Concolic Testing for Deep Neural Networks

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska and Daniel Kroening

Arxiv 2018

Table of Contents

Table of Contents

Introduction

Major concerns have been raised about the readiness of applying DNNs to safety- and security-critical systems, where faulty behaviour carries the risk of endangering human lives or potential damage to business.

To address these concerns critical systems implemented with DNNs, or comprising DNNs components, needs to be thoroughly tested and certified.

None of previous work leverages concolic testing to explore the execution paths of a program that are hard to cover by other techniques such as random testing.

Concolic testing is particularly well-suited for DNNs.

- Input space is high-dimensional, which makes random testing difficult.
- the number of "execution paths" too large to be completely covered by symbolic execution.

Introduction

Currently, test requirements for DNNs lack a unified format.

To enable working with a broad spectrum of test requirements, they utilise Quantified Linear Arithmetic over Rationals (QLAR).

For a given set R of test requirements, we gradually generate test cases to improve coverage by alternating between concrete execution and symbolic analysis.

Given an unsatisfied test requirement r, it is transformed into its corresponding form $\delta(r)$ by means of a heuristic function δ.

Then, for the current set $\mathcal{T}$ of test cases, they identify a pair $(t, \delta(r))$ of test case $t \in \mathcal{T}$ and requirement $\delta(r)$ such that t is close to satisfying r according to an evaluation based on concrete execution.

Introduction

After that, symbolic analysis is applied to $(t, \delta(r))$ to obtain a new concrete test case t', which is then added to the existing test suite, i.e., $\mathcal{T} = \mathcal{T} \cup t'$.

This process repeats until we reach a satisfactory level of coverage.

The generated test suite $\mathcal{T}$ is given to a *robustness oracle*, which detects whether $\mathcal{T}$ includes adversarial examples.

Introduction

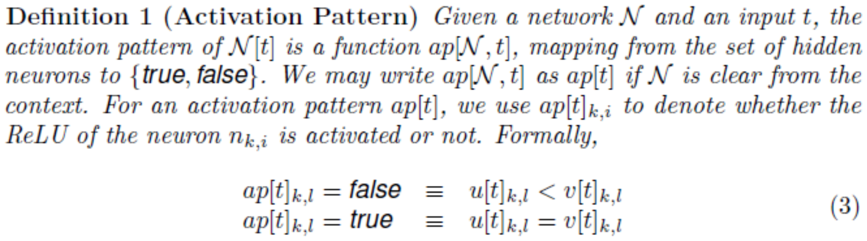
Current work on the robustness of DNNs can be categorised as offensive or defensive.

The concrete execution and symbolic analysis interleave until a certain level of coverage on program statements, branches, execution paths, etc., is reached.

The key factor affecting the performance of concolic testing is the heuristics used to select another execution path.

Table of Contents

Test Requirements



Definition 1 (Activation Pattern) *Given a network $\mathcal{N}$ and an input t, the activation pattern of $\mathcal{N}[t]$ is a function $ap[\mathcal{N}, t]$, mapping from the set of hidden neurons to $\{\textsf{true}, \textsf{false}\}$. We may write $ap[\mathcal{N}, t]$ as $ap[t]$ if $\mathcal{N}$ is clear from the context. For an activation pattern $ap[t]$, we use $ap[t]_{k,i}$ to denote whether the ReLU of the neuron $n_{k,i}$ is activated or not. Formally,*

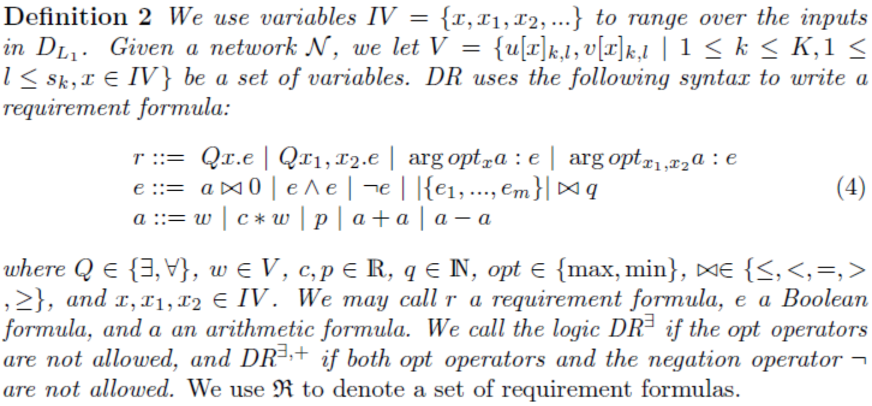
$$\begin{array}{lcl} ap[t]_{k,l} = \textsf{false} & \equiv & u[t]_{k,l} < v[t]_{k,l} \\ ap[t]_{k,l} = \textsf{true} & \equiv & u[t]_{k,l} = v[t]_{k,l} \end{array} \qquad (3)$$

Finding a test suite to cover all activation patterns in a DNN is *intractable*. This is compounded by the fact that a real-world DNN can easily have *millions of neurons*.
Therefore, they adopt the testing approach to first identify a subset of the activation patterns according to certain cover criteria or test requirements, and then generate test cases to cover these activation patterns.

Test Requirements

Quantified Linear Arithmetic over Rationals (QLAR)

They adopt QLAR to express the *DNN Requirement*, abbreviated as DR.



Definition 2 *We use variables* $IV = \{x, x_1, x_2, ...\}$ *to range over the inputs in* D_{L_1}. *Given a network* $\mathcal{N}$, *we let* $V = \{u[x]_{k,l}, v[x]_{k,l} \mid 1 \leq k \leq K, 1 \leq l \leq s_k, x \in IV\}$ *be a set of variables. DR uses the following syntax to write a requirement formula:*

$$\begin{aligned} r &::= Qx.e \mid Qx_1, x_2.e \mid \arg opt_x a : e \mid \arg opt_{x_1,x_2} a : e \\ e &::= a \bowtie 0 \mid e \wedge e \mid \neg e \mid |\{e_1, ..., e_m\}| \bowtie q \\ a &::= w \mid c * w \mid p \mid a + a \mid a - a \end{aligned} \tag{4}$$

where $Q \in \{\exists, \forall\}$, $w \in V$, $c, p \in \mathbb{R}$, $q \in \mathbb{N}$, $opt \in \{\max, \min\}$, $\bowtie \in \{\leq, <, =, >, \geq\}$, *and* $x, x_1, x_2 \in IV$. *We may call* r *a requirement formula,* e *a Boolean formula, and* a *an arithmetic formula. We call the logic* $DR^{\exists}$ *if the opt operators are not allowed, and* $DR^{\exists,+}$ *if both opt operators and the negation operator* $\neg$ *are not allowed.* We use $\mathfrak{R}$ to denote a set of requirement formulas.

Intuitively, formula $\exists x.r$ expresses that there exists an input x such that r is true.

Test Requirements

Semantics

Definition 3 *Given a set $\mathcal{T}$ of test cases and a requirement r, the satisfiability relation $\mathcal{T} \models r$ is defined as follows.*

- $\mathcal{T} \models \exists x.e$ *if there exists some* $t \in \mathcal{T}$ *such that* $\mathcal{T} \models e[x \mapsto t]$, *where* $e[x \mapsto t]$ *is to substitute the occurences of* x *with* t.
- $\mathcal{T} \models \exists x_1, x_2.e$ *if there exist two inputs* $t_1, t_2 \in \mathcal{T}$ *such that* $\mathcal{T} \models e[x_1 \mapsto t_1][x_2 \mapsto t_2]$
- $\mathcal{T} \models \arg\min_x a : e$ *returns an input* $t \in \mathcal{T}$ *such that,* $\mathcal{T} \models e[x \mapsto t]$ *and for all* $t' \in \mathcal{T}$ *such that* $\mathcal{T} \models e[x \mapsto t']$ *we have* $a[x \mapsto t] \leq a[x \mapsto t']$.
- $\mathcal{T} \models \arg\min_{x_1,x_2} a : e$ *returns two inputs* $t_1, t_1 \in \mathcal{T}$ *such that,* $\mathcal{T} \models e[x_1 \mapsto t_1][x_2 \mapsto t_2]$ *and for all* $t_1', t_2' \in \mathcal{T}$ *such that* $\mathcal{T} \models e[x_1 \mapsto t_1'][x_2 \mapsto t_2']$ *we have* $a[x_1 \mapsto t_1][x_2 \mapsto t_2] \leq a[x_1 \mapsto t_1'][x_2 \mapsto t_2']$.

Test Requirements

Semantics

The cases for $\arg\max$ *formulas are similar to those for* $\arg\min$, *by replacing* $\leq$ *with* $\geq$. *The cases for* $\forall$ *formulas are similar to those for* $\exists$ *in the standard way. Note that, when evaluating* $\arg opt$ *formulas (e.g.,* $\arg\min_x a : e$*), if an input* t *is returned, we may need the value* $(\min_x a : e)$ *as well. We use* $val(t, r)$ *to denote such a value for the returned input* t *and the requirement formula* r. *For the evaluation of Boolean expression* e *over an input* t, *we have*

- $\mathcal{T} \models a \bowtie 0$ *if* $a \bowtie 0$
- $\mathcal{T} \models e_1 \wedge e_2$ *if* $\mathcal{T} \models e_1$ *and* $\mathcal{T} \models e_2$
- $\mathcal{T} \models \neg e$ *if not* $\mathcal{T} \models e$
- $\mathcal{T} \models |\{e_1, ..., e_m\}| \bowtie q$ *if* $|\{e_i \mid \mathcal{T} \models e_i, i \in \{1, ..., m\}\}| \bowtie q$

For the evaluation of arithmetic expression a *over an input* t,

- $u[t]_{k,l}$ *and* $v[t]_{k,l}$ *have their values from the activations of the DNN,* $c * u[t]_{k,l}$ *and* $c * v[t]_{k,l}$ *have the standard meaning for* c *being the coefficient,*
- p, $a_1 + a_2$, *and* $a_1 - a_2$ *have the standard semantics.*

Test Requirements

Test Criteria

Proposition 1 *Given a $DR^{\exists,+}$ requirement r, a test suite $\mathcal{T}$ and a subspace $X \subseteq D_{L_1}$, if all test cases in $\mathcal{T}$ are also in X, we have that $X \models r$ implies $\mathcal{T} \models r$ but not vice versa.*

Definition 4 (Test Criterion) *Given a network $\mathcal{N}$, a set $\mathfrak{R}$ of test requirements expressed as DR formulas, and a test suite $\mathcal{T}$, the test criterion $M(\mathfrak{R}, \mathcal{T})$ is as follows:*

$$M(\mathfrak{R}, \mathcal{T}) = \frac{|\{r \in \mathfrak{R} \mid \mathcal{T} \models r\}|}{|\mathfrak{R}|} \tag{7}$$

$$M(\mathfrak{R}, \mathcal{T}) \leq M(\mathfrak{R}, X) \leq M(\mathfrak{R}, D_{L_1}) = 1.0$$

Test Requirements
Computational Complexity

Theorem 1 *Given a network $\mathcal{N}$, a DR requirement formula r with a constant number of $\exists$, $\forall$, $\arg\max$, $\arg\min$ operators, and a test suite $\mathcal{T}$, the checking of $\mathcal{T} \models r$ can be done in polynomial time with respect to the size of $\mathcal{T}$.*

Theorem 2 *Given a network $\mathcal{N}$, a DR requirement formula r with a constant number of $\exists$, $\arg\max$, $\arg\min$ operators, and a subspace $X \subseteq D_{L_1}$, the checking of $X \models r$ is NP-complete. This conclusion also holds for $DR^{\exists}$ and $DR^{\exists,+}$ requirements.*

Table of Contents

Concrete Requirements

In this section, they use $DR^{\exists,+}$ formulas to express several important requirements for DNNs, including Lipschitz continuity and test criteria.

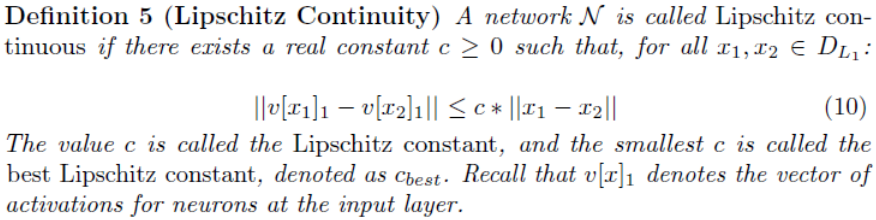
Given a real number b, they generate a finite set $S(D_{L_1}, b)$ of subspaces of D_{L_1} such that for all inputs $x_1, x_2 \in D_{L_1}$, if $||x_1 - x_2|| < b$ then there exists a subspace $X \in S(D_{L_1}, b)$ such that $x_1, x_2 \in X$.

Usually, every subspace $X \in S(D_{L_1}, b)$ can be represented with a box constraint.

$$\bigwedge_{i=1}^{s_1} x(i) - u \leq 0 \wedge x(i) - l \geq 0$$

Concrete Requirements

Lipschitz Continuity



Definition 5 (Lipschitz Continuity) *A network $\mathcal{N}$ is called* Lipschitz continuous *if there exists a real constant $c \geq 0$ such that, for all $x_1, x_2 \in D_{L_1}$:*

$$||v[x_1]_1 - v[x_2]_1|| \leq c * ||x_1 - x_2|| \tag{10}$$

The value c is called the Lipschitz constant, *and the smallest c is called the* best Lipschitz constant, *denoted as c_{best}. Recall that $v[x]_1$ denotes the vector of activations for neurons at the input layer.*

The computation of c_{best} is an NP-hard problem and a smaller c can significantly improve the performance of verification algorithms.

Concrete Requirements

Lipschitz Continuity

The testing of Lipschitz continuity can be guided by having the following requirements:

Definition 6 (Lipschitz Requirements) *Given a real number $c > 0$ and an integer $b > 0$, a set $\mathfrak{R}_{Lip}(b, c)$ of Lipschitz requirements is*

$$\begin{array}{c}\{\exists x_1, x_2.(\|v[x_1]_1 - v[x_2]_1\| - c * \|x_1 - x_2\| > 0) \\ \wedge x_1, x_2 \in X \mid X \in \mathcal{S}(D_{L_1}, b)\}\end{array} \tag{11}$$

Intuitively, for each $X \in S(D_{L_1}, b)$, this requirement expresses the existence of two inputs x_1 and x_2 such that $\mathcal{N}$ breaks the Lipschitz constant c.

Concrete Requirements

Neuron Cover

The Neuron Cover (NC) is an adaptation of the statement cover in software testing.

Definition 7 *The neuron cover for a DNN $\mathcal{N}$ is to find a testsuite $\mathcal{T}$ of inputs such that, for any hidden neuron $n_{k,i}$, there exists test case $t \in \mathcal{T}$ such that $ap[t]_{k,i} = \mathit{true}$.*

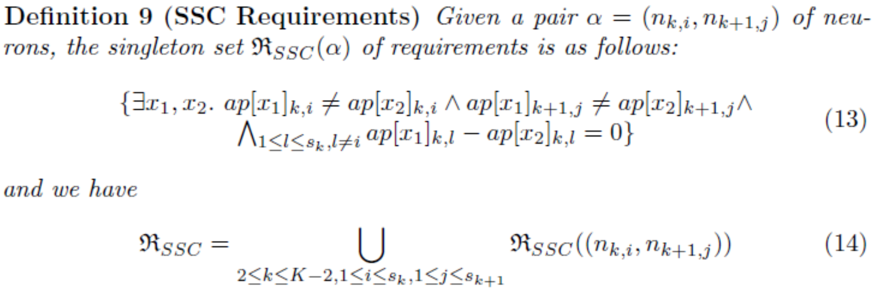
Definition 8 (NC Requirements) *The set $\Re_{NC}$ of requirements is*

$$\{\exists x.ap[x]_{k,i} = \mathit{true} \mid 2 \leq k \leq K-1, 1 \leq i \leq s_k\} \tag{12}$$

Concrete Requirements

Modified Condition/Decision Cover (MC/DC)

According to Sign-Sign Cover (SSC), each neuron $n_{k+1,j}$ can be regarded as a decision such that these neurons at the precedent layer (i.e., the k-th layer) are conditions.



Definition 9 (SSC Requirements) *Given a pair $\alpha = (n_{k,i}, n_{k+1,j})$ of neurons, the singleton set $\mathfrak{R}_{SSC}(\alpha)$ of requirements is as follows:*

$$\begin{array}{c}\{\exists x_1, x_2.\ ap[x_1]_{k,i} \neq ap[x_2]_{k,i} \wedge ap[x_1]_{k+1,j} \neq ap[x_2]_{k+1,j} \wedge \\ \bigwedge_{1 \leq l \leq s_k, l \neq i} ap[x_1]_{k,l} - ap[x_2]_{k,l} = 0\}\end{array} \tag{13}$$

and we have

$$\mathfrak{R}_{SSC} = \bigcup_{2 \leq k \leq K-2, 1 \leq i \leq s_k, 1 \leq j \leq s_{k+1}} \mathfrak{R}_{SSC}((n_{k,i}, n_{k+1,j})) \tag{14}$$

Concrete Requirements

Neuron Boundary Cover

The Neuron Boundary Cover (NBC) aims to cover neuron activation values that exceed pre-specified bounds.

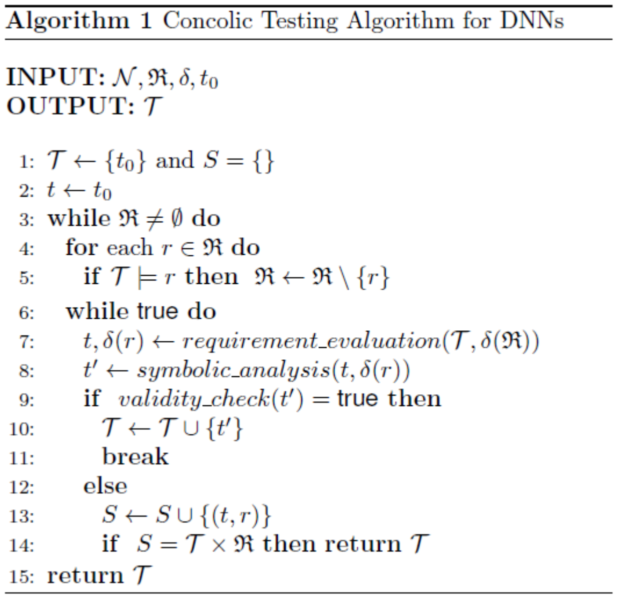
Definition 10 (Neuron Boundary Cover Requirements) *Given two sets of bounds* $h = \{h_{k,i}\}_{2\leq k\leq K-1, 1\leq i\leq s_k}$ *and* $l = \{l_{k,i}\}_{2\leq k\leq K-1, 1\leq i\leq s_k}$, *the set* $\mathfrak{R}_{NBC}(h,l)$ *of requirements is*

$$\begin{gathered}\{\exists x.\ u[x]_{k,i} - h_{k,i} > 0,\ \exists x.\ u[x]_{k,i} - l_{k,i} < 0 \mid \\ 2 \leq k \leq K-1, 1 \leq i \leq s_k\}\end{gathered} \tag{15}$$

where $h_{k,i}$ *and* $l_{k,i}$ *are the upper and lower bounds on the activation value of a neuron* $n_{k,i}$.

Table of Contents

Overall Design



Algorithm 1 Concolic Testing Algorithm for DNNs

INPUT: $\mathcal{N}, \mathfrak{R}, \delta, t_0$
OUTPUT: $\mathcal{T}$

```
1: $\mathcal{T} \leftarrow \{t_0\}$ and $S = \{\}$
2: $t \leftarrow t_0$
3: while $\mathfrak{R} \neq \emptyset$ do
4:   for each $r \in \mathfrak{R}$ do
5:     if $\mathcal{T} \models r$ then $\mathfrak{R} \leftarrow \mathfrak{R} \setminus \{r\}$
6:   while true do
7:     $t, \delta(r) \leftarrow requirement\_evaluation(\mathcal{T}, \delta(\mathfrak{R}))$
8:     $t' \leftarrow symbolic\_analysis(t, \delta(r))$
9:     if $validity\_check(t') =$ true then
10:      $\mathcal{T} \leftarrow \mathcal{T} \cup \{t'\}$
11:      break
12:    else
13:      $S \leftarrow S \cup \{(t, r)\}$
14:      if $S = \mathcal{T} \times \mathfrak{R}$ then return $\mathcal{T}$
15: return $\mathcal{T}$
```

Overall Design

requirement_evaluation (Line 7), aims to find a pair $(t, \delta(r))$ of input and requirement which, according to our concrete evaluation, are the most promising in finding a new test case t' to satisfy the requirement r.

After obtaining $(t, \delta(r))$, *symbolic_analysis* (Line 8), is applied to have a new concrete input t'.

Then *validity_check* (Line 9) is applied to check if the new input is valid or not.

The algorithm either if all test requirements in $\mathcal{R}$ have been satisfied, i.e., $\mathcal{R} = \{\}$, or no further requirement in $\mathcal{R}$ can be satisfied, i.e., $S \cup \mathcal{T} = \mathcal{T} x \mathcal{R}$.

Overall Design

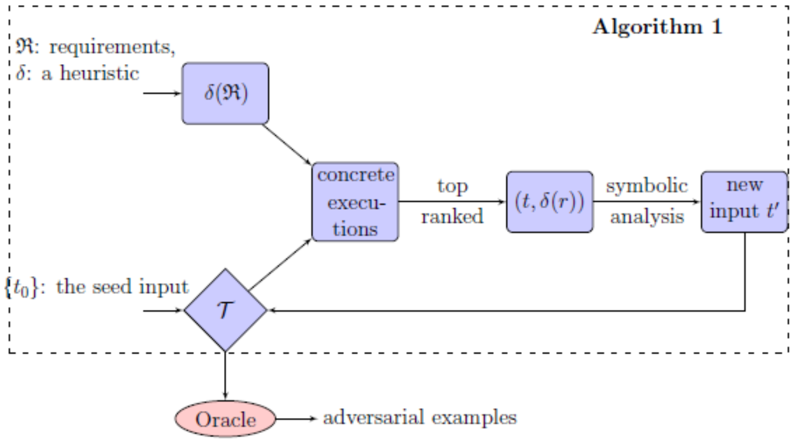


Table of Contents

Requirement Evaluation

Given a set of requirements $\mathcal{R}$ that have not been satisfied, a heuristic δ, and the current set $\mathcal{T}$ of test cases, the goal is to select a concrete input $t \in T$ together with a requirement $r' = \delta(r)$ for some $r \in R$, both of which will be used later in a symbolic approach to find the next concrete input t'.

The general idea of obtaining $(t, \delta(r))$ is as follows:

$$r = \arg\max_{r}\{val(t, \delta(r)) \mid r \in \mathfrak{R}\}.$$

Requirement Evaluation

Heuristics - Lipschitz Continuity

When a Lipschitz requirement r is unsatisfiable on $\mathcal{T}$, we transform it into $\delta(r)$ as follows:

$$\arg\max_{x_1,x_2} . ||v[x_1]_1 - v[x_2]_1|| - c * ||x_1 - x_2|| : x_1, x_2 \in X$$

Requirement Evaluation
Heuristics - Neuron Cover- Neuron Boundary Cover

Neuron Cover:
When a Neuron Cover requirement r is unsatisfiable on $\mathcal{T}$, we transform it into $\delta(r)$ as follows:

$$\arg\max_x c_k \cdot u_{k,i}[x] : \text{true}$$

Neuron Boundary Cover:

$$\arg\max_x u[x]_{k,i} - h_{k,i} : \text{true}$$
$$\arg\max_x l_{k,i} - u[x]_{k,i} : \text{true}$$

Requirement Evaluation

Heuristics - Sign-Sign Cover

When a Sign-Sign Cover requirement r is unsatisfiable on $\mathcal{T}$, we transform it into $\delta(r)$ as follows:

$$\arg\max_{x} -|u[x]_{k,i}| : \text{true}$$

Intuitively, given the decision neuron $n_{k+1,j}$, it selects the condition that is closest to the change of activation sign (i.e., smallest $|u[x]_{k,i}|$).

Table of Contents

Symbolic Generation of New Concrete Inputs

Given a concrete input t and a transformed requirement $r' = \delta(r)$, we need to find the next concrete input t' by symbolic analysis.

Thanks to the use of DR, for each symbolic analysis method, its application to different test criteria can be formulated under a unified logic framework.

Symbolic Generation of New Concrete Inputs

Symbolic Analysis using Linear Programming (LP)

The following linear constraints synthesize a set of inputs that exhibit the same ReLU behaviour as x.

$$\{\mathbf{u_{k,i}} \geq 0 \wedge \mathbf{v_{k,i}} = \mathbf{u_{k,i}} \mid ap[x]_{k,i} \geq 0, k \in [2, K), i \in [1..s_k]\}$$
$$\cup\{\mathbf{u_{k,i}} < 0 \wedge \mathbf{v_{k,i}} = 0 \mid ap[x]_{k,i} < 0, k \in [2, K), i \in [1..s_k]\}$$

$$\{\mathrm{u}_{k,i} = \sum_{1 \leq j \leq s_{k-1}} \{w_{k-1,j,i} \cdot \mathrm{v}_{k-1,j}\} + \delta_{k,i} \mid k \in [2, K], i \in [1..s_k]\}$$

The symbolic analysis for finding a new input t' from a pair (t, r') of input and requirement is equivalent to finding a new activation pattern.

Note that, to make sure that the obtained test case is meaningful, in the LP model an objective is added to minimize the distance between t and t'.

Symbolic Generation of New Concrete Inputs

Neuron Cover

The symbolic analysis of neuron cover takes the input test case t and requirement r', let us say, on the activation of neuron $n_{k,i}$, and it shall return a new test t' such that the test requirement is satisfied by the network instance $N[t']$.

As a result, given $N[t]$'s activation pattern $ap[t]$, we can build up a new activation pattern ap' such that

$$\{ap'_{k,i} = \neg ap[t]_{k,i} \wedge \forall k_1 < k : \bigwedge_{0 \leq i_1 \leq s_{k_1}} ap'_{k_1,i_1} = ap[t]_{k_1,i_1}\}$$

Symbolic Generation of New Concrete Inputs
Neuron Cover

This activation pattern specifies the following conditions:

- $n_{k,i}$'s activation sign is negated: this ensures the aim of the symbolic analysis to activate $n_{k,i}$.
- In the new activation pattern ap', the neurons before layer k preserve their activation signs as in $ap[t]$. Though there may exist various activation patterns that make $n_{k,i}$ activated, for the use of LP modeling, one particular combination of activation signs must be pre-determined.
- Other neurons are irrelevant, as the sign of $n_{k,i}$ is only affected by the activation values of those neurons in previous layers.

Finally, by applying the LP modeling in [5] to the activation pattern, and if there exists a feasible solution, then it will become the new test case t', which makes the DNN satisfy the requirement r'.

Symbolic Generation of New Concrete Inputs

SS Cover

When it comes to the SS Cover, to satisfy the requirement r' we need to find a new test case such that, with respect to the input t, activation signs of $n_{k+1,j}$ and $n_{k,i}$ are negated, while other signs of other neurons at layer k are kept the same as in the case of input t.

To achieve this, the following activation pattern ap' is built up for the LP modeling:

$$\{ap'_{k,i} = \neg ap[t]_{k,i} \wedge ap'_{k+1,j} = \neg ap[t]_{k+1,j} \\ \wedge \forall k_1 < k : \bigwedge_{1 \leq i_1 \leq s_{k_1}} ap'_{k_1,i_1} = ap[t]_{k_1,i_1}\}$$

Symbolic Generation of New Concrete Inputs

Neuron Boundary Cover

In case of the neuron boundary cover, the symbolic analysis aims to find an input t' such that the neuron $n_{k,i}$'s activation value exceeds either its higher bound $h_{k,i}$ or its lower bound $l_{k,i}$.

To achieve this, while preserving the DNN activation pattern as $ap[t]$, we add one of the following constraints into the LP program:

- If $u[x]_{k,i} - h_{k,i} > l_{k,i} - u[x]_{k,i}$: $\mathbf{u_{k,i}} > h_{k,i}$;
- otherwise: $\mathbf{u_{k,i}} < l_{k,i}$.

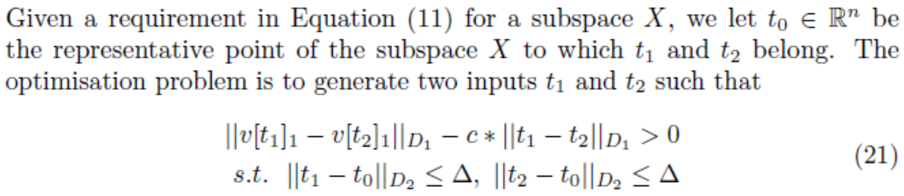
Symbolic Generation of New Concrete Inputs

Symbolic Analysis using Global Optimization

The symbolic analysis for finding a new input can also be implemented by solving the global optimization problem.
That is, by specifying the test requirement as an optimization objective, we apply global optimization to find a test case that makes the test requirement satisfied.

- For the Neuron Cover, the objective is thus to find a t' such that the specified neuron $n_{k,i}$ has $ap[t']_{k,i} = \textbf{true}$.
- In case of the SS Cover, the optimization objective becomes
$$ap[t']_{k,i} \neq ap[t]_{k,i} \wedge ap[t']_{k+1,j} \neq ap[t]_{k+1,j} \wedge \bigwedge_{i' \neq i} ap[t']_{k,i'} = ap[t]_{k,i}$$
- When it comes to the Neuron Boundary Cover, the objective of finding a new input t' can be one of the two forms: 1) $u[t']_{k,i} > h_{k,i}$ or 2) $u[t']_{k,i} < l_{k,i}$.

Symbolic Generation of New Concrete Inputs



Given a requirement in Equation (11) for a subspace X, we let $t_0 \in \mathbb{R}^n$ be the representative point of the subspace X to which t_1 and t_2 belong. The optimisation problem is to generate two inputs t_1 and t_2 such that

$$\begin{aligned} &||v[t_1]_1 - v[t_2]_1||_{D_1} - c * ||t_1 - t_2||_{D_1} > 0 \\ &s.t. \;\; ||t_1 - t_0||_{D_2} \leq \Delta, \;\; ||t_2 - t_0||_{D_2} \leq \Delta \end{aligned} \tag{21}$$

The above problem can be efficiently solved by a novel *alternating compass search* scheme.

Symbolic Generation of New Concrete Inputs

Algorithms for Lipschitz Test Case Generation - Stage 1

$$\min_{t_1} F(t_1, t_0) = -||v[t_1]_1 - v[t_0]_1||_{D_1}$$
$$\text{s.t.} \quad ||t_1 - t_0||_{D_2} \leq \Delta$$

The above objective enables the algorithm to search for an optimal t_1 in the space of a norm ball or hypercube centred on t_0 with radius Δ, such that the norm distance of $v[t_1]_1$ and $v[t_0]_1$ is as large as possible.

To solve the above the problem, we use the compass search method. If it is convergent and we can find an optimal t_1 as

$$t_1^* = \arg\min_{t_1} F(t_1, t_0) \quad \text{s.t.} \ ||t_1 - t_0||_{D_2} \leq \Delta$$

Symbolic Generation of New Concrete Inputs

Algorithms for Lipschitz Test Case Generation - Stage 2

$$\min_{t_2} F(t_1^*, t_2) = -||v[t_2]_1 - v[t_1^*]_1||_{D_1}$$
$$\text{s.t.} \quad ||t_2 - t_0||_{D_2} \leq \Delta$$

If it is convergent at t_2^*, and we still cannot find such a input pair, we modify the objective function again.

Symbolic Generation of New Concrete Inputs

Algorithms for Lipschitz Test Case Generation - Stage 3

In this case, they return the best input pair we can find, i.e., t_1^* and t_2^* , and the largest Lipschitz constant.

In summary, they start from the given t_0 to search for an image t_1 in a norm ball or hypercube (the optimization trajectory on the norm ball space is denoted as $S(t_0, \Delta(t_0))$ such that $Lip(t_0, t_1) > c$ (this step is symbolic execution).

If they cannot find it, they modify the optimization objective function by replacing t_0 with t_1^* (the best concrete input found in this optimization trace) to initiate another optimization trajectory on the space, i.e., $S(t_1^*, \Delta(t_0))$.

This process is repeated until the optimization trace gradually covers the whole norm ball space $S(\Delta(t_0))$.

Oracle

Definition 11 (Validity Checking) *Given a set O of correctly classified inputs (e.g., the training dataset) and a real number b, a test case $t' \in \mathcal{T}$ passes the validity checking if*

$$\exists t \in O : \quad ||t - t'|| \leq b \tag{24}$$

Definition 12 (Robustness Oracle) *Given a set O of correctly classified inputs, a test case t' passes the robustness oracle if*

$$\arg\max_j v[t']_{K,j} = \arg\max_j v[O(t')]_{K,j} \tag{25}$$

A Summary of Coverage-based DNN Testing

Table 1: Comparison between different coverage-based DNN testing methods

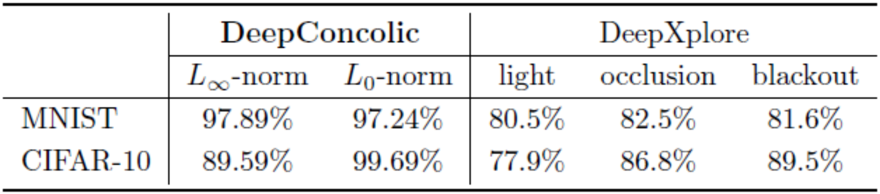
	DeepConcolic	DeepXplore [3]	DeepTest [4]	DeepCover [5]	DeepGauge [6]
Coverage criteria	NC, SSC, NBC etc.	NC	NC	MC/DC	NBC etc.
Test generation	concolic	dual-optimisation	greedy search	symbolic execution	gradient descent methods
DNN inputs	single	multiple	single	single	single
Image inputs	single/multiple	multiple	multiple	multiple	multiple
Distance metric	L_∞, L_0-norm	L_1-norm	Jaccard distance	L_∞-norm	L_∞-norm

Table of Contents

Experimental Results

Comparison with DeepXplore - Neuron Cover

Table 2: Neuron coverage of DeepConcolic and DeepXplore



	DeepConcolic		DeepXplore		
	L_∞-norm	L_0-norm	light	occlusion	blackout
MNIST	97.89%	97.24%	80.5%	82.5%	81.6%
CIFAR-10	89.59%	99.69%	77.9%	86.8%	89.5%

Experimental Results

Comparison with DeepXplore - Adversarial

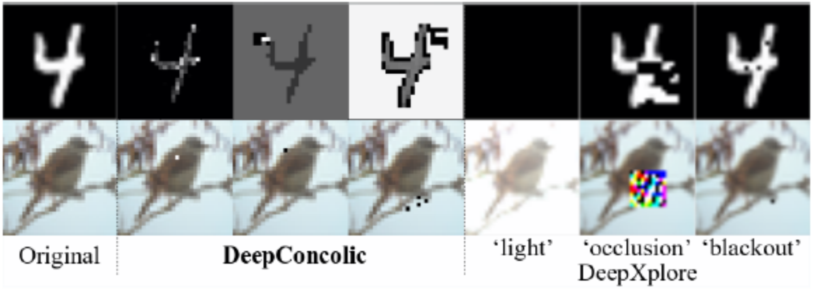
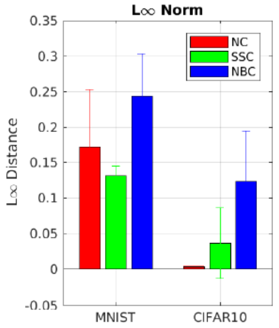


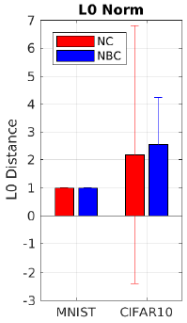
Figure: It is worth noting that, although DeepConcolic does not impose particular domain-specific constraints on the original image as DeepXplore does, the concolic testing procedure automatically generates test cases that resemble "human perception".

Experimental Results

Concolic Testing Results on Different Test Criteria

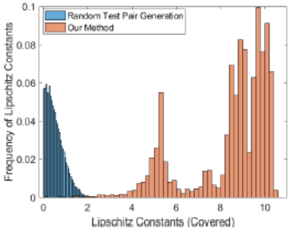
	L_∞-norm						L_0-norm					
	MNIST			**CIFAR-10**			**MNIST**			**CIFAR-10**		
	coverage percentage	adversary /test suite	minimum distance	coverage percentage	adversary /test suite	minimum distance	coverage percentage	adversary /test suite	minimum distance	coverage percentage	adversary /test suite	minimum distance
NC	97.89%	9.69%	0.0039	89.59%	0.32%	0.0039	97.24%	0.07%	1	99.69%	13.91%	1
SSC	94.10%	0.37%	0.1215	100%	1.74%	0.0039	–	–	–	–	–	–
NBC	60.74%	0.83%	0.0806	85.57%	7.01%	0.0113	48.52%	0.06%	1	100%	4.30%	1



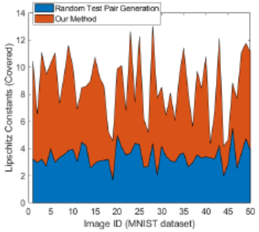


Experimental Results

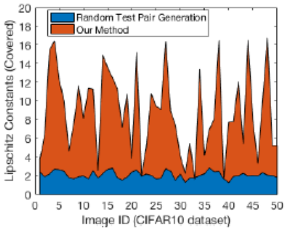
Experimental Results for Lipschitz Constant Testing



(a)



(b)



(c)