Test Case Generation and Reduction by Automated Input-Output Analysis

Prachi Saraph, Mark Last, and Abraham Kandel
Introduction

● Black-Box Testing
  ● Apply an Input
  ● Observe the corresponding output
  ● Compare Observed output with expected

● Large Number of Inputs
  → Huge number of Test Cases
Introduction

• Choose most important test cases and removing the redundant ones

• How?
  • Manual
  • Automatic
Input-Output Analysis

• Identifies the input attributes which affect the value of a particular output

• Concentrates on relationships between inputs and outputs
Machine Learning Approaches

- NN-based mechanism for identification of test cases that are likely to find faults
  
  \[\text{C. Anderson et al., 1995}\]

- NN is used to detect faults in mutated versions of software (Regression Testing)
  
  \[\text{M. Vanmali, 2002}\]

- IFN is used to identify I-O relationships
  
  \[\text{M. Last et al., 2003}\]
Neural Networks – Background

- Supervised Learning

- Learn weights on each edge (Synapse) to get a general model of mapping between inputs and outputs based on training data
Neural Network based Testing

- NN Construction and Training
- Pruning and Feature Ranking
- Rule Extraction
- Test Case Generation
Phase I: NN Construction and Training

- 3-Layer Feed Forward Network using EBP

**Obtaining Training Set:**
- Randomize input values
- Feed them into the original software
- Storing output generated
Phase II (a) : NN Pruning

- Less important connections → Lower Weights

- Pruning:
  - Remove edges with lower weights as long as the predictive accuracy after removing a link stays within the acceptable limits
Phase II (b) : Feature Ranking

- **Method 1 (Sorting):**
  - Sort inputs according to the product
    \[
    \text{weights from input layer to hidden layer} \times \text{Corresponding weights from hidden layer to output layer}
    \]

- **Method 2 (Pruning):**
  - Apply pruning method until all edges are removed
  - Note the order in which input nodes get all their nodes removed
Phase III: Rule Extraction

- Express I-O relations pertained by pruning as if-then rules
- Use clustering to discretize hidden unit activation values
- Link Inputs to outputs through discretized hidden values
Phase IV : Test Case Generation

- After the completion of the pruning phase the possible data values of the attributes are used as equivalence classes to build test cases.
## Case Study: Employment Application Approval System

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Legend</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application ID Number</td>
<td>UID</td>
<td>None</td>
</tr>
<tr>
<td>Degree</td>
<td>B.sc. / M.sc. / PhD</td>
<td>Input</td>
</tr>
<tr>
<td>Years of Experience</td>
<td>0 – 10</td>
<td>Input</td>
</tr>
<tr>
<td>Years out of College</td>
<td>0 – 10</td>
<td>Input</td>
</tr>
<tr>
<td>Certification</td>
<td>Yes / No</td>
<td>Input</td>
</tr>
<tr>
<td>Employment History</td>
<td>0 – 10</td>
<td>Input</td>
</tr>
<tr>
<td>Immigration Status</td>
<td>Citizen / Permanent Resident / Work Permit</td>
<td>Input</td>
</tr>
<tr>
<td>Number of References</td>
<td>0 – 3</td>
<td>Input</td>
</tr>
<tr>
<td>Employment Approval</td>
<td>Yes / No</td>
<td>Output</td>
</tr>
<tr>
<td>Full Time</td>
<td>Yes / No</td>
<td>Output</td>
</tr>
<tr>
<td>Part Time</td>
<td>Yes / No</td>
<td>Output</td>
</tr>
</tbody>
</table>
Case Study: Employment Application Approval System

- Random numbers were generated in the range of every input

- The inputs were fed to the application code, which produced the outputs

- The size of a training data set and a test data set was 1000 examples
Case Study: Employment Application Approval System

- The cycle of training, pruning and rule-extraction was run on a training data set at least ten times.

- Two stopping criteria:
  - Upper limit on number of training epochs (1500)
  - Minimum accuracy on training data (set to 97%)

- The total number of test cases can be calculated by taking a Cartesian product of the corresponding data values. (58,080)
Results

- After Pruning Phase, The links retained in the network for output of ‘Full Time’ corresponded to two inputs:
  - years of experience
  - employment history

- Number of test cases needed for this output = $|\text{years of experience}| \times |\text{employment history}| = 11 \times 10 = 110$

- For the other two outputs, 120 test cases were needed

- Total Number = 230 vs 58,080 $\Rightarrow$ Huge Reduction
Rule Extraction Phase

- By clustering hidden unit activation levels and determining ranges of inputs that can generate each level, the following was found:
  - Values 4 – 10 for the input attribute ‘years of experience’ resulted in activation value of 1 while the rest of the values resulted in activation value of -1
  - Values 6 – 10 for the input attribute ‘Employment History’ generated activation value of 1 and the rest generated a value of -1
Results

- For this output ‘Full Time’ the following rule is generated:
  
  If (years of experience) and (employment history=6)
  
  Employment Hours: Full Time=Yes (1)
  
  Else
  
  Employment Hours: Full Time=No (0)

- By investigating the code, the accuracy of this rule turned out to be 100%
Test cases after rule-extraction phase

- By investigating previous extracted rules, we can build two equivalence classes for each of the two influential inputs:
  - Years of experience 1: [0-3]
  - Years of experience 2: [4-10]
  - Employment history 1: [1-5]
  - Employment history 2: [6-10]

- Each equivalence class can be represented by one value.
Yet More Test case Reduction

- Thus, to cover combinations of these two input attributes, we need 4 test cases for this output instead of previous 110

- By repeating the procedure for each of the other outputs, we get 10 test cases for the whole application

- Those 10 test cases carry some redundancies, so applying some minimization algorithm can further reduce them to 4 test cases!!
Drawbacks

- Generating and running random test cases to create the training set incurs some overhead that wasn’t addressed properly in the paper.

- The method they propose for generating the training set implicitly assumes that you have a fault-free version of the program (which is not always the case).

- The authors didn’t actually give a basis or an experimental framework for choosing the NN learning or pruning parameters.
Digging Deeper

- Although the authors didn’t mention it explicitly, their approach is mainly useful in regression testing.

- Another idea is to utilize this approach in Oracle generation out of the specifications of the program (By providing I-O pairs that are valid under specifications as he training set).
Questions
Thank You