AutoDA: Automated Decision-based Iterative Adversarial Attacks

Qi-An Fu, Yinpeng Dong, Hang Su, Jun Zhu, Chao Zhang

Tsinghua University
Adversarial Examples

• DNNs have been integrated into security-critical applications.
  • e.g., autonomous driving, healthcare, and finance.

• DNN classifiers are vulnerable to adversarial examples.
  • Small adversarial perturbations can fool DNNs.

Alps: 94.39%

Dog: 99.99%

Dong et al. 2018
Adversarial Attack & Defense

• Threat models
  • Distance metrics: $l_2$ or $l_\infty$.
  • Attacker’s goal: targeted or untargeted.
  • Attacker’s knowledge about the target model: white-box or black-box.

• **Black-box** attacks
  • Scored-based.
  • Transfer-based.
  • Decision-based.

• Defense
  • Adversarial training.

Attack with less knowledge about the target model is usually more challenging and practical!
Automated Attacks?

- Developing adaptive attacks is necessary to evaluate defenses.
  - Designed by expert case by case.
  - Requiring lots of manual trial-and-error efforts.

- Decision-based black-box attack.
  - Jacobian-based attacks.
  - Boundary attack. based on heuristics
  - Evolutionary attack.
  - HSJ attack. based on zeroth-order optimization
  - Sign-OPT attack.
Program Synthesis & AutoML

Program Synthesis
• Objective: find programs satisfying some specifications/constraints.
• Search space: programs.
• Use solvers:
  • e.g., SAT solvers, SMT solvers.

Neural Architecture Search (NAS)
• Objective: find neural network architectures achieving higher accuracy.
• Search space: constructed from expert-designed layers.
• Use advanced search method:
  • e.g., reinforcement learning, gradient-based methods.

The Problem of Automatically Discovering Decision-based Attacks

More “Logical”

More “Numerical”
AutoDA

- **Automated** Decision-based Iterative Adversarial Attacks.
- For simplicity, focus on untargeted attacks.
- Intuition: Boundary attack & Evolutionary attack.
  - Their implementations share a quite similar control flow.
  - Their main difference lies in a loop-free code segment.
  - This code segment uses only a dozen of mathematical operations.

Fix the control flow using an algorithm template

Search for the loop-free code segment

Define Search Space

Define Search Method

Define Evaluation Method
Random-walk Framework for $l_2$ Decision-based Attacks

• Proposed in the Boundary attack.
• Used by many later decision-based attacks.

Data: original example $x_0$, adversarial starting point $x_1$;
Output: adversarial example $x$ such that the $l_2$ distortion $\|x - x_0\|_2$ is minimized;
Initialization: $x \leftarrow x_1$; $d_{\text{min}} \leftarrow \|x - x_0\|_2$;
while query budget is not reached do
    $x' \leftarrow \text{generate}(x, x_0)$;
    if $x'$ is adversarial and $\|x' - x_0\|_2 < d_{\text{min}}$ then
        $x \leftarrow x'$; $d_{\text{min}} \leftarrow \|x - x_0\|_2$;
    end if
    Update the success rate of whether $x'$ is adversarial;
    Adjust hyperparameters according to the success rate;
end while

Brendel et al. 2018
Search Space

• Only search for the `generate()` function.

• Define the search space as **programs** expressed in a DSL.
  • 10 basic scalar and vector mathematical operations.
  • Loop-free, SSA form programs.
  • Accept 3 arguments $x, x_0, n$.

• Adequate **expressiveness**:
  • Enough to express the Boundary attack’s `generate()` function.

• Affordable **complexity**.

<table>
<thead>
<tr>
<th>ID</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADD.SS</td>
<td>scalar-scalar addition</td>
</tr>
<tr>
<td>2</td>
<td>SUB.SS</td>
<td>scalar-scalar subtraction</td>
</tr>
<tr>
<td>3</td>
<td>MUL.SS</td>
<td>scalar-scalar multiplication</td>
</tr>
<tr>
<td>4</td>
<td>DIV.SS</td>
<td>scalar-scalar division</td>
</tr>
<tr>
<td>5</td>
<td>ADD.VV</td>
<td>vector-vector element-wise addition</td>
</tr>
<tr>
<td>6</td>
<td>SUB.VV</td>
<td>vector-vector element-wise subtraction</td>
</tr>
<tr>
<td>7</td>
<td>MUL.VS</td>
<td>vector-scalar broadcast multiplication</td>
</tr>
<tr>
<td>8</td>
<td>DIV.VS</td>
<td>vector-scalar broadcast division</td>
</tr>
<tr>
<td>9</td>
<td>DOT.VV</td>
<td>vector-vector dot product</td>
</tr>
<tr>
<td>10</td>
<td>NORM.V</td>
<td>vector $\ell_2$ norm</td>
</tr>
</tbody>
</table>
Search Method

• Random search combined with two pruning techniques and two priors.

• Pruning techniques:
  • *Inputs check*: meaningful attacks should use all 3 inputs arguments.
  • *Distance test*: `generate()` should reduce the distance between adversarial example $x$ and original example $x_0$.

• Priors:
  • *Compact program*: generate less unused statements.
  • *Predefined statements*: the distance $d$ and the angle $u$ between $x$ and $x_0$. 
Program Evaluation Method

• Use a small and fast EfficientNet classifier on class 0 & 1 from CIFAR-10.
  • Can process more than 60,000 images/second on a single GTX 1080 Ti GPU.
• Evaluate programs on a handful of examples to save GPU time.
• $l_2$ distortion ratio. $\frac{||x-x_0||_2}{||x_1-x_0||_2}$
  • The extra $||x_1 - x_0||_2$ is for reducing the impact of the starting points.
• Two rounds evaluation:
  • 1\textsuperscript{st} round: evaluate programs with 100 steps, only keep the best program in each batch.
  • 2\textsuperscript{nd} round: evaluate programs with 10,000 steps.
We implemented a prototype of AutoDA from scratch.
- About 4,000 lines of C++.
- About 2,000 lines of Python.

Program generator generates programs with the two priors, and filters bad programs.

Program evaluator evaluates programs against the classifier on GPU.

Communications between CPU and GPU tasks are done asynchronously in large batches.
Searching for Programs Experiments

• 50 runs. Each run allows 500 million queries to the classifier.
• About 125 billion random programs are generated.
  • 45.475% of them failed in the inputs check.
  • 54.497% of them failed in the distance test.
  • Only 0.028% of them survived both.
• Distribution of the lowest $l_2$ distortion ratios found in each of the 50 runs: average at 0.01797 with a standard deviation of 0.00043.
AutoDA 1st & 2nd: The top-2 programs with lowest $l_2$ distortion ratios

Hyperparameter $x_0$

Original example

Adversarial example $x$

Random noise $n$

Hyperparameter

```
def AutoDA_1st(s0, v1, v2, v3):
    v4 = SUB.VV(v1, v2)
    s5 = NORM.V(v4)
    v6 = DIV.VS(v4, s5)
    v8 = MUL.VS(v3, s0)
    v11 = MUL.VS(v8, s5)
    v17 = ADD.VV(v8, v6)
    s18 = DOT.VV(v17, v8)
    v21 = MUL.VS(v4, s18)
    v22 = ADD.VV(v21, v2)
    v23 = SUB.VV(v22, v11)
    return v23
```

Next point to walk to $x'$

```
def AutoDA_2nd(s0, v1, v2, v3):
    v4 = SUB.VV(v1, v2)
    s5 = NORM.V(v4)
    v6 = DIV.VS(v4, s5)
    v7 = MUL.VS(v3, s0)
    v8 = ADD.VV(v7, v6)
    s9 = NORM.V(v2)
    v10 = MUL.VS(v8, s5)
    s11 = DOT.VV(v10, v7)
    v12 = DIV.VS(v6, s9)
    v17 = ADD.VV(v6, v12)
    v20 = MUL.VS(v17, s11)
    v21 = ADD.VV(v1, v20)
    v23 = SUB.VV(v21, v20)
    return v23
```

10 statements

13 statements
Benchmark Experiments

• Expert-designed baselines
  • Boundary attack.
  • Evolutionary attack. \[\text{Random-walk based. Inspired our method.}\]
  • HopSkipJump attack (HSJA). (S&P 2020)
    • HSJA (default) & HSJA* (grid search).
  • Sign-OPT attack. (ICLR 2020)

• Benchmark metrics
  • Median $l_2$ distortion vs. queries curve.
  • Attack success rate vs. queries curve.
Benchmark Experiments on CIFAR-10 models

- ResNet50
- DenseNet
- DPN
- DLA

### Graph Details

**X-axis:** Queries

**Y-axis:**
- $L_2$ Distortion
- Attack Success Rate ($\delta = 1.0$)
- Attack Success Rate ($\delta = 0.5$)

**Legend:**
- Boundary
- Evolutionary
- Sign-OPT
- HSJA
- HSJA*
- AutoDA 1st
- AutoDA 2nd
Benchmark Experiments on CIFAR-10 models

<table>
<thead>
<tr>
<th>Model</th>
<th>Queries 2,000</th>
<th>Queries 4,000</th>
<th>Queries 20,000</th>
<th>Queries 20,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary</td>
<td>10.7% 28.4% 100.0%</td>
<td>10.6% 28.5% 100.0%</td>
<td>10.7% 28.4% 100.0%</td>
<td>10.7% 28.4% 100.0%</td>
</tr>
<tr>
<td>Evolutionary</td>
<td>64.9% 96.3% 100.0%</td>
<td>66.9% 95.8% 100.0%</td>
<td>64.9% 96.3% 100.0%</td>
<td>64.9% 96.3% 100.0%</td>
</tr>
<tr>
<td>Sign-OPT</td>
<td>76.1% 98.8% 100.0%</td>
<td>77.8% 98.9% 100.0%</td>
<td>76.1% 98.8% 100.0%</td>
<td>76.1% 98.8% 100.0%</td>
</tr>
<tr>
<td>HSJA</td>
<td>91.9% 96.6% 97.1%</td>
<td>94.2% 97.9% 98.3%</td>
<td>91.9% 96.6% 97.1%</td>
<td>91.9% 96.6% 97.1%</td>
</tr>
<tr>
<td>HSJA*</td>
<td>67.4% 92.6% 100.0%</td>
<td>71.7% 92.9% 100.0%</td>
<td>67.4% 92.6% 100.0%</td>
<td>67.4% 92.6% 100.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Queries 2,000</th>
<th>Queries 4,000</th>
<th>Queries 20,000</th>
<th>Queries 20,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boundary</td>
<td>3.000 1.636 0.178</td>
<td>2.847 1.579 0.166</td>
<td>3.000 1.636 0.178</td>
<td>3.000 1.636 0.178</td>
</tr>
<tr>
<td>Evolutionary</td>
<td>0.793 0.399 0.154</td>
<td>0.754 0.378 0.142</td>
<td>0.793 0.399 0.154</td>
<td>0.793 0.399 0.154</td>
</tr>
<tr>
<td>Sign-OPT</td>
<td>0.611 0.288 0.131</td>
<td>0.586 0.273 0.123</td>
<td>0.611 0.288 0.131</td>
<td>0.611 0.288 0.131</td>
</tr>
<tr>
<td>HSJA</td>
<td>0.399 0.252 0.149</td>
<td>0.361 0.228 0.137</td>
<td>0.399 0.252 0.149</td>
<td>0.399 0.252 0.149</td>
</tr>
<tr>
<td>HSJA*</td>
<td>0.732 0.402 0.162</td>
<td>0.680 0.376 0.152</td>
<td>0.732 0.402 0.162</td>
<td>0.732 0.402 0.162</td>
</tr>
</tbody>
</table>

**Attack success rate (ε = 1.0) vs. queries**

- Though our search space is based on the Boundary attack, AutoDA 1st & 2nd are much stronger than it.

- AutoDA 1st & 2nd converge faster before ~7k queries, while converge to slightly worse adversarial examples than Sign-OPT.
Benchmark Experiments on Adv. Trained & ImageNet models

ResNet50 ($\ell_2$ adversarially trained)  WRN ($\ell_\infty$ adversarially trained)  WRN (ImageNet)  ResNet101 (ImageNet)

- Boundary
- Evolutionary
- Sign-OPT
- HSJA
- HSJA*
- AutoDA 1st
- AutoDA 2nd

(a) Boundary Evolutionary Sign-OPT HSJA HSJA* AutoDA 1st AutoDA 2nd
Conclusion

• A novel solution to automatically discover decision-based iterative adversarial attacks.

• A way to construct a search space of decision-based iterative attacks.

• An effective random search algorithm to efficiently explore the search space.

• A prototype of AutoDA
  • The discovered attacks are simple yet powerful;
  • They show comparable performance than SOTA expert-designed attacks;
  • Suggesting these expert-designed attacks are near optimal in our search space.
Thanks for listening!

Q&A

Contact: Qi-An Fu, fugoes.qa@gmail.com