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

Improve Unit Tests (Spatial Pooler & Temporal Memory)

Ali Haider
ali.haider@stud.fra-uas.de

Naveed Ahmad
naveed.ahmad@stud.fra-uas.de

Ali Raza Kharl
ali.kharl@stud.fra-uas.de

***Abstract*—Hierarchical temporal memory is the technique that works on the principles of neo cortex. Spatial Pooler and Temporal Memory are the important components of HTM. It transforms the binary input patterns into the sparse distribution representation which then acts as an input to temporal memory algorithm to make predictions. Temporal Memory learns sequences of Sparse Distributed Representations (SDRs) formed by the Spatial Pooling algorithm and makes predictions of what the next input SDR will be. As unit tests run to maintain code health, ensure code coverage, and find errors and faults before your customers do. Run your unit tests frequently to make sure your code is working properly.**

Keywords—Neo Cortex, Spatial Pooler, Temporal Memory, Sparse Distribution Representations, Unit Test.

I. Introduction

Sparse Distribution Representation (SDRs) are represented with thousands of bits. Small percentage of bits are 1's and rest are 0's. The bits correspond to neurons in neo cortex. The important component of HTM is the use of SDR. Every bit of an SDR corresponds to a meaning. The set of active bits represent the semantic meaning of the SDR. Overlap of the SDRs can be determined by the number of active bits in the same location. SDRs are generalized structure which can represent any type of information in them. Although every bit has a meaning, the structure of SDR is such that we only must store certain percentage of active bits in order to represent the SDR. [1] Spatial Pooler (SP) may be a learning algorithm that's outlined to imitate the neuron's usefulness of the human brain. Basically, if a brain sees one thing at different times, it is aiming to reinforce the neural connections that respond to the input result within the recognition of the object. Additionally, if several comparable SDRs are displayed to the SP algorithm, it'll strengthen the columns that are dynamic according to the on bits within the SDRs. In case the number of training cycles is huge enough, the SP will be able to recognize the objects by creating a distinctive set of active columns inside the required measure of SDR for distinctive objects. [2]

The Temporal Memory algorithm does two things: it learns sequences of Sparse Distributed Representations (SDRs) formed by the Spatial Pooling algorithm, and it makes predictions. In the Temporal Memory algorithm, when a cell becomes active, it forms connections to other cells that were active fair earlier. Cells can predict when they will become active by looking at their connections. If all the cells do this, collectively they can store and review sequences, and they can predict what is likely to happen another. [3]

When unit testing is an intrinsic part of your code, it has the biggest impact on the quality of your code. Create unit tests that evaluate the behavior of the code in response to standard, boundary, and incorrect cases of input data, as well as any explicit or implicit assumptions made by the code as soon as you write a function or other block of application code. With test driven development, you create the unit tests before you write the code, so you use the unit tests as both design documentation and functional specifications.

II. Methods

1) HTM

HTM regions also uses sparse distributed representations. An HTM region consists of columns which comprises of cells. Each column is connected to a unique subset of input bits, and different input bits provide different result for activation of the columns. Columns with the strongest activation inhibit(deactivate) columns with weaker activation. The result is a sparse distributed representation of the input.[1]

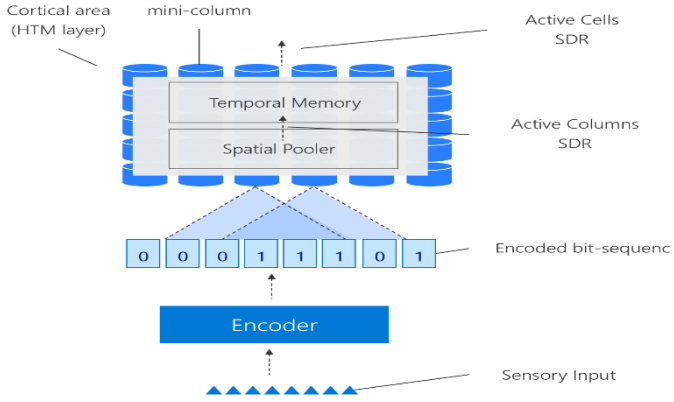

Figure 1: HTM Architecture [4]

2 Spatial Pooler

The spatial pooler learns feed-forward connections between input bits and columns. It acknowledges an Info or a vector of one size and changes into an output vector of different size with a sparse number of activated bits. It keeps a decent sparsity and cross-over properties. The most principal capacity of the spatial pooler is to change over a local input to a scanty pattern.

Functionality of Spatial Pooler:

1) Begin with a set number of bits for your input. These bits could be sensory data, or they could come from a different part of the HTM system.
2) Each column has an associated dendritic segment, serving as the connection to the input space. Each dendrite segment has a set of potential synapses representing a (random) subset of the input bits.
3) Each potential synapse has a permanence value. These values are randomly initialized around the permanence threshold. Based on their permanence values, some of the potential synapses will already be connected; the permanences are greater than the threshold value. For any given input, determine how many connected synapses on each column are connected to active (ON) input bits. These are active synapses. Spatial Pooling Algorithm has many parameters that affect the dimensionality, learning mechanisms, and overall performance.[1]

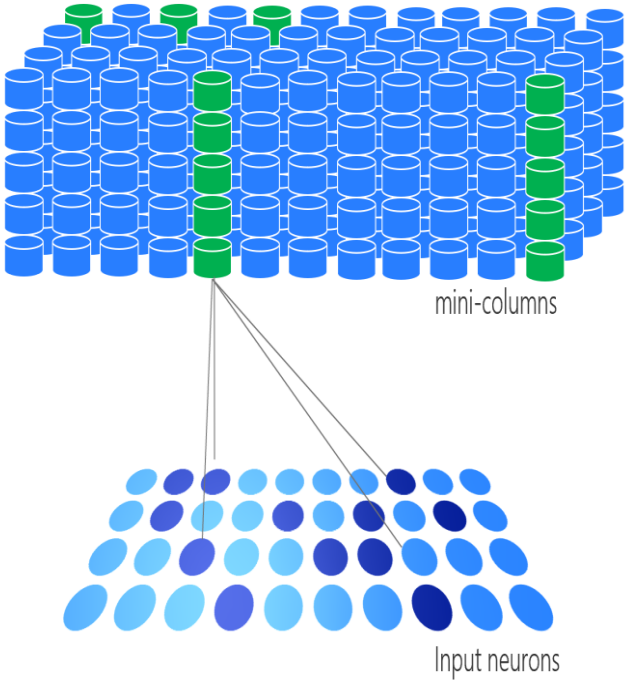

Figure 2: Spatial Pooler Neurons as input [4]

3. **Temporal Memory**

Temporal Memory algorithm predicts what the next input SDR will be based on sequences of Sparse Distributed Representations (SDRs) produced by the Spatial Pooling algorithm. Each column in the SDR consists of many cells. Each cell can have three states: active, predictive, and inactive. These cells should have one proximal segment and many distal dendrite segments. The proximal segment is the connection of the column and several bits in the input space. The distal dendrite segments represent the connection of the cell to nearby cells.

The Temporal Memory algorithm starts where the Spatial Pooling algorithm leaves off, with a set of active columns representing the feed-forward input. In the pseudocode below, a time step consists of the following computations:

1. Receive a set of active columns, evaluate them against predictions, and choose a set of active cells:
    a. For each active column, check for cells in the column that have an active distal dendrite segment (i.e., cells that are in the "predictive state" from the previous time step), and activate them. If no cells have active segments, activate all the cells in the column, marking this column as "bursting". The resulting set of active cells is the representation of the input in the context of prior input.
    b. For each active column, learn about at least one distal segment. For every bursting column, choose a segment that had some active synapses at any permanence level. If there is no such segment, grow a new segment on the cell with the fewest segments, breaking ties randomly. On each of these learning segments, increase the permanence on every active synapse, decrease the permanence on every inactive synapse, and grow new synapses to cells that were previously active.
2. Activate a set of dendrite segments: for every dendrite segment on every cell in the layer, count how many connected synapses correspond to currently active cells (computed in step 1). If the number exceeds a threshold, that dendrite segment is marked as active. The collection of cells with active distal dendrite segments will be the predicted cells in the next time step.[1]

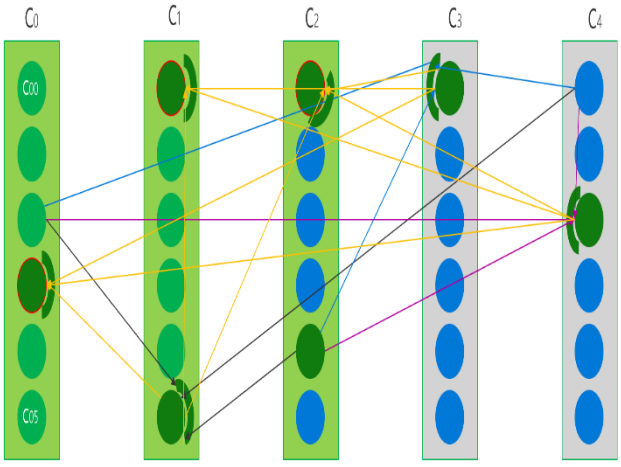

Figure 3: Temporal Memory [4]

4 Implementation of Unit Tests

To implement unit tests of spatial pooler we need to know how compute and Init methods are used to create unit test methods.

Initially, we have created a new unit test project as a console application and got references from already neo-cortex projects. After creation, we have added two public methods to configure spatial pooler and temporal memory parameters were specifically selected as the table illustrate:

Parameters	Values
PotentialRadius	5
PotentialPct	0.5
GlobalInhibition	false
LocalAreaDensity	-1.0
NumActiveColumnsPerInhArea	3.0
StimulusThreshold	0.0
SynPermInactiveDec	0.01
SynPermActiveInc	0.1
SynPermConnected	0.1
MinPctOverlapDutyCycles	0.1
MinPctActiveDutyCycles	0.1
DutyCyclePeriod	10
MaxBoost	10
RandomGenSeed	42

Table 1: Spatial Pooler and Temporal Memory Parameters used in experiments.

The Test File name for Spatial pooler Test isn**SpatialPoolerTestNEWBYAliRazaKharl.**cs which contains four Unit Tests **confirmSPConstruction2(), testCompute2_1(), TestZeroOverlap_NoStimulusThreshold_GlobalInhibition1_1()** and **testZeroOverlap_StimulusThreshold_GlobalInhibition1(PoolerMode poolerMode)** and Test File name for temporal memory Test are **TemporalMemoryTestNEWByAliHaider.cs** and **TemporalMemoryTestNEWByNaveedAhmed.cs** both files contain 12 unit test methods some of them are **TestActivatedunpredictedActiveColumn()**, **TestWithTwoActiveColumns()** and **TestAdaptSegmentToCentre()** as depicted in Figure 5. For all the tests we declare some default parameters for Connection and applied the same by calling GetDefaultParams() method.

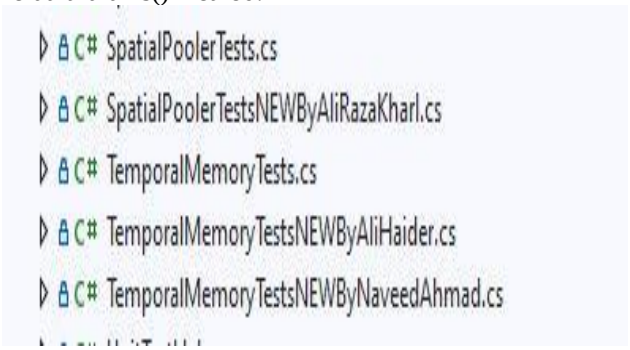

Figure 5: Location of File

These parameters are used in two ways, first one is to create a parameter class method. Another way to create HTMConfig class method assign above mention values accordingly.

```
Public void setupParameters()
{
parameters = Parameters.getAllDefaultParameters(); ///parameter class object
parameters.Set(KEY.INPUT_DIMENSIONS, new int[] { 5 });
parameters.Set(KEY.COLUMN_DIMENSIONS, new int[] { 5 });
parameters.Set(KEY.POTENTIAL_RADIUS, 5); /// potentialRadius must be set to the inputWidth
/// other parameters
}

private HtmConfig SetupHtmConfigDefaultParameters()
      {
         var htmConfig = new HtmConfig(new int[] { 32, 32 }, new int[] { 64, 64 })
         {
            PotentialRadius = 16,
            PotentialPct = 0.5,
            GlobalInhibition = false,
            LocalAreaDensity = -1.0,

            ///Other parameters

         };

         return htmConfig;
      }
```

After Created private method in each unit tests we must create object of that class for example to create a temporal memory unit test these objects implemented.

Similar to Spatial Pooler, Temporal memory has a same method to calculate the output SDR: tm.Compute(int[] activeColumns, bool learn). This method takes the result of active columns in the stable output SDR from the Spatial Pooler (training is off) with learning option to produce a ComputeCycle object which holds the information of winner cells and predictive cells.

```
Public ComputeCycle Compute(int[] activeColumns, bool learn)
```

In spatial pooler unit test class, we have improved four tests by changes some values of parameters used in test. Like in **confirmSPConstruction2()** test we have confirmed spatial pooler construction by passing some new values to its parameters and the test pass.

With learning is turned on, if the predictive cells become active in the next input, the permanence values of its active synapses will be incremented, and its inactive ones will be diminished. However, if the system predicts some wrong cells, the cells' active synapses will be punished as their synaptic permanence is reduced. Without learning, there are no change in the synapses of the segments.

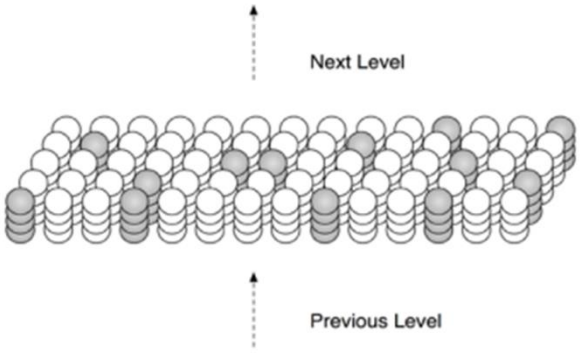

Figure 4: Bursting Columns

There are three functions used in temporal memory class where the random function is used; however, we just created a unit test of bursting column. It activates all the cells in an unpredicted active column, chooses a winner cell, and if learning is turned on, either adapts or creates a segment. grow Synapses is invoked in this segment.

```
public BurstingResult BurstColumn(Connections conn,
Column column, List<DistalDendrite>
matchingSegments,
        ICollection<Cell> prevActiveCells,
ICollection<Cell> prevWinnerCells, double
permanenceIncrement, double permanenceDecrement,
          Random random, bool learn)
```

In Temporal memory, we have implemented 12-unit test cases. In **TestActivatedunpredictedActiveColumn()** test we have implemented a function which uses RANDOM function.
In the Unit test, we get the best cell from the parent Index column and count the distal dendrite from the best cell by using random values in it. In **TestRandomMostUsedCell()** test we have got the most used cell in a column. In this test whenever we run the test it predicts the new values, and it lies between three values. And these expected values vary depending upon how much we choose the column dimension. In **TestArrayNotContainingCells()** test we have implemented a test that checks an array that does not contain any active cell in it. For that, we used **compute cycle method** from temporal memory class. compute cycle holds all important states calculated during a TM computational cycle. In **testNumberOfColumns()** test we check how many columns in a column dimension. And did some other tests as well check how many active cells, winner cells and predictive cells were from two active columns.

III. Results

All test methods in all three files run successfully as shown in figure.

Figure 5: Spatial Pooler Test result

Figure6: Temporal MemoryNEWBYAliHaider test Run

Every time run **TestRandomMostUsedCell()** it predicts a new value as shows in figure 7.

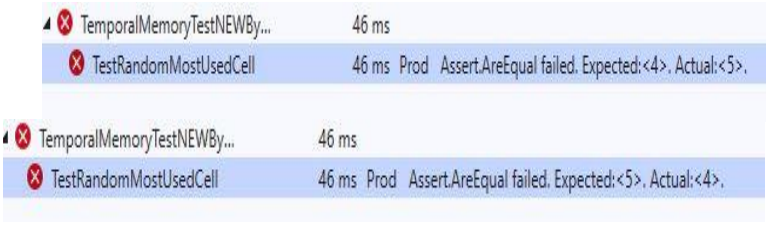

Figure 7: TestRandomMostUsedCell Result

In file **TemporalMemoryTestNEWByAliHaider.cs** the output of first test **TestActivatedunpredictedActiveColumn()**. As it uses random function, so the outcome are best cell and count parent column index from best cells as shows

```
    [TestMethod]
      [TestCategory("Prod")]
      public void TestActivatedunpredictedActiveColumn()
      {
          HtmConfig htmConfig = GetDefaultTMParameters();
          Connections cn = new Connections(htmConfig);
          TemporalMemory tm = new TemporalMemory();
          tm.Init(cn); ///use connection for specified object to build to
implement algoarithm
          Random random = cn.HtmConfig.Random;
          int[] prevActiveColumns = { 1, 2, 3, 4 }; ///
          Column column = cn.GetColumn(6); /// Retrieve column 6
          IList<Cell> preActiveCells = cn.GetCellSet(new int[] { 0, 1, 2,
3 }); /// 4 pre-active cells
          IList<Cell> preWinnerCells = cn.GetCellSet(new int[] { 0, 1 });
///Pre- winners cells from pre avtive once
          List<DistalDendrite> matchingsegments = new
List<DistalDendrite>(cn.GetCell(3).DistalDendrites); ///Matching
segment from Distal dentrite list
          var BustingResult = tm.BurstColumn(cn, column,
matchingsegments,
                        preActiveCells, preWinnerCells, 0.10, 0.10,
                                  new ThreadSafeRandom(100), true);
          // Assert.AreEqual(, BustingResult);
          Assert.AreEqual(6,
BustingResult.BestCell.ParentColumnIndex);
          Assert.AreEqual(1,
BustingResult.BestCell.DistalDendrites.Count());
      }
```

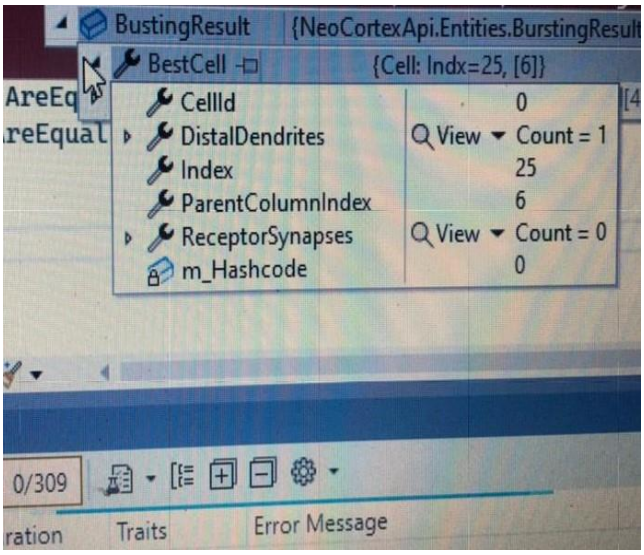

Figure 8: Best Cell output

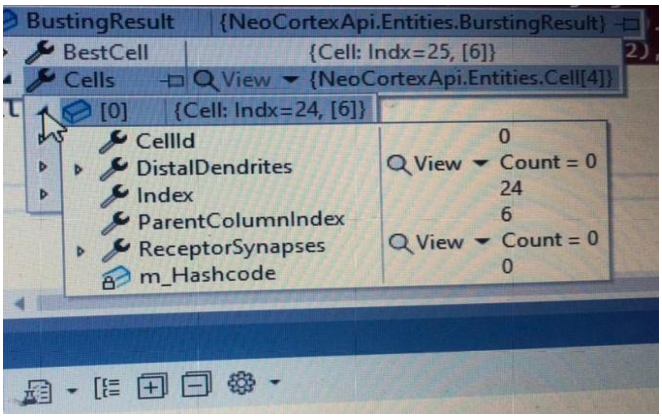

Figure 9: Parent Column Index

Figure 8 and 9 shows the output as random values set to 100.

In **TestNumberofColumn()** test as output show

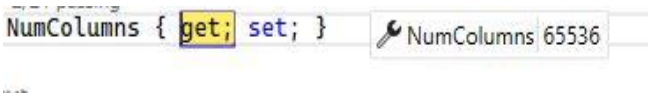

Figure 10: Number of Columns

In above test results could be performed as per expectation. After every time change the value it passes the test at a certain iteration. Not every test needs to have a pass all the time. I can be failed that doesn't mean the code has any error as figure 7 depict that every run we run the test gives a new expected value due to it using a random function.

IV. Discussion

Using Spatial Pooler and Temporal Memory, this paper proposes a machine learning solution for Unit Tests. Machine intelligence has recently evolved a new paradigm in the form of spatial pooler and temporal memory theory, which represents the structural and algorithmic components of the neocortex.

Spatial Pooler and temporal memory are novel techniques in the field of machine intelligence that was developed. In this research, I aim to improve the performance of the spatial pooler and temporal memory by using public classes as a basis for their development. we proposed five-unit tests in spatial pooler and 13-unit tests in temporal memory, which we validate and benchmark against existing ones. Using datasets from the spatial pooler and temporal memory repository. Algorithm's inference of patterns and structures that the system recognizes still needs a lot of development.

The results demonstrate that the proposed unit tests can improve the performance of the spatial pooler and temporal memory and their performance is equivalent to that of other standard machine learning techniques such as the decision tree, among others. The random function cannot be handled easily but can be managed at a small real because units test can handle it easily My future research will focus on further enhancing the algorithm design and investigating the application of the spatial pooler and temporal memory for mathematical issues, among other things. In the realm of machine learning, Deep Learning is a vital component of the use of Artificial Intelligence. The effectiveness of these systems is determined by their behavior and performance

V. References

[1] J. H. and S. Ahmad, "How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites," ArXiv arXiv:1601.00720 [q-NC], 2016.

[2] . J. Hawkins, . S. Ahmad and D. Dubinsky, *Cortical learning algorithm and hierarchical temporal memory*, p. 1–68.

[3] S. Ahmad, M. Lewis and C. Lai, *Temporal Memory Algorithm*, pp. 1-12.

[4] [Online]. Available: https://github.com/ddobric/neocortexapi/tree/master/docs/images.