Learning Boolean Circuits from Examples for Approximate Logic Synthesis

Sina Boroumand, Christos-Savvas Bouganis, George A. Constantinides
Outline

- Motivation and Background
- Problem Definition
- LSE: Logic Synthesis from Examples
- Evaluation
- Conclusion and Future Works
Motivation: Approximate Logic Synthesis

- Computing systems becoming energy hungry
- Applications tolerate computation inaccuracy
  - Robotic
  - Pattern recognition
  - Machine learning
- Approximate Logic Synthesis (ALS)
  - Process of automatically converting a high-level design description into a Boolean circuit
    - Without violating an error constraint
ALS: Techniques

- Structure of exact model is *known*
  - *e.g.* BLASYS, SALSA, and Circuit Carving

- Structure is *unknown*
  - Synthesize from examples
  - Fully/Partially specified truth table
    - CleanSlate, Espresso
Proposed Solution: LSE (Logic Synthesis from Examples)
Mutual Information

- For two random variables $X$ and $Y$, the mutual information $I(X; Y)$ measures the average reduction in uncertainty about $X$ that results by learning the value of $Y$, or vice versa.
Problem Definition

Input:  
Sequence of training inputs \( X = (x_i) \)  
Corresponding sequence of labels \( Y = (y_i) \)  
Maximum acceptable area \( A_{\text{max}} \)

Output:  
Approximate circuit \( f : X \rightarrow Y \)

\[
f = \arg \max_{\tilde{f}} I \left( Y ; \tilde{f}(X) \right) \quad \text{s.t.} \quad \text{Area}(f) \leq A_{\text{max}}
\]

\( I(\cdot) \) denoted as Mutual Information
LSE: Algorithm

- Builds the circuit iteratively node by node by increasing the Mutual Information
  - Modified version of muesli [Oliveira et al., 1994]
  - Learning functionality of Boolean nodes is based on Memorization [Chatterjee, 2018]

Algorithm (1/5)

Training Input Samples

<table>
<thead>
<tr>
<th>#</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Primary Inputs

Primary Output

Input

Output
Algorithm (2/5)

Candidate List \((C)\)

- a
- b
- c
- d
- e

Active List \((A)\)

- c  Most informative node
- a
- b

Active Node
Algorithm (3/5)

Candidate List \((C)\)

- a
- b
- c
- d
- e

Active List \((A)\)

- Most informative node: f
- Active Node: a
- Active Node: b
- Active Node: d
- Active Node: e

Find \(f_0(a, b)\) such that \(I(c, f_0, y) > I(c, a; y)\)

If Success
- add \(f_0\) to \(C\)
- reset active list

Else
- add another node to active list
- increment the active node

Initial support = 2
Algorithm (5/5)

Candidate List (C)

Ending Condition:
\[ \varepsilon(v, y) \geq \varepsilon', \text{ for at least one } v \in C \]

NMI: \[ \varepsilon(v, y) = \frac{I(v; y)}{H(y)} \]
Evaluation

- Benchmarks
  - MCNC logic synthesis suite (10 logic labeled benchmarks randomly selected)
  - Binary MNIST
    - ‘0’ to ‘4’ classified as class 0
    - ‘5’ to ‘9’ classified as class 1
    - Images converted to 1-bit pixels

- Comparison
  - Espresso, CleanSlate
Evaluation Framework Flow: MCNC benchmarks

Design Flow

- Benchmark (BLIF)
- Sample Generator
- Train Samples (PLA)
- Test Samples (PLA)
- LSE
- Logic Optimization (ABC)
- Analyzer
- Design Database
- User's design requirements
- Database Query
- Design Pick
MCNC: Area

- Area versus test error
- The pareto frontier design points are only shown
- The level approximation of LSE can be explicitly controlled by $\varepsilon'$
- LSE outperforms other techniques
- For benchmarks with large inputs both Espresso and CleanSlate were unable to synthesize
MNIST: Binary classification

- MNIST dataset
  - Training set: 60k examples
  - Test set: 10k examples

- Both CleanSlate and Espresso were unable to generate any circuit

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ACCURACY (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TRAINING</td>
</tr>
<tr>
<td>LSE-8</td>
<td>94.68</td>
</tr>
<tr>
<td>LSE-10</td>
<td>94.80</td>
</tr>
<tr>
<td>Memorization [3]</td>
<td>99.00</td>
</tr>
<tr>
<td>Random Guess</td>
<td>50.00</td>
</tr>
</tbody>
</table>

LSE-8 (support size = 8)
LSE-10 (support size = 10)
Conclusion and Future Works

- We proposed a technique to approximately synthesize a circuit from a set of input/output based on searching using Mutual Information.
- We shown that LSE provides a good generalization and low area/delay utilization compared to existing methods.
- The proposed method is limited to single-bit outputs, thus further studies need to be undertaken to benefit from word-level search to generate multi-output circuits efficiently.
Questions?

s.boroumand18@imperial.ac.uk