The *Science* and *Engineering* of Testing

Software 1.0 and 2.0

Subhajit Roy
subhajit@iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology Kanpur

Outline I

Outline II

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

What is common in these movies?

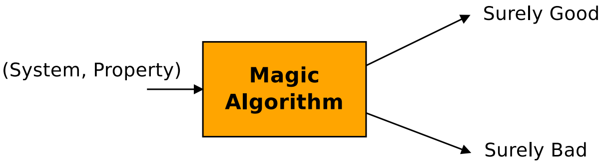
What is common in these movies?

What is common in these movies?

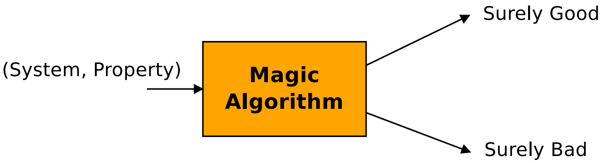
... only because they did not test/verify their AI!

Image credits: http://imdb.com

Our Dream Engine



Our Dream Engine



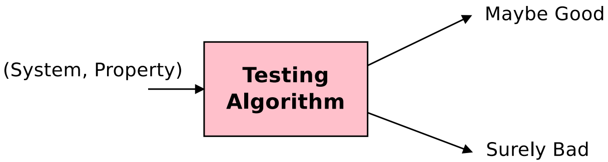
Impossible for a Turing-complete system!

Life is all about compromises

The "pass all good" versus "block all bad" dilemma

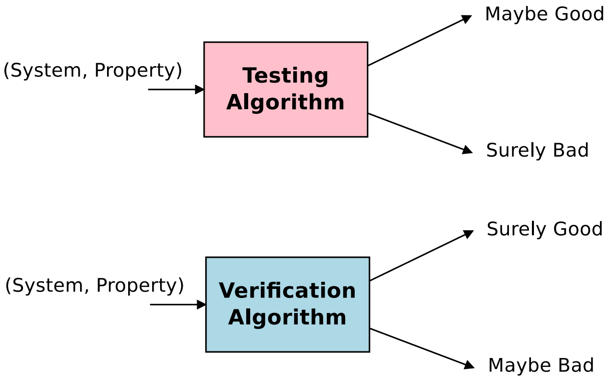
Life is all about compromises

The “pass all good” versus “block all bad” dilemma



Life is all about compromises

The “pass all good” versus “block all bad” dilemma



Outline I

Outline II

What is Testing?

What is Testing?

What is program testing?

Searching for inputs that *fail* the program.

What is Testing?

What is program testing?

Searching for inputs that *fail* the program.

What is Testing?

What is program testing?

Searching for inputs that *fail* the program. Exciting a program with a set of inputs—in a search for *any* input that shows a bug.

What is Testing?

What is program testing?

Searching for inputs that *fail* the program. Exciting a program with a set of inputs—in a search for *any* input that shows a bug.

What is program verification?

Synthesizing a mathematical proof for the *correctness* of the program.

What is Testing?

What is program testing?

Searching for inputs that *fail* the program. Exciting a program with a set of inputs—in a search for *any* input that shows a bug.

What is program verification?

Synthesizing a mathematical proof for the *correctness* of the program.

What is Testing?

What is program testing?

Searching for inputs that *fail* the program. Exciting a program with a set of inputs—in a search for *any* input that shows a bug.

What is program verification?

Synthesizing a mathematical proof for the *correctness* of the program. Build an argument—that the program behaves correctly on *all* inputs.

Testing Ingredients

- How to identify a bug?

Testing Ingredients

- How to identify a bug?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing? **Test Adequacy**
- How much of the system is visible?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing? **Test Adequacy**
- How much of the system is visible?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing? **Test Adequacy**
- How much of the system is visible? **System Visibility**
- What SDLC stage are we in?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing? **Test Adequacy**
- How much of the system is visible? **System Visibility**
- What SDLC stage are we in?

Testing Ingredients

- How to identify a bug? — Test Oracle
- When to stop testing? — Test Adequacy
- How much of the system is visible? — System Visibility
- What SDLC stage are we in? — Testing Stage
- How to test?

Testing Ingredients

- How to identify a bug? — Test Oracle
- When to stop testing? — Test Adequacy
- How much of the system is visible? — System Visibility
- What SDLC stage are we in? — Testing Stage
- How to test?

Testing Ingredients

- How to identify a bug? **Test Oracle**
- When to stop testing? **Test Adequacy**
- How much of the system is visible? **System Visibility**
- What SDLC stage are we in? **Testing Stage**
- How to test? **Testing Algorithm**

System Visibility

- Black-box testing

System Visibility

- Black-box testing
- White-box testing

System Visibility

- Black-box testing
- White-box testing

System Visibility

- Black-box testing
- White-box testing
- Grey-box testing

SDLC Stage

- unit testing

SDLC Stage

- unit testing
- integration testing

SDLC Stage

- unit testing
- integration testing
- system testing

SDLC Stage

- unit testing
- integration testing
- system testing
- acceptance testing

Constructing an Oracle

- Functional specification

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification
  - Easy to implement less efficient baseline, regression testing etc.

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification
  - Easy to implement less efficient baseline, regression testing etc.
- Emsembles as oracles

Constructing an Oracle

- Functional specification
    - Rigorous mathematical description of the input-output relation, rarely available
    - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
    - Robustness, fairness, secure, etc.
- Differential testing
    - Another system for the functional specification
    - Easy to implement less efficient baseline, regression testing etc.
- Emsembles as oracles
    - Majority voting amongst a group of systems to establish the ground-truth (eg. random forest)

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification
  - Easy to implement less efficient baseline, regression testing etc.
- Emsembles as oracles
  - Majority voting amongst a group of systems to establish the ground-truth (eg. random forest)
- Metamorphic relation (between inputs) as oracle

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification
  - Easy to implement less efficient baseline, regression testing etc.
- Emsembles as oracles
  - Majority voting amongst a group of systems to establish the ground-truth (eg. random forest)
- Metamorphic relation (between inputs) as oracle
  - Relation between outputs of pair of inputs is often easy to establish; eg. ADD(x, y) = ADD(x+1, y+1) + 2

Constructing an Oracle

- Functional specification
  - Rigorous mathematical description of the input-output relation, rarely available
  - Humans in the loop
- Properties as oracles: can be seen as a Hoare Triple $\{Pre\}\ S\ \{Post\}$
  - Robustness, fairness, secure, etc.
- Differential testing
  - Another system for the functional specification
  - Easy to implement less efficient baseline, regression testing etc.
- Emsembles as oracles
  - Majority voting amongst a group of systems to establish the ground-truth (eg. random forest)
- Metamorphic relation (between inputs) as oracle
  - Relation between outputs of pair of inputs is often easy to establish; eg. ADD(x, y) = ADD(x+1, y+1) + 2
  - Referred to as *metamorphic testing*

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

How to select test inputs?

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

How to select test inputs?

Principle: Test via an *equivalence partitioning* on inputs

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

How to select test inputs?

Principle: Test via an *equivalence partitioning* on inputs

Equivalence classes

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

How to select test inputs?

Principle: Test via an *equivalence partitioning* on inputs

Equivalence classes

- Define a partitioning on the input-space

Testing Adequacy

Possibly unbounded number of values (eg. strings as input)

How to select test inputs?

Principle: Test via an *equivalence partitioning* on inputs

Equivalence classes

- Define a partitioning on the input-space
- Pick a representative input from each partition

Defining Equivalence Partitions

- Semantics of input vector

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S'

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S'

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S' *kills* a mutatant
  - Referred to as *mutation testing*

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S' *kills* a mutatant
  - Referred to as *mutation testing*
- Coverage metrics

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S' *kills* a mutatant
  - Referred to as *mutation testing*
- Coverage metrics
  - coverage metric is defined on set of control-flow or dataflow entities (recall *metric space*)

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S' *kills* a mutatant
  - Referred to as *mutation testing*
- Coverage metrics
  - coverage metric is defined on set of control-flow or dataflow entities (recall *metric space*)
  - inputs I_1 and I_2 belong to different partitions if they *cover* a different subset of these entities

Defining Equivalence Partitions

- Semantics of input vector
  - inputs I_1 and I_2 belong to different partitions if they are *likely* to create different outputs (positive and negative numbers, empty and non-empty strings etc.)
  - eg. product of matrices
- System behavior
  - inputs I_1 and I_2 belong to different partitions if they create different behaviors in the system S and a *mutated* system S'   *kills* a mutatant
  - Referred to as *mutation testing*
- Coverage metrics
  - coverage metric is defined on set of control-flow or dataflow entities (recall *metric space*)
  - inputs I_1 and I_2 belong to different partitions if they *cover* a different subset of these entities
  - the coverage metric quantifies the *goodness* of a test-suite

Control-flow graph

Primary data-structure for flow analysis of programs.

Control-flow graph

Primary data-structure for flow analysis of programs.

- Each node in the CFG is a basic-block : piece of straight-line code i.e. a sequence of instructions with single entry and single exit (no jumps or jump targets)

Control-flow graph

Primary data-structure for flow analysis of programs.

- Each node in the CFG is a basic-block : piece of straight-line code i.e. a sequence of instructions with single entry and single exit (no jumps or jump targets)

Control-flow graph

Primary data-structure for flow analysis of programs.

- Each node in the CFG is a basic-block : piece of straight-line code i.e. a sequence of instructions with single entry and single exit (no jumps or jump targets)

  basic block
- Any possible control flow from one basic-block to another is represented by a control-flow edge

Control-flow graph

Primary data-structure for flow analysis of programs.

- Each node in the CFG is a basic-block : piece of straight-line code i.e. a sequence of instructions with single entry and single exit (no jumps or jump targets) **basic block**
- Any possible control flow from one basic-block to another is represented by a control-flow edge

Control-flow graph

Primary data-structure for flow analysis of programs.

- Each node in the CFG is a basic-block : piece of straight-line code i.e. a sequence of instructions with single entry and single exit (no jumps or jump targets) **basic block**
- Any possible control flow from one basic-block to another is represented by a control-flow edge **control-flow edge**
- Two important blocks: entry block and exit block

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)
- Condition coverage (Predicate coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)
- Condition coverage (Predicate coverage)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)
- Condition coverage (Predicate coverage)
- Modified condition and decision coverage (MCDC)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)
- Condition coverage (Predicate coverage)
- Modified condition and decision coverage (MCDC)

Control-flow coverage a.k.a Code coverage

- Statement coverage (Line coverage, Node coverage)
- Branch coverage (Decision Coverage)
- Condition coverage (Predicate coverage)
- Modified condition and decision coverage (MCDC)
- Path Coverage

Coverage Metrics

Does 100% statement coverage imply 100% branch coverage?

Coverage Metrics

Does 100% statement coverage imply 100% branch coverage?

Does 100% branch coverage imply 100% statement coverage?

Coverage Metrics

Does 100% statement coverage imply 100% branch coverage?

Does 100% branch coverage imply 100% statement coverage?

Does 100% branch coverage imply 100% path coverage?

Coverage Metrics

Does 100% statement coverage imply 100% branch coverage?

Does 100% branch coverage imply 100% statement coverage?

Does 100% branch coverage imply 100% path coverage?

Does 100% path coverage means that the program has no bug?

Testing Algorithms

Solves an *optimization problem* w.r.t. the test goal (eg. coverage metric, property violation likelihood etc.)

Testing Algorithms

Solves an *optimization problem* w.r.t. the test goal (eg. coverage metric, property violation likelihood etc.)

- Randomized Algorithms (greedy search, fuzzing)

Testing Algorithms

Solves an *optimization problem* w.r.t. the test goal (eg. coverage metric, property violation likelihood etc.)

- Randomized Algorithms (greedy search, fuzzing)
- Symbolic Algorithms (symbolic execution, concolic execution)

Testing Algorithms

Solves an *optimization problem* w.r.t. the test goal (eg. coverage metric, property violation likelihood etc.)

- Randomized Algorithms (greedy search, fuzzing)
- Symbolic Algorithms (symbolic execution, concolic execution)
- Evolutionary Search (genetic algorithms (like differential evolution), ant colony)

Testing Algorithms

Solves an *optimization problem* w.r.t. the test goal (eg. coverage metric, property violation likelihood etc.)

- Randomized Algorithms (greedy search, fuzzing)
- Symbolic Algorithms (symbolic execution, concolic execution)
- Evolutionary Search (genetic algorithms (like differential evolution), ant colony)
- Gradient-based Search (gradient descent)

Symbolic Execution

Analyse this

What inputs cause this program to violate the assertion?

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```

Symbolic Execution

Analyze this

OK, let's answer this!

Symbolic Execution

Analyze this

OK, let's answer this!

Symbolic Execution

Analyze this

OK, let's answer this!

A customer buys 4 apples and 2 bananas for Rs. 50. Another customer buys 5 apples and 4 bananas for Rs. 70. What is the cost of each item?

Symbolic Execution

Analyze this

OK, let's answer this!

A customer buys 4 apples and 2 bananas for Rs. 50. Another customer buys 5 apples and 4 bananas for Rs. 70. What is the cost of each item?

== **Use symbols to represent unknowns!** **==**

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

Explore all paths!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```

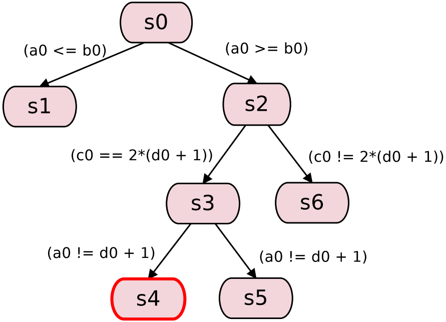
Explore all paths!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```



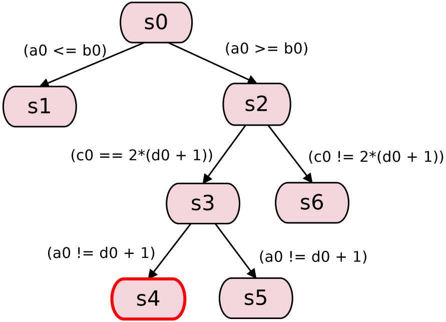
Explore all paths!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```



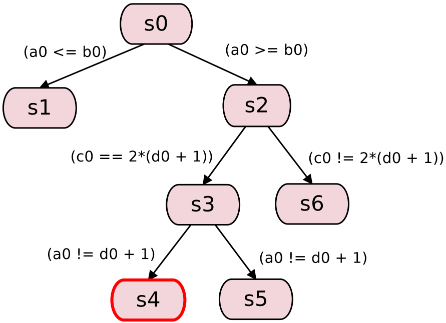
Explore all paths!

Symbolic Execution

Simple idea

Execute a program with symbolic inputs!

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```



Explore all paths!

Symbolic Execution Tree

Concolic Execution

ConcolicExecution

Concolic Execution

ConcolicExecution

$\vec{t} := random()$

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)

Concolic Execution

ConcolicExecution

$\vec{t} := \mathit{random}()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $\mathit{out}, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, \mathit{out}$) $==$ Error **then**

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if

Concolic Execution

ConcolicExecution

$\vec{t} := \mathit{random}()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $\mathit{out}, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, \mathit{out}$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

Concolic Execution

ConcolicExecution

$\vec{t} := \mathit{random}()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $\mathit{out}, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, \mathit{out}$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

φ symbolically encodes a program path

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

φ symbolically encodes a program path

Path Condition (PC)

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

φ symbolically encodes a program path

Path Condition (PC)

SearchHeuristic performs test selection/prioritization

Concolic Execution

ConcolicExecution

$\vec{t} := random()$
$T := \{\vec{t}\}$
while $\neg$ goal **do**
  $out, \varphi :=$ ConcolicRun($\vec{t}$)
  if TestOracle($\vec{t}, out$) $==$ Error **then**
    fail
  end if
  $\varphi' :=$ SearchHeuristic(φ)
  $\vec{t} :=$ Solve(φ')
  T.append($\vec{t}$)
end while
return T

φ symbolically encodes a program path — **Path Condition (PC)**

SearchHeuristic performs test selection/prioritization — **Search Heuristic**

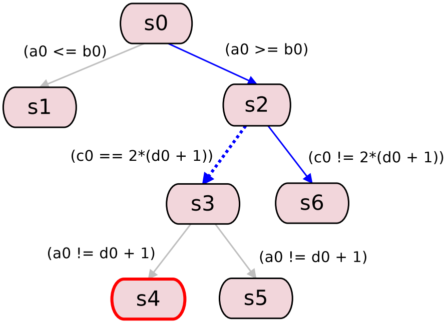
Search Heuristic

Search Heuristic

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```

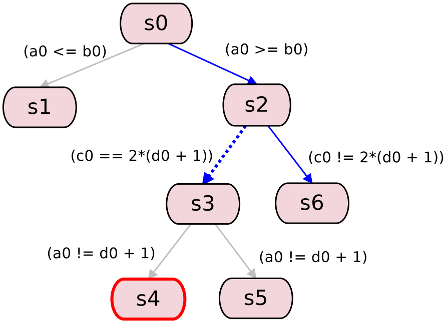
Search Heuristic

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```



Search Heuristic

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```

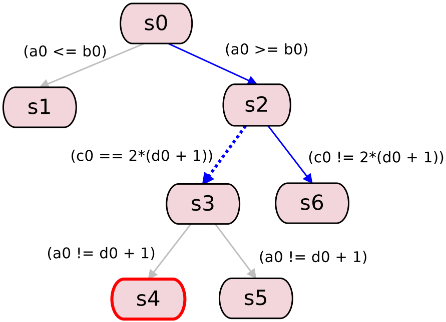


How to modify PCs for next input?

Current: $(a_0 \geq b_0) \wedge (c_0 \neq 2 * (d_0 + 1))$

Search Heuristic

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```

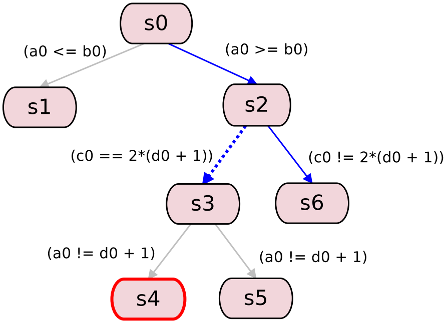


How to modify PCs for next input?

Current: $(a_0 \geq b_0) \wedge (c_0 \neq 2 * (d_0 + 1))$

Search Heuristic

```
int main(){
 input(a,b,c,d);
 if ( a <= b){
    c++;
 }
 else {
    d++;
    if ( c == 2*d)
        assert(a > d)
 }
}
```



How to modify PCs for next input?

Current: $(a_0 \geq b_0) \wedge (c_0 \neq 2 * (d_0 + 1))$
Next: $(a_0 \geq b_0) \wedge \neg(c_0 \neq 2 * (d_0 + 1))$

Homework

Design search heuristics for *depth-first search* and *breadth-first search*

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]
3. If t is new, result, cov $\leftarrow$ Run(t)

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]
3. If t is new, result, cov $\leftarrow$ Run(t)
4. if result is a crash, declare t as *Crashing Input*

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]
3. If t is new, $\text{result}, \text{cov} \leftarrow \text{Run(t)}$
4. if result is a crash, declare t as *Crashing Input*
5. if cov increases coverage, declare t as *Interesting Input andaddittoworklis*

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]
3. If t is new, result, cov $\leftarrow$ Run(t)
4. if result is a crash, declare t as *Crashing Input*
5.

   if cov increases coverage, declare t as *Interesting Input and add t to* worklis
6. $e \leftarrow$ selectEnergy(t, result, cov)

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow \text{selectNextInput(worklist)}$[1]
3. If t is new, $\text{result}, \text{cov} \leftarrow \text{Run(t)}$
4. if result is a crash, declare t as *Crashing Input*
5. if cov increases coverage, declare t as *Interesting Input andaddittoworklis*
6. $e \leftarrow \text{selectEnergy}(\text{t}, \text{result}, \text{cov})$
7. $\text{worklist} \leftarrow \text{worklist} + \text{mutate}(\text{t}, \text{e})$

[1] t is not removed from worklist

Fuzzing Algorithm

1. worklist $\leftarrow$ seedinputs
2. $t \leftarrow$ selectNextInput(worklist)[1]
3. If t is new, result, cov $\leftarrow$ Run(t)
4. if result is a crash, declare t as *Crashing Input*
5. if cov increases coverage, declare t as *Interesting Input andadditto* worklis
6. $e \leftarrow$ selectEnergy(t, result, cov)
7. worklist $\leftarrow$ worklist + mutate(t, e)
8. Goto 2

[1] t is not removed from worklist

Fuzzing

Fuzzing

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs
while not timeout **do**

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    t ← selectNextInput(worklist)   ▷ not removed from worklist

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
  t ← selectNextInput(worklist) ▷ not removed from worklist
  if t is new **then**

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs
while not timeout **do**
  $t \leftarrow$ selectNextInput(worklist) $\triangleright$ not removed from worklist
  if t is new **then**
    $result, cov \leftarrow \text{Run}(t)$

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    t ← selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow$ Run(t)
    end if

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    t ← selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov$ ← Run(t)
    end if
    if result is a crash **then**

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs
while not timeout **do**
  $t \leftarrow$ selectNextInput(worklist) ▷ not removed from worklist
  if t is new **then**
    $result, cov \leftarrow \text{Run}(t)$
  end if
  if result is a crash **then**
    declare t as *Crashing Input*

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    t ← selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow$ Run(t)
    end if
    if result is a crash **then**
        declare t as *Crashing Input*
    end if

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
  $t \leftarrow$ selectNextInput(worklist) ▷ not removed from worklist
  if t is new **then**
    $result, cov \leftarrow \text{Run}(t)$
  end if
  if result is a crash **then**
    declare t as *Crashing Input*
  end if
  if cov increases coverage **then**

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    $t \leftarrow$ selectNextInput(worklist) ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow \text{Run}(t)$
    end if
    if result is a crash **then**
        declare t as *Crashing Input*
    end if
    if cov increases coverage **then**
        declare t as *Interesting Input*

Fuzzing

Fuzzing

```
worklist ← seedinputs
while not timeout do
    t ← selectNextInput(worklist)        ▷ not removed from worklist
    if t is new then
        result, cov ← Run(t)
    end if
    if result is a crash then
        declare t as Crashing Input
    end if
    if cov increases coverage then
        declare t as Interesting Input
        worklist.add(t)
```

Fuzzing

Fuzzing

```
worklist ← seedinputs
while not timeout do
    t ← selectNextInput(worklist)        ▷ not removed from worklist
    if t is new then
        result, cov ← Run(t)
    end if
    if result is a crash then
        declare t as Crashing Input
    end if
    if cov increases coverage then
        declare t as Interesting Input
        worklist.add(t)
        if it is crashing then
```

Fuzzing

Fuzzing

```
worklist ← seedinputs
while not timeout do
    t ← selectNextInput(worklist)        ▷ not removed from worklist
    if t is new then
        result, cov ← Run(t)
    end if
    if result is a crash then
        declare t as Crashing Input
    end if
    if cov increases coverage then
        declare t as Interesting Input
        worklist.add(t)
        if it is crashing then
            mark as unique crash
```

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    t ← selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow \text{Run}(t)$
    end if
    if result is a crash **then**
        declare t as *Crashing Input*
    end if
    if cov increases coverage **then**
        declare t as *Interesting Input*
        worklist.add(t)
        if it is crashing **then**
            mark as *unique crash*
        end if

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs
while not timeout **do**
  $t \leftarrow$ selectNextInput(worklist) $\triangleright$ not removed from worklist
  if t is new **then**
    $result, cov \leftarrow \text{Run}(t)$
  end if
  if result is a crash **then**
    declare t as *Crashing Input*
  end if
  if cov increases coverage **then**
    declare t as *Interesting Input*
    worklist.add(t)
    if it is crashing **then**
      mark as *unique crash*
    end if
  end if

Fuzzing

Fuzzing

worklist $\leftarrow$ seedinputs
while not timeout **do**
    $t \leftarrow$ selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow \text{Run}(t)$
    end if
    if result is a crash **then**
        declare t as *Crashing Input*
    end if
    if cov increases coverage **then**
        declare t as *Interesting Input*
        worklist.add(t)
        if it is crashing **then**
            mark as *unique crash*
        end if
    end if
    $e \leftarrow \text{selectEnergy}(\text{t}, \text{result}, \text{cov})$

Fuzzing

Fuzzing

worklist ← seedinputs
while not timeout **do**
    $t \leftarrow$ selectNextInput(worklist)   ▷ not removed from worklist
    if t is new **then**
        $result, cov \leftarrow \text{Run}(t)$
    end if
    if result is a crash **then**
        declare t as *Crashing Input*
    end if
    if cov increases coverage **then**
        declare t as *Interesting Input*
        worklist.add(t)
        if it is crashing **then**
            mark as *unique crash*
        end if
    end if
    $e \leftarrow \text{selectEnergy}(\text{t}, \text{result}, \text{cov})$
    worklist ← worklist.*add*(mutate(t, e))

Fuzzing

Fuzzing

```
worklist ← seedinputs
while not timeout do
    t ← selectNextInput(worklist)        ▷ not removed from worklist
    if t is new then
        result, cov ← Run(t)
    end if
    if result is a crash then
        declare t as Crashing Input
    end if
    if cov increases coverage then
        declare t as Interesting Input
        worklist.add(t)
        if it is crashing then
            mark as unique crash
        end if
    end if
    e ← selectEnergy(t, result, cov)
    worklist ← worklist.add(mutate(t, e))
end while
```

Gradient Descent

Consider optimizing w.r.t. an objective function $\varphi(\vec{w})$

Gradient Descent

Gradient Descent

Consider optimizing w.r.t. an objective function $\varphi(\vec{w})$

Gradient Descent

$\vec{w} := random()$

Gradient Descent

Consider optimizing w.r.t. an objective function $\varphi(\vec{w})$

Gradient Descent

$\vec{w} := random()$
while not converged **do**

Gradient Descent

Consider optimizing w.r.t. an objective function $\varphi(\vec{w})$

Gradient Descent

$\vec{w} := \mathit{random}()$
while not converged **do**
  $\vec{w} := \vec{w} - \alpha \cdot \nabla_{\vec{w}}\varphi$

Gradient Descent

Consider optimizing w.r.t. an objective function $\varphi(\vec{w})$

Gradient Descent

$\vec{w} := \mathit{random}()$
while not converged **do**
  $\vec{w} := \vec{w} - \alpha \cdot \nabla_{\vec{w}}\varphi$
end while

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs
- Bug Localization

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs
- Bug Localization

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs
- Bug Localization
- Debugging

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs
- Bug Localization
- Debugging

Challenges in program testing

- Test Generation
  optimization problem: minimum tests to find max bugs/generate highest coverage
- Test Prioritization
  ranking problem: higher ranked tests more likely to find bugs
- Test Selection
  selection problem: select "k" tests that are most likely to catch errors
- Differential/Regression testing
  constrained optimization: select tests that lead to different behaviors
- Testing multithreaded and distributed systems
- Testing for security bugs
- Bug Localization
- Debugging
- Repair

Outline I

Outline II

- Neural networks as Programs
- Attacks on Neural Networks

5. Testing neural networks
   - Visibility and Stage
   - Neural Network Oracles
   - Test Adequacy
   - Algorithms to achieve high NN coverage

6. Conclusion

Properties of an algorithm

- Soundness (Precision)

Properties of an algorithm

- Soundness (Precision)
- Completeness (Recall)

Properties of an algorithm

- Soundness (Precision)
- Completeness (Recall)
- Termination

Soundness versus Completeness

Given a claim $\mathcal{C}$ and an algorithm $\mathcal{A}$ that attempts to validate the claim:

Soundness versus Completeness

Given a claim $\mathcal{C}$ and an algorithm $\mathcal{A}$ that attempts to validate the claim:

- **Soundness:** $\mathcal{A}$ is sound if whenever $\mathcal{A}$ signals YES, $\mathcal{C}$ holds

Soundness versus Completeness

Given a claim $\mathcal{C}$ and an algorithm $\mathcal{A}$ that attempts to validate the claim:

- **Soundness:** $\mathcal{A}$ is sound if whenever $\mathcal{A}$ signals YES, $\mathcal{C}$ holds
- **Completeness:** $\mathcal{A}$ is complete if it always signals YES whenever $\mathcal{C}$ holds

Soundness/Completeness of Testing/Verification

If the claim is on the *correctness* of the program, can you deduce the soundness/completeness of testing/verification?

Soundness/Completeness of Testing/Verification

If the claim is on the *correctness* of the program, can you deduce the soundness/completeness of testing/verification?

- Verification is sound but testing is not

Soundness/Completeness of Testing/Verification

If the claim is on the *correctness* of the program, can you deduce the soundness/completeness of testing/verification?

- Verification is sound but testing is not
- Testing is complete but verification is not

Soundness/Completeness of Testing/Verification

If the claim is on the *correctness* of the program, can you deduce the soundness/completeness of testing/verification?

- Verification is sound but testing is not
- Testing is complete but verification is not

Soundness/Completeness of Testing/Verification

If the claim is on the *correctness* of the program, can you deduce the soundness/completeness of testing/verification?

- Verification is sound but testing is not
- Testing is complete but verification is not

Note: If the claim is on the presence of bugs, then testing is sound but not complete.

Properties of a function

Continuity

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x + \delta) \sim f(x)$

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \underset{\delta \to 0}{lim} f(x+\delta) \sim f(x)$

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x + \delta) \sim f(x)$

Differentiability

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x+\delta) \sim f(x)$

Differentiability

- a function is differentiable if derivative exists at all possible values of the arguments in the function domain

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x+\delta) \sim f(x)$

Differentiability

- a function is differentiable if derivative exists at all possible values of the arguments in the function domain
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f'(x) = \frac{f(x+\delta)-f(x)}{\delta}$

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x + \delta) \sim f(x)$

Differentiability

- a function is differentiable if derivative exists at all possible values of the arguments in the function domain
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f'(x) = \frac{f(x+\delta)-f(x)}{\delta}$
- if derivative does not exist at a point, the set of *subderivatives* is a non-empty closed interval $[a, b]$, where a, b are the one-sided limits:

Properties of a function

Continuity

- a function is continuous if a continuous variation of the arguments induces a continuous variation in the outputvalue of the function, i.e. no abrupt changes in output (discontinuities)
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f(x+\delta) \sim f(x)$

Differentiability

- a function is differentiable if derivative exists at all possible values of the arguments in the function domain
- $\forall x \in Domain(f).\ \lim_{\delta \to 0} f'(x) = \frac{f(x+\delta)-f(x)}{\delta}$
- if derivative does not exist at a point, the set of *subderivatives* is a non-empty closed interval $[a, b]$, where a, b are the one-sided limits:
  - $a = \lim_{h \to 0^+} \frac{x+h}{h}$, $b = \lim_{h \to 0^-} \frac{x+h}{h}$

Nicer Properties

Lipshitz continuous

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Nicer Properties

Lipshitz continuous

- a function is Lipshitz continuous if there exists a bound on how fast the function can change, i.e. its slope

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Nicer Properties

Lipshitz continuous

- a function is Lipshitz continuous if there exists a bound on how fast the function can change, i.e. its slope
- $\forall x_1, x_2.\ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Nicer Properties

Lipshitz continuous

- a function is Lipshitz continuous if there exists a bound on how fast the function can change, i.e. its slope
- $\forall x_1, x_2.\ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$
- K is called the Lipshitz constant; the smallest K is the best-Lipshitz constant

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Nicer Properties

Lipshitz continuous

- a function is Lipshitz continuous if there exists a bound on how fast the function can change, i.e. its slope
- $\forall x_1, x_2.\ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$
- K is called the Lipshitz constant; the smallest K is the best-Lipshitz constant
- In general, it is defined over two metric spaces (X, d_X) and (Y, d_Y), with $f : X \rightarrow Y$:

$$\exists K.\ d_Y(f(X), f(Y)) \leq K \cdot d_X(X, Y)$$

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Nicer Properties

Lipshitz continuous

- a function is Lipshitz continuous if there exists a bound on how fast the function can change, i.e. its slope
- $\forall x_1, x_2.\ |f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$
- K is called the Lipshitz constant; the smallest K is the best-Lipshitz constant
- In general, it is defined over two metric spaces (X, d_X) and (Y, d_Y), with $f : X \rightarrow Y$:

$$\exists K.\ d_Y(f(X), f(Y)) \leq K \cdot d_X(X, Y)$$

Hölder continuous

Continuously differentiable $\subset$ Lipschitz continuous $\subset$ α-Hölder continuous

Gradients and more

Piecewise-defined function

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each 'piece' of the function being continuous or differentiable in its own sub-domain

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each ‘piece’ of the function being continuous or differentiable in its own sub-domain

Multivariate functions

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each 'piece' of the function being continuous or differentiable in its own sub-domain

Multivariate functions

- Multivalued functions can be seen as a function with a single argument, that is nothing but a *vector* of all its arguments

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each 'piece' of the function being continuous or differentiable in its own sub-domain

Multivariate functions

- Multivalued functions can be seen as a function with a single argument, that is nothing but a *vector* of all its arguments
- $f(a_1, a_2, \ldots, a_n) \equiv f(\vec{a}), a_1 \in D_1, a_2 \in D_2, \ldots, a_n \in D_n, \vec{a} \in D_1 \times D_2 \times \ldots D_n$

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each 'piece' of the function being continuous or differentiable in its own sub-domain

Multivariate functions

- Multivalued functions can be seen as a function with a single argument, that is nothing but a *vector* of all its arguments
- $f(a_1, a_2, \ldots, a_n) \equiv f(\vec{a}), a_1 \in D_1, a_2 \in D_2, \ldots, a_n \in D_n, \vec{a} \in D_1 \times D_2 \times \ldots D_n$
- Gradient: $\nabla_{\vec{a}} f = [\frac{\partial f}{a_1}, \frac{\partial f}{a_2}, \ldots, \frac{\partial f}{a_n}]^T$

Gradients and more

Piecewise-defined function

- defined by multiple subfunctions, each subfunction operating in a sub-domain; these sub-domains together make the domain the the function
- piecewise continuity and differentiability refer to each 'piece' of the function being continuous or differentiable in its own sub-domain

Multivariate functions

- Multivalued functions can be seen as a function with a single argument, that is nothing but a *vector* of all its arguments
- $f(a_1, a_2, \ldots, a_n) \equiv f(\vec{a}), a_1 \in D_1, a_2 \in D_2, \ldots, a_n \in D_n, \vec{a} \in D_1 \times D_2 \times \ldots D_n$
- Gradient: $\nabla_{\vec{a}} f = [\frac{\partial f}{a_1}, \frac{\partial f}{a_2}, \ldots, \frac{\partial f}{a_n}]^T$
- Subgradient: vector of (partial) subderivatives

Vector of functions

Let $\vec{f} = [f_1, f_2, \ldots, f_k]$ be a vector over multiple variables $\vec{x} \equiv (x_1, x_2, \ldots, x_n)$

Jacobian

Vector of functions

Let $\vec{f} = [f_1, f_2, \ldots, f_k]$ be a vector over multiple variables $\vec{x} \equiv (x_1, x_2, \ldots, x_n)$

Jacobian

- Jacobian matrix (J) is a $n \times k$ matrix: $[\nabla_{\vec{x}} f_1, \nabla_{\vec{x}} f_2, \ldots, \nabla_{\vec{x}} f_k]^T$

Vector of functions

Let $\vec{f} = [f_1, f_2, \ldots, f_k]$ be a vector over multiple variables $\vec{x} \equiv (x_1, x_2, \ldots, x_n)$

Jacobian

- Jacobian matrix (J) is a $n \times k$ matrix: $[\nabla_{\vec{x}} f_1, \nabla_{\vec{x}} f_2, \ldots, \nabla_{\vec{x}} f_k]^T$
- $\vec{f}(\vec{x}) + J(\vec{x}) \cdot h$, for small h is the best linear approximator of $\vec{f}$ at $\vec{x}$

Vector of functions

Let $\vec{f} = [f_1, f_2, \ldots, f_k]$ be a vector over multiple variables $\vec{x} \equiv (x_1, x_2, \ldots, x_n)$

Jacobian

- Jacobian matrix (J) is a $n \times k$ matrix: $[\nabla_{\vec{x}} f_1, \nabla_{\vec{x}} f_2, \ldots, \nabla_{\vec{x}} f_k]^T$
- $\vec{f}(\vec{x}) + J(\vec{x}) \cdot h$, for small h is the best linear approximator of $\vec{f}$ at $\vec{x}$
- When $n = k$, the Jacobian determinent exists and provides 'local' information of a function; eg. a function is invertible at a point if the Jacobian determinent is non-zero

Vector of functions

Hessian

Vector of functions

Hessian

- Matrix of second-order derivatives:
  $H(f) = J(\nabla(f))^T, i.e.(H_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$

Vector of functions

Hessian

- Matrix of second-order derivatives:
  $H(f) = J(\nabla(f))^T, i.e. (H_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$
- Hessian matrix of a convex function is positive semi-definite

Vector of functions

Hessian

- Matrix of second-order derivatives:
  $H(f) = J(\nabla(f))^T, i.e.(H_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$
- Hessian matrix of a convex function is positive semi-definite
- If the Hessian is positive-definite (negative-definite) at $\vec{x}$, then f attains an isolated local minimum (maximum) at $\vec{x}$. If the Hessian has both positive and negative eigenvalues, then it is a saddle point. Otherwise the test is inconclusive.

Vector of functions

Hessian

- Matrix of second-order derivatives:
  $H(f) = J(\nabla(f))^T, i.e.(H_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$
- Hessian matrix of a convex function is positive semi-definite
- If the Hessian is positive-definite (negative-definite) at $\vec{x}$, then f attains an isolated local minimum (maximum) at $\vec{x}$. If the Hessian has both positive and negative eigenvalues, then it is a saddle point. Otherwise the test is inconclusive.
- We can define Taylor expansion using Jacobians and Hessians.

Vector of functions

Hessian

- Matrix of second-order derivatives:
  $H(f) = J(\nabla(f))^T, i.e.(H_f)_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$
- Hessian matrix of a convex function is positive semi-definite
- If the Hessian is positive-definite (negative-definite) at $\vec{x}$, then f attains an isolated local minimum (maximum) at $\vec{x}$. If the Hessian has both positive and negative eigenvalues, then it is a saddle point. Otherwise the test is inconclusive.
- We can define Taylor expansion using Jacobians and Hessians.
- If the gradient is computed, the Hessian can be approximated by a linear number of scaler operations

Optimization strategies

- Randomized search

Optimization strategies

- Randomized search
- LP, ILP, MILP

Optimization strategies

- Randomized search
- LP, ILP, MILP
- Gradient based

Optimization strategies

- Randomized search
- LP, ILP, MILP
- Gradient based
- Evolutionary algorithms

MILP

- LP relaxation,

MILP

- LP relaxation,
- branch and bound,

MILP

- LP relaxation,
- branch and bound,
- cutting planes

Metric Space

- definition

Metric Space

- definition
- distances (L_2 norm, L_∞, Jaccard index)

Outline I

Outline II

- Neural networks as Programs
- Attacks on Neural Networks

5. Testing neural networks
   - Visibility and Stage
   - Neural Network Oracles
   - Test Adequacy
   - Algorithms to achieve high NN coverage

6. Conclusion

Computations in Neural Networks

Fully Connected layer

$$\vec{T} = \begin{pmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ & \dots & \\ b_1 & b_2 & \dots \end{pmatrix} \circ ReLU$$

Computations in Neural Networks

Fully Connected layer

$$\vec{T} = \begin{pmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ & \dots & \\ b_1 & b_2 & \dots \end{pmatrix} \circ ReLU$$

Computations in Neural Networks

Fully Connected layer

$$\vec{T} = \begin{pmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ & \dots & \\ b_1 & b_2 & \dots \end{pmatrix} \circ ReLU$$

Multiple layers

$$\text{Multiple Layers: } T_1 \circ T_2 \circ \dots$$

Computations in Neural Networks

Fully Connected layer

$$\vec{T} = \begin{pmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ & \dots & \\ b_1 & b_2 & \dots \end{pmatrix} \circ ReLU$$

Multiple layers

$$\text{Multiple Layers: } T_1 \circ T_2 \circ \dots$$

Computations in Neural Networks

Fully Connected layer

$$\vec{T} = \begin{pmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ & \dots & \\ b_1 & b_2 & \dots \end{pmatrix} \circ ReLU$$

Multiple layers

$$\text{Multiple Layers: } T_1 \circ T_2 \circ \dots$$

The last layer: turning to probabilities

(Softmax)

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Neural networks are programs!

Mappings to programs

Neural networks are programs!

Mappings to programs

- neurons = variables

Neural networks are programs!

Mappings to programs

- neurons = variables
- affine transformations = computations with variables (neurons)

Neural networks are programs!

Mappings to programs

- neurons = variables
- affine transformations = computations with variables (neurons)
- activation functions = conditional branching (or (cond ? a : b))

Neural networks are programs!

Mappings to programs

- neurons = variables
- affine transformations = computations with variables (neurons)
- activation functions = conditional branching (or (cond ? a : b))
- multiple layers = sequence of function calls

Applying PL techniques on NNs

Easy programs: no function calls, no array accesses, no pointers. . .

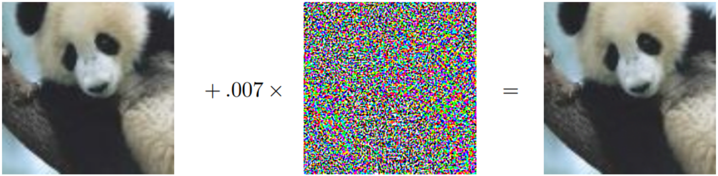
Applying PL techniques on NNs

Easy programs: no function calls, no array accesses, no pointers. . .

Challenge: modeling the large number of branches (ReLUs)—path explosion

The billboard on adversarial examples

The famous picture[2] on adversarial examples:



$$\boldsymbol{x} + .007 \times \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)) = \boldsymbol{x} + \epsilon\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

$\boldsymbol{x}$	$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"panda"	"nematode"	"gibbon"
57.7% confidence	8.2% confidence	99.3 % confidence

[2] Goodfellow et al. Explaining and Harnessing Adversarial Examples, ICLR 2015". (https://arxiv.org/pdf/1412.6572.pdf)

Creating Adversarial Examples

The Fast Sign Method (Goodfellow et al. 2015)[3]

$\vec{x} + \epsilon \; sgn(\Delta_x J(\theta, \vec{x}, y))$

[3] Goodfellow et al. Explaining and Harnessing Adversarial Examples, ICLR 2015.

[4] Carlini and Wagner. Towards Evaluating the Robustness of Neural Networks, 2016.

Creating Adversarial Examples

The Fast Sign Method (Goodfellow et al. 2015)[3]

$\vec{x} + \epsilon \; sgn(\Delta_x J(\theta, \vec{x}, y))$

Carlini-Wagner (Carlini and Wagner. 2015)[4]

minimize $\mathcal{D}(\vec{x} + \epsilon, \vec{x})$ such that $\mathcal{C}(\vec{x} + \epsilon) = t, x + \epsilon \in [0, 1]^n$

[3] Goodfellow et al. Explaining and Harnessing Adversarial Examples, ICLR 2015.

[4] Carlini and Wagner. Towards Evaluating the Robustness of Neural Networks, 2016.

Adversarial Examples: NLP models

$$\mathcal{C}(\vec{x} + \epsilon) \neq \mathcal{C}(\vec{x})$$

such that[5]:

- ϵ is small (within some threshold)

[5] Jin et al. TextFool: Fool your Model with Natural Adversarial Text
[6] hotflip, pruthi, deepwordbug, morpheus
[7] lzantot, bert-attack, faster-alzantot, iga, bae, kuleshov, pso, pwws, textfooler
[8] https://nicholas.carlini.com/code/audio_adversarial_examples/

Adversarial Examples: NLP models

$$\mathcal{C}(\vec{x} + \epsilon) \neq \mathcal{C}(\vec{x})$$

such that[5]:

- ϵ is small (within some threshold)
- adding noise, '$+$', can be syntactic[6] (adding "typos") or semantic[7] (replace words having low cosine similary on word embeddings for synonym extraction)

---

[5] Jin et al. TextFool: Fool your Model with Natural Adversarial Text

[6] hotflip, pruthi, deepwordbug, morpheus

[7] lzantot, bert-attack, faster-alzantot, iga, bae, kuleshov, pso, pwws, textfooler

[8] https://nicholas.carlini.com/code/audio_adversarial_examples/

Adversarial Examples: NLP models

$$\mathcal{C}(\vec{x} + \epsilon) \neq \mathcal{C}(\vec{x})$$

such that[5]:

- ϵ is small (within some threshold)
- adding noise, '$+$', can be syntactic[6] (adding "typos") or semantic[7] (replace words having low cosine similary on word embeddings for synonym extraction)

[5] Jin et al. TextFool: Fool your Model with Natural Adversarial Text
[6] hotflip, pruthi, deepwordbug, morpheus
[7] lzantot, bert-attack, faster-alzantot, iga, bae, kuleshov, pso, pwws, textfooler
[8] https://nicholas.carlini.com/code/audio_adversarial_examples/

Adversarial Examples: NLP models

$$\mathcal{C}(\vec{x} + \epsilon) \neq \mathcal{C}(\vec{x})$$

such that[5]:

- ϵ is small (within some threshold)
- adding noise, '$+$', can be syntactic[6] (adding "typos") or semantic[7] (replace words having low cosine similary on word embeddings for synonym extraction)

Also, audio adversarial examples for TTS engines.[8]

[5] Jin et al. TextFool: Fool your Model with Natural Adversarial Text

[6] hotflip, pruthi, deepwordbug, morpheus

[7] lzantot, bert-attack, faster-alzantot, iga, bae, kuleshov, pso, pwws, textfooler

[8] https://nicholas.carlini.com/code/audio_adversarial_examples/

Adversarial Examples: Code

- Source code classification is getting popular for tasks like comment generation, suggesting method names and code completion;

[9] Zhang et al. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models Authors. AAAI'2020; Yefet et al. Adversarial Examples for Models of Code. OOPSLA'20; Zhou et al. Adversarial Robustness of Deep Code Comment Generation, ArXiv 2021, Bielik and Vechev. Adversarial Robustness for Code. ICML'20

Adversarial Examples: Code

- Source code classification is getting popular for tasks like comment generation, suggesting method names and code completion;
- Attacks on code models attempt to create semantically equivalent code snippets that make the classifiers behave differently[9]

[9] Zhang et al. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models Authors. AAAI'2020; Yefet et al. Adversarial Examples for Models of Code. OOPSLA'20; Zhou et al. Adversarial Robustness of Deep Code Comment Generation, ArXiv 2021, Bielik and Vechev. Adversarial Robustness for Code. ICML'20

Formalizing Robustness

(Local) Robustness

A "small" perturbation does not change the outcome significantly

$$\forall x \in I.\ \forall e \in E.\ f(x) \simeq f(x + e)$$

Formalizing Robustness

(Local) Robustness

A "small" perturbation does not change the outcome significantly

$$\forall x \in I.\ \forall e \in E.\ f(x) \simeq f(x+e)$$

Formalizing Robustness

(Local) Robustness

A "small" perturbation does not change the outcome significantly

$$\forall x \in I.\ \forall e \in E.\ f(x) \simeq f(x+e)$$

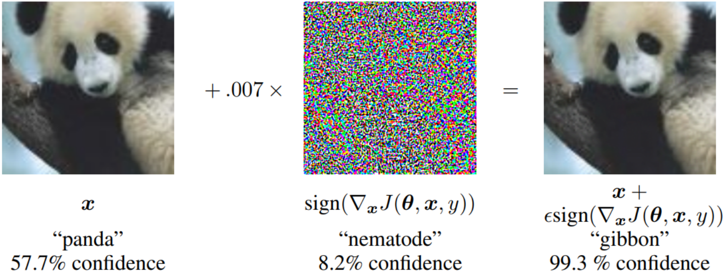


Image Credit: Goodfellow et al. Explaining and Harnessing Adversarial Examples. 2013.

Data Poisoning Attacks

Use of croudsourced data-sets or active/online learning (like spam filters) opens up an opportunity to affect the learned model via a few specially crafted data-points

[10] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In Asian Conference on Machine Learning, 2011.

[11] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020

Data Poisoning Attacks

Use of croudsourced data-sets or active/online learning (like spam filters) opens up an opportunity to affect the learned model via a few specially crafted data-points

Attacks on availability

[10] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In Asian Conference on Machine Learning, 2011.

[11] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020

Data Poisoning Attacks

Use of croudsourced data-sets or active/online learning (like spam filters) opens up an opportunity to affect the learned model via a few specially crafted data-points

Attacks on availability

- Add a few mislabelled data points that change the decision boundary.

[10] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In Asian Conference on Machine Learning, 2011.

[11] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020

Data Poisoning Attacks

Use of croudsourced data-sets or active/online learning (like spam filters) opens up an opportunity to affect the learned model via a few specially crafted data-points

Attacks on availability

- Add a few mislabelled data points that change the decision boundary.
- Easier to perform on SVMs, as even a single data instance can disturb the maximum margin hyperplane[10].

[10] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In Asian Conference on Machine Learning, 2011.

[11] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020

Data Poisoning Attacks

Use of croudsourced data-sets or active/online learning (like spam filters) opens up an opportunity to affect the learned model via a few specially crafted data-points

Attacks on availability

- Add a few mislabelled data points that change the decision boundary.
- Easier to perform on SVMs, as even a single data instance can disturb the maximum margin hyperplane[10].
- DNNs seem robust to this attack as their decision boundaries are more complex.[11]

[10] B. Biggio, B. Nelson, and P. Laskov. Support vector machines under adversarial label noise. In Asian Conference on Machine Learning, 2011.

[11] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020

Data Poisoning Attacks (cont.)

Create Backdoors

Insert the backdoor pattern *within a few examples from the legitimate domain* labelled to the backdoor class[12]

[12] Miller et al. Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks. Proceedings of IEEE'2020.

Attacks on data privacy

Different types of attacks possible[13]:

- Membership inference attacks (eg. if person p was in training set)

[13] Rigaki and Garcia, A Survey of Privacy Attacks in Machine Learning, ArXiv 2021

Attacks on data privacy

Different types of attacks possible[13]:

- Membership inference attacks (eg. if person p was in training set)
- Reconstruction attacks (eg. recreate the training dataset)

[13] Rigaki and Garcia, A Survey of Privacy Attacks in Machine Learning, ArXiv 2021

Attacks on data privacy

Different types of attacks possible[13]:

- Membership inference attacks (eg. if person p was in training set)
- Reconstruction attacks (eg. recreate the training dataset)
- Property inference attacks (eg. the gender ratio of the dataset)

[13] Rigaki and Garcia, A Survey of Privacy Attacks in Machine Learning, ArXiv 2021

Attacks on data privacy

Different types of attacks possible[13]:

- Membership inference attacks (eg. if person p was in training set)
- Reconstruction attacks (eg. recreate the training dataset)
- Property inference attacks (eg. the gender ratio of the dataset)
- Model extraction (or "stealing") attacks (eg. train a new classifier)

[13] Rigaki and Garcia, A Survey of Privacy Attacks in Machine Learning, ArXiv 2021

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)
- Learn adversarial examples on the bootstrap model

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)
- Learn adversarial examples on the bootstrap model
- Query the victim model on these examples

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)
- Learn adversarial examples on the bootstrap model
- Query the victim model on these examples
- Retrain the bootstrap model on these query responses

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)
- Learn adversarial examples on the bootstrap model
- Query the victim model on these examples
- Retrain the bootstrap model on these query responses

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Reverse Engineering or "Models stealing" attacks

Attempt to "steal" cloud hosted models (MLaaS)

Cloudleak[14] uses active learning:

- Train a "bootstrap" model (say, use a pre-trained model)
- Learn adversarial examples on the bootstrap model
- Query the victim model on these examples
- Retrain the bootstrap model on these query responses

Significant reduction in the number of queries

[14] Yu et al. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. NDSS'22

Is the model even safe to use?

Safety of drones and robots

[15] Rudy Bunel et al. Piecewise linear neural networks verification: A comparative study. In NeurIPS 2018

[16] Rüdiger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In arXiv 2017.

[17] Julian et al. Policy compression for aircraft collision avoidance systems. In DASC 2016.

Is the model even safe to use?

Safety of drones and robots

- TwinStream[15] (Property: output is positive)

[15] Rudy Bunel et al. Piecewise linear neural networks verification: A comparative study. In NeurIPS 2018

[16] Rüdiger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In arXiv 2017.

[17] Julian et al. Policy compression for aircraft collision avoidance systems. In DASC 2016.

Is the model even safe to use?

Safety of drones and robots

- TwinStream[15] (Property: output is positive)
- CollisionAvoidance[16] (Property: Check if vehicles will collide)

[15] Rudy Bunel et al. Piecewise linear neural networks verification: A comparative study. In NeurIPS 2018

[16] Rüdiger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In arXiv 2017.

[17] Julian et al. Policy compression for aircraft collision avoidance systems. In DASC 2016.

Is the model even safe to use?

Safety of drones and robots

- TwinStream[15] (Property: output is positive)
- CollisionAvoidance[16] (Property: Check if vehicles will collide)
- ACAS Xu[17] (Property: generate advisories such that aircrafts don't collide)

[15] Rudy Bunel et al. Piecewise linear neural networks verification: A comparative study. In NeurIPS 2018

[16] Rüdiger Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In arXiv 2017.

[17] Julian et al. Policy compression for aircraft collision avoidance systems. In DASC 2016.

Outline I

Outline II

- Neural networks as Programs
- Attacks on Neural Networks

5. Testing neural networks
   - Visibility and Stage
   - Neural Network Oracles
   - Test Adequacy
   - Algorithms to achieve high NN coverage

6. Conclusion

Recall

- Why test neural networks?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? Test Oracle
  - When to stop testing?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible? **System Visibility**
  - What SDLC stage are we in?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible? **System Visibility**
  - What SDLC stage are we in?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible? **System Visibility**
  - What SDLC stage are we in? **Testing Stage**
  - How to test?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible? **System Visibility**
  - What SDLC stage are we in? **Testing Stage**
  - How to test?

Recall

- Why test neural networks?
  - NNs are (conventionally) trained *only* to achieve high accuracy,
  - We often want our network to achieve other properties: security, fairness etc.
- Testing ingredients
  - How to identify a bug? **Test Oracle**
  - When to stop testing? **Test Adequacy**
  - How much of the system is visible? **System Visibility**
  - What SDLC stage are we in? **Testing Stage**
  - How to test? **Testing Algorithm**

Visibility

- In most cases, we will assume a *white-box* setting, or at least a *graybox* setting;

Visibility

- In most cases, we will assume a *white-box* setting, or at least a *graybox* setting;
- Sometimes NNs need a *blackbox* setting, eg. in MLaaS systems

Visibility

- In most cases, we will assume a *white-box* setting, or at least a *graybox* setting;
- Sometimes NNs need a *blackbox* setting, eg. in MLaaS systems
  - Prominent strategy in such cases is analysis using local surrogate models trained using active learning[CloudLeak]

Visibility

- In most cases, we will assume a *white-box* setting, or at least a *graybox* setting;
- Sometimes NNs need a *blackbox* setting, eg. in MLaaS systems
  - Prominent strategy in such cases is analysis using local surrogate models trained using active learning[CloudLeak]

Visibility

- In most cases, we will assume a *white-box* setting, or at least a *graybox* setting;
- Sometimes NNs need a *blackbox* setting, eg. in MLaaS systems
  - Prominent strategy in such cases is analysis using local surrogate models trained using active learning[CloudLeak]

We will consider *unit testing* stage.

Challenges to creating Oracles

Challenges in formulating preconditions

Challenges to creating Oracles

Challenges in formulating preconditions

- The input distribution is not well-known; is only specified by a set of examples

Challenges to creating Oracles

Challenges in formulating preconditions

- The input distribution is not well-known; is only specified by a set of examples
- The *ground-truths* for arbitrary instances are not known

Challenges to creating Oracles

Challenges in formulating preconditions

- The input distribution is not well-known; is only specified by a set of examples
- The *ground-truths* for arbitrary instances are not known

- Properties: deterministic, probabilistic

Challenges to creating Oracles

Challenges in formulating preconditions

- The input distribution is not well-known; is only specified by a set of examples
- The *ground-truths* for arbitrary instances are not known

- Properties: deterministic, probabilistic
- Domain-specific constraints on inputs (g. occlusion, blackout)

Solution #1: Using a "closeness" assumption

Given a neural network $\mathcal{N} : \mathcal{F} \rightarrow L$, classifying features $x \in \mathcal{F}$ to labels $l \in \mathcal{L}$, an annotated example set $\mathcal{E}$, any data-point x that is "close" to a given example, $e \in \mathcal{E}$ is valid and has the same label as e i.e.,

$$Valid(\langle x, l \rangle) = \{\langle x, l \rangle \mid ||x - e|| < \epsilon \wedge l = \mathcal{N}(e), e \in \mathcal{E}, x \in \mathcal{F}\}$$

Solution #1: Using a "closeness" assumption

Given a neural network $\mathcal{N} : \mathcal{F} \rightarrow L$, classifying features $x \in \mathcal{F}$ to labels $l \in \mathcal{L}$, an annotated example set $\mathcal{E}$, any data-point x that is "close" to a given example, $e \in \mathcal{E}$ is valid and has the same label as e i.e.,

$$Valid(\langle x, l \rangle) = \{\langle x, l \rangle \mid ||x - e|| < \epsilon \wedge l = \mathcal{N}(e), e \in \mathcal{E}, x \in \mathcal{F}\}$$

Closeness Assumption

Solution #1: Using a "closeness" assumption

Given a neural network $\mathcal{N} : \mathcal{F} \rightarrow L$, classifying features $x \in \mathcal{F}$ to labels $l \in \mathcal{L}$, an annotated example set $\mathcal{E}$, any data-point x that is "close" to a given example, $e \in \mathcal{E}$ is valid and has the same label as e i.e.,

$$Valid(\langle x, l \rangle) = \{\langle x, l \rangle \mid ||x - e|| < \epsilon \wedge l = \mathcal{N}(e), e \in \mathcal{E}, x \in \mathcal{F}\}$$

Closeness Assumption

- DeepConcolic divides the input-space into (overlapping) subspaces, $S(D, \epsilon)$ where D is a distance metric and ϵ is a given threshold such that if $||x_1 - x_2|| \leq \epsilon$ then there exists a valid subspace $X \in S(D, \epsilon)$ s.t. $x_1, x_2 \in X$. All analysis is only restricted to such valid subspaces $\{X \mid X \in S(D, \epsilon)\}$;

Solution #1: Using a "closeness" assumption

Given a neural network $\mathcal{N} : \mathcal{F} \rightarrow L$, classifying features $x \in \mathcal{F}$ to labels $l \in \mathcal{L}$, an annotated example set $\mathcal{E}$, any data-point x that is "close" to a given example, $e \in \mathcal{E}$ is valid and has the same label as e i.e.,

$$Valid(\langle x, l \rangle) = \{\langle x, l \rangle \mid ||x - e|| < \epsilon \wedge l = \mathcal{N}(e), e \in \mathcal{E}, x \in \mathcal{F}\}$$

Closeness Assumption

- DeepConcolic divides the input-space into (overlapping) subspaces, $S(D, \epsilon)$ where D is a distance metric and ϵ is a given threshold such that if $||x_1 - x_2|| \leq \epsilon$ then there exists a valid subspace $X \in S(D, \epsilon)$ s.t. $x_1, x_2 \in X$. All analysis is only restricted to such valid subspaces $\{X \mid X \in S(D, \epsilon)\}$;
- DeepSafe uses k-means to cluster the input-space into valid-clusters and attempts to find robustness regions

Solution #2: Using domain-specific mutations

Well-designed mutations can be used such that muatations on the provided examples, $e \in \mathcal{E}$, are likely to maintain the validity of the input with the same label as $\mathcal{N}(e)$.

Solution #2: Using domain-specific mutations

Well-designed mutations can be used such that muatations on the provided examples, $e \in \mathcal{E}$, are likely to maintain the validity of the input with the same label as $\mathcal{N}(e)$.

Use of mutations

Solution #2: Using domain-specific mutations

Well-designed mutations can be used such that muatations on the provided examples, $e \in \mathcal{E}$, are likely to maintain the validity of the input with the same label as $\mathcal{N}(e)$.

Use of mutations

- DeepXplore uses domain-specific mutations on the provided input images like lighting modifications (simulating different times of the day), introducing occlusion (cars blocking view), a small-fraction of pixels blackened (effect of dirt on camera);

Solution #2: Using domain-specific mutations

Well-designed mutations can be used such that muatations on the provided examples, $e \in \mathcal{E}$, are likely to maintain the validity of the input with the same label as $\mathcal{N}(e)$.

Use of mutations

- DeepXplore uses domain-specific mutations on the provided input images like lighting modifications (simulating different times of the day), introducing occlusion (cars blocking view), a small-fraction of pixels blackened (effect of dirt on camera);
- Deep test uses nine different realistic image transformations (changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog effect, and rain effect)—comprising linear, affine, and convolutional transformations.

Solution #3: Using generative models

Generative models are capable of learning a target distribution, and have been used to synthesize new points in the input-space:

Solution #3: Using generative models

Generative models are capable of learning a target distribution, and have been used to synthesize new points in the input-space:

Use of Generative models

Solution #3: Using generative models

Generative models are capable of learning a target distribution, and have been used to synthesize new points in the input-space:

Use of Generative models

- Dola et al. use variational autoencoders (VAE) to learn the input-distribution;

Solution #3: Using generative models

Generative models are capable of learning a target distribution, and have been used to synthesize new points in the input-space:

Use of Generative models

- Dola et al. use variational autoencoders (VAE) to learn the input-distribution;
- DeepRoad uses Generative Advarsarial Networks (GAN) to generative realistic road scenes to test self-driving controller DNNs.

Solution #4: Using ensembles

Assumes that networks *mostly* work well and ensembles are effective

Use of ensembles

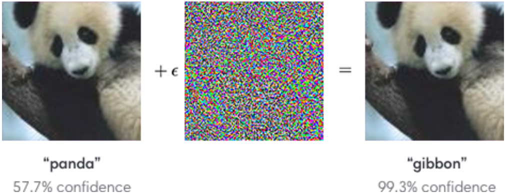
Solution #4: Using ensembles

Assumes that networks *mostly* work well and ensembles are effective

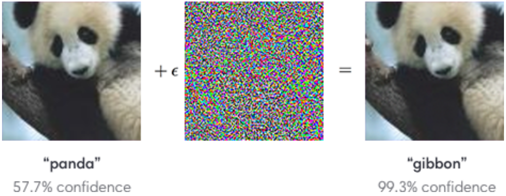
Use of ensembles

- DeepXplore uses *majority voting* on an ensemble of networks to establish ground-truth

Properties



Properties

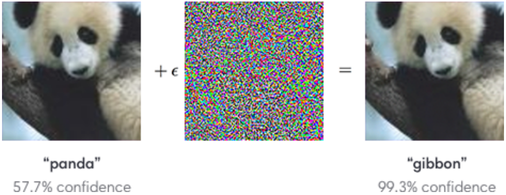


(Local) Robustness

"small" perturbations don't alter outcome significantly at x_0

$$\forall \epsilon \in \text{Noise}.\ \mathcal{N}(x_0) \simeq \mathcal{N}(x_0 + \epsilon)$$

Properties

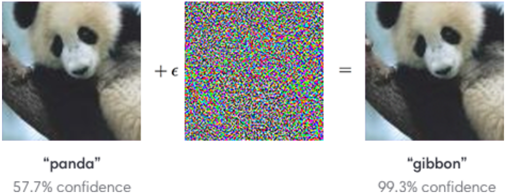


(Local) Robustness

"small" perturbations don't alter outcome significantly at x_0

$$\forall \epsilon \in \text{Noise}.\ \mathcal{N}(x_0) \simeq \mathcal{N}(x_0 + \epsilon)$$

Properties



(Local) Robustness

"small" perturbations don't alter outcome significantly at x_0

$$\forall \epsilon \in \text{Noise}.\ \mathcal{N}(x_0) \simeq \mathcal{N}(x_0 + \epsilon)$$

(Global) Robustness

"small" perturbations don't alter outcome significantly *anywhere*

$$\forall x \in X.\ \forall \epsilon \in \text{Noise}.\ \mathcal{N}(x) \simeq \mathcal{N}(x + \epsilon)$$

Image Credit: Goodfellow et al. 2013.

Properties

Independence-based Fairness (CARE)

$$\forall l \in \mathcal{L}.\ P(Y = l|\mathcal{F}(s)) - P(Y = l|\mathcal{F} \neq s) \leq \epsilon$$

$s \in \mathcal{F}$ is a sensitive attribute.

Properties

Independence-based Fairness (CARE)

$$\forall l \in \mathcal{L}.\ P(Y = l | \mathcal{F}(s)) - P(Y = l | \mathcal{F} \neq s) \leq \epsilon$$

$s \in \mathcal{F}$ is a sensitive attribute.

Backdoor attack success rate (CARE)

$$SR(t) = P(\mathcal{N}(x) = t \mid x \in Z, Z \subseteq X) \leq \epsilon$$

$t \in X$ is the target label, $Z \subseteq X$ is the set of adversarial inputs.

Properties

Independence-based Fairness (CARE)

$$\forall l \in \mathcal{L}.\ P(Y = l|\mathcal{F}(s)) - P(Y = l|\mathcal{F} \neq s) \leq \epsilon$$

$s \in \mathcal{F}$ is a sensitive attribute.

Backdoor attack success rate (CARE)

$$SR(t) = P(\mathcal{N}(x) = t \mid x \in Z, Z \subseteq X) \leq \epsilon$$

$t \in X$ is the target label, $Z \subseteq X$ is the set of adversarial inputs.

Safety property violation rate (CARE)

$$VR(\rho) = P(\mathcal{N} \not\models \rho \mid x \in X) \leq \epsilon$$

ρ is a critical safety property.

Neural Network Coverage Metrics

- Lipshitz Continuity (LC): all neurons are within a provided Lipshitz constant bound (DeepConcolic)

Neural Network Coverage Metrics

- Lipshitz Continuity (LC): all neurons are within a provided Lipshitz constant bound (DeepConcolic)
- Neuron Coverage (NC): all neurons that have been activated in some test; similar to statement coverage (DeepConcolic)

Neural Network Coverage Metrics

- Lipshitz Continuity (LC): all neurons are within a provided Lipshitz constant bound (DeepConcolic)
- Neuron Coverage (NC): all neurons that have been activated in some test; similar to statement coverage (DeepConcolic)
- Sign Sign Coverage (SSC): have we seen all possible activation/deactivation combinations of all pairs of neurons in adjacent layers; similar to MC/DC (DeepConcolic)

Neural Network Coverage Metrics

- Lipshitz Continuity (LC): all neurons are within a provided Lipshitz constant bound (DeepConcolic)
- Neuron Coverage (NC): all neurons that have been activated in some test; similar to statement coverage (DeepConcolic)
- Sign Sign Coverage (SSC): have we seen all possible activation/deactivation combinations of all pairs of neurons in adjacent layers; similar to MC/DC (DeepConcolic)
- Neuron Boundary Coverage (NBC): all neuron activation values that *exceed* a given bound (DeepConcolic)

Neural Network Coverage Metrics

- Lipshitz Continuity (LC): all neurons are within a provided Lipshitz constant bound (DeepConcolic)
- Neuron Coverage (NC): all neurons that have been activated in some test; similar to statement coverage (DeepConcolic)
- Sign Sign Coverage (SSC): have we seen all possible activation/deactivation combinations of all pairs of neurons in adjacent layers; similar to MC/DC (DeepConcolic)
- Neuron Boundary Coverage (NBC): all neuron activation values that *exceed* a given bound (DeepConcolic)
- Neuron Contribution Coverage (NCC): contribution of each edge on the value of a neuron activation value (DeepCon),

$$u^i_{h,j}(x) = w^i_{h,j} \cdot f_{i-1,h}(x) \qquad U^i_j(x) = \{u^i_{h,j}(x) \mid 0 \leq h \leq s_{i-i}\}$$

$$nu^i_{h,j} = \frac{u^i_{h,j}(x) - min(U^i_j(x))}{max(U^i_j(x)) - min(U^i_j(x))}$$

Algorithm Summary

- Greedy search (DeepTest),

Algorithm Summary

- Greedy search (DeepTest),
- Gradient Descent (DeepGauge, DiffAI, DSE, DeepXplore, DeepCon),

Algorithm Summary

- Greedy search (DeepTest),
- Gradient Descent (DeepGauge, DiffAI, DSE, DeepXplore, DeepCon),
- Concolic Execution (DeepConcolic, DeepCover, NeuroSPF, DeepCheck),

Algorithm Summary

- Greedy search (DeepTest),
- Gradient Descent (DeepGauge, DiffAI, DSE, DeepXplore, DeepCon),
- Concolic Execution (DeepConcolic, DeepCover, NeuroSPF, DeepCheck),
- Coverage-guided Graybox Fuzzing (DLFuzz, TensorFuzz)

Compile to program (DeepCheck)

As already discussed, a neural network can be compiled to a program:

DeepCheck

Compile to program (DeepCheck)

As already discussed, a neural network can be compiled to a program:

DeepCheck

- The i^{th} output neuron:

$$y_i = C_{i,0} \cdot x_0 + C_{i,1} \cdot x_1 + \cdots + C_{i,n-1} \cdot x_{n-1} + B_i$$

Compile to program (DeepCheck)

As already discussed, a neural network can be compiled to a program:

DeepCheck

- The i^{th} output neuron:

$$y_i = C_{i,0} \cdot x_0 + C_{i,1} \cdot x_1 + \cdots + C_{i,n-1} \cdot x_{n-1} + B_i$$

- Using concolic execution, this coefficient can be assembled as:

$$C_{i,j} = \sum_{p \in paths(i,j)} \Big(\prod_{e \in edges(p)} w(e) \Big)$$

Compile to program (DeepCheck)

DeepCheck

Compile to program (DeepCheck)

DeepCheck

- Use the value of the coefficients to identify the most important input (assign *importance scores*) to be mutated (akin to gradient-based approaches)

Compile to program (DeepCheck)

DeepCheck

- Use the value of the coefficients to identify the most important input (assign *importance scores*) to be mutated (akin to gradient-based approaches)
- Attack generation: output of j^{th} neuron, $f_j(X) = B_j + \sum_{i=1}^{t} C_{j,i} \cdot X_i$, so attack constraint is:

$$\exists X. \ \wedge_{j=1, j \neq l'} f_j(X) < f_{l'}(X) \wedge PathCond$$

where $PC = \wedge_{j=1}^{A}(B_j + \sum_{i=1}^{t} C_{j,i} \cdot X_i \{\leq, >\} 0)$, A is the number of activation functions; the symbolic constraints can be limited to the important neurons for efficiency

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

Activation (Pattern)

- ap[t](i)= $W_i \cdot X[t] + B_i\{\leq, >\}0$

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)= $W_i \cdot X[t] + B_i\{\leq, >\}0$ **Activation of a neuron**

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t

- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ **Activation of a neuron**

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t
- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ **Activation of a neuron**
  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t
- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ **Activation condition (AC)**
- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)= $W_i \cdot X[t] + B_i\{\leq, >\}0$ **Activation of a neuron**
  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t
- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ **Activation condition (AC)**
- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ Activation of a neuron

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t

- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ Activation condition (AC)

- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron Activation Tree

  Keeps track of which activations have been seen.

- DC: $\wedge_{i \neq k} o_i > o_k$, for desired class i

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ **Activation of a neuron**

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t

- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ **Activation condition (AC)**

- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron **Activation Tree**

  Keeps track of which activations have been seen.

- DC: $\wedge_{i \neq k} o_i > o_k$, for desired class i

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ Activation of a neuron

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t

- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ Activation condition (AC)

- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron Activation Tree

  Keeps track of which activations have been seen.

- DC: $\wedge_{i \neq k} o_i > o_k$, for desired class i Decision Condition (DC)

- Same AC with different DCs search all decisions with same activation

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t

  Activation (Pattern)

- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ — Activation of a neuron

  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t

- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ — Activation condition (AC)

- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron — Activation Tree

  Keeps track of which activations have been seen.

- DC: $\wedge_{i \neq k} o_i > o_k$, for desired class i — Decision Condition (DC)

- Same AC with different DCs search all decisions with same activation

Concolic Execution (DeepConcolic, SpaceScanner)

Activation Tree

- *activation pattern (ap)* of a test t, $ap[t]$, is a bitvector recording the activation state of all neurons; $ap[t](i)$ (or $ap[t]_{k_1,k_2}$) is the activation neuron for the i^{th} neuron (or k_2 neuron on k_1 layer) for test t
  Activation (Pattern)
- ap[t](i)$= W_i \cdot X[t] + B_i\{\leq, >\}0$ Activation of a neuron
  W_i: incoming edge-wts. of neuron i, $X[t]$: input vector for test t
- AC: $\wedge_{i=0}^{|\mathcal{N}|} NeuronActivate(i)$ Activation condition (AC)
- NN unfolded as a tree, each neuron is a prefix of the activation bitvector till that neuron Activation Tree
  Keeps track of which activations have been seen.
- DC: $\wedge_{i \neq k} o_i > o_k$, for desired class i Decision Condition (DC)
- Same AC with different DCs search all decisions with same activation
  Amplification

Coverage guided concolic execution

DeepConcolic

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.
  - $ap'[t]_{k,i} = \neg ap[t]_{k,i} \wedge \underset{k_1 < k}{\forall} \bigwedge_{0 \leq i_q \leq s_{k_1}} ap'[t]_{k_1,i_i} = ap[t]_{k_1,i_1}$

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.
  - $ap'[t]_{k,i} = \neg ap[t]_{k,i} \wedge \underset{k_1<k}{\forall} \underset{0 \leq i_q \leq s_{k_1}}{\bigwedge} ap'[t]_{k_1,i_i} = ap[t]_{k_1,i_1}$

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.
  - $ap'[t]_{k,i} = \neg ap[t]_{k,i} \wedge \underset{k_1<k}{\forall} \underset{0 \leq i_q \leq s_{k_1}}{\bigwedge} ap'[t]_{k_1,i_i} = ap[t]_{k_1,i_1}$ Neuron Cov.
  - $\{\exists x_1, x_2. (||o[x_1] - o[x_2]|| - c \cdot ||x_1 - x_2|| > 0) \wedge x_1, x_2 \in X \mid X \in S(D, b)\}$
    $X \in S(D, b)$ is any of the valid regions, o(x) is the output layer value corresponding to input x

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.
  - $ap'[t]_{k,i} = \neg ap[t]_{k,i} \wedge \underset{k_1<k}{\forall} \underset{0 \leq i_q \leq s_{k_1}}{\bigwedge} ap'[t]_{k_1,i_i} = ap[t]_{k_1,i_1}$ Neuron Cov.
  - $\{\exists x_1, x_2. \, (||o[x_1] - o[x_2]|| - c \cdot ||x_1 - x_2|| > 0) \wedge x_1, x_2 \in X \mid X \in S(D, b)\}$ $X \in S(D, b)$ is any of the valid regions, o(x) is the output layer value corresponding to input x

Coverage guided concolic execution

DeepConcolic

- Classic concolic execution: start with seed tests (examples) $\mathcal{E}$, and maximize coverage metrics (NC, SSC, NBC, Lipshitz Test) to generate new tests, eg.
  - $ap'[t]_{k,i} = \neg ap[t]_{k,i} \wedge \underset{k_1<k}{\forall} \underset{0\leq i_q \leq s_{k_1}}{\bigwedge} ap'[t]_{k_1,i_i} = ap[t]_{k_1,i_1}$ **Neuron Cov.**
  - $\{\exists x_1, x_2. \; (||o[x_1] - o[x_2]|| - c \cdot ||x_1 - x_2|| > 0) \wedge x_1, x_2 \in X \mid X \in S(D, b)\}$
    $X \in S(D, b)$ is any of the valid regions, o(x) is the output layer value corresponding to input x **Lipshitz coverage**
- Given coverage requirememts, $\mathcal{R}$, return test with highest score:

$$t = \underset{r}{argmax}\{val(t) \mid r \in \mathcal{R}\}$$

Probabilistic Symbolic Execution (SpaceScanner)

SpaceScanner

Probabilistic Symbolic Execution (SpaceScanner)

SpaceScanner

- Probability of a Decision (by volume computation):

Probabilistic Symbolic Execution (SpaceScanner)

SpaceScanner

- Probability of a Decision (by volume computation):

Probabilistic Symbolic Execution (SpaceScanner)

SpaceScanner

- Probability of a Decision (by volume computation):

Decision Probability

$$Pr(\mathcal{D}) = \sum_{\varphi \in AC \wedge DC} \int_x \mathbb{1}_{x \models \varphi} \cdot p(x) dx \tag{1}$$

$$= \sum_{\varphi \in AC \wedge DC} \frac{Vol(\phi \wedge s_i)}{s_i} \cdot \sum_x p(s_i) \tag{2}$$

Assuming that the input distribution is discretized into a histogram, $H : s_i \mapsto Pr(s_i)$ p(x) is the input distribution

Gradient-guided Optimization

DeepXplore

$$(\sum_{i \neq j} \mathcal{N}_i(x)[c] - \lambda_1 \cdot \mathcal{N}_j(x)[c]) + \lambda_2 \cdot f_n(x)$$

Gradient-guided Optimization

DeepXplore

$$(\sum_{i \neq j} \mathcal{N}_i(x)[c] - \lambda_1 \cdot \mathcal{N}_j(x)[c]) + \lambda_2 \cdot f_n(x)$$

- It attempts to maximize (1) differential behavior amonst an ensemble of networks (first term), (2) neuron coverage (second term)

Gradient-guided Optimization

DeepXplore

$$(\sum_{i \neq j} \mathcal{N}_i(x)[c] - \lambda_1 \cdot \mathcal{N}_j(x)[c]) + \lambda_2 \cdot f_n(x)$$

- It attempts to maximize (1) differential behavior amonst an ensemble of networks (first term), (2) neuron coverage (second term)
- A network $\mathcal{N}_j$ is selected at random, and its differential behavior from others is maximized to draw out faulty behaviors

Gradient-guided Optimization

DeepXplore

$$(\sum_{i \neq j} \mathcal{N}_i(x)[c] - \lambda_1 \cdot \mathcal{N}_j(x)[c]) + \lambda_2 \cdot f_n(x)$$

- It attempts to maximize (1) differential behavior amonst an ensemble of networks (first term), (2) neuron coverage (second term)
- A network $\mathcal{N}_j$ is selected at random, and its differential behavior from others is maximized to draw out faulty behaviors
- A deactivated neuron f_n is selected at random, and we attempt to activate it

Data-driven + Symbolic solvers

Solvers like Reluplex can effectively solve *local robustness* at x_0:

$$\exists_{x,x_0}.\ Symbolic[N](x) - \mathcal{N}(x_0) > \epsilon \wedge ||x - x_0|| \leq \delta$$

(note: $\mathcal{N}(x_0)$ is a constant)

Data-driven + Symbolic solvers

Solvers like Reluplex can effectively solve *local robustness* at x_0:

$$\exists_{x,x_0}.\ Symbolic[N](x) - \mathcal{N}(x_0) > \epsilon \wedge ||x - x_0|| \leq \delta$$

(note: $\mathcal{N}(x_0)$ is a constant)

they can be used to solve global robustness by:

$$\exists_{x_1,x_2}.\ Symbolic[N](x_1) - Symbolic(\mathcal{N}(x_2) > \epsilon \wedge ||x_1 - x_2|| \leq \delta$$

However, it is not efficient:

- Symbolically encoding two copies of the network does not scale

Data-driven + Symbolic solvers

Solvers like Reluplex can effectively solve *local robustness* at x_0:

$$\exists_{x,x_0}.\ Symbolic[N](x) - \mathcal{N}(x_0) > \epsilon \wedge ||x - x_0|| \leq \delta$$

(note: $\mathcal{N}(x_0)$ is a constant)

they can be used to solve global robustness by:

$$\exists_{x_1,x_2}.\ Symbolic[N](x_1) - Symbolic(\mathcal{N}(x_2) > \epsilon \wedge ||x_1 - x_2|| \leq \delta$$

However, it is not efficient:

- Symbolically encoding two copies of the network does not scale
- Reluplex like solvers work best when the check is restricted to small neighborhoods

Data-driven + Symbolic solvers

DeepSafe

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);
- Use the local robustness check with Reluplex on each region;

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);
- Use the local robustness check with Reluplex on each region;
- for each region r:

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);
- Use the local robustness check with Reluplex on each region;
- for each region r:
  - if no adversarial example is found, r is robust

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);
- Use the local robustness check with Reluplex on each region;
- for each region r:
  - if no adversarial example is found, r is robust
  - if an adversarial example x is found, r is shrunk to eliminate x

Data-driven + Symbolic solvers

DeepSafe

- Cluster the examples $\mathcal{E}$ via k-means: each cluster identifies a *region* characterized by a centroid (x_0), radius (r) and label (l);
- Use the local robustness check with Reluplex on each region;
- for each region r:
  - if no adversarial example is found, r is robust
  - if an adversarial example x is found, r is shrunk to eliminate x
  - if a region size reduces to zero, $\mathcal{N}$ is non-robust

Metamorphic Testing

Metamorphic Testing (HOMRS)

Outline I

Outline II

Conclusion

Application and impact of these techniques, other works on verification, repair, interpretation, explanation,