Slice Testing With Neural Network

Mahesh Kumar Tiwari[1], Rinku Raheja[2]

[1, 2] *Assistant Professor, Computer Science Department, National P.G. College, Lucknow, U.P, India*

Abstract: **- Slice Testing is a testing technique in which a program is divided into slices in order to identify the segments that will be affected by any change or to identify which set of statement (grouped as slices) will be executed at a particular instant. In this paper we have tried to mix the concept of slice testing with Neural Network using Object Oriented Programming.**

In this we will take execution time of each slice as the weight factor in the evaluation of net input. This allows us to identify the overall architecture of the program and helps in evaluating the overall performance and dependencies of the system. Here, each slice will act as a node of the network for which net input will be calculated. We are focused on implementing this technique in an Object Oriented Program as nowadays all of the software development is based on the concept of Object Oriented Programming.

Keywords**: - Slicing, Static Slicing, Forward Slicing, Backward Slicing, Dynamic Slicing, Object Oriented Programming.**

I. INTRODUCTION

This paper is based on the concepts of Slicing and Artificial Neural Networks both these fields have evolved over the time in various ways. Here we have described what Slice based testing and Artificial Neural network is and how these concepts can be merged and used to calculate the performance of the program. Slices as described by Weiser are created on the basis of Slicing Criteria which is of the form <P,V> where P is the point of interest of the program and V is the variable. A slice consists of all the statements in a program which are affected by the defined slicing criteria. A slice can be computed by two techniques, forward slicing and backward slicing. Once the slices are determined we calculate the execution time of slices with help of Junit. With this time we will calculate the net input for the whole program. Now let us look into the details of each topic.

II. PROGRAM SLICING

As mentioned that this technique has evolved in many ways so there are many ways which can be used to get the program slices. Some of these are mentioned below:

Forward Slicing- A forward slice contains the set of statements that might get affected by the slicing criteria i.e., it provides answer to the following question "which statements will be affected by the slicing criterion?"

Backward Slicing- A backward slice contains all parts of the program that might directly or indirectly affect the slicing criterion. Thus a static backward slice provides the answer to the question: "which statements affect the slicing criterion?"

Static Slicing- A static slice contains all the statements that might directly or indirectly affect the slicing criteria. The size of slice is large in comparison to Dynamically sliced program. There are no assumptions made on input while slicing.

Dynamic Slicing- Here the slicing criteria is defined as <I, P, V> where I is the input, P is the point of interest and V is the variable for which we wish to monitor the program. Here we include only those statements that directly affect the program.

Static and Dynamic slicing are performed in combination with Forward and Backward slicing. As clear by the definitions Dynamic Slicing is more suitable for Debugging Object oriented Programs.

Example of Backward Static and Backward Dynamic is shown in *Figure-1 and Figure-2* for a C++ program.

```
1. cin>>n;
2. cout<<n;
3. int a=2;
4. if (n>2)
5.   a=4;
6. else
7.   a=6;
8. cout<<a;
```

The above figure shows Backward Static Slicing with Criteria<8, a> and the figure below shows Backward Dynamic Slicing with Criteria<3, 8, a>.

```
1. cin>>n;
2. cout<<n;
3. int a=2;
4. if (n>2)
5.   a=4;
6. else
7.    a=6;
8. cout<<a;
```

III. ARTIFICIAL NEURAL NETWORK

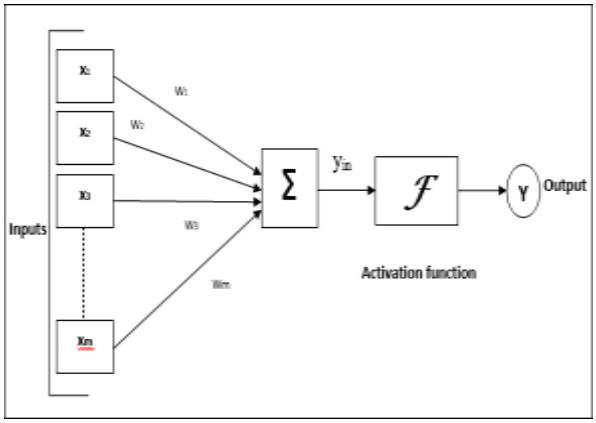
Artificial Neural Network ANN is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also

referred to as nodes or neurons, are simple processors which operate in parallel.

Every neuron is connected with other neuron through a connection link. Each connection link is associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

Model of Artificial Neural Network

The following diagram shows the structure of an Artificial Neural Network .



For this model net input can be calculated as follows –

$$y_{in} = x1.w1 + x2.w2 + x3.w3 \ldots xm.wm$$

i.e., Net input $y_{in} = \sum$ mixi.wi

IV. SLICING AND UNIT TESTING

Now we are simply going to analyze the program in Java by creating slices of the program and calculating its execution time.

Consider the following program which implements four methods add(), multiply(), subtract(), divide.

```
1.  class Methods
2.  {
3.  public int add(int a, int b)
4.  {       return a+b;     }
5.  public int multiply(int a,int b)
6.  {       return a*b;    }
7.  public int subtract(int a,int b)
8.  {       return a-b;    }
9.  //Division of whole no.s
10. public int divide(int a, int b)
11. {    if (b>0 && a>=0)
12. return a/b;
13. else
14. return -1;
15. }}
16. public class Operations {
17. public static void main(String[] args)
18. {   Methods m=new Methods();
19. System.out.println("Sum:"+m.add(10,12));
20. System.out.println("Product:"+m.multiply(100,2));
21. System.out.println("Subtraction:"+m.subtract(48,
    60));
22. System.out.println("Quotient:"+m.divide(48, 2));
23. }
24. }
```

Figure - Sample Code

Slices for the above program based on the

```
1.  class Methods
2.  {
3.  public int add(int a, int b)
4.  {       return a+b;     }
5.  public int multiply(int a,int b)
6.  {       return a*b;    }
7.  public int subtract(int a,int b)
8.  {       return a-b;    }
9.  //Division of whole no.s
10. public int divide(int a, int b)
11. {    if (b>0 && a>=0)
12. return a/b;
13. else
14. return -1;
15. }}
16. public class Operations {
17. public static void main(String[] args)
18. {   Methods m=new Methods();
19. System.out.println("Sum:"+m.add(10,12));
20. System.out.println("Product:"+m.multiply(100,2));
21. System.out.println("Subtraction:"+m.subtract(48,
    60));
22. System.out.println("Quotient:"+m.divide(48, 2));
23. }
24. }
```

Slicing Criterion: < [10, 12], 19, add () >

Figure - S1

different slicing criteria are shown below:

```
1.                  class Methods
2.  {
3.  public int add(int a, int b)
4.  {       return a+b;     }
5.  public int multiply(int a,int b)
6.  {       return a*b;    }
7.  public int subtract(int a,int b)
8.  {       return a-b;    }
```

```
9.  //Division of whole no.s
10. public int divide(int a, int b)
11. {    if (b>0 && a>=0)
12. return a/b;
13. else
14. return -1;
15. }}
16. public class Operations {
17. public static void main(String[] args)
18. {   Methods m=new Methods();
19. System.out.println("Sum:"+m.add(10,12));
20. System.out.println("Product:"+m.multiply(100,2));
21. System.out.println("Subtraction:"+m.subtract(48,
    60));
22. System.out.println("Quotient:"+m.divide(48, 2));
23. }
24. }
```

Slicing Criterion: <[100,2],20,multiply ()>

Figure – S2

```
1.  class Methods
2.  {
3.  public int add(int a, int b)
4.  {       return a+b;     }
5.  public int multiply(int a,int b)
6.  {       return a*b;    }
7.  public int subtract(int a,int b)
8.  {       return a-b;    }
9.  //Division of whole no.s
10. public int divide(int a, int b)
11. {    if (b>0 && a>=0)
12. return a/b;
13. else
14. return -1;
15. }}
16. public class Operations {
17. public static void main(String[] args)
18. {   Methods m=new Methods();
19. System.out.println("Sum:"+m.add(10,12));
20. System.out.println("Product:"+m.multiply(100,2));
21. System.out.println("Subtraction:"+m.subtract(48,
    60));
22. System.out.println("Quotient:"+m.divide(48, 2));
23. }
24. }
```

Slicing Criterion:<[48,60],21,subtract()>

Figure – S3

```
1.  class Methods
2.  {
3.  public int add(int a, int b)
4.  {       return a+b;     }
5.  public int multiply(int a,int b)
6.  {       return a*b;    }
7.  public int subtract(int a,int b)
8.  {       return a-b;    }
9.  //Division of whole no.s
10. public int divide(int a, int b)
11. {    if (b>0 && a>=0)
12. return a/b;
13. else
14. return -1;
15. }}
16. public class Operations {
17. public static void main(String[] args)
18. {   Methods m=new Methods();
19. System.out.println("Sum:"+m.add(10,12));
20. System.out.println("Product:"+m.multiply(100,2));
21. System.out.println("Subtraction:"+m.subtract(48,
    60));
22. System.out.println("Quotient:"+m.divide(48, 2));
23. }
24. }
```

Slicing Criterion:<[48,2],22,divide()>

Figure – S4

The test file prepared to test these slices has been created and is shown in the figures below:

```
@Test
   public void testAdd() {
      System.out.println("add");
      int a = 12;
      int b = 32;
      Methods instance = new Methods();
      int expResult = 44;
      int result = instance.add(a, b);
      assertEquals(expResult, result);
   }
```

Figure Test Method for add()

```
@Test
   public void testMultiply() {
      System.out.println("multiply");
      int a = 100;
      int b = 2;
      Methods instance = new Methods();
      int expResult = 200;
      int result = instance.multiply(a, b);
      assertEquals(expResult, result);
   }
```

Figure Test Method for multiply()

```
@Test
   public void testSubtract() {
      System.out.println("subtract");
      int a = 20;
      int b = 5;
      Methods instance = new Methods();
      int expResult = 15;
      int result = instance.subtract(a, b);
      assertEquals(expResult, result);
   }
```

Figure Test Method for subtract()

```
@Test
   public void testDivide() {
      System.out.println("divide");
      int a = 6;
      int b = 2;
      Methods instance = new Methods();
      int expResult = 3;
      int result = instance.divide(a, b);
      assertEquals(expResult, result);
   }
```

Figure Test Method for divide()

Now the execution time calculated for each slice (S1, S2, S3, S4) using Junit are 0.088s, 0.085 s, 0.077 s, 0.078 s, respectively. Also, total time taken to test all the slices altogether is 0.117 s.

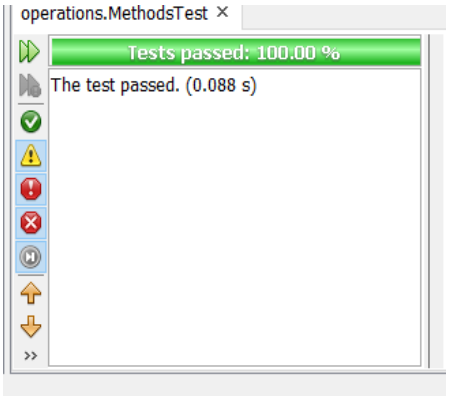


Figure Test result for S1

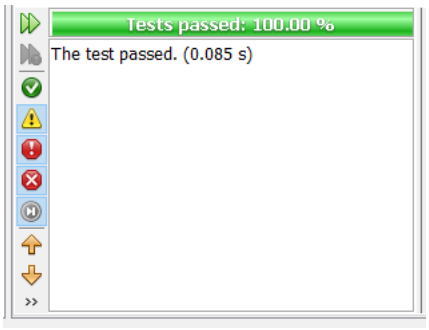


Figure Test result for S2

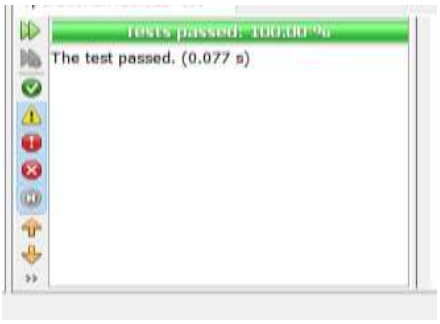


Figure Test result for S3

Figure Test result for S4

Note- These figures may wary as per the system.

V. IMPLEMENTING NEURAL NETWORK CONCEPT

As stated earlier

$$y_{in} = x1.w1 + x2.w2 + x3.w3 \ldots xm.wm$$

i.e., Net input $y_{in} = \sum mixi.wi$

Here, x1,x2,… are the inputs and w1, w2, w3,… are the weights(execution time).

So, for the above program

y_{in}=S1*0.088+S2*0.085+S3*0.077+S4*0.78

VI. CONCLUSION AND FUTURE SCOPE

So, with the above proposed methods we can easily get the picture of an Object Oriented Program. This gives us the capability to derive numerical facts which provide us information about the performance of an Object Oriented Program. Here, we can also see the capability of Slice Testing in reducing the redundancy while creating independent slices.

In future we can show the implementation of the above method with inheritance and other Object Oriented Programming concepts. Further, instead of taking execution time as weight factor we can take lines of code (LOC) or some other factor as weight. Also, the implementation of y_{in} could be used in some other testing technique enhancing the capabilities.

REFERENCES

[1] Durga Prasad Mohapatra, Rajib Mall and Rajeev Kumar1 Department of Computer Science and Engineering Indian Institute of Technology Kharagpur Kharagpur, WB 721 302, India. “An Overview of Slicing Techniques for Object-Oriented Programs”.
[2] Jianjun Zhao, ”Dynamic Slicing of Object Oriented Program”.
[3] N.Sasirekha, A. Edwin Robert and Dr. M.Hemalatha. ”PROGRAM SLICING TECHNIQUES AND ITS APPLICATIONS”, International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3, July 2011.
[4] Sonam Agarwal and Arun Prakash Agarwal. “Program Slicing using Test Cases”, International Journal of Computer Applications (0975 – 8887) Volume 60– No.10, December 2012 .
[5] Iqbaldeep Kaur, Navneet Kaur, Amandeep Ummat, Jaspreet Kaur, Navjot Kaur Dept. of CSE, Chandigarh Engineering College, Landran, Punjab, India. “Research Paper on Object Oriented Software Engineering”, IJCST Vol. 7, ISSUE 4, OCT - DEC 2016.
[6] Zeping Yu and Gongshen Liu, School of Electronic Information and Electrical Engineering Shanghai Jiao Tong University. “Sliced Recurrent Neural Networks”.