Trust, Resilience and Interpretability of AI Models

Susmit Jha

Computer Science Laboratory,
SRI International
susmit.jha@sri.com

Abstract. In this tutorial, we present our recent work on building trusted, resilient and interpretable AI models by combining symbolic methods developed for automated reasoning with connectionist learning methods that use deep neural networks. The increasing adoption of artificial intelligence and machine learning in systems, including safety-critical systems, has created a pressing need for developing scalable techniques that can be used to establish trust over their safe behavior, resilience to adversarial attacks, and interpretability to enable human audits. This tutorial is comprised of three components: review of techniques for verification of neural networks, methods for using geometric invariants to defend against adversarial attacks, and techniques for extracting logical symbolic rules by reverse engineering machine learning models. These techniques form the core of TRINITY: Trusted, Resilient and Interpretabile AI framework being developed at SRI. In this tutorial, we identify the key challenges in building the TRINITY framework, and report recent results on each of these three fronts.

1 Introduction

The rapid integration of intelligent and autonomous agents into our industrial and social infrastructure has created an immediate need for establishing trust between these agents and their human users. Decision-making and planning algorithms central to the operation of these systems currently lack the ability to explain the choices and decisions that they make. This is particularly problematic when the results returned by these algorithms are counter-intuitive. It is important that intelligent agents become capable of responding to inquiries from human users. For example, when riding in an autonomous taxi, we might expect to query the AI driver using questions similar to those we would ask a human driver, such as "why did we not take the Bay Bridge", and receive a response such as "there is too much traffic on the bridge" or "there is an accident on the ramp leading to the bridge or in the middle lane of the bridge." These explanations are essentially formulae in propositional logic formed by combining the atomic propositions corresponding to the user-observable system and the environment states using Boolean connectives.

Even though the decisions of intelligent agents are the consequence of algorithmic processing of perceived system and environment states, the straightforward approach of reviewing this processing is not practical. There are three key reasons for this. First, AI algorithms use internal states and intermediate variables to make decisions which may not be observable or interpretable by a typical user. For example, reviewing decisions made by the A* planning algorithm [38] could reveal that a particular state was never considered in the priority queue. But this is not human-interpretable, because a user may not be familiar with the details of how A* works. Second, the efficiency and effectiveness of many AI algorithms relies on their ability to intelligently search for optimal decisions without deducing information not needed to accomplish the task, but some user inquiries may require information that was not inferred during the original execution of the algorithm. For example, a state may never be included in the queue of a heuristic search algorithm like A* because either it is unreachable or it has very high cost. Thus, the ability to explain why this state is not on the computed path will require additional effort. Third, artificial intelligence is often a composition of numerous machine learning and decision-making algorithms, and explicitly modeling each one of these algorithms is not practical. Instead, we need a technique which can treat these algorithms as black-box oracles, and obtain explanations by observing their output on selected inputs. This is the first challenge addressed in the TRINITY framework of improving interpretability of AI models by extracting logical symbolic rules.

Among AI models, deep neural networks (DNNs) have emerged as an ubiquitous choice of representation in machine learning due to the relative ease and computational efficiency of training these models in the presence of large amounts of data. The massive increase in computational power fueled by Moore's law and the emergence of architectures supporting parallel processing at a large scale have made it possible to train these highly nonlinear deep learning networks with thousands of parameters using millions of samples in a reasonable amount of time. This has led to a quantum leap in the prediction accuracy of machine learned models, and encouraged their rapid adoption in different aspects of our social, economic and military infrastructure. Deep neural networks currently provide state-of-the-art results in various applications ranging from computer vision, network security, natural language processing to automatic control.

Unlike other traditional system design approaches, there are few known and scalable methods to verify DNN models. This is the second challenge addressed by the TRINITY framework for building trusted AI systems by developing techniques for verifying DNNs [11,12] that has been implemented in a publicly available open-source tool, Sherlock. Sherlock uses mixed-integer linear programming (MILP) solver but it does not merely compile the verification into an MILP problem. Sherlock first uses sound piecewise linearization of the nonlinear activation function to define an encoding of the neural network semantics into mixed-integer constraints involving real-valued variables and binary variables that arise from the (piecewise) linearized activation functions. Such an encoding into MILP is a standard approach to handling piecewise linear functions. As such, the input con-

straints $\phi(\mathbf{x})$ are added to the MILP and next, the output variable is separately maximized and minimized to infer the corresponding guarantee that holds on the output. This enables us to infer an assume-guarantee contract on the overall deep neural network. Sherlock augments this simple use of MILP solving with a local search that exploits the local continuity and differentiability properties of the function represented by the network. These properties are not exploited by MILP solvers which typically use a branch-and-cut approach. On the other hand, local search alone may get "stuck" in local minima. Sherlock handles local minima by using the MILP solver to search for a solution that is "better" than the current local minimum or conclude that no such solution exists. Thus, by alternating between inexpensive local search iterations and relatively expensive MILP solver calls, Sherlock can exploit local properties of the neural network function but at the same time avoid the problem of local minima, and thus, solve the verification of deep neural networks more efficiently.

Further, DNN models have been shown to be very brittle and vulnerable to specially crafted adversarial perturbations to examples: given an input x and any target classification t, it is possible to find a new input x' that is similar to x but classified as t. These adversarial examples often appear almost indistinguishable from natural data to human perception and are yet incorrectly classified by the neural network. Recent results have shown that accuracy of neural networks can be reduced from close to 100% to below 5% using adversarial examples. This creates a significant challenge in deploying these deep learning models in security-critical domains where adversarial activity is intrinsic, such as cyber-networks, and surveillance. The use of neural networks in computer vision and speech recognition have brought these models into the center of security-critical systems where authentication depends on these machine learned models. How do we ensure that adversaries in these domains do not exploit the limitations of machine learning models to go undetected or trigger a non-intended outcome? The third challenge addressed in TRINITY framework is to use geometric methods for identifying invariants in training data that can be used for detecting adversarial examples.

2 Interpretability of AI Models

2.1 Motivating Example

We describe a motivating example to illustrate the problem of providing human-interpretable explanations for the results of an AI algorithm. We consider the A* planning algorithm [38], which enjoys widespread use in path and motion planning due to its optimality and efficiency. Given a description of the state space and transitions between states as a weighted graph where weights are used to encode costs such as distance and time, A* starts from a specific node in the graph and constructs a tree of paths starting from that node, expanding paths in a best-first fashion until one of them reaches the predetermined goal node. At each iteration, A* determines which of its partial paths is most promising

and should be expanded. This decision is based on the estimate of the cost-to-go to the goal node. Specifically, A* selects an intermediate node n that minimizes $\texttt{totalCost}(n) = \texttt{partialCost}(n) + \texttt{guessCost}(n)$, where `totalCost` is the estimated total cost of the path that includes node n, obtained as the sum of the cost ($\texttt{partialCost}(n)$) of reaching n from the initial node, and a heuristic estimate of the cost ($\texttt{guessCost}(n)$) of reaching the goal from n . The heuristic function `guessCost` is problem-specific: e.g., when searching for the shortest path on a Manhattan grid with obstacles, a good `guessCost` is the straight line distance from the node n to the final destination. Typical implementations of A* use a priority queue to perform the repeated selection of intermediate nodes. This priority queue is known as the open set or fringe. At each step of the algorithm, the node with the lowest `totalCost` value is removed from the queue, and "expanded", This means that the `partialCost` values of its neighbors are updated accordingly based on whether going through n improves them, and these neighbors are added to the queue. The algorithm continues until some goal node has the minimum cost value, `totalCost`, in the queue, or until the queue is empty (in which case no plan exists). The `totalCost` value of the goal node is then the cost of the optimal path. We refer readers to [38] for a detailed description of A*. In rest of this section, we illustrate the need for providing explanations using a simple example map and application of A* on it to find the shortest path.

Figure 1 depicts the result of running A* on a 50×50 grid, where cells that form part of an obstacle are colored red. The input map (Figure 1 (a)) shows the obstacles and free space. A* is run to find a path from lower right corner to upper left corner. The output map is shown in Figure 1 (b).

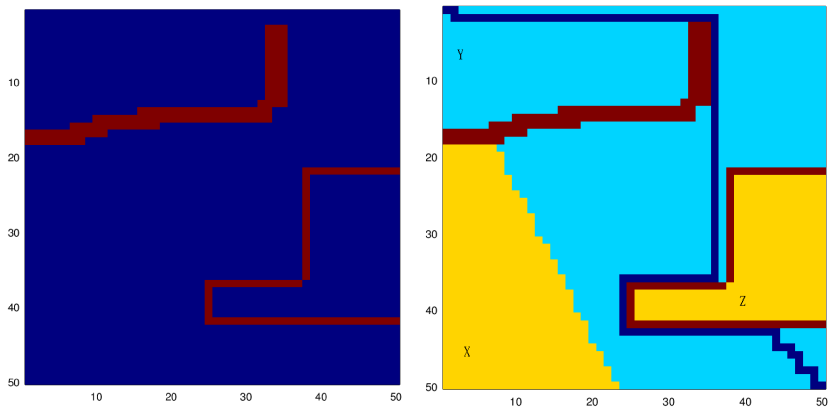

Fig. 1: (a) Input map to A* (b) Output showing final path and internal states of A*. Cells on the computed optimal path are colored dark blue. Cells which entered A*'s priority queue are colored light cyan, and those cells that never entered the queue are colored yellow.

Consider the three cells X,Y,Z marked in the output of A* in Figure 1 (b) and the following inquiries on the optimal path discovered by A*:

- *Why was the cell Y not selected for the optimal path?* Given the output and logged internal states of the A* algorithm, we know that Y was considered as a candidate cell but discarded due to non-optimal cost.
- *Why was the cell X not selected for the optimal path?* If we logged the internal states of the A* algorithm, we would find that X was not even considered as a candidate and it never entered the priority queue of the A* algorithm. But this is not a useful explanation because a non-expert user cannot be expected to understand the concept of a priority queue, or the details of how A* works.
- *Why was the cell Z not selected for the optimal path?* The cell Z was also never inserted into the priority queue and hence, it was never a candidate to be selected on the optimal path similar to cell X. When responding to a user query about why X and Z were not selected in the optimal path, we cannot differentiate between the two even if all the internal decisions and states of the A* algorithm were logged. So, we cannot provide the intuitively expected explanation that Z is not reachable due to obstacles, while X is reachable but has higher cost than the cells that were considered.

This example scenario illustrates the need for new information to provide explanation in addition to the usual deduction by AI algorithm while solving the original decision making problem.

2.2 Logic Extraction and Formal Synthesis

Our approach relies on learning logical explanations in the form of sparse Boolean formula from examples that are obtained by carefully selected introspective simulations of the decision-making algorithm. The area of active learning Boolean formula from positive and negative examples has been studied in literature [1,33] in both exact and probably approximately correct (PAC) setting. Exact learning Boolean formula [3, 34] requires a number of examples exponential in the size of the vocabulary. Under the PAC setting, learning is guaranteed to find an approximately correct concept given enough independent samples [2, 43, 50]. It is known that k-clause conjunctive normal form Boolean formula are not PAC learnable with polynomial sample-size, even though monomials and disjunctive normal form representations are PAC learnable [13, 50]. Changing the representation from CNF to DNF form can lead to exponential blow-up. In contrast, we consider only sparse Boolean formula and our goal is to learn the exact Boolean formula with probabilistic confidence, and not its approximation. Efficient learning techniques exist for particular classes of Boolean formulae such as monotonic and read-one formulae [18, 23], but explanations do not always take these restricted forms, and hence, our focus on sparse Boolean formulae is better suited for this context.

Another related research area is the newly emerged field of formal synthesis, which combines induction and deduction for automatic synthesis of systems

from logical or black-box oracle specifications [25, 29]. Unlike active learning, formal synthesis is also concerned with defining techniques for the generation of interesting examples and not just its inductive generalization, much like our approach. While existing formal synthesis techniques have considered completion of templates by inferring parameters [6, 52, 55], composition of component Boolean functions or uplifting to bitvector form [10, 21, 25, 63], inferring transducers and finite state-machines [7, 8, 17], and synthesis of invariants [56, 57], our work is the first to consider sparsity as a structural assumption for learning Boolean formulae.

The need for explanations of AI decisions to increase trust of decision-making systems has been noted in the literature [40]. Specific approaches have been introduced to discover explanations in specific domains such as MDPs [14], HTNs [22] and Bayesian networks [64]. Explanation of failure in robotic systems by detecting problems in the temporal logic specification using formal requirement analysis was shown to be practically useful in [51]. Inductive logic programming [15] has also been used to model domain-specific explanation generation rules. In contrast, we propose a domain-independent approach to generate explanations by treating the decision-making AI algorithm as an oracle. Domain-independent approaches have also been proposed in the AI literature for detecting sensitive input components that determine the decision in a classification problem [53,60]. While these approaches work in a quantitative setting, such as measuring sensitivity from the gradient of a neural network classifier's ouput, our approach is restricted to the discrete, qualitative setting. Further, we not only detect sensitive inputs (support of Boolean formulae) but also generate the explanation.

2.3 Sparse Boolean Formula Learning for Explanations

A decision-making AI algorithm `Alg` can be modelled as a function that computes the values of output variables `out` given input variables `in`, that is,

$$\texttt{Alg} : \texttt{in} \rightarrow \texttt{out}$$

The outputs are the decision variables, while the inputs include the environment and system states as observed by the system through the perception pipeline. While the decision and state variables can be continuous and real valued, the inquiries and explanations are framed using predicates over these variables, such as comparison of a variable to some threshold. These predicates can either be directly provided by the user or the developer of the AI system, or they can be automatically extracted from the implementation of the AI system by including predicates that appear in the control flow of the AI system. These must be predicates over the input and output variables, that is, `in` and `out`, which are understood by the users. Our approach exploits the sparsity of Boolean formula for learning the explanations and so, the vocabulary can include all possible predicates and variables that might be useful for explaining AI decisions. We propose methods to efficiently find relevant variables where these methods only depend logarithmically on the size of the vocabulary. This ensures that the definition of vocabulary can conveniently include all possible variables, and our approach

can automatically find the relevant subset and synthesize the corresponding explanation.

We denote the vocabulary of atomic predicates used in the inquiry from the user and the provided explanation from the system by $\mathcal{V}$. We can separate the vocabulary $\mathcal{V}$ into two subsets: $\mathcal{V}_Q$ used to formulate the user inquiry and $\mathcal{V}_R$ used to provide explanations.

$$\mathcal{V}_Q = \{q_1, q_2, \ldots q_m\}, \mathcal{V}_R = \{r_1, r_2, \ldots r_n\} \textit{ where } q_i, r_i : \texttt{in} \cup \texttt{out} \rightarrow \texttt{Bool}$$

Intuitively, $\mathcal{V}$ is the shared vocabulary that describes the interface of the AI algorithm and is understood by the human-user. For example, the inquiry vocabulary for a planning agent may include propositions denoting selection of a waypoint in the path, and the explanation vocabulary may include propositions denoting presence of obstacles on a map.

An *inquiry* ϕ_Q from the user is an observation about the output (decision) of the algorithm, and can be formulated as a Boolean combination of predicates in the vocabulary $\mathcal{V}_Q$. Hence, we can denote it as $\phi_Q(\mathcal{V}_Q)$ where the predicates in $\mathcal{V}_Q$ are over the set $\texttt{in} \cup \texttt{out}$, and the corresponding grammar is:

$$\phi_Q := \phi_Q \wedge \phi_Q \mid \phi_Q \vee \phi_Q \mid \neg\phi_Q \mid q_i \texttt{ where } q_i \in \mathcal{V}_Q$$

While conjunction and negation are sufficient to express any Boolean combination, we include disjunction and implication for succinctness of inquiries. Similarly, the *response* $\phi_R(\mathcal{V}_R)$ is a Boolean combination of the predicates in the vocabulary $\mathcal{V}_R$ where the predicates in $\mathcal{V}_R$ are over the set $\texttt{in} \cup \texttt{out}$, and the corresponding grammar is:

$$\phi_R := \phi_R \wedge \phi_R \mid \phi_R \vee \phi_R \mid \neg\phi_R \mid r_i \texttt{ where } r_i \in \mathcal{V}_R$$

Definition 1. *Given an AI algorithm* `Alg` *and an inquiry* $\phi_Q(\mathcal{V}_Q)$, $\phi_R(\mathcal{V}_R)$ *is a* necessary and sufficient explanation *when* $\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)$ *where* $\mathcal{V}_R, \mathcal{V}_Q$ *are predicates over* $\texttt{in} \cup \texttt{out}$ *as explained earlier, and* $\texttt{out} = \texttt{Alg(in)}$. $\phi_R(\mathcal{V}_R)$ *is a* sufficient explanation *when* $\phi_R(\mathcal{V}_R) \Rightarrow \phi_Q(\mathcal{V}_Q)$.

If the algorithm $\texttt{out} = \texttt{Alg(in)}$ could be modeled explicitly in appropriate logic, then the above definition could be used to generate explanations for a given inquiry using techniques such as satisfiability solving. However, such an explicit modeling of these algorithms is currently outside the scope of existing logical deduction frameworks, and is impractical for large and complicated AI systems even from the standpoint of the associated modeling effort. The AI algorithm `Alg` is available as an executable function; hence, it can be used as an oracle that can provide an outputs for any given input. This motivates oracle-guided learning of the explanation from examples using the notion of confidence associated with it.

Definition 2. *Given an AI algorithm* `Alg` *and an inquiry* $\phi_Q(\mathcal{V}_Q)$, $\phi_R(\mathcal{V}_R)$ *is a* necessary and sufficient explanation *with probabilistic confidence* κ *when* $Pr(\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)) \geq \kappa$, *where* $\mathcal{V}_R, \mathcal{V}_Q$ *are predicates over* $\texttt{in} \cup \texttt{out}$ *as explained earlier,* $\texttt{out} = \texttt{Alg(in)}$ *and* $0 \leq \kappa \leq 1$. *The probability of satisfaction*

of $\phi_R(\mathcal{V}_R) \iff \phi_Q(\mathcal{V}_Q)$ *is computed using uniform distribution over the variables in* $\mathcal{V}$*. This uniform distribution is not an assumption over the context in which an AI algorithm* `Alg` *is used. This uniform distribution is only used to estimate the probability of finding the correct explanation. Similarly,* $\phi_R(\texttt{in})$ *is a* sufficient explanation *with confidence* κ *when* $Pr(\phi_R(\mathcal{V}_R) \Rightarrow \phi_Q(\mathcal{V}_Q)) \geq \kappa$*.*

The oracle used to learn the explanation uses the AI algorithm. It runs the AI algorithm on a given input in_i to generate the decision output out_i, and then marks the input as a positive example if $\phi_Q(out_i)$ is true, that is, the inquiry property holds on the output. It marks the input as a negative example if $\phi_Q(out_i)$ is not true. We call this an *introspection oracle* which marks each input as either positive or negative.

Definition 3. *An* introspection oracle $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ *for a given algorithm* `Alg` *and inquiry* ϕ_Q *takes an input* in_i *and maps it to a positive or negative label, that is,* $\mathcal{O}_{\phi_Q,\texttt{Alg}} : \texttt{in} \rightarrow \{\oplus, \ominus\}$*.*

$\mathcal{O}_{\phi_Q,\texttt{Alg}}(in_i) = \oplus$ *if* $\phi_Q(\mathcal{V}_Q(out_i))$ *and* $\mathcal{O}_{\phi_Q,\texttt{Alg}}(in_i) = \ominus$ *if* $\neg\phi_Q(\mathcal{V}_Q(out_i))$*, where* $out_i = \texttt{Alg}(in_i)$*, and* $\mathcal{V}_Q(out_i)$ *is the evaluation of the predicates in* $\mathcal{V}_Q$ *on* out_i

We now formally define the problem of learning Boolean formula with specified confidence κ given an oracle that labels the examples.

Definition 4. *The problem of oracle-guided learning of Boolean formula from examples is to identify (with confidence* κ*) the target Boolean function* ϕ *over a set of atomic propositions* $\mathcal{V}$ *by querying an oracle* $\mathcal{O}$ *that labels each input* in_i *(which is an assignment to all variables in* $\mathcal{V}$*) as positive or negative* $\{\oplus, \ominus\}$ *depending on whether* $\phi(in_i)$ *holds or not, respectively.*

We make the following observations which relates the problem of finding explanations for decisions made by AI algorithms to the problem of learning Boolean formula.

Observation 1 *The problem of generating explanation* ϕ_R *for the AI algorithm* `Alg` *and an inquiry* ϕ_Q *is equivalent to the problem of oracle-guided learning of Boolean formula* ϕ_R *using oracle* $\mathcal{O}_{\phi_Q,\texttt{Alg}}$ *as described in Definition 4.*

$\phi[r_i]$ denotes the restriction of the Boolean formula ϕ by setting r_i to `true` in ϕ and $\phi[\overline{r_i}]$ denotes the restriction of ϕ by setting r_i to `false`. A predicate r_i is in the support of the Boolean formula ϕ, that is, $r_i \in \texttt{support}(\phi)$ if and only if $\phi[r_i] \neq \phi[\overline{r_i}]$.

Observation 2 *The explanation* ϕ_R *over a vocabulary of atoms* $\mathcal{V}_R$ *for the AI algorithm* `Alg` *and a user inquiry* ϕ_Q *is a sparse Boolean formula, that is,* $|\texttt{support}(\phi_R)| << |\mathcal{V}_R|$*.*

These observations motivate the following problem definition for learning sparse Boolean formula.

Definition 5. *Boolean function ϕ is called k-sparse if $|\texttt{support}(\phi_R)| \leq k$. The problem of oracle-guided learning of k-sparse Boolean formula from examples is to identify (with confidence κ) the target k-sparse Boolean function ϕ over a set of atomic propositions $\mathcal{V}$ by querying an oracle $\mathcal{O}$ that labels each input in_i (which is an assignment to all variables in $\mathcal{V}$) as positive or negative $\{\oplus, \ominus\}$ depending on whether $\phi(in_i)$ holds or not, respectively.*

The explanation of decisions made by an AI algorithm can be generated by solving the problem of oracle-guided learning of k-sparse Boolean formula. We recently formulated two algorithms to learn sparse Boolean formula where the size of required examples grows logarithmically (in contrast to exponentially in the general case) with the size of the overall vocabulary. The first algorithm is based on a binary search in the Hamming space first described in our earlier work [27]. The second algorithm is based on random walk in the Boolean hypercube reported in our earlier work [28]. We refer the reader to [27,28] for technical details of the formulation, and detailed case studies for empirical evaluation.

3 Verification of DNNs

At an abstract level, a deep neural network computes a function from a set of inputs to some set of outputs. The question that we address here is as follows:

> Given a neural network (NN), and constraints (assumptions) which define a set of inputs to the network, provide a *tight* over-approximation (guarantee) of the output set.

This serves as one of the main primitives in verification of neural networks. Deep neural networks are very common in applications such as image classification and autonomous control. In image classification networks, since each image is a point in the high dimensional pixel space, a polyhedral set may be used to represent all possible bounded perturbations to a given image. If, for such a set, we can guarantee that the output of the classification remains unaltered, then we have proved that the neural network is *robust* to bounded pixel noise. Besides image classification, neural networks are increasingly used in the control of autonomous systems, such as self-driving cars, unmanned aerial vehicles, and other robotic systems. A typical approach to verify these systems involves a *reachability computation* to estimate the forward reachable set as time evolves. Using this, it is possible to prove that, no matter what the initial condition of a system is, it always reaches a target region in finite time. For instance, we wish prove that, an autonomous car whose inputs are provided by a neural network controller's feedback, will remain within a fixed lateral distance from the center of the road (desired trajectory), while remaining under the speed limit.

We address the output range analysis problem for a neural network with a single output. The extension to multiple output neural networks is straightforward. Let $x \in \Re^n$ be an input to a NN, and $y \in \Re$ be the output of the network. A

typical neural network consists of layers, where each layer computes some function on the outputs of the previous layer and feeds it's output to the next layer. That is, for a k layer neural network, we get a composition of k functions. Each function is a matrix multiplication, followed by an element wise computation of an activation function. A k layer neural network with N neurons in each hidden layer is described by a set of matrices: $[(W_0, b_0), \ldots, (W_{k-1}, b_{k-1}), (W_k, b_k)]$.

Definition 6 (ReLU Unit). *Each neuron in the network implements a non-linear function σ linking its input value to the output value. We consider ReLU units that implement the function $\sigma(z) : \max(z, 0)$. We extend the definition of σ to apply component-wise to vectors z as $\sigma(z) : (\sigma(z_1), \sigma(z_2) \ldots \sigma(z_n))$.*

Taking σ to be the ReLU function, we describe the overall function defined by a given network N as follows:

Definition 7 (Function Computed by neural networks). *Given a neural network N as described above, the function $F : \Re^n \to \Re$ computed by the neural network is given by the composition $F := F_k \circ \cdots \circ F_0$ wherein $F_i(z) : \sigma(W_i z + b_i)$ is the function computed by the i^{th} hidden layer with F_0 denoting the function linking the inputs to the first layer and F_k linking the last layer to the output.*

3.1 Range Estimation

Let N be a neural network with n input vector, x, a single output y, and weights $[(W_0, b_0), \ldots, (W_k, b_k)$. Let F_N be the function defined by such a network. The general problem of verifying neural network and establishing an assume-guarantee contract on its inputs and outputs can be simplified to range estimation problem by suitably transforming the inputs and outputs such that the assumption constraints are described by a polyhedron and the guarantee constraints to be derived over the outputs can be represented as intervals. The universal approximation property of neural networks can be used to approximately encode such transformation as a part of the network itself. Thus, we focus on range estimation problem and rely on reducing other verification problems to it.

The *range estimation* problem is defined as follows:

- Inputs: Neural Network N, input constraints $P : Ax \leq b$ and a tolerance parameter $\delta > 0$.
- Output: An interval $[\ell, u]$ such that $(\forall\ x \in P)\ F_N(x) \in [\ell, u]$. i.e, $[\ell, u]$ contains the range of F_N over inputs $x \in P$. Furthermore, we require the interval to be *tight*:

$$(\max_{x \in P}\ F_N(x) \geq u - \delta),\ (\min_{x \in P}\ F_N(x) \leq \ell + \delta)\,.$$

We will assume that the input polyhedron P is compact: i.e, it is closed and has a bounded volume. It was shown in [32] that even proving simple properties is NP complete. Simple properties, like proving that there exists an assignment from input set to an output set, which respects the constraints imposed by the neural network. So, one of the fundamental challenges in this problem is to tackle the exponential nature.

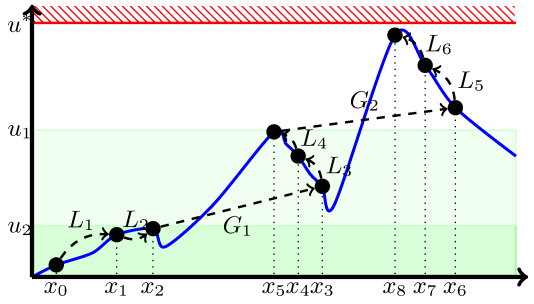

Fig. 2: A schematic figure showing our approach showing alternating series of local search $L_1, \ldots, L_6$ and "global search" G_1, G_2 iterations. The points x_2, x_5, x_8 represent local minima wherein our approach transitions from local search iterations to global search iterations.

3.2 MILP based Approach

Without loss of generality, we will focus on estimating the upper bound u. The case for the lower bound will be entirely analogous. First, we note that a single Mixed Integer Linear Programming (MILP) problem can be formulated, and then query a solver to directly compute u. Unfortunately, that can be quite expensive in practice. Therefore, our approach will combine a series of MILP feasibility problems alternating with local search steps.

Figure 2 shows a pictorial representation of the overall approach. The approach incrementally finds a series of approximations to the upper bound $u_1 < u_2 < \cdots < u^*$, culminating in the final bound $u = u^*$.

1. The first level u_1 is found by choosing a randomly sampled point $x_0 \in P$.
2. Next, we perform a series of local iteration steps resulting in samples $x_1, \ldots, x_i$ that perform gradient ascent until these steps cannot obtain any further improvement. We take $u_2 = F_N(x_i)$.
3. A "global search" step is now performed to check if there is any point $x \in P$ such that $F_N(x) \geq u_2 + \delta$. This is obtained by solving a MILP feasibility problem.
4. If the global search fails to find a solution, then we declare $u^* = u_2 + \delta$.
5. Otherwise, we obtain a new *witness* point x_{i+1} such that $F_N(x_{i+1}) \geq u_2 + \delta$.
6. We now go back to the local search step.

The ideas discussed here for the output range analysis have been implemented in Sherlock [12]. For neural networks with multiple outputs, we can individually find the bounds for each of the network outputs, and then combine them to form a hyper-rectange in the output dimensional space. This can be extended to using a template polyhedron to obtain tighter bounds, in the output dimension, described in the next section. In general, we can obtain guarantees on the output from a given class defined by the constraint template used in the minimization/maximization step of the presented approach. Our current implementation in Sherlock built on top of MILP solvers requires the template to be linear.

3.3 Application: Reachability Analysis

We briefly describe how we can use the above algorithm to verify behaviors of autonomous systems, with neural networks as controllers. Details are presented in [11]. A closed loop system, $\mathcal{C}$, is described by the neural networks f_p, for the system model, and f_h, for the control law. The plant model function f_p, gives the state of the system in the next time step, given the states of the system at the current time step. That is, $x(t+1) = f_p(x(t), f_h(x(t)))$, where $x(t) \in \Re^n$, is the state of the system at time time t, in an n dimensional space.

Thus, given an initial state X_0 (represented as a polyhedron over the state space), we wish to compute symbolic representations for sets $X_1, X_2, \ldots, X_K$ wherein X_i represents the reachable states of the closed loop system given by the composition of the plant f_p and the feedback law f_h, in i steps. Here K is some fixed time horizon. We will use range computation, as a primitive for checking reachability, invariance and stability properties.

The computation of the reach sets of the closed loop system starts with an effort to compute over-approximation of the post operator:

$$\mathsf{post}(X; f_p, f_h) : \ \{\mathbf{x} \in \mathbb{R}^n \mid (\exists \mathbf{x}_0 \in X)\ \mathbf{x} = f_p(\mathbf{x}_0, f_h(\mathbf{x}_0))\}\,.$$

For an input set X, the exact set of the output map of the post operator can be prohibitively expensive to compute: in general, it's a union of polyhedrons, the count of which is exponential in the number of neurons in the two given networks f_p and f_h. Instead, we use a single polyhedron $P(X)$ that approximates the post condition.

For that purpose, we use a template polyhedra:

Definition 8 (Template Polyhedra). *A template T is a set of expressions $T : \ \{\mathbf{e}_1, \ldots, \mathbf{e}_r\}$ wherein each $\mathbf{e}_i$ is an linear expression of the form $\mathbf{c}_i^t\mathbf{x}$ over the state variables. A template polyhedron P over a template T is of the form:*

$$\bigwedge_{j=1}^{r} \ell_j \leq \mathbf{e}_j \leq u_j\,,$$

for bounds ℓ_j, u_j over each template expression $\mathbf{e}_j$.

For a fixed template T, the reachable sets are represented by template polyhedra over the above templates. The post condition operation $\mathsf{post}(X; f_p, f_h)$, can now be substituted by a template-based post-condition operator $\mathsf{post}_T(X; f_p, f_h)$ that produces bounds ℓ_j, u_j for each $\mathbf{e}_j \in T$ by solving the following optimization problem:

$$\ell_j(u_j) : \ \min(\max)\ \mathbf{e}_j[\mathbf{x}]\ \mathsf{s.t.}\mathbf{x}_0 \in X_0,\ \mathbf{u} = f_h(\mathbf{x}_0),\ \mathbf{x} = f_p(\mathbf{x}_0, \mathbf{u})\,.$$

The above optimization problem, is defined using neural network functions f_h and f_p. However, the combination of local search and MILP encoding used in our tool SHERLOCK can be modified almost trivially to solve this optimization problem. Furthermore, the guarantees used in SHERLOCK extend. Thus, we

guarantee that the reported result is no more than ϵ away from the true value, for the given tolerance parameter ϵ.

The computation of reach sets can be extended beyond single step using Sherlock. It is possible to use the tool for a k step reachability $\mathsf{post}_T^{(k)}(X; f_p, h)$ with the tolerance factor ϵ, in a very straight forward manner. A fundamental goal in computing reachable sets is to prove that the system trajectories converge to a target set, starting from the initial set. It suffices to show that the reach sets computed eventually land inside the target set T, in a finite number of time steps. Thus, the problem of checking reachability of a target set T is performed iteratively as

Table 1: Performance results on networks trained on functions with known maxima and minima . **Legend:** x number of inputs, k number of layers, N: total number of neurons, T: CPU time taken, Nc: number of nodes explored. All the tests were run on a Linux server running Ubuntu 17.04 with 24 cores, and 64GB RAM (DNC : Did Not Complete)

				23 cores				single core				
				Sherlock		Monolithic		Sherlock		Monolithic		Reluplex
ID	x	k	N	T	Nc	T	Nc	T	Nc	T	Nc	T
N_0	2	1	100	**1s**	94	2.3**s**	24	0.4**s**	44	0.3**s**	25	9.0
N_1	2	1	200	2.2**s**	166	3.6**s**	29	0.9**s**	71	0.8**s**	38	**1m**50**s**
N_2	2	1	500	7.8**s**	961	12.6**s**	236	2s	138	2.9**s**	257	15**m**59**s**
N_3	2	1	500	1.5**s**	189	0.5**s**	43	0.6**s**	95	0.2**s**	53	12**m**25**s**
N_4	2	1	1000	3**m**52**s**	32E3	3**m**52**s**	3E3	1**m**20**s**	4.8E3	35.6**s**	5.3E3	**1h**06**m**
N_5	3	7	425	4**s**	6	6.1**s**	2	1.7**s**	2	0.9**s**	2	DNC
N_6	3	4	762	3**m**47**s**	3.3E3	4**m**41**s**	3.6E3	37.8**s**	685	56.4**s**	2.2E3	DNC
N_7	4	7	731	3.7**s**	1	7.7s	2	3.9**s**	1	3.1**s**	2	**1h**35**m**
N_8	3	8	478	6.5**s**	3	40.8**s**	2	3.6**s**	3	3.3**s**	2	DNC
N_9	3	8	778	18.3**s**	114	**1m**11**s**	2	12.5**s**	12	4.3**s**	73	DNC
N_{10}	3	26	2340	50**m**18**s**	4.6E4	**1h**26**m**	6E4	17**m**12**s**	2.4E4	18**m**58**s**	1.9E4	DNC
N_{11}	3	9	1527	5**m**44**s**	450	55**m**12**s**	6.4E3	56.4**s**	483	130.7**s**	560	DNC
N_{12}	3	14	2292	24**m**17**s**	1.8E3	**3h**46**m**	2.4E4	8**m**11**s**	2.3E3	**1h**01**m**	1.6E4	DNC
N_{13}	3	19	3057	**4h**10**m**	2.2E4	61**h**08**m**	6.6E4	**1h**7**m**	1.5E4	15**h**1**m**	1.5E5	DNC
N_{14}	3	24	3822	72**h**39**m**	8.4E4	111**h**35**m**	1.1E5	5**h**57**m**	3E4	timeout	-	DNC
N_{15}	3	127	6845	2**m**51**s**	1	timeout	-	3**m**27**s**	1	timeout	-	DNC

We did comparisons with a recent solver for deep neural networks called Reluplex [32], and detailed study is available in [11,12]. Even though Reluplex is an SMT solver, it can be used to perform set propagation using a binary search wrapper. The preliminary comparison shows that Sherlock is much faster than

Reluplex used with a binary search wrapper. Another set of comparisons was using Sherlock, against a monolithic mixed integer linear programming (MILP) solver. The results of the comparison has been presented in Table 1

We used Sherlock for verifying properties of various closed-loop cyberphysical systems that have neural networks as controller. We could prove that in finite number of steps, the sets did converge to the goal region. Details of the technical approach and empirical evaluation are presented in [11, 12].

4 Resilience of DNNs

There has been a recent explosion of methods for adversarial attacks on neural network models along with techniques for making neural networks resilient to attacks. No single resilient mechanism has yet been discovered which can be used against any feasible attack method. We take a different approach to resilience by focusing on the identification of suspicious adversarial examples. The overall idea is to ensure that the machine learning models can identify adversarial attacks, and not provide a prediction on them instead of providing a wrong prediction. An approach to detect these adversarial examples will act as a runtime monitor that finds the limits of the machine learning model.

We recently developed an approach to detect adversarial examples by identifying a low dimensional manifold in which the training data lie, and then measuring the distance of a new sample to this manifold. The manifold corresponds to a geometric invariant of the training data. Adversarial examples often rely on lying outside this manifold, and since the model was learned using data samples in the manifold, the model naturally mis-predicts on examples farther away from the manifold. In our experiments, we used the CleverHans system [46] and employed the Projected Gradient Descent (PGD) attack method implemented in it. This is an implementation of a very recent attack method described in Madry et al [42]. We control the strength of the attack using one of the parameters in this method ϵ that bounds the maximum distortion of adversarial example compared to the original input. Increasing this norm bound generates adversarial examples with higher confidence.

Our empirical study on MNIST [39] and CIFAR10 [36] datasets suggests that adversarial examples not only lie farther away from the data manifold, but this distance from manifold of an adversarial example increases with the confidence of adversarial examples. Consequently, our detection approach can more easily detect adversarial examples generated with higher norm bound and hence, more likely to cause mis-prediction in the machine learned model. Our efforts constitute a first step towards formulating a computational geometric approach to identifying boundaries of a machine learning model, and using it to detect adversarial attacks. The details are described in [26].

4.1 Adversarial Attacks and Defenses

Multiple methods have been proposed in literature to generate adversarial examples as well as defend against adversarial examples. Adversarial example gen-

eration methods include both white-box and black-box attacks on neural networks [19,47,48,61], targeting feedforward classification networks [9], generative networks [35], and recurrent neural networks [49]. These methods leverage gradient based optimization for normal examples to discover perturbations that lead to mis-prediction - the techniques differ in defining the neighborhood in which perturbation is permitted and the loss function used to guide the search. For example, one of the earliest attacks [19] used a fast sign gradient method (FGMS) that looks for a similar image x' in the L^∞ neighborhood of x. Given a loss function $Loss(x, l)$ specifying the cost of classifying the point x as label l, the adversarial example x' is calculated as

$$x' = x + \epsilon \cdot sign(\nabla_x Loss(x, l_x)$$

FGMS was improved to iterative gradient sign approach (IGSM) in [37] by using a finer iterative optimization strategy where the attack performs FGMS with a smaller step-width α, and clips the updated result so that the image stays within the ϵ boundary of x. In this approach, the i-th iteration computes the following:

$$x'_{i+1} = clip_{\epsilon,x}(x'_i + \alpha \cdot sign(\nabla_x Loss(x, l_x)))$$

In contrast to FGSM and IGSM, DeepFool [45] attempts to find a perturbed image x' from a normal image x by finding the closest decision boundary and crossing it. In practice, DeepFool relies on local linearized approximation of the decision boundary. Another attack method that has received a lot of attention is Carlini attack that relies on finding a perturbation that minimizes change as well as the hinge loss on the logits (pre-softmax classification result vector). The attack is generated by solving the following optimization problem:

$$min_\delta[||\delta||_2 + c \cdot \max(\ Z(x')_{l_x} - maxZ(x')_i : i \neq l_x, -\kappa\ )]$$

where Z denotes the logits, l_x is the ground truth label, κ is the confidence (raising which will force searcher for larger perturbations), and c is a hyper-parameter that balances the perturbation and the hinge loss. Another attack method is projected gradient method (PGM) proposed in [42]. PGD attempts to solve this constrained optimization problem:

$$\max_{||x^{adv} - x||_\infty \leq \epsilon} Loss(x^{adv}, l_x)$$

where S is the constraint on the allowed perturbation usually given as bound ϵ on the norm, and l_x is the ground truth label of x. Projected gradient descent is used to solve this constrained optimization problem by restarting PGD from several points in the l_∞ balls around the data points x. This gradient descent increases the loss function $Loss$ in a fairly consistent way before reaching a plateau with a fairly well-concentrated distribution and the achieved maximum value is considerably higher than that of a random point in the data set. We focus on this PGD attack because it is shown to be a universal first order adversary [42], that is, developing detection capability or resilience against PGD also implies defense against many other first order attacks.

Defense of neural networks against adversarial examples is more difficult compared to generating attacks. Madry et al. [42] propose a generic saddle point formulation where $\mathcal{D}$ is the underlying training data distribution, $Loss(\theta, x, l_x)$ is a loss function at data point x with ground truth label l_x for a model with parameter θ:

$$\min_{\theta} E_{(x,y)\sim\mathcal{D}}[\max_{||x^{adv}-x||_\infty \leq \epsilon} Loss(\theta, x^{adv}, l_x)]$$

This formulation uses robust optimization over the expected loss for worst-case adversarial perturbation for training data. The internal maximization corresponds to finding adversarial examples, and can be approximated using IGSM [37]. This approach falls into a category of defenses that use *adversarial training* [59]. Instead of training with only adversarial examples, using a mixture of normal and adversarial examples in the training set has been found to be more effective [45, 61]. Another alternative is to augment the learning objective with a regularizer term corresponding to the adversarial inputs [19]. More recently, logit pairing has been shown to be an effective approximation of adversarial regularization [31].

Another category of defense against adversarial attacks on neural networks are defensive distillation methods [48]. These methods modify the training process of neural networks to make it difficult to launch gradient based attacks directly on the network. The key idea is to use distillation training technique [24] and hide the gradient between the pre-softmax layer and the softmax outputs. Carlini and Wagner [9] found methods to break this defense by changing the loss function, calculating gradient directly from pre-softmax layer and transferring attack from easy-to-attack network to distilled network. More recently, Athalye et al. [4] showed that it is possible to bypass several defenses proposed for the whitebox setting.

Our approach falls into the category of techniques that focus on only detecting adversarial examples. Techniques based on manually identified statistical features [20] or a dedicated learning model [44] trained separately to identify adversarial examples have been previously proposed in literature. These explicit classification methods do not generalize well across different adversarial example generation techniques.

In contrast to these defensive methods, our approach does not require any augmentation of training data, modification of the training process or change in the learned model. The design and training of the neural network is independent to the manifold based filtering approach. Thus, our approach to detection is orthogonal to learning robust machine learning models and can benefit from these methods. Further, we do not require access to the adversarial example generation method, and thus this defense is likely to generalize well across different attack methods. Our approach relies on just identifying the manifold of typical data which need not be even labeled and hence, this method is more practical in contexts where labeled training data is very difficult to obtain.

4.2 Manifold Learning

Learning manifold in which data points lie has been itself an active area of research [41,54,58,62]. ISOMAP, t-SNE and spectral embedding have been proposed to learn the data manifold. The spectral embedding method performs dimensionality reduction in a way that preserves dot products between data points as closely as possible by minimizing $\sum_i (x_i^T x_j - y_i^T y_j)^2$ where y_i is embedding of x_i. ISOMAP [62] embeds the data points in a low dimensional space while preserving the geodesic distances between data points. The geodesic distances are measured in terms of shortest paths between the points in a graph formed by computing k-nearest neighbors and introducing an edge between the neighbors. After computing the geodesic distances, spectral methods can be used to compute the embeddings that preserve this geodesic distance instead of Euclidean distance. t-distributed Stochastic Nearest Embedding (t-SNE) [41] is another method for computing manifold. It constructs a probability distribution over pairs of high-dimensional data points in such a way that similar objects have a high probability of being picked. This is followed by defining a similar distribution in the low dimension and minimized the KL divergence between the two distributions.

LLE [54] is another graph-based dimensionality reduction method that tries to preserve the local linear structure. LLE linearly approximates each data point in the training set manifold with its closest neighbors where the approximation is learned using linear regression. LLE requires computations of the k-nearest neighbors followed by computing the weight matrix W that represents each point as a linear combination of its neighbors. W is computes such that the overall reconstruction error $\sum_i ||x_i - \sum_j W_{ij} x_j||^2$ is minimized subject to constraints that $W_{ij} = 0$ when x_i and x_j are not neighbors, and $\sum_j W_{ij} = 1$ for all i. The low dimensional embedding is computed in LLE by minimizing the following objective: $\sum_i ||y_i - \sum_j W_{ij} y_j||^2$, where y_i denotes the low dimensional embedding of x_i, and we can normalize the representation by requiring $\sum_i y_i = 0$ and $Y^T Y = I$. W is constructed locally for each point, but the low dimensional embeddings y_i are computed globally in a single optimization step. This enables LLE to uncover global structure. Further, the embedding discovered by LLE is scale and rotation independent due to constraints on y_i. Our experiments found LLE to be most effective because of LLE's better discovery of nonlinearity, and sharper embedding in lower dimension.

Our approach relies on computing the distance of the new sample point from the manifold of training data. The kernel density estimation can be used to measure the distance $d(x)$ of x from the data manifold of training set. Specifically, $d(x) = \frac{1}{|X|} \sum_{x_i \in X} k(x_i, x)$, where X is the full data set and $k(\cdot, \cdot)$ is a kernel function such as Gaussian or a simple L_∞ or L_2 norm. In case of using Gaussian kernel, the bandwidth σ needs to be carefully selected to avoid 'spiky' density estimate or an overly smooth density estimate. A typical good choice for bandwidth is a value that maximizes the log-likelihood of the training data [30]. Further, we can restrict the set of training points to be considered from the full set X to

a set of immediate neighbors of x. The neighborhood can be defined using the maximum distance or bound on the number of neighbors. In our experiments, we use L_∞ norm with bound on the number of neighbors which yielded good result.

It has been hypothesized in literature [5, 16] that the deeper layers of a deep neural network provide more linear and unwrapped manifolds in comparison to the input space. Thus, the task of identifying the manifold becomes easier as we progress from the input space to the more abstract feature spaces all the way to the logit space. But the adversarial perturbations are harder to detect at higher levels and might get hidden by the lower layers of the neural network. In our experiments, we learned manifolds in input space as well as the logit space.

4.3 Empirical Evaluation

We evaluated our approach on MNIST dataset [39] and CIFAR10 dataset [36]. We report the key findings in this section.

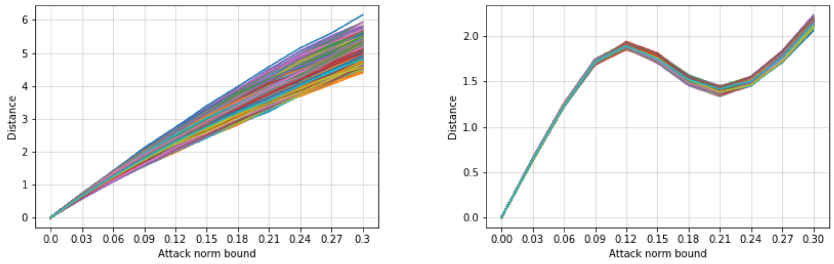

Fig. 3: Increase in adversarial distance from manifold for MNIST in input space (Left) and logit space (Right). Each line of different color shows the increase in distance with attack norm for one sample of a 1000 images. The distance monotonically increased in each of the 100 experiments in the input space. The logit space shows increase in distance with norm up to a threshold after which the distance decreases before again increasing. This is because of high norm bound allowing occasional discovery of 'clever' adversarial examples that are closer to the logit manifold though farther from the input manifold.

As the norm bound in the PGD method for generating adversarial examples is increased, the distance of adversarial examples from the manifold increases. While the success of attack on the neural network increases with high norm bound, it also becomes easier to detect these adversarial examples. We observed this behavior to be common across MNIST and CIFAR10 data set as illustrated in Figure 4. The distance from manifold monotonically increases in the input space but in the logit space, higher norm bound beyond a threshold allows the attack method to find examples that decrease the distance from logit manifold

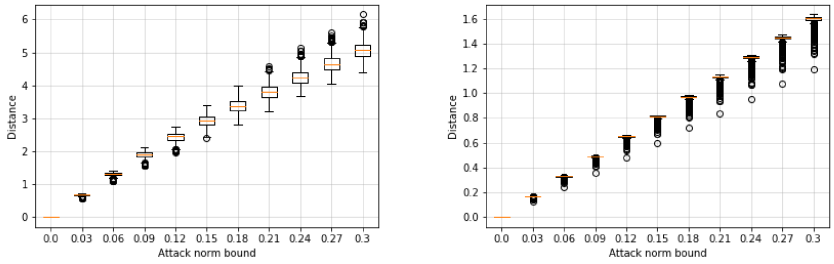

Fig. 4: Increase in adversarial example's distance from input manifold with increase in attack norm: Left:MNIST, Right: CIFAR. The boxes in the box plot denote the first and third quartile of the distance at a given attack norm.

even though they are farther from the input manifold. The consistent rise and fall of distance in logit space for all the 100 samples is likely a property of the used PGD method. This result is illustrated in Figure 3. The detection rate of adversarial examples for MNIST as well as CIFAR10 improves with increase in norm bound and increased distance from the manifold as illustrated in Figure 5.

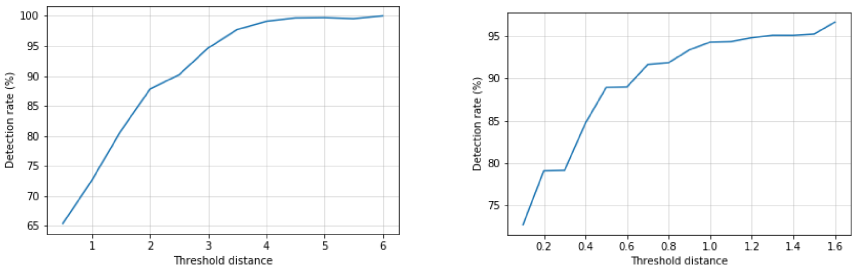

Fig. 5: (Left) Detection rate for MNIST data set (Right) Detection Rate for CIFAR

5 Conclusion

We summarized recent work on building trusted, resilient and interpretable AI models. We identify the key challenges for each of these three fronts, and describe recently proposed techniques that make progress on these challenges. These techniques comprise the key elements of TRINITY framework for high-assurance artificial intelligence being developed at SRI.

Acknowledgement

The author acknowledges support from the US ARL Cooperative Agreement W911NF-17-2-0196 on Internet of Battlefield Things (IoBT) and National Science Foundation(NSF) #1750009 and #1740079.

References

1. Abouzied, A., Angluin, D., Papadimitriou, C., Hellerstein, J.M., Silberschatz, A.: Learning and verifying quantified boolean queries by example. In: ACM symposium on Principles of database systems. pp. 49–60. ACM (2013)
2. Angluin, D.: Computational learning theory: survey and selected bibliography. In: ACM symposium on Theory of computing. pp. 351–369. ACM (1992)
3. Angluin, D., Kharitonov, M.: When won't membership queries help? In: ACM symposium on Theory of computing. pp. 444–454. ACM (1991)
4. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. arXiv preprint arXiv:1802.00420 (2018)
5. Bengio, Y., Mesnil, G., Dauphin, Y., Rifai, S.: Better mixing via deep representations. In: International Conference on Machine Learning. pp. 552–560 (2013)
6. Bittner, B., Bozzano, M., Cimatti, A., Gario, M., Griggio, A.: Towards pareto-optimal parameter synthesis for monotonie cost functions. In: FMCAD. pp. 23–30 (Oct 2014)
7. Boigelot, B., Godefroid, P.: Automatic synthesis of specifications from the dynamic observation of reactive programs. In: TACAS. pp. 321–333 (1997). https://doi.org/10.1007/BFb0035397
8. Botinčan, M., Babić, D.: Sigma*: Symbolic learning of input-output specifications. In: POPL. pp. 443–456 (2013). https://doi.org/10.1145/2429069.2429123
9. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. arXiv preprint arXiv:1608.04644 (2016)
10. Cook, B., Kroening, D., Rümmer, P., Wintersteiger, C.M.: Ranking function synthesis for bit-vector relations. FMSD **43**(1), 93–120 (2013). https://doi.org/10.1007/s10703-013-0186-4
11. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Learning and verification of feedback control systems using feedforward neural networks. IFAC-PapersOnLine **51**(16), 151–156 (2018)
12. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods Symposium. pp. 121–138. Springer (2018)
13. Ehrenfeucht, A., Haussler, D., Kearns, M., Valiant, L.: A general lower bound on the number of examples needed for learning. Information and Computation **82**(3), 247 – 261 (1989). https://doi.org/10.1016/0890-5401(89)90002-3
14. Elizalde, F., Sucar, E., Noguez, J., Reyes, A.: Generating Explanations Based on Markov Decision Processes, pp. 51–62 (2009)
15. Feng, C., Muggleton, S.: Towards inductive generalisation in higher order logic. In: 9th International Workshop on Machine learning. pp. 154–162 (2014)
16. Gardner, J.R., Upchurch, P., Kusner, M.J., Li, Y., Weinberger, K.Q., Bala, K., Hopcroft, J.E.: Deep manifold traversal: Changing labels with convolutional features. arXiv preprint arXiv:1511.06421 (2015)

17. Godefroid, P., Taly, A.: Automated synthesis of symbolic instruction encodings from i/o samples. SIGPLAN Not. **47**(6), 441–452 (Jun 2012). https://doi.org/10.1145/2345156.2254116
18. Goldsmith, J., Sloan, R.H., Szörényi, B., Turán, G.: Theory revision with queries: Horn, read-once, and parity formulas. Artificial Intelligence **156**(2), 139–176 (2004)
19. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). arXiv preprint arXiv:1412.6572
20. Grosse, K., Manoharan, P., Papernot, N., Backes, M., McDaniel, P.: On the (statistical) detection of adversarial examples. arXiv preprint arXiv:1702.06280 (2017)
21. Gurfinkel, A., Belov, A., Marques-Silva, J.: Synthesizing Safe Bit-Precise Invariants, pp. 93–108 (2014)
22. Harbers, M., Meyer, J.J., van den Bosch, K.: Explaining simulations through self explaining agents. Journal of Artificial Societies and Social Simulation (2010), http://EconPapers.repec.org/RePEc:jas:jasssj:2009-25-1
23. Hellerstein, L., Servedio, R.A.: On pac learning algorithms for rich boolean function classes. Theoretical Computer Science **384**(1), 66–76 (2007)
24. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
25. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-guided component-based program synthesis. In: ICSE. pp. 215–224. IEEE (2010)
26. Jha, S., Jang, U., Jha, S., Jalaian, B.: Detecting adversarial examples using data manifolds. In: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM). pp. 547–552. IEEE (2018)
27. Jha, S., Raman, V., Pinto, A., Sahai, T., Francis, M.: On learning sparse boolean formulae for explaining AI decisions. In: NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings. pp. 99–114 (2017)
28. Jha, S., Sahai, T., Raman, V., Pinto, A., Francis, M.: Explaining ai decisions using efficient methods for learning sparse boolean formulae. Journal of Automated Reasoning pp. 1–21 (2018)
29. Jha, S., Seshia, S.A.: A theory of formal synthesis via inductive learning. Acta Informatica, Special Issue on Synthesis (2016)
30. Jones, M.C., Marron, J.S., Sheather, S.J.: A brief survey of bandwidth selection for density estimation. Journal of the American Statistical Association **91**(433), 401–407 (1996)
31. Kannan, H., Kurakin, A., Goodfellow, I.: Adversarial logit pairing. arXiv preprint arXiv:1803.06373 (2018)
32. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
33. Kearns, M., Li, M., Valiant, L.: Learning boolean formulas. J. ACM **41**(6), 1298–1328 (Nov 1994)
34. Kearns, M., Valiant, L.: Cryptographic limitations on learning boolean formulae and finite automata. Journal of the ACM (JACM) **41**(1), 67–95 (1994)
35. Kos, J., Fischer, I., Song, D.: Adversarial examples for generative models. arXiv preprint arXiv:1702.06832 (2017)
36. Krizhevsky, A., Nair, V., Hinton, G.: The cifar-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html (2014)
37. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533 (2016)

38. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
39. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
40. Lee, J., Moray, N.: Trust, control strategies and allocation of function in human-machine systems. Ergonomics **35**(10), 1243–1270 (1992)
41. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)
42. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
43. Mansour, Y.: Learning boolean functions via the fourier transform. In: Theoretical advances in neural computation and learning, pp. 391–424 (1994)
44. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. arXiv preprint arXiv:1702.04267 (2017)
45. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2574–2582 (2016)
46. Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F., Matyasko, A., Hambardzumyan, K., Juang, Y.L., Kurakin, A., Sheatsley, R., et al.: cleverhans v2. 0.0: an adversarial machine learning library. arXiv preprint arXiv:1610.00768 (2016)
47. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 506–519. ACM (2017)
48. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. pp. 372–387. IEEE (2016)
49. Papernot, N., McDaniel, P., Swami, A., Harang, R.: Crafting adversarial input sequences for recurrent neural networks. In: Military Communications Conference, MILCOM 2016-2016 IEEE. pp. 49–54. IEEE (2016)
50. Pitt, L., Valiant, L.G.: Computational limitations on learning from examples. Journal of the ACM (JACM) **35**(4), 965–984 (1988)
51. Raman, V., Lignos, C., Finucane, C., Lee, K.C.T., Marcus, M.P., Kress-Gazit, H.: Sorry Dave, I'm Afraid I can't do that: Explaining unachievable robot tasks using natural language. In: Robotics: Science and Systems (2013)
52. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.: Counterexample-Guided Quantifier Instantiation for Synthesis in SMT, pp. 198–216 (2015), http://dx.doi.org/10.1007/978-3-319-21668-3_12
53. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why Should I Trust You?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016). https://doi.org/10.1145/2939672.2939778
54. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. science **290**(5500), 2323–2326 (2000)
55. Sankaranarayanan, S., Miller, C., Raghunathan, R., Ravanbakhsh, H., Fainekos, G.: A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In: Annual Allerton Conference on Communication, Control, and Computing. pp. 1610–1617 (Oct 2012). https://doi.org/10.1109/Allerton.2012.6483413
56. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: HSCC. pp. 221–230 (2010). https://doi.org/10.1145/1755952.1755984

57. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. FMSD **32**(1), 25–55 (2008). https://doi.org/10.1007/s10703-007-0046-1
58. Saul, L.K., Roweis, S.T.: Think globally, fit locally: unsupervised learning of low dimensional manifolds. Journal of machine learning research **4**(Jun), 119–155 (2003)
59. Shaham, U., Yamada, Y., Negahban, S.: Understanding adversarial training: Increasing local stability of neural nets through robust optimization. arXiv preprint arXiv:1511.05432 (2015)
60. Štrumbelj, E., Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. KIS **41**(3), 647–665 (2014). https://doi.org/10.1007/s10115-013-0679-x
61. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
62. Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science **290**(5500), 2319–2323 (2000)
63. Urban, C., Gurfinkel, A., Kahsai, T.: Synthesizing Ranking Functions from Bits and Pieces, pp. 54–70 (2016), http://dx.doi.org/10.1007/978-3-662-49674-9_4
64. Yuan, C., Lim, H., Lu, T.C.: Most relevant explanation in bayesian networks. J. Artif. Intell. Res.(JAIR) **42**, 309–352 (2011)