Various Testing Technique

Md Firoz Alam
2023pis5097

Malaviya National Institute of Technology Jaipur

Submitted to:
Prof.Gridhari Singh.

May 27, 2024

Outline

Differential Testing

1. Differential testing, also known as differential fuzzing, is a popular software testing technique that attempts to detect bugs, by providing the same input to a series of similar applications (or to different implementations of the same application), and observing differences in their execution.
2. Differential Testing (DT) assesses a system's functionality by comparing the behavior of multiple implementations or models performing the same task. These systems are tested with identical inputs, and discrepancies in their outputs help identify potential faults. Two primary issues in DT are identifying the faulty system, as **the correct answer isn't known**, and **ensuring the test set's quality is sufficient for confidence in the system's correctness**.

Explain With Example:-

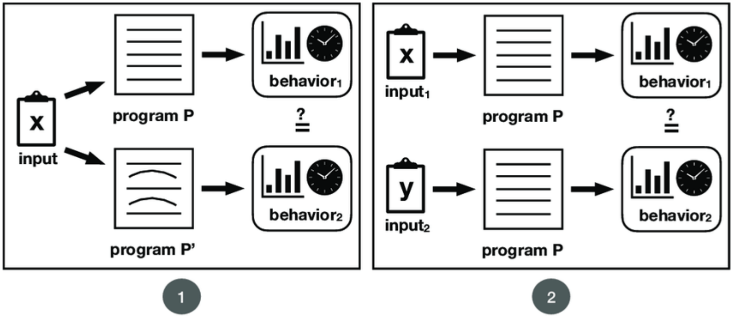


Figure: 1

Cont...

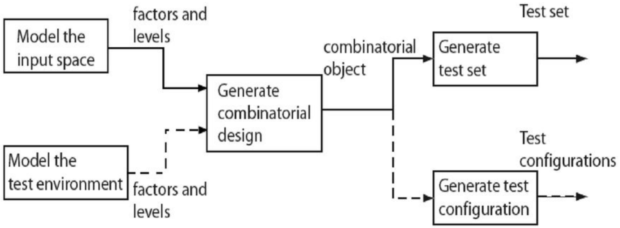
DeepXplore was the pioneering method to apply DT to test deep learning (DL) models. It utilized public datasets like MNIST, ImageNet, VirusTotal, Udacity video dataset, and Drebin to test state-of-the-art DL models. DeepXplore demonstrated that combining neuron activation coverage with DT could generate inputs highlighting incorrect behaviors by selecting inputs that lead to wrong classifications for most models. These inputs could then be used to retrain and improve the tested models. Despite this, the accuracy achieved for the MNIST dataset was 98.96

Combinatorial testing

1. Combinatorial Testing (CT) is a software testing approach that evaluates the quality of test suites and software systems by focusing on interactions between input data values. Instead of exhaustively testing all input combinations, CT targets specific interactions (like pairs or triples of values), making the testing process more structured and efficient. This method is particularly useful for testing deep learning (DL) systems, where it helps to identify interactions between neurons in the network.
2. **Efficiency:-** CT focuses on representative interactions, avoiding exhaustive searches and enhancing efficiency.
   Application in DL:- CT is used to investigate neuron interactions within DL models, aiding in defect detection.

Combinatorial Testing Process:-

Combinatorial test design process



Modeling of input space or the environment is not exclusive and one might apply either one or both depending on the application under test.

© Aditya P. Mathur 2009

Figure: 1

Cont....

Combinatorial Smoke Testing:- Smoke Testing (ST) is a degree of testing performed by developers on the first software builds to ensure that all of the basic/core features are operating properly. Due to a number of issues like configuration, regression, code, and environmental issues, the environment is not well served by the software builds. So, after the development team has completed the initial build, it is subjected to basic testing before being submitted to the next stage of testing. This initial level of testing is known as smoke testing. Using this testing technique, the authors of performed experiments on both supervised (classification) and unsupervised (clustering) learning and they came up with universal smoke tests using techniques like boundary value analysis, and equivalence classes, which verifies the fundamental functions that can be performed without failing.

Metamorphic Testing(MT)

1. Metamorphic Testing (MT) is a valuable approach for testing programs when defining a precise oracle is challenging or impossible. MT leverages known relations between input and output, termed metamorphic relations (MRs), to identify faults in programs. For instance, MRs could stipulate that certain transformations applied to input data should not alter the output. By using MRs, follow-up test cases can be generated from initial test cases, and if the relationship is violated, a fault is revealed, even if the exact fault location or correct output is unknown.
2. Metamorphic Testing (MT) is a crucial technique for testing software when defining a precise oracle is difficult or unfeasible. MT uses known relations between inputs and outputs, called metamorphic relations (MRs), to detect faults. MRs specify how changes in input data should affect the output. By applying MRs, follow-up test cases are derived from initial ones, and if the expected relationships are violated, a fault is identified. This method enhances the software reliability even if the exact fault location or correct output is unkown.

Cont...

1. However, identifying MRs tailored for testing DL models poses challenges. While various MRs have been proposed, their effectiveness in detecting bugs may vary. Contextual hijackers have been suggested to aid in selecting the most appropriate MRs for testing DL models, showcasing promise in enhancing MR selection.
2. Moreover, tools like DeepTest have been developed to leverage MT for testing self-driving car decision systems, ensuring accurate behavior under diverse conditions. Despite these advancements, deploying MT for testing DL models in safety-critical Virtualized and Embedded Systems (VBS) remains relatively unexplored, presenting its own set of challenges.
3. In an experimental setup inspired by DeepTest, researchers designed MRs to test trained DL models on the MNIST dataset. By applying various transformations to the input images, such as rotations, shifts, shearing, and zooming, they assessed the model's performance under different conditions. Their experiments revealed that DL models trained with augmented data generally perform better .

MT Explain With Example:-

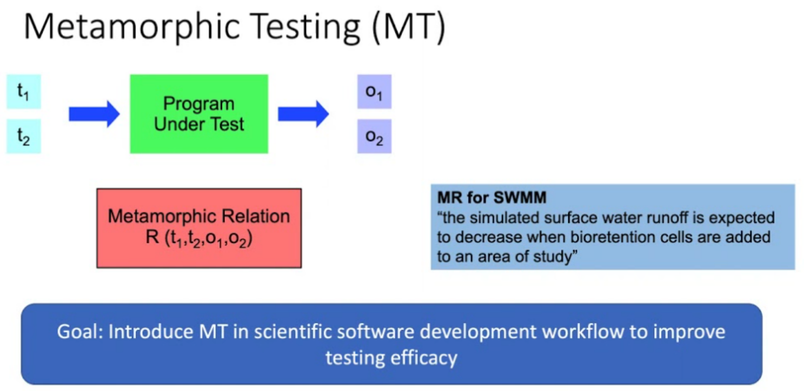

Metamorphic Testing (MT)
t_1
t_2
Program Under Test
o_1
o_2
Metamorphic Relation R (t_1,t_2,o_1,o_2)
MR for SWMM
"the simulated surface water runoff is expected to decrease when bioretention cells are added to an area of study"
Goal: Introduce MT in scientific software development workflow to improve testing efficacy

Figure: 1

Adversarial pertubation Testing(APT)

1. Adversarial Perturbation Testing (APT) is a Testing technique derived from adversarial attacks on machine learning systems, which can be repurposed to assess the robustness of Deep Learning (DL) models. Adversarial examples, intentionally crafted inputs, are designed to deceive DL models into producing incorrect outputs. These examples are crafted to be similar to correctly classified inputs, making them challenging to detect by humans or traditional means.
2. There are two main types of adversarial examples: **targeted** and **untargeted**. Targeted examples aim to make the model classify an input as a specific incorrect class, while untargeted examples focus on misclassifying an input without a specific target. Adversarial examples can be generated using algorithms or gradient ascent methods, resulting in images that are imperceptible to humans but confidently misclassified by DL models.

Adversarial Perturbations Testing Process:-

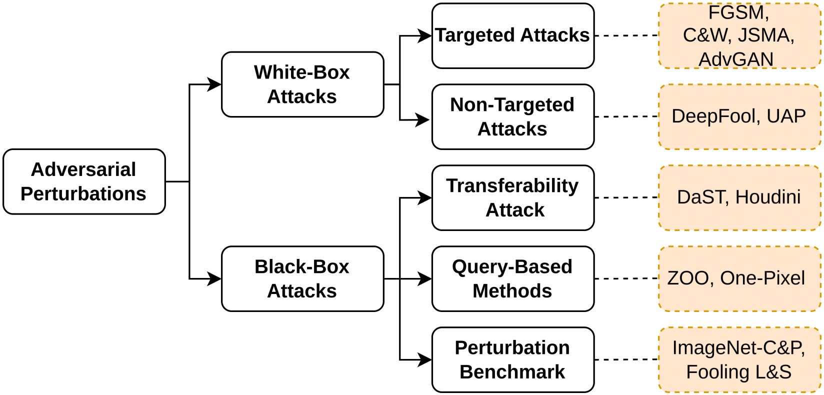


Figure: 1

Cont...

1. Adversarial Perturbation Testing (APT) leverages adversarial examples to construct test cases that expose misclassifications in DL models. **CleverHans** is a widely-used toolkit for automatically generating adversarial inputs, aiding in quantifying the robustness of DL classifiers. Tools like DeepFool iteratively linearize DL classifiers to calculate minimal perturbations necessary for misclassification.
2. Moreover, Generative Adversarial Networks (GANs) can enhance APT by using a discriminator network to evaluate inputs generated by a generative network. GANs can learn to produce adversarial examples, effectively becoming APT tools themselves. For instance, DeepRoad combines GANs with Metamorphic Testing (MT) to create adversarial examples for testing self-driving car decision systems. However, GAN training can be challenging, leading to non-convergence issues and increasing testing complexity.

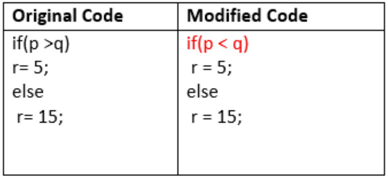
Cont...

1. Recent advancements include algorithms that formalize the space of adversaries in neural networks, crafting adversarial examples based on accurate mappings between input and output. These algorithms have demonstrated high success rates in producing adversarial examples while modifying input features minimally. Additionally, hardness measures have been introduced to quantify the vulnerability of samples to adversarial perturbations.
2. APT leverages adversarial examples to reveal vulnerabilities in DL models, aiding in assessing their robustness. However, challenges such as GAN training complexity and non-convergence issues persist, highlighting the ongoing efforts to enhance APT techniques for more effective DL model testing

Mutation Testing(MT)

1. Mutation testing is a method of software testing in which program or source code is deliberately changed, followed by testing of the changed code.
   Mutation testing is a technique that focuses on measuring the adequacy of test data and to check the efficiency of test data.
2. Mutation testing is a way to measure the quality of test cases, along with that actual testing of program units is an added benefit.
   Mutation testing deals with changing the code and then analysis of test case, therefore it comes under **white box testing**.
3. To determining if a set of test data or test cases are useful, by **deliberately** introducing various bugs in the program. Re-testing with the original test data/cases to determine if the bugs are detected. This is basically to **test the test cases**.

Mutation Testing Example



Original Code	Modified Code
if(p >q)	if(p < q)
r= 5;	r = 5;
else	else
r= 15;	r = 15;

Figure: 1

Mutation Testing

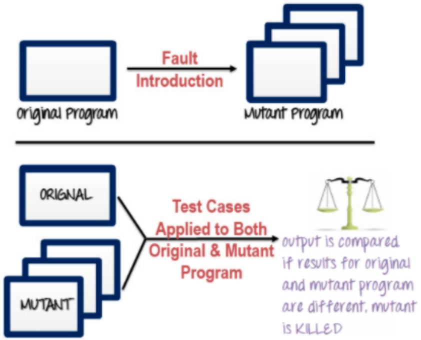


Figure: 2

Steps of Mutation Process:-

1. Step 1: Faults are introduced into the source code of the program by creating many versions called mutants. Each mutant should contain a single fault, and the goal is to cause the mutant version to fail which demonstrates the effectiveness of the test cases.
2. Step 2: Test cases are applied to the original program and also to the mutant program. A test case should be adequate, and it should able to detect faults in a program.
3. Step 3: Compare the results of original and mutant program.
4. Step 4: If the original program and mutant programs generate the different output, then that the mutant is killed by the test case. Hence the test case is good enough to detect the change between the original and the mutant program.
5. Step 5: If the original program and mutant program generate same output, Mutant is kept alive. In such cases, more effective test cases need to be created that kill all mutants.

Ends

Thank you

Ends