Adversarial Robustness for Code

Pavol Bielik, Martin Vechev
pavol.bielik@inf.ethz.ch, martin.vechev@inf.ethz.ch

Department of Computer Science
Adversarial Robustness

**Vision**

\[ \text{panda} + \text{noise} = \text{gibbon} \]

Explaining and Harnessing Adversarial Examples. Goodfellow et. al. ICLR’15

**Sound**

Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML’18 workshop
Adversarial Robustness for Code

Vi**sion**

Explaining and Harnessing Adversarial Examples. Goodfellow et. al. ICLR’15

**Sound**

Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML'18 workshop

**Code**

refactoring
Deep Learning + Code

Prior Works

90% Accuracy
Adversarial Robustness for Code

Prior Works

90% Accuracy

Robustness

Accuracy
Adversarial Robustness for Code

Prior Works

Code Classification: 90% Accuracy
Code Captioning: 4%-50% Robustness
Bug Detection: 88% Accuracy
Code Search: 84% Robustness
Code Completion: Type Inference
Loop Invariants: Variable Naming
Bug Repair: Neural Decompilation
Program Translation: 2016

This Work

Neural Decompilation: 2017
Program Translation: 2018
Bug Repair: 2019

Accuracy
Robustness
Adversarial Robustness Example

Input Program $x$

```
...v = parseInt(
  color.substr(1),
  radix
)
...```

Goal (Adversarially Robustness):
Model is correct for all label preserving program transformations

Model $f(x) \rightarrow y$

```
...v = parseInt(
  hex.substr(42),
  radix
)
...```

Program Properties $y$

```
...v_{\text{num}} = parseInt_{\text{num}}(
  hex_{\text{str}}.substr_{\text{str}}(1),
  radix_{\text{num}}
)
...```

(Type Inference)

- variable renaming
- constant replacement
- semantic equivalence
- remove assignment
Our Work: Three Key Techniques

Abstain

1. Allows model not to make a prediction if uncertain

```javascript
v = parseInt(hex.abs.substr(1), radix.abs)
```
Our Work: Three Key Techniques

1. Abstain
2. Adversarial Training

\[ \delta = \text{hex} \rightarrow \text{color} \]
Our Work: Three Key Techniques

1. Abstain
2. Adversarial Training
3. Representation Learning
Our Work: Three Key Techniques

1. Abstain

   \( v = \text{parseInt}(\text{hex}^{\text{abs}}.\text{substr}^{\text{abs}}(1), \text{radix}^{\text{abs}}) \)

2. Adversarial Training

   \( \delta = \text{hex} \rightarrow \text{color} \)

   \( \nu^{\text{num}} = \text{parseInt}^{\text{num}}(\text{color} \cdot \text{substr}(1), \text{radix}) \)

3. Representation Learning

   \( \alpha(x + \delta) \)

84% robustness
Our Work: Three Key Techniques

1. Abstain

\[ v = \text{parseInt}(\text{hex}^{abs}.\text{substr}^{abs}(1), \text{radix}^{abs}) \]

2. Adversarial Training

\[ \delta = \text{hex} \rightarrow \text{color} \]

\[ v^\text{num} = \text{parseInt}(\text{color}.\text{substr}(1), \text{radix}) \]

3. Representation Learning

\[ \alpha(x + \delta) \]

\[ \text{parseInt}(\_, \_) \]

4. Refinement
Learning to Abstain

Leads to a simpler optimization problem

Property prediction problem is undecidable

Model should be both Robust and Accurate

Predict Class

Abstains

Model should be only Robust
Learning to Abstain

Main Insight
Combine Robustness + Learning to Abstain

How to Abstain?

Leads to a simpler optimization problem
Property prediction problem is undecidable
Our Work: Three Key Techniques

1. Abstain

\[ \delta = \text{hex} \rightarrow \text{color} \]

2. Adversarial Training

\[ \alpha(x + \delta) \]

3. Representation Learning

\[ \text{parseInt}(\text{num}) \]

4. Refinement

Learned Jointly
Adversarial Training

measures the model performance  ground-truth label

Standard training

\[ \min \text{loss}(\theta, x, y) \]

Adversarial training

\[ \min \left[ \max \text{loss}(\theta, x + \delta, y) \right] \]
\[ \delta \in S(x) \]

Label preserving program transformations

1 Define the space \( S \) of program transformations

2 Solve the inner max loss efficiently
Label Preserving Program Transformations

**Word Substitution**
- Constants, Binary Operators, ...
  - $7 + \delta$ -> $42 - \delta$
  - $\text{radix } + \text{ offset}$ -> $\text{radix } - \text{ offset}$

**Word Renaming**
- Rename Variables, Parameters, Fields, Method Names, ...
  - `def getID() {...}` -> `def get_id() {...}`
  - `client.Name` -> `client.name`

**Sequence Substitution**
- Adding Dead Code, Reordering Statements, ...
  - `a = get_id()` -> `b = 42
  - `b = 42` -> `a = get_id()`
Adversarial Training

Standard training

\[
\min \text{loss}(\theta, x, y)
\]

Adversarial training

\[
\min [\max \text{loss}(\theta, x + \delta, y)]
\]

\[
\delta \in S(x)
\]

Label preserving program transformations

1. Define the space \( S \) of program transformations
2. Solve the inner \( \max \text{loss} \) efficiently
Solving the Inner $\max$ loss Efficiently

Gradient Based Optimization

$\theta \leftarrow \theta - \nabla \text{loss}(\theta, x + \delta, y)$

$\delta \in S(x)$

Limitations

- $54\% \rightarrow 54\%$
  - standard \hspace{1cm} adversarial
- same or worse robustness
- Discrete and disruptive changes
- Highly structured and large programs
- hard optimization problem
- no structural transformations

Adversarial Examples for Models of Code.
Yefet et. al. ArXiv’20
Solving the Inner $\max \text{loss}$ Efficiently

**Gradient Based Optimization**

$$\theta \leftarrow \theta - \nabla \text{loss}(\theta, x + \delta, y)$$

$$\delta \in S(x)$$

**Refine $S$**

$$\min \left[ \max \text{loss}(\theta, x + \delta, y) \right]$$

$$\delta \in S(\alpha(x))$$

... $v = \text{parseInt}($

```javascript
  color.substr(1),
  radix
)...
```

...
Solving the Inner max loss Efficiently

Gradient Based Optimization
\[ \theta \leftarrow \theta - \nabla \text{loss}(\theta, x + \delta, y) \quad \delta \in S(x) \]

Refine \( S \)
\[ \min [\max \text{loss}(\theta, x + \delta, y)] \quad \delta \in S(\alpha(x)) \]

reduces the search space
leads to an easier optimization

... v = parseInt( color.substr(1), radix ) ...

...
Solving the Inner $\max \text{loss}$ Efficiently

Gradient Based Optimization

$$\delta \in S(x)$$

$$\theta \leftarrow \theta - \nabla \text{loss}(\theta, x + \delta, y)$$

Refine S

$$\min \left[ \max \text{loss}(\theta, x + \delta, y) \right]$$

$$\delta \in S(\alpha(x))$$

- orthogonal to gradient optimization
- supports all transformations
- reduces the search space
- leads to an easier optimization
Our Work: Three Key Techniques

1. Abstain

\[ \delta = \text{hex} \rightarrow \text{color} \]

2. Adversarial Training

\[ \alpha(x + \delta) \]

3. Representation Learning

4. Refinement

Learned Jointly

\[ \text{parseInt}(num(\_\_, \_\_)) \]
## Representation Learning

1. **Programs as Graphs**
   - Learning to Represent Programs with Graphs. Allamanis et. al. ICLR’18
   - Generative Code Modeling with Graphs. Brockschmidt et. al. ICLR’19

2. **Define Refinement**
   - \( \alpha : \langle V, E, \xi \rangle \rightarrow \langle V, E' \subseteq E, \xi \rangle \)

\[
G = \langle V, E, \xi \rangle
\]

- nodes
- attributes
- edges

\[
v = x + 7
\]
Representation Learning

Programs as Graphs

Learning to Represent Programs with Graphs. Allamanis et. al. ICLR’18
Generative Code Modeling with Graphs. Brockschmidt et. al. ICLR’19

1  Programs as Graphs

\[ v = x + 7 \]

\[ G = \langle V, \ E, \ \xi \rangle \]

\[ \alpha : \langle V, E, \xi \rangle \rightarrow \langle V, E', \subseteq E, \xi \rangle \]

2  Define Refinement

Remove Graph Edges

All decisions are made locally
Representation Learning

1 Programs as Graphs
   Learning to Represent Programs with Graphs.
   Allamanis et. al. ICLR'18
   Generative Code Modeling with Graphs.
   Brockschmidt et. al. ICLR'19

2 Define Refinement
   \( \alpha : \langle V, E, \xi \rangle \rightarrow \langle V, E' \subseteq E, \xi \rangle \)

3 Optimize \( \alpha \)
   \[ \arg \min_{\alpha} \sum_{(x, y) \in \mathcal{D}} |\alpha(x)| \]
   subject to
   \[ \text{loss}(\theta, x, y) \approx \text{loss}(\theta, \alpha(x), y) \]

\[ v = x + 7 \]

\[ G = \langle V, E, \xi \rangle \]

nodes attributes
edges

\[ \alpha \]:

Remove Graph Edges
Minimize Graph Size

\[ v = x + 7 \]
Our Work: Three Key Techniques

1. Abstain

2. Adversarial Training

3. Representation Learning

4. Refinement

Learned Jointly

\[ \delta = \text{hex} \rightarrow \text{color} \]

\[ \alpha(x + \delta) \]
Evaluation

Type Inference

Task

Type Inference

<table>
<thead>
<tr>
<th>Task</th>
<th>Labeled</th>
<th>Target Classes (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v = parseInt(hex.str.substr(1), radix)</td>
<td>string, number, boolean, void</td>
<td></td>
</tr>
<tr>
<td>() ⇒ string, () ⇒ number,</td>
<td>() ⇒ boolean, () ⇒ void</td>
<td></td>
</tr>
<tr>
<td>any</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Models

<table>
<thead>
<tr>
<th>Models</th>
<th>LSTM</th>
<th>DeepTyper</th>
<th>Graph Neural Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTM  + 1 layer GNN + LSTM</td>
<td>GNN Transformer</td>
<td>GNN GCN GNN GGNN</td>
</tr>
</tbody>
</table>

DeepTyper: Deep Learning Type Inference. Hellendoorn et. al, FSE'18

more complex type inference

Typilus: Neural Type Hints. Allamanis et. al. PLDI’20

LambdaNet: Probabilistic Type Inference using Graph Neural Networks. Wei et. al. ICLR’20

JavaScript

TypeScript

Task: Typilus: Neural Type Hints. Allamanis et. al. PLDI’20

LambdaNet: Probabilistic Type Inference using Graph Neural Networks. Wei et. al. ICLR’20

Models

<table>
<thead>
<tr>
<th>Models</th>
<th>LSTM</th>
<th>DeepTyper</th>
<th>Graph Neural Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTM  + 1 layer GNN + LSTM</td>
<td>GNN Transformer</td>
<td>GNN GCN GNN GGNN</td>
</tr>
</tbody>
</table>

DeepTyper: Deep Learning Type Inference. Hellendoorn et. al, FSE'18

more complex type inference

Typilus: Neural Type Hints. Allamanis et. al. PLDI’20

LambdaNet: Probabilistic Type Inference using Graph Neural Networks. Wei et. al. ICLR’20

JavaScript

TypeScript
Our Work: Three Key Techniques

1st Model

2nd Model

3rd Model

4 Refinement
## Evaluation

<table>
<thead>
<tr>
<th>Component</th>
<th>Accuracy</th>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Training</td>
<td>89.3%</td>
<td>54.9%</td>
</tr>
<tr>
<td>Adversarial Training</td>
<td>90.3%</td>
<td>54.3%</td>
</tr>
<tr>
<td>All Components</td>
<td>88.4%</td>
<td>83.8%</td>
</tr>
</tbody>
</table>

-1%  
+29%
### Evaluation

<table>
<thead>
<tr>
<th></th>
<th>GNN&lt;sub&gt;Transformer&lt;/sub&gt;</th>
<th>Accuracy</th>
<th>Robustness</th>
<th>Abstain</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Training</strong></td>
<td></td>
<td>89.3%</td>
<td>54.9%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Adversarial Training</strong></td>
<td></td>
<td>90.3%</td>
<td>54.3%</td>
<td>-</td>
</tr>
<tr>
<td>0%</td>
<td>All Components</td>
<td>88.4%</td>
<td>83.8%</td>
<td>-</td>
</tr>
<tr>
<td>99%</td>
<td>All Components</td>
<td>99.0%</td>
<td>99.6%</td>
<td>61.3%</td>
</tr>
<tr>
<td>100%</td>
<td>All Components</td>
<td>99.9%</td>
<td>99.9%</td>
<td>75.9%</td>
</tr>
</tbody>
</table>

**Target Accuracy**

**Allows training highly accurate & robust models**
Adversarial Robustness for Code

For more experiments and results, please refer to the extended version of our paper.

We only scratched the surface, more work in domain of code is needed and is being done, e.g.:

- Optimization-guided binary diversification to mislead neural networks for malware detection. Sharif et. al. ArXiv
- Semantic Robustness of Models of Source Code. Ramakrishnan et. al., ArXiv