Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Implement Unit-Tests for the SpatialPooler AdaptSynapses method

Jiban Kumar Ray
jiban.ray@stud.fra-uas.de

Nishat Shama
nishat.shama@stud.fra-uas.de

***Abstract*—Implementing unit tests in software development is significant because it helps ensure that individual components of the software function correctly and reliably. However, unit testing is often considered to be an arduous task. This study employs the implementation of unit tests for the AdaptSynapses method in the Spatial Pooler (SP) algorithm used in the Hierarchical Temporal Memory (HTM) architecture, which is a computational framework inspired by the structure and function of the neocortex. To ensure the reliability and robustness of the method in different situations, we did a diverse range of test case scenarios to test the provided AdaptSynapses method with changing threshold values and Permanence values. During the task accomplishment, we gained valuable insights into how this method connects mini-columns and sensory input. As part of our efforts, we also analyzed many existing unit tests by introducing a range of inputs with varying complexity. Finally, with the visual studio feature, we did cross-validation of the test cases to evaluate the algorithm's performance. We observed all the test cases successfully covering the complete test AdaptSynapses method in SpatialPooler class. Hence all of them resulted in 100% code coverage. The study effectively motivated us to write unit tests that are more comprehensive and of higher quality.**

Keywords—Spatial Pooler, sparse coding, competitive learning, hierarchical temporal memory, Code Coverage, AdaptSynapses, HtmConfig, Unit Test, testAdaptSynapsesForSP

I. INTRODUCTION

In the world of software development, unit testing is a crucial aspect of building a highly effective and reliable software program. Unit testing, in particular, involves testing individual methods of classes and components of the test suite to ensure they function correctly. To perform unit testing, code coverage tools, a feature in visual studio is used to measure the effectiveness of a test suite in covering the codebase. The code coverage analysis is vital to identify areas of the codebase that are not adequately tested, and further improvement in the code. Here, this project implements unit tests for the AdaptSynapses method, which is a critical component for improving the performance of the Spatial Pooler (SP) algorithm used in machine learning. [1]

The development of the code for this project can be found in the neocortexapi repository. The primary objective of this study is to implement unit tests that will fully cover the provided AdaptSynapses method in the SpatialPooler class. Eventually, it aims to achieve a maximum of 100% code coverage for the AdaptSynapses method, thereby ensuring complete testing of its functionality. The findings from this project will provide valuable insights into the effectiveness of unit testing and the importance of building a robust test suite.

A. *Hierarchical Temporal Memory (HTM)*

Hierarchical Temporal Memory (HTM) is a computational framework for building intelligent systems modeled after the neocortex. It consists of layers of processing nodes arranged hierarchically, with each neuron processing input by connecting to other neurons in a sparse, scattered network.[4] Using time-series data, HTM architecture can carry out pattern recognition, anomaly detection, and prediction tasks. [2] The algorithm's spatial pooler, a crucial part of HTM, uses a sparse distributed representation to learn spatial patterns in incoming data, while the temporal memory learns temporal patterns. HTM offers a strong and adaptable foundation for creating intelligent systems suitable for a variety of applications, including robotics, computer vision, and natural language processing. Fig. 1 is an example of the HTM architecture.

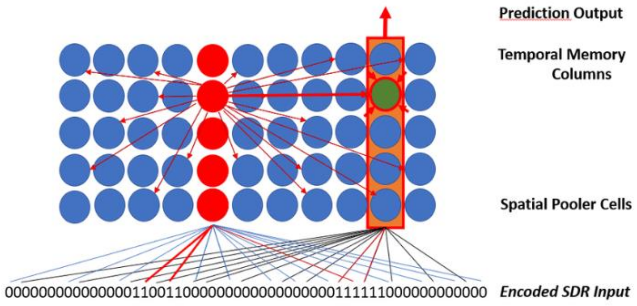

Figure 1 HTM Architecture

B. *Spatial Pooler (SP)*

The Spatial Pooler algorithm is a key component of the Hierarchical Temporal Memory (HTM) architecture, which models the neocortex's structure and function. The algorithm is initialized using an HtmConfig object that contains parameters that govern its behavior. As an unsupervised learning algorithm, the Spatial Pooler identifies patterns in input data and makes predictions based on them. [3] It plays a crucial role in deep learning neural networks by pre-processing input data such as images or speech waveforms to create a sparse representation of the data, which reduces its dimensionality and enhances processing and analysis. The Spatial Pooler uses filters or receptive fields to detect specific features or patterns in input data, and it determines the activation values of neurons

based on how closely input data matches these filters. Below Fig. 2 shows Spatial Pooler Architecture in HTM.

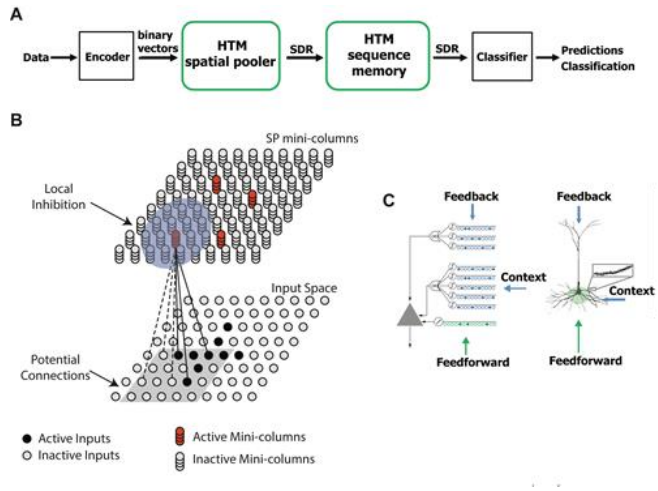

Figure 2 HTM Spatial Pooler Architecture (A) An HTM system. (B) Spatial pooler converts inputs (bottom) to SDRs (top). (C) The SP models the feedforward connections onto the proximal dendrite.

C. *Adapt Synapses*

The AdaptSynapses method in the SpatialPooler class adjusts the synaptic connections between mini-columns and sensory input through a Learning rule, allowing the spatial pooler to adapt and learn from input data. The algorithm involves initializing, processing input, competing, adapting, and repeating. The Fig. 3 explains the process of adapting synapses in a spatial pooler.

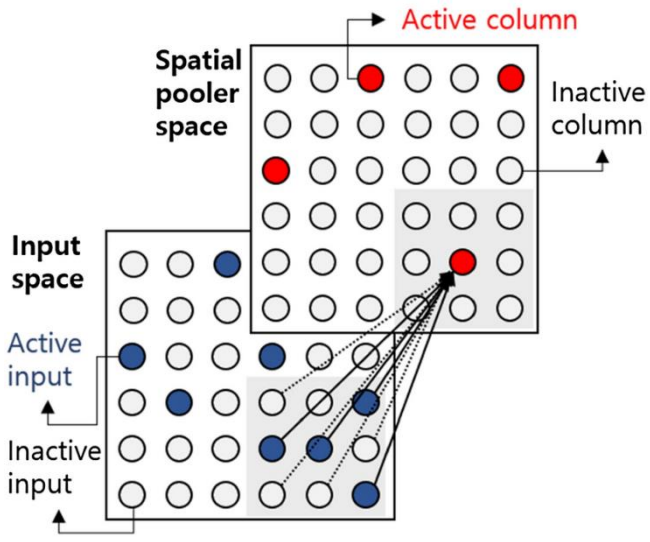

Figure 3 Adapt Synapses Principle

It strengthens active synapses while punishing inactive ones by incrementing or decrementing their permanence values based on activity. The algorithm updates all synapse permanence's to input bits for each mini column, removing or adding synapses as necessary.

II. METHODOLOGY

In this segment, we explain the methodology involved in implementing unit tests for the AdaptSynapses method in the SpatialPooler class and are guided by the discussion above in the introduction. We aimed to test the given algorithm's ability to learn and make predictions under various conditions by conducting sequence experiments using unit tests. We begin by outlining the experimental framework, followed by several unit tests, analyzing the code coverage tests and the outcomes we obtained.

A. *Spatial Pooler Initialization*

The SpatialPooler Initialization is the process of setting up the necessary configurations and parameters for the Spatial Pooler algorithm to start working. It involves defining the input space, number of columns, receptive field size, and the number of active columns that the algorithm should produce for a given input. This initialization process is crucial for the Spatial Pooler to learn and identify patterns in the input data stream.

There are two ways of Initializing the Spatial Pooler's -

- Using Parameters, which is obsolete and mostly compatible with Java implementations.
- Using HtmConfig, which is a configuration object that contains all the required parameters for the algorithm to function effectively.

The HtmConfig initilization we used in all our test cases as shown in below Listing 1. This updated initialization method is recommended over the old one for better performance and reliability of the algorithm.

Listing 1 Code HtmConfigInitialization

```
private HtmConfig HtMConfigInitialization()
{
        HtmConfig htmConfig = new HtmConfig()
        {
           InputDimensions = new int[] { 32, 32 },
           ColumnDimensions = new int[] { 64, 64 },
           CellsPerColumn = 32,
           ActivationThreshold = 10,
           LearningRadius = 10,
           MinThreshold = 9,
           MaxNewSynapseCount = 20,
           MaxSynapsesPerSegment = 225,
           MaxSegmentsPerCell = 225,
           InitialPermanence = 0.21,
           ConnectedPermanence = 0.5,
           PermanenceIncrement = 0.10,
           PermanenceDecrement = 0.10,
           PredictedSegmentDecrement = 0.1,
           Random = new ThreadSafeRandom(42),
           RandomGenSeed = 42
        };

        return htmConfig;
    }
```

B. *Steps involved in Adapt Synapses*

The method adjusts the permanence values of synapses between input bits and columns to improve the accuracy and

efficiency of the algorithm's learning and prediction capabilities. Here are some basic steps involved in the process:

- Make an array of actInputIndexes containing the indices of the active input bits in inputVector.
- Make a new array permChanges with a size equal to the number of inputs in the HTM configuration. All values of permChanges are set to the negative value of SynPermInactiveDec.
- Set the values at the indices in actInputIndexes to SynPermActiveInc.
- Loop over each active column in the model.
- Get the corresponding Column object using the GetColumn method of the Connections class.
- Make an array actInputIndexes containing the indices of the active input bits in inputVector.
- Make a new array permChanges with a size equal to the number of inputs in the HTM configuration. All values of permChanges are set to the negative value of SynPermInactiveDec.
- Set the values at the indices in actInputIndexes to SynPermActiveInc.
- Loop over each active column in the model:
- Get the corresponding Column object using the GetColumn method of the Connections class.

C. *Unit Test Approaches*

While implementing the test cases below aspects of the unit tests were considered as well-designed Unit Tests being fast, repeatable, and highlighting discrepancies with precision.

- Setting the environment: First we implement unit test file and import the necessary classes and methods. Then, made instance of the Spatial Pooler class and set the parameters for the test, such as the input vector, active columns, and HTM configuration.
- Call the AdaptSynapses method: We passed the above instance of the Spatial Pooler class to the AdaptSynapses method along with the required input parameters (i.e., conn, inputVector, and activeColumns).
- Verification: Here we check the permanence values of the synapses in the active columns of the Spatial Pooler by accessing the Column and Pool objects using the GetColumn and GetPool methods. Then assert necessary changes to the synapse permanences.

Details of the unit-test codes are linked in the result segment with respective tests. In Listing 2 below the code for Initializing with HtmConfig are represented for a test case named TestAdaptSynapsesWithSinglePermanences.

Listing 2 Initialization of test case TestAdaptSynapsesWithSinglePermanence

```
var htmConfig = SetupHtmConfigDefaultParameters();
htmConfig.InputDimensions = new int[] { 8 };
htmConfig.ColumnDimensions = new int[] { 4 };
htmConfig.SynPermInactiveDec = 0.01;
htmConfig.SynPermActiveInc = 0.1;
htmConfig.WrapAround = false;
mem = new Connections(htmConfig);
sp = new SpatialPooler();
sp.Init(mem);

mem.HtmConfig.SynPermTrimThreshold = 0.05;

int[][] potentialPools = new int[][] {
new int[]{ 1, 1, 1, 1, 0, 0, 0, 0 }
};
```

For the above mentioned test cases, execution of the AdaptSynapses method with parameters are shown in below at Listing 3.

Listing 3 Execution of test case TestAdaptSynapsesWithSinglePermanence

```
sp.AdaptSynapses(mem, inputVector, activeColumns);

for (int i = 0; i < mem.HtmConfig.NumColumns; i++)
{
    double[] perms =
mem.GetColumn(i).ProximalDendrite.RFPool.GetDense
Permanences(mem.HtmConfig.NumInputs);
   for (int j = 0; j < truePermanences[i].Length; j++)
     {
      Assert.IsTrue(Math.Abs(truePermanences[i][j] -
perms[j]) <= 0.01);
     }
```

III. RESULTS

A. *Unit Test Results*

With reference to the above methodology, 8-unit tests cases for the AdaptSynapses method of SpatialPooler class are implemented. Details of the unit tests code are linked to respective tests Methods.

- TestAdaptSynapsesWithMinThreshold
- TestAdaptSynapsesWithMaxThreshold
- TestAdaptSynapsesWithSinglePermanences
- TestAdaptSynapsesWithTwoPermanences
- TestAdaptSynapsesWithThreePermanences
- TestAdaptSynapsesWithFourPermanences
- TestAdaptSynapsesWithNoColumns
- TestAdaptSynapsesWithNoColumnsNoInputVector

We run all the above listed unit tests in visual studio test explorer and found all of them cover the complete method

AdaptSynapses of SpatialPooler.cs class. Below Fig 4 represents that all the test case are passed.

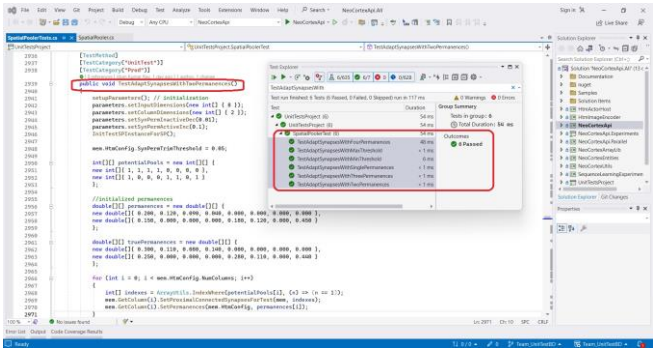

Figure 4 Execution of test results

B. *Code Coverage Analysis*

We analyzed each of the test cases with visual studio code coverage feature and we found that our new test case coverage is 100%. Into the adaptSynapses method, there are fewer conditions to avoid checking (if, else, or for-loop) which makes to achieve a maximum code coverage. However, the last two-unit test methods achieve a lower code coverage which are between 59% to 70%. Table I below summarized all unit tests code coverage results:

TABLE I. CODE COVERAGE ANALYSIS

Test Cases	*Covered (%Blocks)*	*Not Covered (%Blocks)*
TestAdaptSynapsesWithMinThreshold	100%	0%
TestAdaptSynapsesWithMaxThreshold	100%	0%
TestAdaptSynapsesWithSinglePermanences	100%	0%
TestAdaptSynapsesWithTwoPermanences	100%	0%
TestAdaptSynapsesWithThreePermanences	100%	0%
TestAdaptSynapsesWithFourPermanences	100%	0%
TestAdaptSynapsesWithNoColumns	59.26%	40.74%
TestAdaptSynapsesWithNoColumnsNoInput Vector	69.76%	30.24%

Code Coverage Results for the unit test TestAdaptSynapsesWithMinThreshold shown in Fig. 5 below. Since the unit test fully covers the provided AdaptSynapses method so the lines displayed in light blue color. In contrast, if code does not cover the complete test case might be highlighted in red.

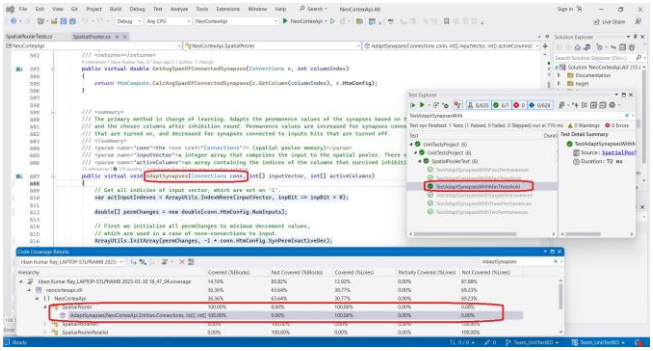

Figure 5 Code Coverage Analysis of test case TestAdaptSynapsesWithMinThreshold

Similarly, the following Fig. 6 represents code coverage result of unit test TestAdaptSynapsesWithTwoPermanences.

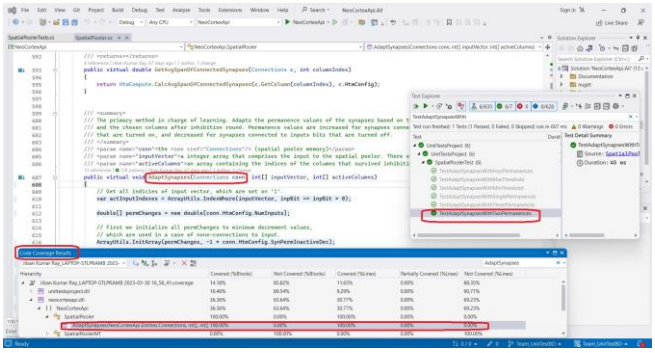

Figure 6 Code Coverage Analysis of test case TestAdaptSynapsesWithTwoPermanences

C. *Future Scope*

Code The implementation of unit tests for the AdaptSynapses method is an important step towards ensuring the accuracy and reliability of the Spatial Pooler algorithm. Unit tests can ensure that the algorithm functions correctly and meets the desired specifications.

There are several potential areas of future scopes. All test cases are not fully covered, those can be analyzed and improve their code coverage through unit testing to achieve maximum code coverage.

Another area for future improvement is to implement integration tests that test the Spatial Pooler algorithm as a whole, rather than just the AdaptSynapses method. Integration tests can help ensure that the different components of the algorithm work together correctly and that the algorithm as a whole meets the desired specifications.

IV. CONCLUSION

In addition to the findings mentioned above, the study also highlights the importance of unit testing in software development, particularly in the context of machine learning and synaptic learning algorithms. Unit testing not only ensures the reliability and robustness of the code but also helps in identifying and fixing any errors or bugs early in the development cycle. The study also emphasizes the significance of code coverage, as achieving high code coverage helps in identifying any gaps or untested scenarios in the code.

Moreover, the AdaptSynapses method's effectiveness in sparse or noisy input scenarios indicates its potential for application in a wide range of real-world problems, such as natural language processing and image recognition. As such, the study's results can be useful for researchers and practitioners in the field of machine learning and synaptic learning, providing them with valuable insights and information to improve their algorithms' performance and functionality.

To further enhance the algorithm's performance and functionality, future research could focus on implementing integration tests for the algorithm as a whole. Integration tests can help in identifying any issues or errors that arise when different components of the algorithm interact with each other. Additionally, analyzing and covering the remaining test cases can further improve the algorithm's reliability and robustness, ensuring its effectiveness in a wider range of scenarios. Overall, the study's findings have significant implications for software development, machine learning, and synaptic learning research, highlighting the importance of rigorous testing practices and the potential of the AdaptSynapses method for solving real-world problems.

ACKNOWLEDGMENT

The members of this project work acknowledge no conflicts of interest regarding the publication of this paper. We thank Damir Dobric and Prof. Dr Andreas Pech for allowing us the opportunity to work on this topic. Through this project we made a strong foundation in software engineering, and we would be delighted to work in this field of software testing and unit testing in the future whenever have any scope.

REFERENCES

[1] Y. Cui, S. Ahmed, and J. Hawkins, “The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding,” Front. Comput. Neurosci. November 2017, vol. 11-2017, doi: 10.3389/fncom.2017.00111.

[2] Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. Neurocomputing 262, 134–147. doi: 10.1016/j.neucom.2017.04.070.

[3] Mnatzaganian, J., Fokoué, E., and Kudithipudi, D. (2017). A mathematical formalization of hierarchical temporal memory's spatial pooler. Front. Robot. AI 3:81. doi: 10.3389/frobt.2016.00081.

[4] K. Mountcastle, V. B. (1997). The columnar organization of the neocortex. Brain 120, 701–722. doi: 10.1093/brain/120.4.701.

[5] Olshausen, B. A., and Field, D. J. (2004). Sparse coding of sensory inputs. Curr. Opin. Neurobiol. 14, 481–487. doi: 10.1016/j.conb.2004.07.007.

[6] Y. Cui, S. Ahmed, and J. Hawkins, “Continuous online sequence learning with an unsupervised neural network model,” Neural Computation, Volume 28, Issue 11, November 2016, pp 2474–2504, doi: 10.1162/NECO_a_00893