Testing Deep Neural Networks

Xiaowei Huang, University of Liverpool
Outline

Safety Problem of AI

Verification (brief)

Testing

Conclusions and Future Works
Human-Level Intelligence
Robotics and Autonomous Systems
Deep neural networks

all implemented with
Figure: safety in image classification networks
AI Can Be Fooled With One Misspelled Word

When artificial intelligence is dumb.

Figure: safety in natural language processing networks
Security

Drowning Dalek commands Siri in voice-rec hack attack

Boffins embed barely-audible-to-humans commands inside vids to fool virtual assistants

By Darren Pauli 11 Jul 2016 at 07:48

Figure: safety in voice recognition networks
AI vs AI: New algorithm automatically bypasses your best cybersecurity defenses

Researchers have created an AI that tweaks malware code, and it easily bypassed an anti-malware AI undetected. Is machine learning ready to face down cybersecurity threats?

By Brandon Vigiarolo | August 2, 2017, 12:25 PM PST

Figure: safety in security systems
Safety Definition: Human Driving vs. Autonomous Driving

Traffic image from “The German Traffic Sign Recognition Benchmark”
Safety Definition: Human Driving vs. Autonomous Driving

Image generated from our tool
Safety Problem: Incidents

Woman dead after being struck by self-driving Uber
Safety Definition: Illustration

- go right
- or straight
- go left
- or straight
- Stop

...
Safety Requirements

- Pointwise Robustness (this talk)
  - if the decision of a pair (input, network) is invariant with respect to the perturbation to the input.
- Network Robustness
- or more fundamentally, Lipschitz continuity, mutual information, etc
- model interpretability
Certification of DNN

https://github.com/TrustAI
Outline

Safety Problem of AI

Verification (brief)

Testing

Conclusions and Future Works
Safety Definition: Traffic Sign Example
Maximum Safe Radius

Definition
The maximum safe radius problem is to compute the minimum distance from the original input $\alpha$ to an adversarial example, i.e.,

$$\text{MSR}(\alpha) = \min_{\alpha' \in \mathcal{D}} \{ \| \alpha - \alpha' \|_k \mid \alpha' \text{ is an adversarial example} \} \quad (1)$$
Existing Approaches

- layer-by-layer exhaustive search, see e.g., [2]¹
- SMT, MILP, SAT based constraint solving, see e.g., [3]²
- global optimisation, see e.g., [6]³
- abstract interpretation, see e.g., [1]⁴

¹ Huang, Kwiatkowska, Wang, Wu, CAV2017
² Katz, Barrett, Dill, Julian, Kochenderfer, CAV2017
³ Ruan, Huang, Kwiatkowska, IJCAI2018
⁴ Gehr, Mirman, Drachsler-Cohen, Tsankov, Chaudhuri, Vechev, S&P2018
Outline

Safety Problem of AI

Verification (brief)

Testing
  Test Coverage Criteria
  Test Case Generation

Conclusions and Future Works
Deep Neural Networks (DNNs)

\[ label = \arg\max_{1 \leq l \leq s_K} u_{K,l} \]
Deep Neural Networks (DNNs)

\[\text{label} = \arg\max_{1 \leq l \leq s_K} u_{K,l}\]

1) neuron activation value
\[u_{k,i} = b_{k,i} + \sum_{1 \leq h \leq s_{k-1}} w_{k-1,h,i} \cdot v_{k-1,h}\]
weighted sum plus a bias;
\[w, b\] are parameters learned

2) rectified linear unit (ReLU):
\[v_{k,i} = \max\{u_{k,i}, 0\}\]
DNN as a program

...  

// 1) neuron activation value

\[ u_{k,i} = b_{k,i} \]

for (unsigned \( h = 0; h \leq s_{k-1}; h += 1 \))
{  
    \[ u_{k,i} += w_{k-1,h,i} \cdot v_{k-1,h} \]
}

\[ v_{k,i} = 0 \]

// 2) ReLU

if \( (u_{k,i} > 0) \)
{  
    \[ v_{k,i} = u_{k,i} \]
}

...
Testing Framework

- Test Coverage Criteria
- Test Case Generation
Examples of Test Coverage Criteria

- Neuron coverage [5]
- Neuron boundary coverage [4]
- MC/DC for DNNs [8]
- Lipschitz continuity

---

5 Pei, Cao, Yang, Jana, SOSP2017.
6 Ma, Xu, Zhang, Sun, Xue, Li, Chen, Su, Li, Liu, Zhao, Wang, ASE2018
7 Sun, Huang, Kroening, ASE2018
Neuron coverage

For any hidden neuron $n_{k,i}$, there exists test case $t \in T$ such that the neuron $n_{k,i}$ is activated: $u_{k,i} > 0$.

Test coverage conditions:

\[
\{ \exists x. u[x]_{k,i} > 0 \mid 2 \leq k \leq K - 1, 1 \leq i \leq s_k \}
\]
Neuron coverage

For any hidden neuron $n_{k,i}$, there exists test case $t \in \mathcal{T}$ such that the neuron $n_{k,i}$ is activated: $u_{k,i} > 0$.

Test coverage conditions:

\[ \exists x. u[x]_{k,i} > 0 \mid 2 \leq k \leq K - 1, 1 \leq i \leq s_k \]

$\Rightarrow \approx$ statement (line) coverage

\[ \cdots \]

// 1) neuron activation value
$u_{k,i} = b_{k,i}$
for (unsigned $h = 0; h \leq s_{k-1}; h += 1$) {
    $u_{k,i} += w_{k-1,h,i} \cdot v_{k-1,h}$
}

$v_{k,i} = 0$

// 2) ReLU
if ($u_{k,i} > 0$) {
    $v_{k,i} = u_{k,i}$ \text{ \textit{\Rightarrow this line is covered}}
}

$\cdots$
Neuron Coverage

Problem of neuron coverage:

- too easy to reach 100% coverage
MC/DC in Software Testing

Developed by NASA and has been widely adopted in e.g., avionics software development guidance to ensure adequate testing of applications with the highest criticality.

Idea: if a choice can be made, all the possible factors (conditions) that contribute to that choice (decision) must be tested.

For traditional software, both conditions and the decision are usually Boolean variables or Boolean expressions.
Example: the decision

\[ d \iff ((a > 3) \lor (b = 0)) \land (c \neq 4) \]  \hspace{1cm} (2)

contains the three conditions \((a > 3)\), \((b = 0)\) and \((c \neq 4)\).

The following two test cases provide 100\% condition coverage (i.e., all possibilities of the conditions are exploited):

1. \((a > 3)\)=True, \((b = 0)\)=True, \((c \neq 4)\)=True, \(d = True\)
2. \((a > 3)\)=False, \((b = 0)\)=False, \((c \neq 4)\)=False, \(d = False\)
Example: the decision

\[ d \iff ((a > 3) \lor (b = 0)) \land (c \neq 4) \] (3)

contains the three conditions \((a > 3)\), \((b = 0)\) and \((c \neq 4)\).

The following six test cases provide 100% MC/DC coverage:

1. \((a > 3) = \text{True}, (b = 0) = \text{True}, (c \neq 4) = \text{True}, d = \text{True}\)
2. \((a > 3) = \text{False}, (b = 0) = \text{False}, (c \neq 4) = \text{False}, d = \text{False}\)
3. \((a > 3) = \text{False}, (b = 0) = \text{False}, (c \neq 4) = \text{True}, d = \text{False}\)
4. \((a > 3) = \text{False}, (b = 0) = \text{True}, (c \neq 4) = \text{True}, d = \text{True}\)
5. \((a > 3) = \text{False}, (b = 0) = \text{True}, (c \neq 4) = \text{False}, d = \text{False}\)
6. \((a > 3) = \text{True}, (b = 0) = \text{False}, (c \neq 4) = \text{True}, d = \text{True}\)
The core idea of our criteria is to ensure that not only the presence of a feature needs to be tested but also the effects of less complex features on a more complex feature must be tested.

For example, check the impact of $n_{2,1}, n_{2,2}, n_{2,3}$ on $n_{3,1}$. 
A neuron pair \((n_{k,i}, n_{k+1,j})\) are two neurons in adjacent layers \(k\) and \(k+1\) such that \(1 \leq k \leq K - 1\), \(1 \leq i \leq s_k\), and \(1 \leq j \leq s_{k+1}\).

(Sign Change of a neuron) Given a neuron \(n_{k,l}\) and two test cases \(x_1\) and \(x_2\), we say that the sign change of \(n_{k,l}\) is exploited by \(x_1\) and \(x_2\), denoted as \(sc(n_{k,l}, x_1, x_2)\), if \(\text{sign}(v_{k,l}[x_1]) \neq \text{sign}(v_{k,l}[x_2])\).
MC/DC for DNNs – Value Change and Distance Change

(Value Change of a neuron) Given a neuron $n_{k,l}$ and two test cases $x_1$ and $x_2$, we say that the value change of $n_{k,l}$ is exploited with respect to a value function $g$ by $x_1$ and $x_2$, denoted as $vc(g, n_{k,l}, x_1, x_2)$, if $g(u_{k,l}[x_1], u_{k,l}[x_2]) = True$.

\[
g(u_{2,1}[x_1], u_{2,1}[x_2]) \equiv \frac{u_{2,1}[x_1]}{u_{2,1}[x_2]} > 5
\]
A neuron pair $\alpha = (n_k,i, n_{k+1},j)$ is SS-covered by two test cases $x_1, x_2$, denoted as $\text{cov}_{SS}(\alpha, x_1, x_2)$, if the following conditions are satisfied by the network instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:

- $sc(n_k,i, x_1, x_2)$;
- $\neg sc(n_k,l, x_1, x_2)$ for all $n_k,l \in P_k \setminus \{i\}$;
- $sc(n_{k+1},j, x_1, x_2)$.
Value-Sign Cover, or VS Cover

Sign-Value Cover, or SV Cover

Value-Value Cover, or VV Cover
Relation

\[ M_N \] denotes the neuron coverage metric

arrows represent “weaker than” relation between metrics
Activation Pattern

- Given a concrete input $x$, $N[x]$ corresponds to a linear model $C$
  - $C$ represents the set of inputs following the same activation pattern
  - One DNN activation pattern corresponds to a program execution path
  - Traverse of all activation patterns $\Rightarrow$ formal verification
  - Too many patterns: e.g., $2^{10,000} \ldots$

---

Safety Coverage [10]\(^9\)

**Definition**

Let each hyper-rectangle \( rec \) contains those inputs with the same pattern of ReLU, i.e., for all \( x_1, x_2 \in rec \) we have \( sign(n_{k,l}, x_1) = sign(n_{k,l}, x_2) \) for all \( n_{k,l} \in \mathcal{H}(N) \).

A hyper-rectangle \( rec \) is safe covered by a test case \( x \), denoted as \( cov_S(rec, x) \), if \( x \in rec \).

\(^9\)Wicker, Huang, Kwiatkowska, TACAS2018


Relation

\[ M_S \]

\[ M_{SS} \quad M_{VS} \quad M_{SV} \quad M_{VV_1, 2} \]

\[ M_N \]

\( M_S \) denotes the safety coverage metric
Problem of safety coverage:

- exponential number of hyper-rectangles to be covered

Therefore, our MC/DC based criteria strikes the balance between intensive testing and computational feasibility (justified by the experimental results).
Relation with a few other criteria from [4]

- $M_{MN}$: multi-section neuron coverage
- $M_{NB}$: neuron boundary coverage
- $M_{TN}$: top-k neuron coverage
What we can do?

- bug finding
- DNN safety statistics
- testing efficiency
- DNN internal structure analysis
Test Case Generation

- optimisation based (symbolic) approach
- concolic testing
- monte carlo tree based input mutation testing
Optimisation based symbolic approach

Formalising the searching of the next test case as an optimisation problem, which can then be solved by e.g.,

- Linear Programming (LP) based, see e.g., [8]^{10}
- Global Optimisation (GO) based, see e.g., [7]^{11}

Concolic approach [9]²

Concolic testing: concrete execution + symbolic analysis

\( \mathcal{R} \): test coverage conditions
\( \delta \): a heuristic

\( \delta(\mathcal{R}) \)

ranked \( \mathcal{R} \)
top ranked

\( t, r \)
symbolic analysis

new input \( t' \)

\( \{ t_0 \} \): seed input

\( \mathcal{T} \)

Oracle

adversarial examples

¹²Sun, Wu, Ruan, Huang, Kwiatkowska, Kroening, ASE2018
Concrete execution (neuron coverage)

- The $t, r$ pair is chosen by concrete executions such that though the specified neuron is not activated by $t$, it should be really close to be activated.

Intuitively, to find the neuron that is closest to be activated

- E.g., $u_{k,i} = -1.0$ is ranked higher than $u_{k,j} = -100.0$
Concrete execution (neuron coverage)

- The $t$, $r$ pair is chosen by concrete executions such that though the specified neuron is not activated by $t$, it should be really close to be activated.

Intuitively, to find the neuron that is closest to be activated

- E.g., $u_{k,i} = -1.0$ is ranked higher than $u_{k,j} = -100.0$

... 

// 1) neuron activation value
$u_{k,i} = b_{k,i}$
for (unsigned $h = 0; h \leq s_{k-1}; h += 1$) {
    $u_{k,i} += w_{k-1,h,i} \cdot v_{k-1,h}$
}

$v_{k,i} = 0$

// 2) ReLU
if ($u_{k,i} > 0$) ≜ not satisfied
{
    $v_{k,i} = u_{k,i}$
}

...

- to select the branching point that is most likely to be satisfied
Symbolic execution (neuron coverage)

Given $t, r$, to find a new input $t'$ s.t. $r$ is satisfied.

$$\{ u'_{k,i} > 0 \land \forall k_1 < k : \bigwedge_{0 \leq i_1 \leq s_{k_1}} a'_{k_1,i_1} = a[p(t)]_{k_1,i_1} \}$$
Symbolic execution (neuron coverage)

Given $t, r$, to find a new input $t'$ s.t. $r$ is satisfied.

\[
\{ u'_{k,i} > 0 \land \forall k_1 < k : \bigwedge_{0 \leq i_1 \leq s_{k_1}} ap'_{k_1,i_1} = ap[t]_{k_1,i_1} \} 
\land \min ||t' - t||_p
\]
Symbolic execution (neuron coverage)

Given $t, r$, to find a new input $t'$ s.t. $r$ is satisfied.

\[
\{ u'_{k,i} > 0 \land \forall k_1 < k : \bigwedge_{0 \leq i_1 \leq s_{k_1}} ap'_{k_1,i_1} = ap[t]_{k_1,i_1} \} \\
\land \min ||t' - t||_p \Rightarrow \text{the symbolic engine}
\]
Symbolic execution (neuron coverage)

Given \( t, r \), to find a new input \( t' \) s.t. \( r \) is satisfied.

\[
\{ u'_{k,i} > 0 \land \forall k_1 < k : \bigwedge_{0 \leq i_1 \leq s_{k_1}} a'_{k_1,i_1} = a[p[t]_{k_1,i_1}]} \land \min||t' - t||_p \Rightarrow \text{the symbolic engine}
\]

- The CPLEX Linear Programming (LP) solver\(^\text{13}\)
  - \( L^\infty \)-norm: maximum difference among all pixels
- The global optimisation method \(^\text{14}\)
  - \( L^0 \)-norm: the number of pixels that have been changed

---


Comparison with DeepXplore

<table>
<thead>
<tr>
<th></th>
<th>DeepConcolic</th>
<th>DeepXplore</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$L_{\infty}$-norm</td>
<td>$L_0$-norm</td>
</tr>
<tr>
<td>MNIST</td>
<td>97.89%</td>
<td>97.24%</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>89.59%</td>
<td>99.69%</td>
</tr>
</tbody>
</table>
Monte carlo tree search based test case generation [10]^{15}

\textsuperscript{15}Wicker, Huang, Kwiatkowska, TACAS2018
Pixel Manipulation

define pixel manipulations $\delta_{X,i} : D \rightarrow D$ for $X \subseteq P_0$ a subset of input dimensions and $i \in I$:

$$\delta_{X,i}(\alpha)(x, y, z) = \begin{cases} 
\alpha(x, y, z) + \tau, & \text{if } (x, y) \in X \text{ and } i = + \\
\alpha(x, y, z) - \tau, & \text{if } (x, y) \in X \text{ and } i = - \\
\alpha(x, y, z) & \text{otherwise}
\end{cases}$$
Safety Testing as Two-Player Turn-based Game
Rewards under Strategy Profile $\sigma = (\sigma_1, \sigma_2)$

- For terminal nodes, $\rho \in Path_F^I$,

\[
R(\sigma, \rho) = \frac{1}{\text{sev}_\alpha(\alpha'_\rho)}
\]

where $\text{sev}_\alpha(\alpha')$ is severity of an image $\alpha'$, comparing to the original image $\alpha$

- For non-terminal nodes, simply compute the reward by applying suitable strategy $\sigma_i$ on the rewards of the children nodes
Players’ Objectives

The goal of the game is for player I to choose a strategy $\sigma_I$ to maximise the reward $R((\sigma_I, \sigma_{II}), s_0)$ of the initial state $s_0$, based on the strategy $\sigma_{II}$ of the player II, i.e.,

$$\arg \max_{\sigma_I} \opt_{\sigma_{II}} R((\sigma_I, \sigma_{II}), s_0). \quad (4)$$

where option $\opt_{\sigma_{II}}$ can be $\max_{\sigma_{II}}$, $\min_{\sigma_{II}}$, or $\nat_{\sigma_{II}}$, according to which player II acts as a cooperator, an adversary, or nature who samples the distribution $\mathcal{G}(\Lambda(\alpha))$ for pixels and randomly chooses the manipulation instruction.
Outline

Safety Problem of AI

Verification (brief)

Testing

Conclusions and Future Works
Conclusions and Future Works

Conclusions

- Testing-DNNs is a one-year old baby.
- It has attracted attentions from both the academia and the industry.
- Both criteria and test case generation need further validations.

Future Works

- Safety problems other than robustness
- DNN specific criteria, to complement the existing ones which borrow ideas from traditional software engineering
- More light-weight test case generation algorithms
- ...
Please make sure I am doing things right.

Thank You Human
Reference

T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev.
In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 948–963.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu.
Safety verification of deep neural networks.

Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer.


Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana.
Deepxplore: Automated whitebox testing of deep learning systems.

Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska.
Reachability analysis of deep neural networks with provable guarantees.

Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska.
Global robustness evaluation of deep neural networks with provable guarantees for L0 norm.

Youcheng Sun, Xiaowei Huang, and Daniel Kroening.
Testing deep neural networks.

Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening.
Concolic testing for deep neural networks.