Detecting Numerical Bugs in Neural Network Architectures

Yuhao Zhang\textsuperscript{1}, Luyao Ren\textsuperscript{1}, Liqian Chen\textsuperscript{2}, Yingfei Xiong\textsuperscript{1}, Shing-Chi Cheung\textsuperscript{3}, Tao Xie\textsuperscript{1}

Peking University\textsuperscript{1}
National University of Defense Technology\textsuperscript{2}
The Hong Kong University of Science and Technology\textsuperscript{3}
Detecting Numerical Bugs in Neural Network Architectures

Background & Motivation
Neural Network Architecture

Neural Network Model

Existing work on NN model:
- Testing
- Verification
- Bug Detection
...
Why Neural Network Architecture?

1. Bugs at model level are difficult to fix
   - Hours, Days, Weeks, Months, ...

2. Bugs in architectures may cause failures in training

3. Quality assurance needs to be provided for architectures
Numerical Bugs

Bugs leading to errors in numerical operations, such as “NaN”, “INF”, or crashes during training or inference.

value: [nan, nan, nan]
value: [nan, nan, nan]
value: [nan, nan, nan]
An Example of Numerical Bugs

... 
1. $y_{\text{softmax}} = \text{tf.nn.softmax}(h_{\text{fc}})$
2. $\text{cross_entropy} = y_{\_} \ast \text{tf.log}(y_{\text{softmax}})$
...
Detecting Numerical Bugs in Neural Network Architectures

- NN Architecture
- Computation Graph
- Static Analysis
- Check Unsafe Operations

Log
Exp
...

Diagram showing the process from NN Architecture to detecting numerical bugs with a computation graph and static analysis.
Infinitely many possible inputs and parameters for an NN architecture

\[ x = 0.14, 0.55, \ldots, 0.99 \]

\[ \sigma(x) = [0,1] \]

1. How about tensors?
2. How can we improve the precision of interval abstraction?
Abstraction for Neural Network Architectures

1. Tensor Partitioning
2. Interval Abstraction with Affine Equality Relation
## Abstraction on Tensors

<table>
<thead>
<tr>
<th></th>
<th>Tensor Expansion</th>
<th>Tensor Smashing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Instantiate every element</td>
<td>Smash an array into one element</td>
</tr>
<tr>
<td></td>
<td>Precise but not scalable</td>
<td>Scalable but not precise</td>
</tr>
</tbody>
</table>

Tensor Partitioning combines the strengths, scalable and precise enough

1. \( A = \text{Matrix}(2, 2) \);
   
2. \( A[1][1] = 1; \)
   
3. \( A[0][0] += A[1][1]; \)

\[
\sigma(A) = \begin{pmatrix} [0,1] & [-1,0] \\ [-1,0] & [1,1] \end{pmatrix}
\]

\[
\sigma(A) = [1] 
\]

\[
\sigma(A) = [-2,2] 
\]
Motivation of Tensor Partitioning

- Many elements of a tensor are subject to the same operations. Provide opportunity to reduce analysis effort
- Some operations like `concatenate` and `split` provide partition positions. Partition positions come free.
Abstraction on Tensors

Tensor Partitioning

• Partition the tensor into a set of disjoint parts
• Smash each part into one element

1. A = Matrix(2,2);
   //within [-1,0]

2. A[1][1] = 1;

3. A[0][0] += A[1][1];

\[\sigma(A) = (-1,0) \quad (-1,1)\]

\[\sigma(A) = (-2,1) \quad (-1,1)\]
Motivation of Affine Equality Relation

Many elementwise affine operations in computation graphs
Affine equality relation is more precise than sole interval abstraction.
Sole Interval Abstraction

1. \( a, b \)
   \( \sigma(a) = [0,1], \sigma(b) = [1,2] \)

2. \( x = a + b; \)
   \( \sigma(x) = [0,1] + [1,2] = [1,3] \)

3. \( y = a - b; \)
   \( \sigma(y) = [0,1] - [1,2] = [-2,0] \)

4. \( z = x + y; \)
   \( \sigma(z) = [1,3] + [-2,0] = [-1,3] \)

Interval abstraction abstracts away the relation between \( x, y, \) and \( z \)
Interval Abstraction with Affine Equality Relation

Affine Equality Relation: \[ \sum_i \omega_i x_i = \omega_0 \]

1. a, b
   \[ \sigma(a) = [0,1], \sigma(b) = [1,2] \]

2. x = a + b;
   \[ \sigma(x) = [0,1] + [1,2] = [1,3] \] \[ x = a + b \]

3. y = a - b;
   \[ \sigma(y) = [0,1] - [1,2] = [-2,0] \] \[ y = a - b \]

4. z = x + y;
   \[ \sigma(z) = [1,3] + [-2,0] = [-1,3] \]
   \[ [0,1] + [0,1] = [0,2] \] \[ z = x + y = 2a \]
Evaluation
A Collection of Neural Network Architectures

- 9 buggy architectures from previous study [1, 2]
- 48 real-world architectures from tensorflow/models [3], containing different NN architectures (including CNN, RNN, GAN, HMM) in various research domains

Main Results

Framework
(Tensor Abstraction + Numerical Abstraction)

DEBAR
(Tensor Partitioning + Affine Equality Relation):
Accuracy: 93.0%, all in 3 minutes, 12.1s on average
100% accuracy on 9 buggy architectures

Tensor Partitioning + Sole Interval Abstraction
Accuracy: 80.6%, 12.1s on average

Instantiate every element in an array

\[ \sigma(A) = \begin{pmatrix} [0,1] & [-1,0] \\ [-1,0] & [1,1] \end{pmatrix} \]

Tensor Expansion + Affine Equality Relation:
33/57 > 30mins; on rest 24, DEBAR doesn’t lost accuracy

Tensor Smashing + Affine Equality Relation:
Accuracy: 87.1%, 12.2s on average

Smash an array into one element
\[ \sigma(A) = [-2,2] \]
Bugs in Real-World Architectures

Found 11 buggy statements in the code repository
Submitted pull requests, and 3 buggy statements have been repaired by the developers
Detect Numerical Bugs in Neural Network Architectures

How to map the buggy operations to the buggy code statements?

Design Abstraction Techniques for Analyzing Neural Architectures

1. Interval Abstraction with Affine Equality Relation
2. Tensor Partitioning

How to analyze the dynamic computation graphs?

Collect 57 Computation Graphs for Future Research