varphi_\varepsilon^{\text{ae}} \coloneqq \varphi^f \wedge \varphi^A \wedge \varphi_\varepsilon^{dist}$. As the next theorem states, this formula indeed expresses the existence of an ε-adversarial example of the autoencoder f in the region A.

Theorem 1. *Let f be an autoencoder, A a region, dist a distance function, $\varepsilon > 0$, and φ_ε^{ae} as defined above. Then, the following two properties hold:*

1. *If an ε-adversarial example exists, then φ_ε^{ae} is satisfiable.*
2. *If φ_ε^{ae} is satisfiable, say by the interpretation $\mathcal{I}$, then $\big(\mathcal{I}(x_{1,1}), \dots, \mathcal{I}(x_{1,l_1})\big)$ is an ε-adversarial example.*

Theorem 1 now suggests a simple procedure to find ε-adversarial examples: simply construct $\varphi_\varepsilon^{\text{ae}}$, run an SMT solver, and return $\big(\mathcal{I}(x_{1,1}), \dots, \mathcal{I}(x_{1,l_1})\big)$ if a satisfying assignment $\mathcal{I} \models \varphi_\varepsilon^{\text{ae}}$ exists. However, the SMT solver might report that $\varphi_\varepsilon^{\text{ae}}$ is unsatisfiable. In this case, Theorem 1 guarantees that no ε-adversarial example exists. The proof of Theorem 1 can be found in the supplementary material[3]. We exploit this property now to approximate the worst-case error of an autoencoder.

Algorithm 1: Computing *wce* up to accuracy δ

Input: Autoencoder f, Region A, distance function *dist*, start value $\varepsilon_0 > 0$, accuracy $\delta > 0$

1 $\varepsilon_{\text{low}} = \varepsilon_{\text{up}} = \varepsilon_0$
2 Construct $\varphi^{\text{ae}}_{\varepsilon_0}$ and check satisfiability using an SMT solver
3 **if** $\varphi^{ae}_{\varepsilon_0}$ *is satisfiable* **then**
4 | Increase ε_{up} by $\varepsilon_{\text{up}} * 2$ until $\varphi^{\text{ae}}_{\varepsilon_{\text{up}}}$ becomes unsatisfiable
5 **else**
6 | Decrease ε_{low} by $\varepsilon_{\text{low}}/2$ until $\varphi^{\text{ae}}_{\varepsilon_{\text{low}}}$ becomes satisfiable or $\varepsilon_{\text{low}} < \delta$ (in which case **return** ε_{low})
7 **end**
8 $\varepsilon^\star \leftarrow \text{Binary-search}_{f,A,dist}(\varepsilon_{\text{low}}, \varepsilon_{\text{up}}, \delta))$ `// involves calls to SMT solver`
9 **return** $\varepsilon^\star$

4.4 Approximating the Worst-Case Error

We now provide an algorithm for approximating the worst-case error of an autoencoder. Our algorithm, which is sketched in pseudocode as Algorithm 1, is based on the method for finding ε-adversarial examples from Section 4.3. Apart from the autoencoder itself, the safety-critical region, and a distance function, it expects two additional arguments: a start value $\varepsilon_0 > 0$ for the search and an accuracy value $\delta > 0$. The start value ε_0 is used as an initial estimate for $wce(f, A)$ and can be either initialized arbitrarily or based on domain knowledge. The accuracy, on the other hand, is a measure of how close the output of Algorithm 1 is to the actual value of $wce(f, A)$. A smaller δ results in a more precise approximation of $wce(f, A)$, but it also increases the computation time.

Algorithm 1 uses a binary search to find a sufficiently close approximation of $wce(f, A)$ (see line 8). To this end, it uses two values $\varepsilon_{\text{low}} < \varepsilon_{\text{up}}$ for which it maintains the invariant that (a) there exists an ε_{low}-adversarial example and (b) there does not exist an ε_{up}-adversarial example in the given region. Hence, $wce(f, A)$ is guaranteed to lie in the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{up}}]$. The initial values for ε_{low} and ε_{up} are obtained by starting with ε_0 and increasing ε_{up} or decreasing ε_{low} until the invariant is established (see lines 1 to 7). Subsequently, the binary search then repeatedly runs the procedure for finding ε-adversarial examples and updates the bounds ε_{low} and ε_{up} accordingly. Algorithm 1 stops once the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{up}}]$ is small enough (i.e. less than 2δ). In summary, Algorithm 1 provides an effective procedure to compute the worst-case error of an autoencoder up to a user-defined accuracy $\delta > 0$, as formalized in the theorem below.

Theorem 2. *Let f be an autoencoder, A a region, and $\delta > 0$. Then, Algorithm 1 terminates eventually and outputs a value $\varepsilon^\star \in [wce(f, A) - \delta, wce(f, A) + \delta]$.*

Theorem 2 follows from Theorem 1 and the fact that the binary search of Algorithm 1 narrows down the interval $[\varepsilon_{\text{low}}, \varepsilon_{\text{up}}]$ until it is smaller than 2δ. The latter fact also implies the termination of Algorithm 1.

The complexity of Algorithm 1 consists of two parts: the binary search and the SMT solver. The number of steps in the binary search is in $\mathcal{O}(log(\frac{wce(f,A)}{\delta}))$. In each step the SMT solver is called once with a runtime that mainly depends on the number of atomic formulas in (the respective) $\varphi_{\varepsilon}^{\text{ae}}$. Under the restrictions in Section 4 there are $\mathcal{O}(n+m)$ many atomic formulas where n is the number of neurons in the autoencoder and m is the number of halfspaces used to construct the safety-critical region. Note that the number of atomic formulas arising from the L_1 and L_∞ distance depends linearly on the dimension of the input/output space of the autoencoder and is hence in $\mathcal{O}(n)$. Even though encoding the problem as a formula is inexpensive, the SMT solver itself is an exponential algorithm as is relies on solving instances of the NP-complete SAT problem.

5 Empirical Evaluation

We evaluate both concepts presented within our QUGA solution: (1) extracting an adversarial example and (2) calculation of quality bounds. For evaluation we use both synthetic and real-world data. For future comparison and reproducibility of our experiments we provide our implementation[3] with the off-the-shelf SMT solver Z3. As Z3 is not specialised for neural nets, our approach is not scalable enough to deal with benchmark datasets such as MNIST or CIFAR-10. This is one of the most urgent challenges we intend to tackle in future work.

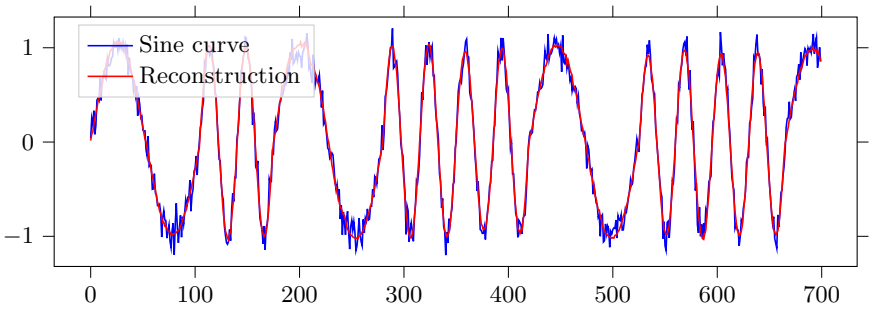

Fig. 2. Synthetic sine curve with two frequencies and noise with its reconstruction by the autoencoder.

5.1 Experiment Setup

We use synthetic time series generated by sine curves with two different frequencies (35 and 105) and random gaussian noise ($\sigma = 0.1$) per time-point. Additionally, we use ECG5000 data from the UCR time series repository [5]. We train autoencoders

[3] `https://github.com/KDD-OpenSource/QUGA`

with a topology of $L = (35, 5, 35)$ with 5 hidden ReLU nodes and 35 linear output nodes using the MSE loss function. Training data consists of time windows of length 35 without overlap. For the sine curve the time windows correspond to 4 clusters: The full sine curve and the beginning, the middle and the end of the large sine curve. We denote them by $C_{full}, C_{beg}, C_{mid}$ and C_{end} respectively. For the ECG5000 dataset, we obtain 8 clusters arising from 2 classes and 4 time windows. We call them C_{i_x} where $i \in \{1, 2, 3, 4\}$ and $x \in \{u, b\}$ indicating the upper or lower part of the respective time window. As critical region A we evaluate a box around the two sine curves with width 0.2 in every dimension. This region contains by construction the majority of training data. For the ECG5000 dataset we extract representative time series for the two main classes and add a margin of 0.25. We visualize the regions along with the training data in Figure 3.

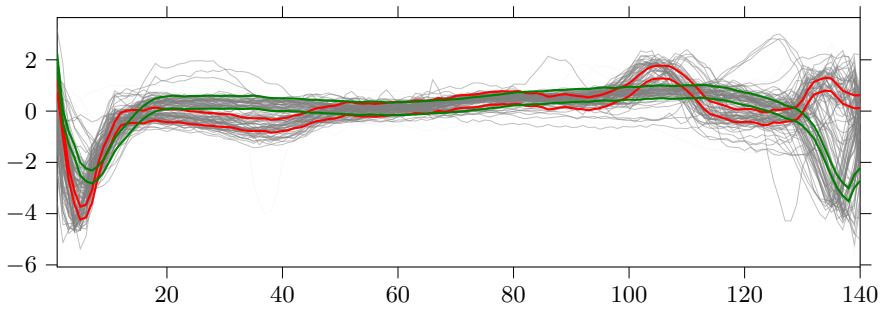

Fig. 3. ECG5000 dataset with two safety-critical regions (red and green) obtained by extracting prototypes for two classes and adding a margin of 0.25.

5.2 Extracting an Adversarial Example

The first observation is that our QUGA approach successfully extracts adversarial examples. We depict the adversarial examples obtained in Figure 4 for the sine curve dataset and in Figure 5 for the two safety-critical regions in the ECG5000 Dataset. Ideally an autoencoder should extract a denoised version of the input. With the adversarial examples we have an indication whether the autoencoder succeeds in doing so. For the sine curve the outputs of the autoencoder on adversarial examples in C_{full}, C_{mid} and C_{end} are much smoother than for the adversarial example in C_{beg}, suggesting that the autoencoder does not denoise as well in C_{beg}. For the ECG5000 dataset the autoencoder seems to denoise for all clusters very well.

5.3 Comparing Quality Bounds with Sampling

We compare the quality bounds obtained by the QUGA approach with accuracy 0.025 to quality bounds obtained by a simple sampling approach. As a competitor

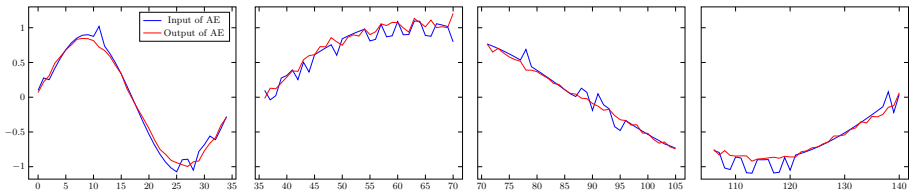

Fig. 4. Adversarial examples for different parts of the sine curve dataset obtained by the QUGA approach maximizing the L_∞-distance between the input and the output of the autoencoder in the respective safety-critical region. The adversarial example on the second plot from the left indicates that this part is denoised less.

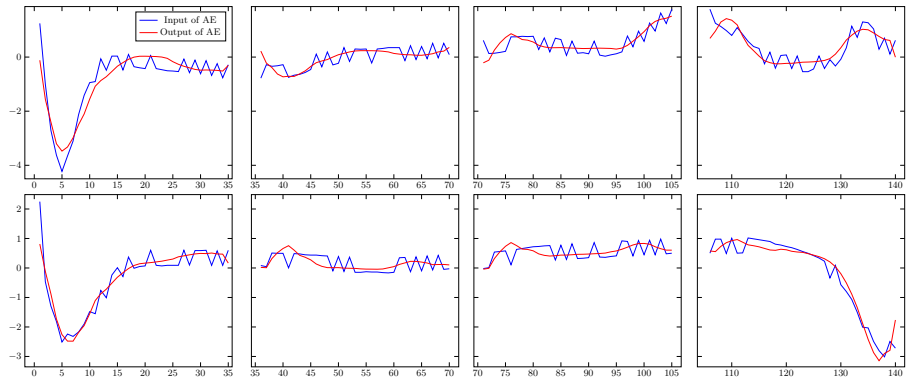

Fig. 5. Adversarial examples for two classes and different time windows of the ECG5000 dataset obtained by the QUGA approach maximizing the L_∞-distance between the input and the output of the autoencoder in the respective safety-critical region. No difference in denoising quality between the different plots can be seen.

to the QUGA approach we sample points in the region, calculate their L_∞ errors and take the maximum as an estimator for the $L_\infty - wce$. Table 1 sums up the results. First of all note, that the QUGA $L_\infty - wce$ bounds are much more precise. The $L_\infty - wce$ bound obtained by the sampling approach yields no upper bound at all, and furthermore the lower bound is much weaker than the lower bound obtained by QUGA in all cases. A clear drawback of sampling is the large amount of samples required to reach our QUGA estimation. In Figure 6 we show runtime of QUGA vs. sampling with their respective error estimations. QUGA as a systematic search scheme is more efficient, while sampling is shown to underestimate worst-case errors.

5.4 Safety Critical Application

We demonstrate our QUGA framework on the ECG5000 dataset, by evaluating the unsupervised training based on two time series clusters. The goal of a traditional evaluation would be to show that all training objects are clearly separated in

	sine curve				*ECG*							
Cluster	C_{full}	C_{beg}	C_{mid}	C_{end}	C_{1_b}	C_{2_b}	C_{3_u}	C_{4_u}	C_{1_u}	C_{2_u}	C_{3_b}	C_{4_b}
QUGA	0.297	0.422	0.297	0.266	1.359	1.016	0.828	1.078	1.453	0.766	0.766	0.953
Sampling	0.211	0.255	0.214	0.189	1.189	0.829	0.651	0.908	1.255	0.563	0.546	0.774

Table 1. Worst-case errors as estimated by sampling and QUGA approach for the sine curve and ECG datasets. The accuracy for the QUGA approach is 0.025.

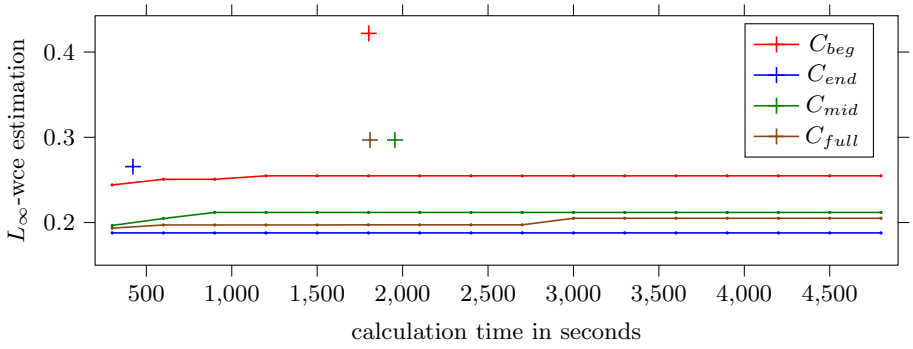

Fig. 6. Depiction of runtime against error estimation for the sine curve dataset. Plus signs indicate the results by QUGA. Lines indicate results by sampling.

the latent space. In contrast, we care about all possible (infinitely many objects) in two safety-critical areas that need to be distinguishable in the latent space. In Figure 7 we see the resulting corridor into which points from the critical regions can be mapped. For the first three time windows we cannot guarantee that the autoencoder keeps points from the two regions distinguishable in the latent space. Both clusters mix-up as the upper bound of the lower cluster is higher than the lower bound of the upper cluster. For the last 35 time steps though a guaranteed separation of all infinitely many points in the critical regions is possible by the autoencoder. With this result we can give a formal quality guarantee of the trained autoencoder. It securely extracts a latent representation for each time series in the safety-critical area that guarantees separation of both clusters. Please note that one could not have used the latent space to check separability directly. We have no control over where the autoencoder maps the safety-critical regions in the latent space. In contrast our QUGA method solves this by symbolic representation of the autoencoder, and the systematic search of possible unsupervised adversarial examples that lead to a mix-up of two clusters. With this we can prove separability for all infinite points in the safety-critical regions and not just on the finite training set.

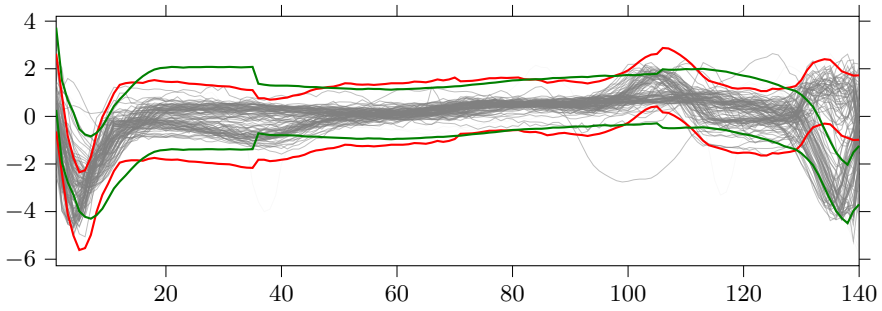

Fig. 7. Image spaces (red and green) into which points from the respective safety-critical regions in Figure 3 can theoretically be mapped by the autoencoder.

6 Conclusion and Future Work

QUGA overcomes major shortcomings of unsupervised learning with autoencoders. We provide the first methodology to bound the error of an autoencoder in a safety-critical region. With our solution framework based on SMT solvers we propose to search for adversarial examples in the infinite search space of a safety-critical region. Therefore, we have defined *unsupervised adversarial examples* as inputs that show maximal error even if these objects are not contained in the training data. Our QUGA approach formulates the autoencoder, the safety-critical region, and the error of the loss function with a logical conjunction of linear constraints. Once we have found an unsupervised adversarial example, it serves as a lower bound for the error while binary search allows to derive an upper bound. We demonstrate the effectiveness of our approach on both synthetically created and real dataset. We show that QUGA finds unsupervised adversarial examples, provides quality guarantees with lower and upper bounds, and outperforms sampling schemes that underestimate the maximum error.
As this is the first work for unsupervised adversarial examples on autoencoders we expect a variety of follow-up research. In particular we aim at unsupervised uncertainty quantification of autoencoders. Furthermore we plan to develop more advanced adversarial attacks for specialized autoencoders in time series domain, as well as the re-use of adversarial examples for re-training autoencoders in safety-critical regions. Moreover we intend to make this work more scalable by incorporating advances from the research area of SMT solvers into our approach.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017)
2. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring neural net robustness with constraints. In: Advances in Neural Information Processing Systems 29. pp. 2613–2621 (2016)

3. Bradley, A.R., Manna, Z.: The calculus of computation - decision procedures with applications to verification (2007)
4. Chazan, S.E., Gannot, S., Goldberger, J.: Deep clustering based on A mixture of autoencoders. In: 29th IEEE Int. Workshop on Machine Learning for Signal Processing. pp. 1–6 (2019)
5. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (July 2015)
6. Chhabra, A., Roy, A., Mohapatra, P.: Strong black-box adversarial attacks on unsupervised machine learning models. CoRR (2019)
7. Dalvi, N.N., Domingos, P.M., Mausam, Sanghai, S.K., Verma, D.: Adversarial classification. In: Proc. of the Tenth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. pp. 99–108 (2004)
8. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: Automated Technology for Verification and Analysis - 15th Int. Symposium. vol. 10482, pp. 269–286 (2017)
9. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy. pp. 3–18 (2018)
10. Gondara, L.: Medical image denoising using convolutional denoising autoencoders. In: IEEE Int. Conf. on Data Mining Workshops. pp. 241–246 (2016)
11. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd Int. Conf. on Learning Representations (2015)
12. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification - 29th Int. Conf. vol. 10426, pp. 97–117 (2017)
13. Le, Q.V., Ranzato, M., Monga, R., Devin, M., Corrado, G., Chen, K., Dean, J., Ng, A.Y.: Building high-level features using large scale unsupervised learning. In: Proc. of the 29th Int. Conf. on Machine Learning (2012)
14. Meng, Q., Catchpoole, D.R., Skillicom, D., Kennedy, P.J.: Relational autoencoder for feature extraction. In: 2017 Int. Joint Conf. on Neural Networks (2017)
15. Min, M.R., Stanley, D.A., Yuan, Z., Bonner, A.J., Zhang, Z.: A deep non-linear feature mapping for large-margin knn classification. In: ICDM 2009, The Ninth IEEE Int. Conf. on Data Mining. pp. 357–366 (2009)
16. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th Int. Conf.. vol. 4963 (2008)
17. Pasa, L., Sperduti, A.: Pre-training of recurrent neural networks via linear autoencoders. In: Advances in Neural Information Processing Systems 27: Annual Conf. on Neural Information Processing Systems 2014. pp. 3572–3580 (2014)
18. Sakurada, M., Yairi, T.: Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proc. of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis (2014)
19. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: 7th Int. Conf. on Learning Representations (2019)
20. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: 2nd Int. Conf. on Learning Representations (2014)
21. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010)
22. Zhao, G., Zhang, M., Liu, J., Wen, J.R.: Unsupervised adversarial attacks on deep feature-based retrieval with gan (2019)

7 Supplementary Material

7.1 Experimental Setup

In order to ease reproducibility we provide the precise parameters of the experiments. First regarding the experiments on the synthetic dataset:
We trained an autoencoder with topology $[35, 5, 35]$ and 5 hidden ReLU nodes for 100 epochs using the Adam-optimizer [1]. The dataset used for training is given by a sine-curve with two frequencies: One type of sine-curve stretches over 35 timesteps and the other type of sine-curve over 105 timesteps. To both types we added random gaussian noise drawn from $\mathcal{N}(0, 0.1)$ on each time step. We trained on 250 long sine-curves and 500 short sine-curves. In order to obtain inputs for the autoencoder we divided the resulting timeseries into non-overlapping windows of length 35 resulting in a dataset of 4 clusters.
We obtained the safety-critical region by adding a margin of 0.1 into both directions onto every time step of the (non-noisy) sine-curves.
Next we consider the experiment on real-world data:
Similarly to the synthetic experiment we trained an autoencoder with topology $[35, 5, 35]$ containing 5 hidden ReLU nodes for 200 epochs using the Adam-optimizer. The dataset (ECG5000) consists of 5000 timeserieses of length 140 with class labels. We split each time series into 4 subseries of length 35 in order to be able to train on them. The safety-critical region was obtained by randomly sampling 20 points of the two most frequent classes, taking their respective mean and adding a safety-margin of 0.25 in both directions. This choice should emulate what a potential domain expert could have done to obtain a safety-critical region without relying on labels.

7.2 Proofs

Theorem 1. *Let f be an autoencoder, A a region, dist a distance function, $\varepsilon > 0$, and $\varphi^{ae}_{\varepsilon}$ as defined above. Then, the following two properties hold:*

1. *If an ε-adversarial example exists in the region defined by A, then $\varphi^{ae}_{\varepsilon}$ is satisfiable.*
2. *If $\varphi^{ae}_{\varepsilon}$ is satisfiable, say by the interpretation $\mathcal{I}$, then $\big(\mathcal{I}(x_{1,1}), \ldots, \mathcal{I}(x_{1,l_1})\big)$ is an ε-adversarial example in the region defined by A.*

Proof. For proving the first statement, we formulate a satisfying interpretation $\mathcal{I}$ for $\varphi^{\text{ae}}_{\varepsilon}$, using an ε-adversarial example, say $v^{\varepsilon} = (v_{1,1}, v_{1,2}, \cdots v_{1,l_1})$, in the region A. For that, we assign valuation to each variable in $\varphi^{\text{ae}}_{\varepsilon}$ in the following manner:

- We set $\mathcal{I}(x_{1,i}) = v_{1,j}$ for each $i \in \{1, \cdots, l_1\}$
- We set $\mathcal{I}(x_{k,j}) = v_{k,j}$, for each $i \in 1, \cdots, l_k$, $k \in \{2, \cdots, N\}$ where $v_{k,j}$ is the output of the Neuron j in Layer k, when input v^{ε} is passed through f.

We prove that the above interpretation indeed satisfies $\varphi_{\varepsilon}^{\text{ae}}$, which is a conjunction of three formulas φ^{f}, φ^{A} and $\varphi_{\varepsilon}^{dist}$. First, we observe that if f consists of a linear neuron, then

$$v_{k,j} = \sum_{i=1}^{l_k - 1} W_{i,j}^{k} v_{k-1,i} + b_j^k$$

for $k \in \{2, \cdots, N\}$, $j \in \{1, \cdots, l_k\}$. This indicates that the formulas $\psi_{k,j}$ are satisfiable, and consequently, φ^{f} is satisfiable, being a conjunction of the former formulas. We can make a similar observation if f consists of ReLU neurons. Second, v^{ε} belongs to the region A and hence, satisfies φ^{A}. Third, following the definition of ε-adversarial example, we have

$$\bigvee_{1 \leq j \leq l_1} \Big[|v_{1,j} - v_{N,j}| > \varepsilon \Big],$$

indicating that $\varphi_{\varepsilon}^{dist}$ (note *dist* refers to L_∞ distance) is satisfied under $\mathcal{I}$ as well.

For proving the second statement, from a satisfying interpretation $\mathcal{I}$ of $\varphi_{\varepsilon}^{\text{ae}}$, we derive a ε-adversarial example in the region defined by A. For brevity, let $\mathcal{I}(x_{k,j}) = v_{k,j}$ for $k \in \{1, \cdots, N\}$ and $j \in \{1, \cdots, l_k\}$ and $v^{\varepsilon} = (v_{1,1}, v_{1,2}, \cdots, v_{1,l_1})$.

Now, we prove that when v^{ε} is passed through f, the output of Neuron j in Layer k is $v_{k,j}$ We prove this fact by induction on the layers of f. For the base case, observe that the output of Neuron j in layer 1 is $v_{1,j}$, since $(v_{1,1}, \cdots, v_{1,l_1})$ is an input to f. For the inductive step observe that if f consists of linear neurons we have

$$v_{k,j} = \sum_{i=1}^{l_k - 1} W_{i,j}^{k} v_{k-1,i} + b_j^k$$

since $\varphi_{k,j}$ is satisfiable under $\mathcal{I}$. Now, due to induction hypothesis, $v_{k-1,i}$ is the output of Neuron i in Layer $k-1$ for $i \in \{1, \cdots, l_{k-1}\}$. The above relation exactly combines the output from Layer $k-1$ as as an autoencoder would do and ensures that $v_{k,j}$ is the output of Neuron j in Layer k.

Moreover, φ^{A} being satisfiable under interpretation $\mathcal{I}$ implies that v^{ε} belongs to the region defined by A. Finally, since $\varphi_{\varepsilon}^{dist}$ is satisfiable under $\mathcal{I}$, we have

$$\bigvee_{1 \leq i \leq l_1} \Big[[v_{1,j} - v_{N,j} > \varepsilon] \vee [v_{N,j} - v_{i,j} > \varepsilon] \Big]$$

indicating that v^{ε} is an ε-adversarial example.

Theorem 2. *Let f be an autoencoder, A a region, and $\delta > 0$. Then, Algorithm 1 terminates eventually and outputs a value $\varepsilon^{\star} \in (wce(f, A) - \delta, wce(f, A) + \delta)$.*

Proof. In Lines 1 to 7 in Algorithm 1, we compute ε_{low} and ε_{up} (which are inputs to Binary-search in Line 8) in such a way that, $\varphi_{\varepsilon_{low}}^{ae}$ is satisfiable, while $\varphi_{\varepsilon_{up}}^{ae}$ is not. We refer $wce(f, A)$ as ε_{wce} for brevity. The Binary-search has the following loop invariants: $\varphi_{\varepsilon_{low}}^{ae}$ is satisfiable; $\varphi_{\varepsilon_{up}}^{ae}$ is not satisfiable; and $\varepsilon_{low} \leq \varepsilon_{wce} \leq \varepsilon_{up}$.

While the first two invariants are consequence of the updates made during the loop in binary search, the third invariant can be proved using contradiction.

For the proof we make use of the following property of wce : $\varphi^{ae}_{\varepsilon'}$ is satisfiable for any $\varepsilon' \leq \varepsilon_{wce}$, and $\varphi^{ae}_{\varepsilon''}$ is not satisfiable for any $\varepsilon'' > \varepsilon_{wce}$. Now, towards contradiction, assume that $\varepsilon_{wce} < \varepsilon_{low}$ or $\varepsilon_{up} < \varepsilon_{wce}$. However, if $\varepsilon_{wce} < \varepsilon_{low}$, the above property of wce is violated since, $\varphi^{ae}_{\varepsilon_{low}}$ is satisfiable. Similarly, if $\varepsilon_{up} < \varepsilon_{wce}$, the property of wce is violated since, $\varphi^{ae}_{\varepsilon_{up}}$ is unsatisfiable. Thus, $\varepsilon_{low} \leq \varepsilon_{wce} \leq \varepsilon_{up}$

The Binary-search terminates when $|\varepsilon_{up} - \varepsilon_{low}| < \delta$ and $\varepsilon^* = (\varepsilon_{up} + \varepsilon_{low})/2$ is returned as an output. Now, in the final step, we have

$$\varepsilon^* - \varepsilon_{wce} \leq \varepsilon^* - \varepsilon_{low} < \varepsilon_{up} - \varepsilon_{low} < \delta, \text{ and}$$
$$\varepsilon^* - \varepsilon_{wce} > \varepsilon^* - \varepsilon_{up} > \varepsilon_{low} - \varepsilon_{up} > -\delta$$

Thus, $\varepsilon^* \in (wce(f, A) - \delta, wce(f, A) + \delta)$.

References

1. Kingma, Diederik, Ba, Jimmy: Adam: A Method for Stochastic Optimization. ICLR (2014)