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

Investigation of Sequence Learning of SP/TM Layer

Ghulam Mustafa
ghulam.mustafa@stud.fra-usa.de

Abdul Samad
abdul.samad@stud.fra-uas.de

Diab elmehdi
el.diab@stud.fra-uas.de

Muhammad Mubashir Ali Khan
muhammad.khan2@stud.fra-uas.de

Treesa Maria Thomas
treesa.thomas@stud.fra-uas.de

Abstract—. Hierarchical Temporal Memory **(**HTM) is based on the supposition that the world has a structure and is therefore predictable. The development of HTM for Artificial Neural Networks has led to an advancement in the field of artificial intelligence and leading the computing intelligence to a new age. In this paper, we studied various learning parameters like *Width(W), Input Bits(N), Max,Min* values and the *number of columns*, that majorly contribute to optimize the sequence learning behavior of spatial pooler and temporal memory layer. We also performed experiment to obtain stability of Spatial Pooler output by tuning the *boost* and *duty cycles*. We evaluated each of these parameters based on the theoretical and practical framework and summarized the results in graphical diagrams.

Keywords— Temporal memory, sequence learning, Spatial pooler, stability

I. Introduction

Hierarchical temporal memory (HTM) is a neuromorphic machine learning algorithm which resembles neocortex functions in the human brain. The HTM architecture comprises a spatial pooler (SP) and temporal memory (TM) with the sparsely, modular and hierarchically characteristical components. The SP sparsely distributes the input data while the TM is in charge of the learning process. The imitation of human brain functionality allows HTM an adaptive core algorithm for various operations such as categorization and data classification of objects. Therefore, the temporal memory is the substratum for all of the neocortical functions. The discrepancy between HTM theory and most other theories of artificial neural grids is possibly the biggest.HTM starts whit the presumption of the memory and the confirmation of sequence of patterns that all the neocortex does. [1]

The main objective of this paper is to analyze various HTM parameters namely cells per column, column dimensions, input bits (N), Width (W), Range (Max and Min values), and Boost by working on simple sequence and complex sequence of inputs and observe how they influence the learning process. The rest of the paper is structured as follows; Section 2 covers HTM Overview. Section 3 explains the working methodology of the whole experiment. Experimental results and conclusion are presented in section 4 and 5 respectively.

II. HTM OVERVIEW

The HTM network is made of regions that are arranged hierarchically as shown in the figure 1. Each of these regions has neurons known as cells. These cells are arranged vertically forming a column such that it responds to one single specific input at a time. These cells can be considered as major units of HTM. These cells also have dendrites, both distant and proximal allowing them to connect with input spaces and neighboring cells in that particular area.

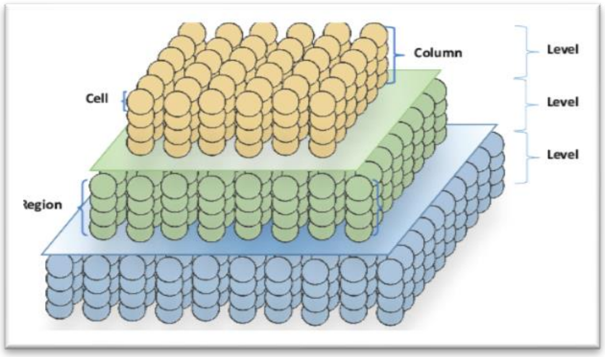


Figure 1: A representation of three layers of HTM arranged hierarchically. Each region contains columns of cells arranged vertically.

A. *Encoder*

The data which we have from the source is encoded here semantically as a sparse distributed representation (SDR). These arrays then go through a spatial pooler making them all into a fixed size. This is done by Hebbian learning, with activating the inactive cells. One thing we need to make sure is to obtain same SDR'S from same inputs. The SDR representation can be seen below: ()

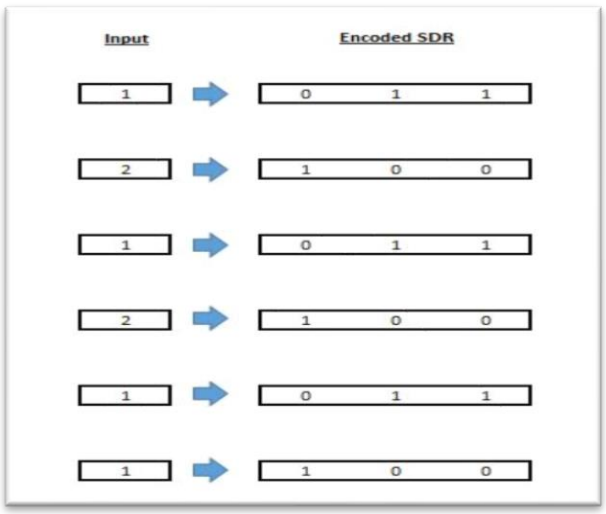


Figure 2: A representation of semantically encoded bits. In this project we have used a scalar encoder that encodes a numeric value to an array of bits. This is necessary to map our data into SDR to be fed into Spatial Pooler as a scalar input.

B. *Spatial Pooler*

The spatial pooler model encodes the binary input data into sparse distributed representations (SDR). The SDR does the job of learning, distinguishing the common traits and making predictions. With the use of a proximal segment, consisting of many synapses, each column is linked to a specific component of the input space. If enough number of active synapses in the inputs is connected to active bits, then we can say that the proximal dendritic segment is being activated and this result in representing the input with its surrounding column. And finally, the column with the most active inputs and synapses inhibits its neighbors and is active (winner). The spatial pooler process can be explained in three stages namely initialization, overlap, inhibition, and learning. [1]

Initialization: All the regional parameters, including column connections to the input space are initialized in this step. Now consider the receptive field (Aj) determines a spot in the input space where the synapses of the jth column are connected. (Aj). Connections of the jth column,

$\vec{b}_j \in R^{n}_{\{0,1\}}$, can be identified to the input space (1).

$$\vec{b}_j = I\,(n_s, j \mid j \mp \varepsilon \; \exists \, Aj\,(r)) \qquad (1)$$

Here I is an indicator function which returns a vector either '0' or '1' in which '1' indicates an input synapse. ns and ε number of potential jth column synapses. Dj is either a j-centered local receptive field with a radius r.

Overlap and Inhibition: In this section the winning columns will be selected (which reflect the input). This can happen only after each column's overlap (α) is being calculated. The jth column overlap is determined by counting its active synapses in the input space which interact with the active bits. The permanence of the potential synapses ρ ⃗j of the jth column is initialized with random values close to the permanence threshold (Pth). Mathematically, this can be achieved by dot product, as in (2).

$$\alpha j = (c\,\vec{}j \odot \rho\,\vec{}j *).x\,\vec{} \qquad (2)$$

$$\rho\,\vec{}j * = I\,(\rho\,\vec{}j \geq Pth) \qquad (3)$$

Learning: Once after the winning columns are decided, the learning process starts updating the permanence values of synapses of the columns as appropriate, but only the synapses of the active columns are modified. [2]

C. *Temporal Memory Module*

Temporal memory model cycle involves the cells of the winning columns. The active cells of the winning columns produce lateral synaptic connections with the active preceding cells. This helps to predict the active state by simply analyzing the distal segments.

Each cell is connected to other cell via distal segments, which has the function of keeping the status of each cell (inactive, predictive, or an active). One cell per active column is selected to be active and to learn the pattern of input. If the input is not expected, i.e. there are no cells in the predictive state, then all the active column cells must burst to be in the active state. In addition, the cell with the greatest number of active synapses, known as the best matching cell, is chosen as a learning cell for learning the sequence of inputs. [3]

III. METHODOLOGY

Following are the parameters on which we have worked to observe how they affect the learning of input sequence in SP/TM layer.

- ***Width***

The number of bits that are set to encode a single value. The "width" of the output signal restriction: w must be odd to avoid centering problems. It is a parameter for encoder. The symbol for width is W.

- ***Input Bits***

The number of bits in the output. Must be greater than or equal to ``w``. It is a parameter for encoder. The symbol for Input Bits is N.

- ***Htm Sparsity***

HTM Sparsity is the ratio of Width (W) and Input Bits (N). It is the combination of W and N, so it is also a parameter for encoder. It is a parameter for encoder.

- ***Max and Min values***

An encoder parameter, which defines the input range. The input sequence must be within that range. Range should be small just related to the input to avoid overlap between the two input values.

- ***Mini Columns***

The output of the encoder is fed to spatial pooler to activate certain mini columns. This add semantic meaning to the input. In the code this parameter is named as column dimensions and this is set at the start of the experiment.

- ***Cells in Columns***

Cells in each column is a TM layer parameter. We need to define the number of cells in each mini column at the start of the experiment. In the code this parameter is named as cells_per_column.

- ***Boost***

A parameter which makes the columns active which are in inhibition of radius. After boosting these becomes active. Boosting is done to provide stability at the SP output. In the code it is written as Max_Boost.

- ***Duty Cycle***

A time duration cycle which notice that which column is active for which time after inhibition. This is also called Active duty cycle which call as a function. In the code we are setting Duty_Cycle_Period to vary this parameter.

By varying these parameters, we will find out how they are affecting the learning process, upon changing these parameters whether the cycles required by the algorithm to learn the input sequence increases of decreases. Learning refers to the 100% prediction of the next input sequence for a particular cycle. Elaboration of how results are observed can be found in results section of this paper.

Case 1. Effect of Width on Learning

This experiment is performed by changing value of width and keeping all other values constant. The values of different parameters are:

```
int inputBits = 50;
{ "W", 9},
//W=1,3,5,7,9,11,13,15,19,29,39,49
{ "N", inputBits},
{ "Radius", -1.0},
{ "MinVal", 0.0},
{ "Periodic", false},
{ "Name", "scalar"},
{ "ClipInput", false},
double max = 10;
```

The experiment is performed using a test named “WidthExperiment”. For each value of Width, the experiment is running for more than 1000 times to obtain a more accurate result.

Case 2. Effect of Input Bits on Learning

This experiment is performed by changing value of width and keeping all other values constant. The values of different parameters are:

```
int inputBits = 50;
{ "W", 9},
{ "N", inputBits},
//N=10,25,50,75,100,150,200,250,300,400,500
{ "Radius", -1.0},
{ "MinVal", 0.0},
{ "Periodic", false},
{ "Name", "scalar"},
{ "ClipInput", false},
double max = 10;
```

The experiment is performed using a test named “InputBitsExperiment”. For each value of Input Bits, the experiment is running for more than 1000 times to obtain a more accurate result.

Case 3. Effect of Max and Min Values on Learning

The aim of this experiment is to study the dependence of learning an input sequence on Range (Max and Min Values).

For this experiment we have kept all the other parameters like input bits (N), Width(W), Cells per Column constant and varying Max values to observe how it effect the learning process. Following are the values set for the parameters:

```
int inputBits = 200;
{ "W", 11}
{ "N", inputBits},
{ "Radius", -1.0},
{ "MinVal", 0.0},
{ "Periodic", false},
{ "Name", "scalar"},
{ "ClipInput", false},
p.Set(KEY.CELLS_PER_COLUMN, 5);
p.Set(KEY.COLUMN_DIMENSIONS, new int[] { 500 });
```

“MaxMinValueExperiment” is used to perform this experiment. There are two input sequences. One is [9 8 7 6 5 4 3 2 1 0] and the other one is [0 10 20 30 40 50 60 70 80 90. For first sequence, we take Max=10 and then changing the values to 15, 20, 25, 30,40, 50 and 100 to see how much cycles our program take to learn the given input sequence. Less cycles it takes means better efficiency. For every Max value we took 30 experiment repetitions and average it to get the final value.

Case 4. Effect of different parameters on learning of a Complex Sequence

Complex sequence is different to dealt with as compared to simple sequence. In the previous cases we have observed the effects of various parameters on simple sequence of numbers. In this case, we have taken a complex sequence **[9 8 7 6 5 9 8 7 6 4]** and observing how the parameters like Cells per column, Width and number of mini columns effect the learning process. We set the parameters as:

```
int inputBits = 400;
{ "W", 15}
{ "N", inputBits},
{ "Radius", -1.0},
{ "MinVal", 0.0},
{ "MaxVal", 10.0},
{ "Periodic", false},
{ "Name", "scalar"},
{ "ClipInput", false},
p.Set(KEY.COLUMN_DIMENSIONS, new int[] { 500});
```

This experiment is performed using “ComplexSequenceExperiment”. First, we have changed the values of cells per column p.Set(KEY.CELLS_PER_COLUMN, C); as 2, 3, 4, 5, 6, 7, 10, 15, 20 and note how much cycles do the algorithm take to learn this complex sequence.

Second, we keep all the other parameters constant and vary the number of active bits (N) between 100 and 800 and observe the results.

Also, we have evaluated the results by changing the parameter p.Set(KEY.COLUMN_DIMENSIONS, new int[]{ }); and set the values 200, 300, 500, 1024, 1500 and 2000, keeping all the other parameters constant.

Case 5. Effect of boost on stability

This experiment is performed by changing the value of boost and duty cycle while keeping all the parameters same. These parameters are,

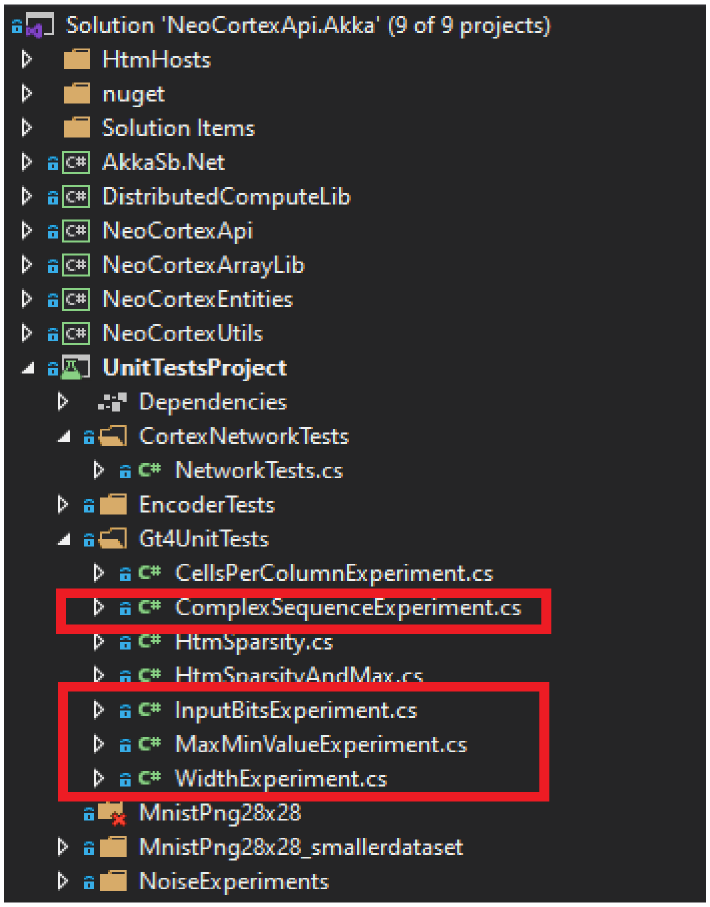
```
p.Set(KEY.MAX_BOOST, 1.0);
p.Set(KEY.DUTY_CYCLE_PERIOD, 100000);
p.Set(KEY.IS_BUMPUP_WEAKCOLUMNS_DISABLED, true);

p.Set(KEY.MAX_BOOST, 5.0);
p.Set(KEY.DUTY_CYCLE_PERIOD, 8000000);
p.Set(KEY.IS_BUMPUP_WEAKCOLUMNS_DISABLED, true);
```

This experiment is performed using a unit test named "SpatialPooler_Stability_Experiment.cs" For each boost value from 10 to 1. This experiment ran for more than 20000 duty cycles to get some accurate and stable results.

IV. C# CODE

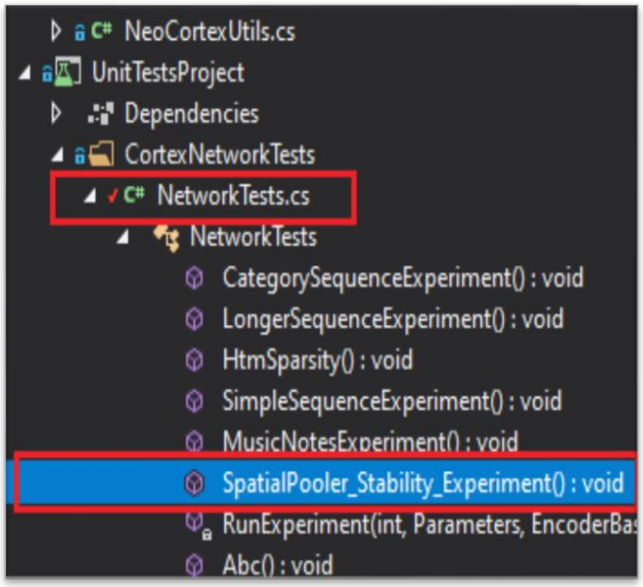
The unit tests used for these experiments can be found under the solution "NeoCortexapi.Akka" in HTM folder, in "UnitTestsProject" there is a folder name Gt4UnitTests which has all the Unit Tests we worked on in these experiments. The Unit Tests are highlighted in figure below.



```
Solution 'NeoCortexApi.Akka' (9 of 9 projects)
  HtmHosts
  nuget
  Solution Items
  AkkaSb.Net
  DistributedComputeLib
  NeoCortexApi
  NeoCortexArrayLib
  NeoCortexEntities
  NeoCortexUtils
  UnitTestsProject
    Dependencies
    CortexNetworkTests
      NetworkTests.cs
    EncoderTests
    Gt4UnitTests
      CellsPerColumnExperiment.cs
      ComplexSequenceExperiment.cs
      HtmSparsity.cs
      HtmSparsityAndMax.cs
      InputBitsExperiment.cs
      MaxMinValueExperiment.cs
      WidthExperiment.cs
    MnistPng28x28
    MnistPng28x28_smallerdataset
    NoiseExperiments
```

In the unit test "SimpleSequenceExperiment", first encoder an spatial pooler is added to the layer and their parameters are passed as this is the newborn learning stage. After training of SP, instance of TM is added to the layer. Now Spatial pooler and temporal memory are trained together (SP is pretrained on pattern). Prediction also takes place during the learning process.

The unit tests under Gt4UnitTests folder "WidthExperiment", "InputBitsExperiment", "MinMaxValueExperiment" and "ComplexSequenceExperiment" contains a loop in which the base experiment "SimpleSequenceExperiment" runs for "loop" number of times. To obtain a more averaged result. Input parameters for different experiments (Width, Input bits, Number of experiment repetitions and Max) is given to the unit test using Data Row. The results from the Unit tests (first cycle at which we get 100% correct prediction for the given sequence) are saved in Excel .csv file using "StreamWriter". Then the graphs are made manually using excel data to observe the result in a graphical form.



```
NeoCortexUtils.cs
UnitTestsProject
  Dependencies
  CortexNetworkTests
    NetworkTests.cs
      NetworkTests
        CategorySequenceExperiment() : void
        LongerSequenceExperiment() : void
        HtmSparsity() : void
        SimpleSequenceExperiment() : void
        MusicNotesExperiment() : void
        SpatialPooler_Stability_Experiment() : void
        RunExperiment(int, Parameters, EncoderBas
        Abc() : void
```

The unit test "SpatialPooler_Stability_Experiment" under CortexNetworkTests folder in the UnitTestProject is used to check the stability of the SP output by varying the Boosting parameter.

V. Result

Number of cycles for 100% Prediction:

It is the minimum number of cycles required to get 100% prediction for the given sequence.

Experiment Repetitions:

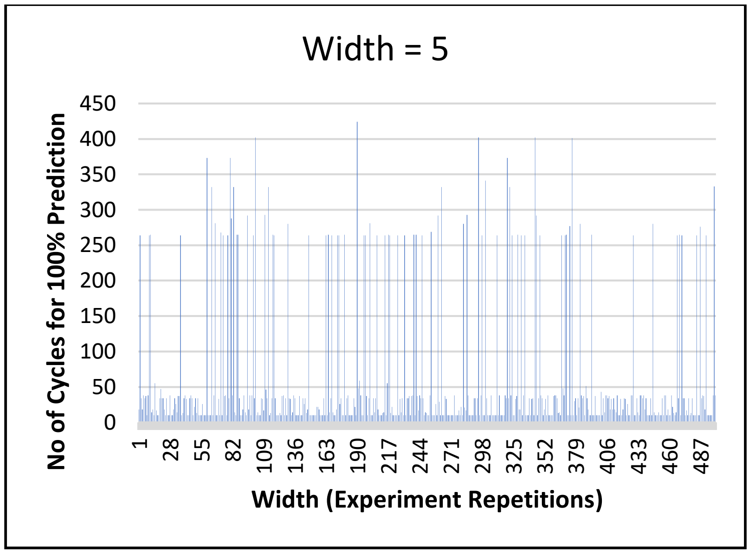
It is the number of times the unit test is running. It is running for multiple times due to fluctuations in the results i.e. to obtain a well averaged result.

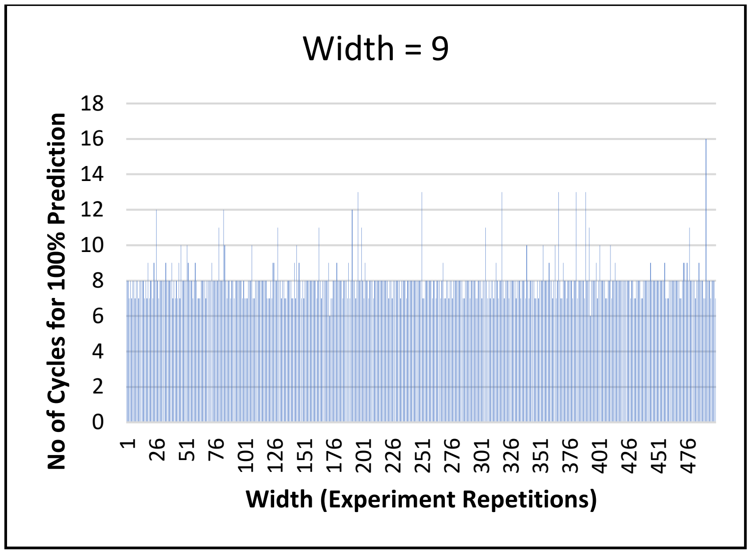
For each case the code is run for multiple cycles. In each cycle the algorithm predicts the next values in the sequence and if the predicted value and the next input value is same, it means that the algorithm has learned that sequence. For an input sequence of 0 to 9, the algorithm tries to predict all the next values and we have recorded the minimum cycles that the algorithm took to predict the 100% correct sequence upon changing the different parameters to see the influence of those parameters on the learning process.

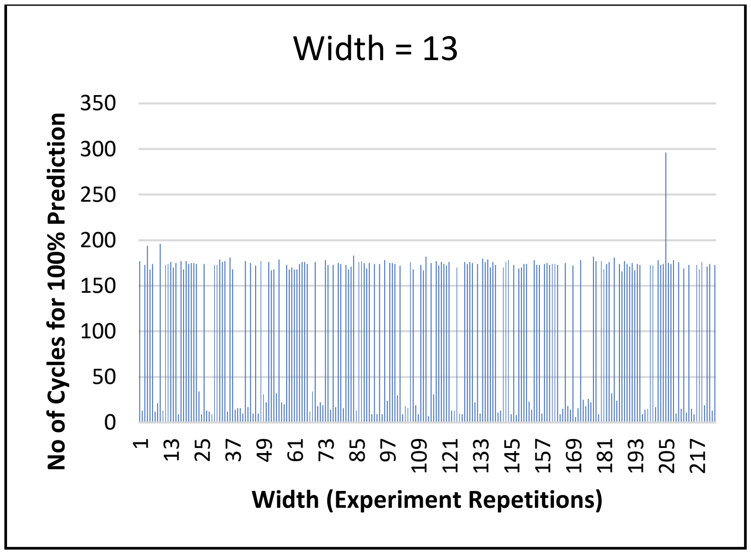
Following cases were subjected with its results:

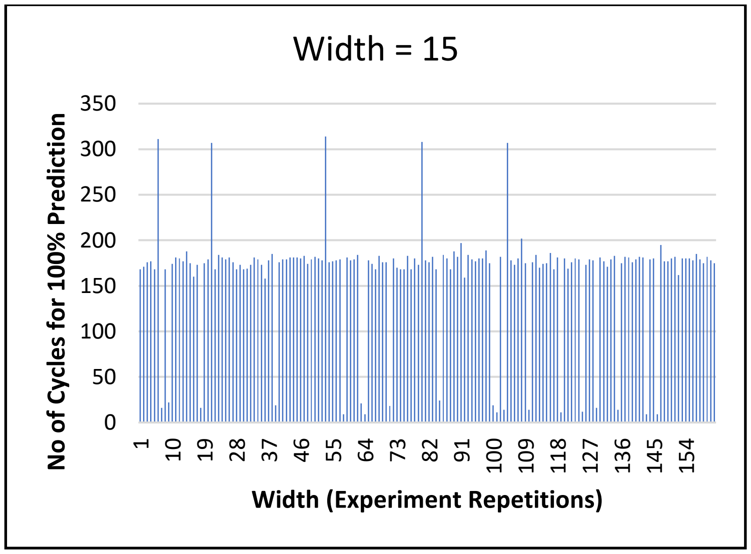
Case 1. Effect of Width on learning

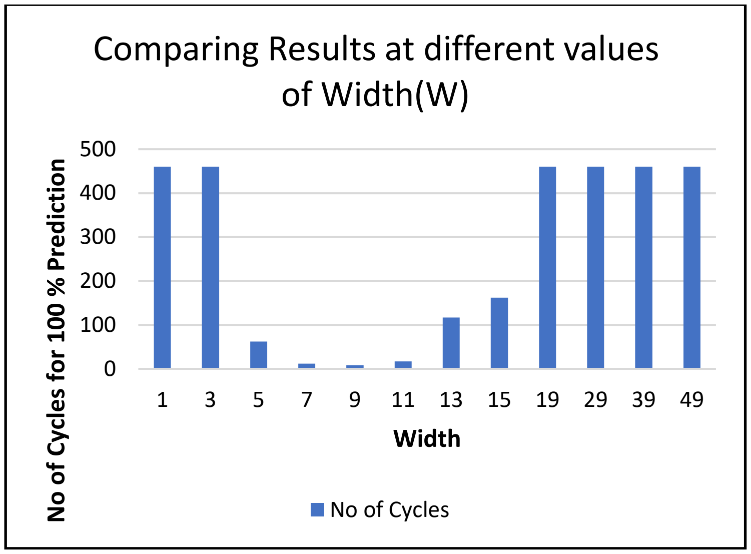
Following are few of the graphs for different values of width on which we worked on:

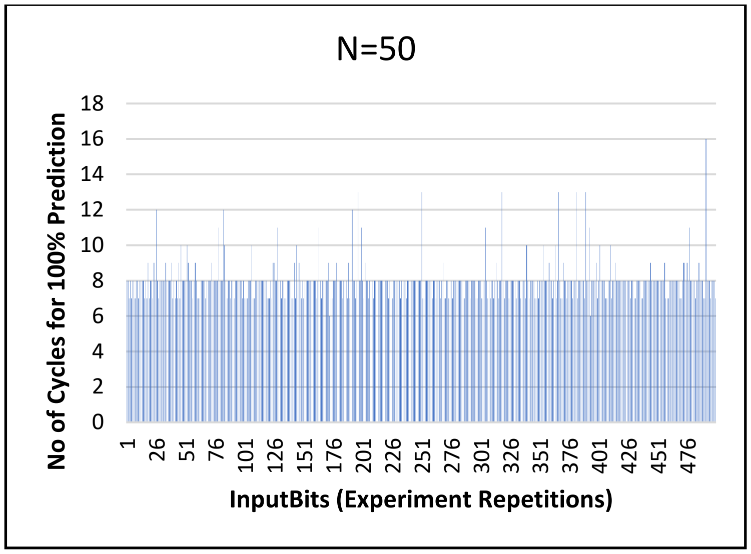










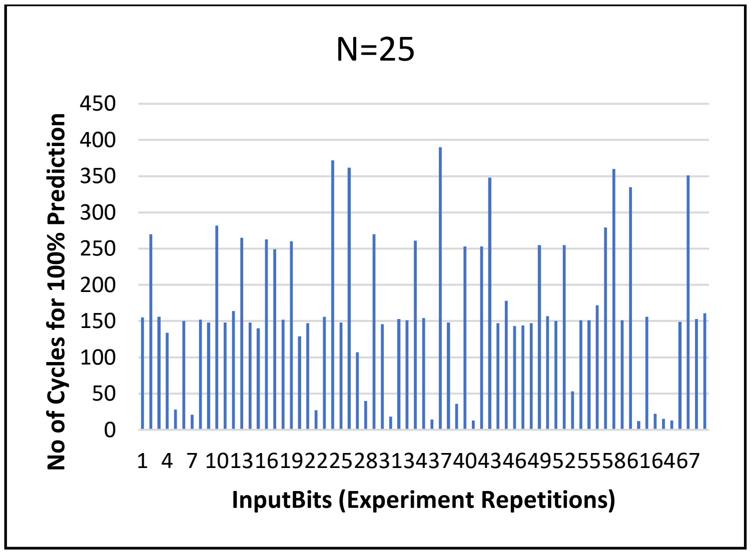


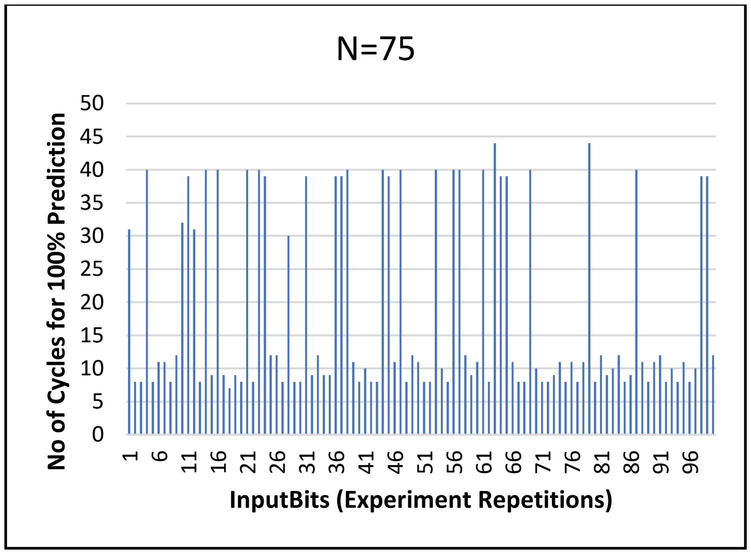
It can be seen from the above graph that, when we keep all the other parameters constant and change the value of Width(W), we can find the most optimal value of width. All the result seen from the above graph are taken from the unit test named "ParameterChangeExperment.cs" by running it for more than 500 times. The average time for the program to run for each value is about 1 hour 20 mins. For max =10 and N= 50, we get the lowest number of cycles to get 100 % match at Width = 8.

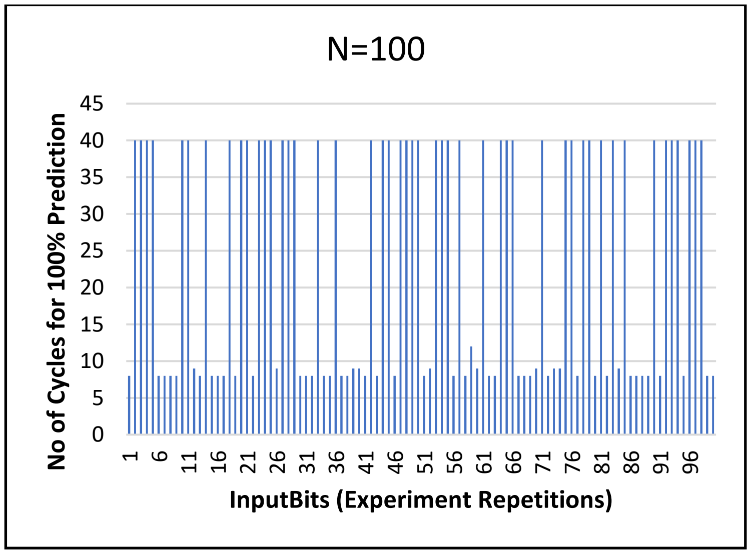
Number of Cycles for 100 % Matches decreases when we increase the value of Width(W) until it reaches its optimal value and after that the Number of Cycles will keep increasing.

Case 2. Effect of Input Bits on learning

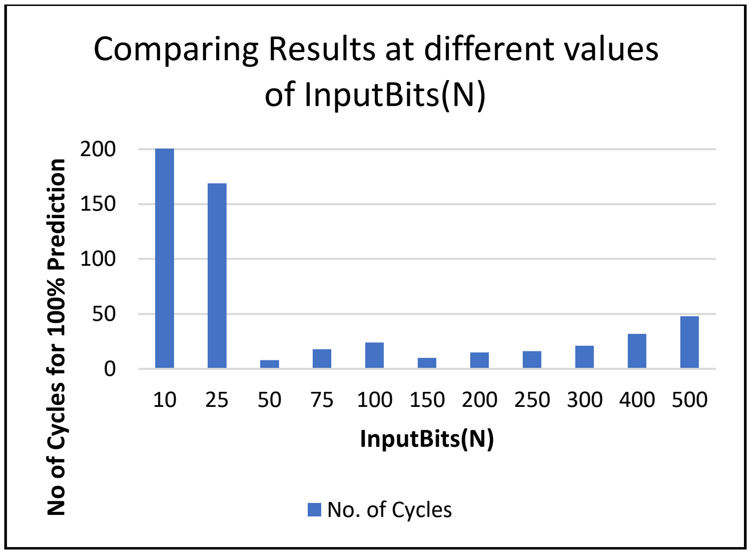
Follow are few of the graphs for different values of Input Bits.







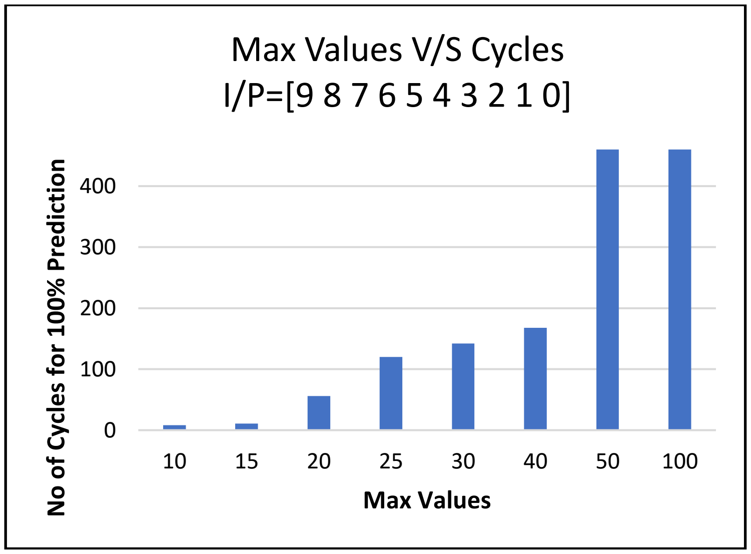
This graph shows the combined results of all the values of Input Bits tested in the experiment.

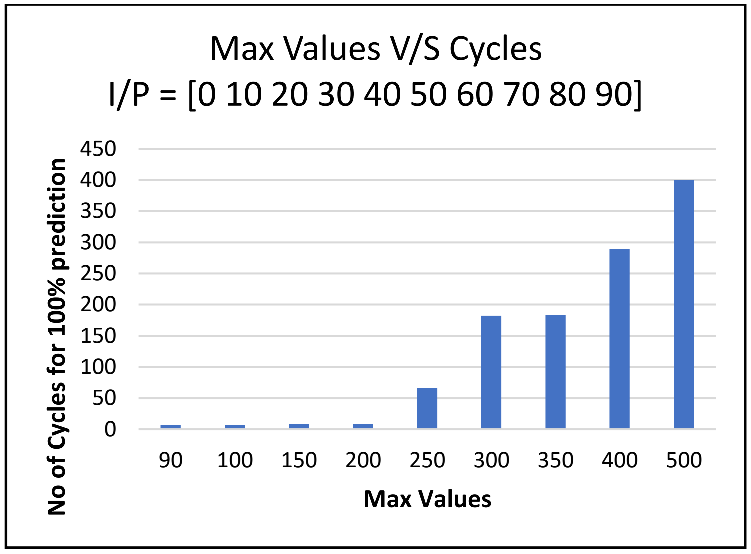


It can be seen from the above graph that, when we keep all the other parameters constant and change the value of Input Bits(N), we can find the most optimal value of Input Bits. All the result seen from the above graph are ran for more than 100 times using the "ParameterChangeExperment.cs" unit test.

The average time for the program to run for each value is about 30 mins. For max =10 and W = 9, we get the lowest number of cycles to get 100 % match at Input Bits(N) = 50. Number of cycles for 100% match decreases when we increase the value of Input Bits(N) until it reaches its optimal value and after that the Number of Cycles will keep increasing.

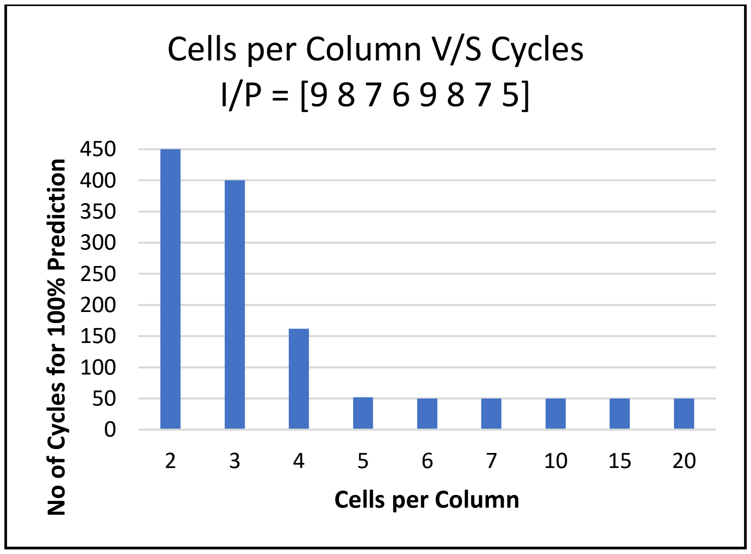
Case 3. Effect of Max and Min Values on Learning





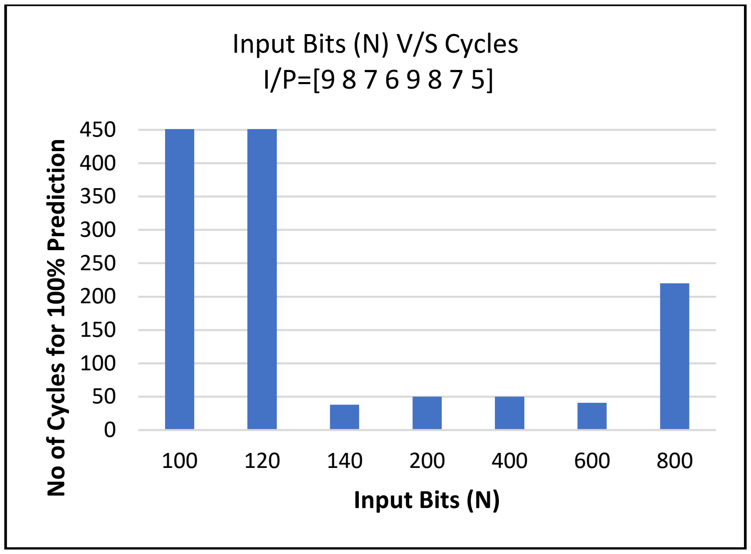
From the above graphs we can clearly observe that Max and Min values have effect on the learning. When we set the Max value to 100 and Min value to 0, for the input sequence of [9 8 7 6 5 4 3 2 1 0], Algorithm didn't learn the sequence in 460 cycles, and upon decreasing the Max values, the algorithm takes less cycles to learn the input sequence. The optimal value for Max was observed to be the greatest number in the input sequence, 10 and 100 for the above input sequences.

Case 4. Effect of different parameters on learning of a Complex Sequence

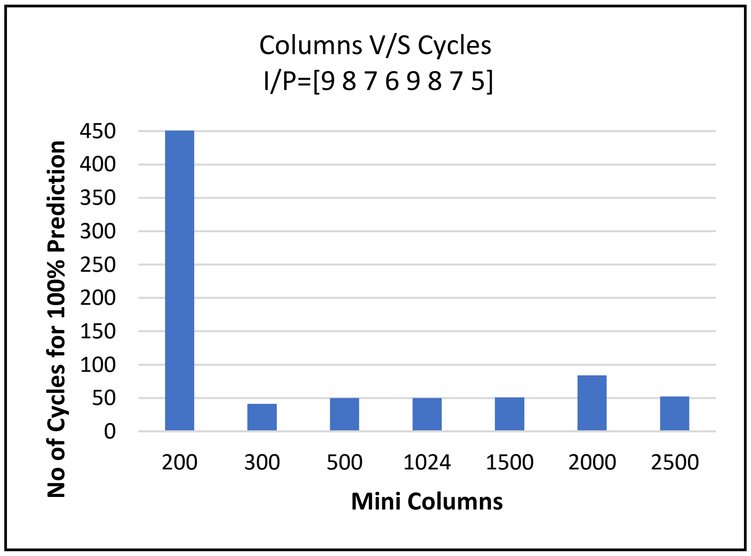


From the above experiment it can be noted that, at least 5 cells per column should be set in order to predict the correct next input to come. Unlike simple sequence of distinct number, where less number of cells are enough to get the optimal number of cycles to learn the sequence.

Secondly, when changing the number of active bits (N), the result is as follows,

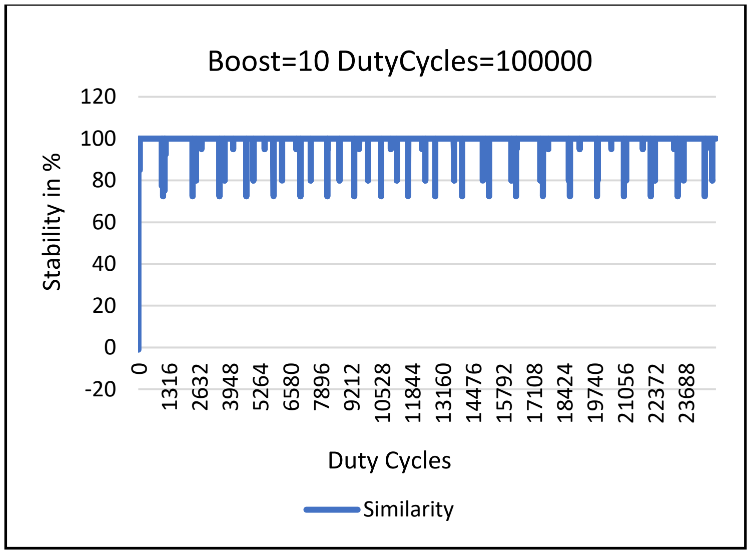


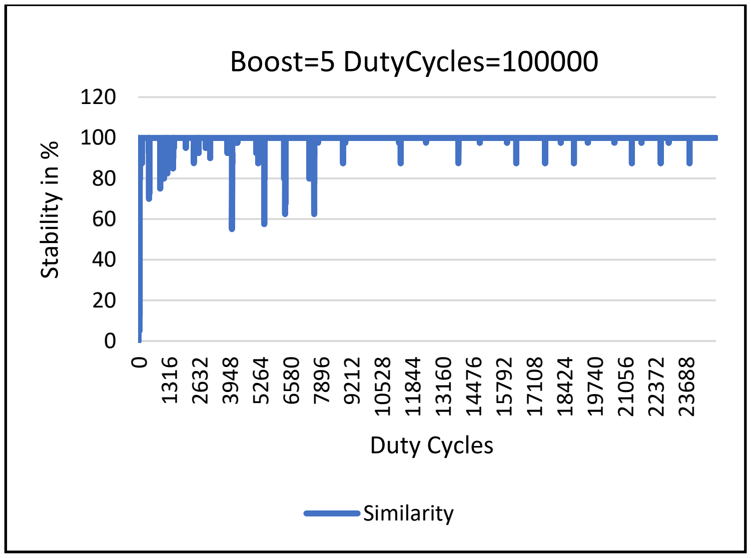
We can see that in order for algorithm to learn a complex sequence, another factor that contribute is N. Choosing the right values of input bits (N) and Width (W) are important. During encoding we need to select such value of W that can contain enough on bits to represent an input value and also the value of N to be as much so that there is no overlap between the SDR's of input bits. For this sequence the optimal values of N are between 140 and 600. Next, by keeping all the other parameters constant and varying the number of mini columns, we observe the following results,

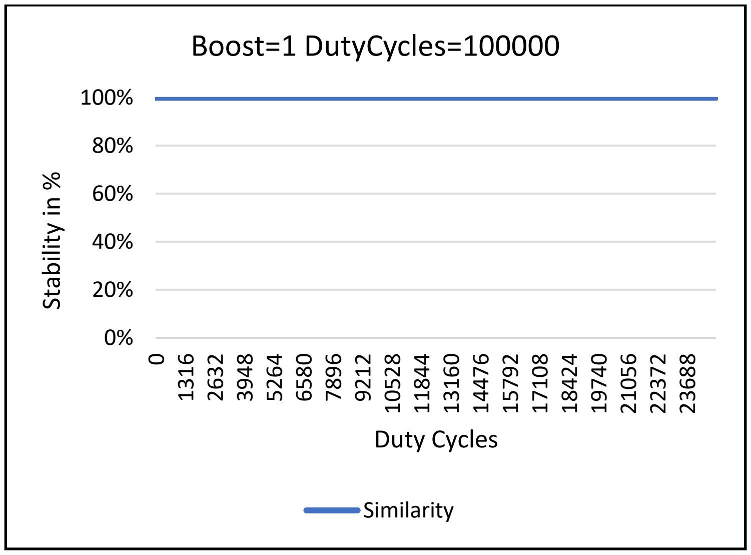


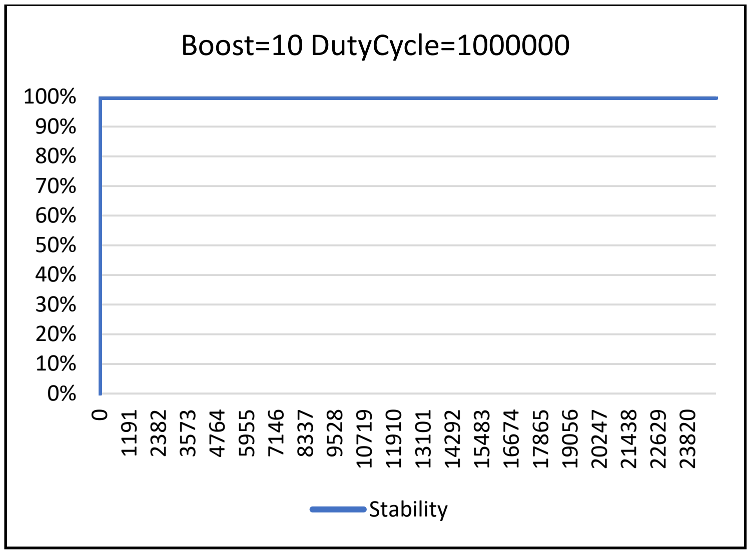
The above graph shows that it is required to have minimum number of columns in the neo cortex so that enough columns can be active in the spatial pooler for an SDR input. Increasing the number of columns beyond that minimum number will have no effect on learning. For this input sequence, minimum number of columns are 300 and for every value above 300, the algorithm takes around 52 cycles to learn the sequence.

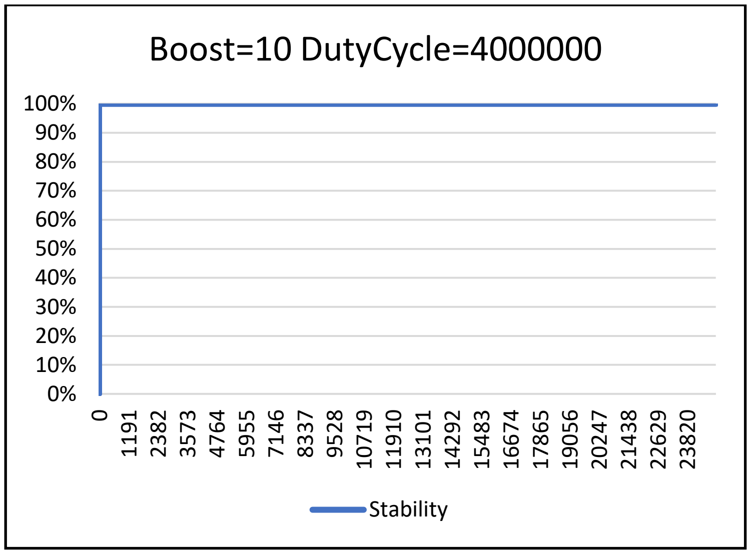
Case 5. Effect of Boost and Duty Cycle for stability of state

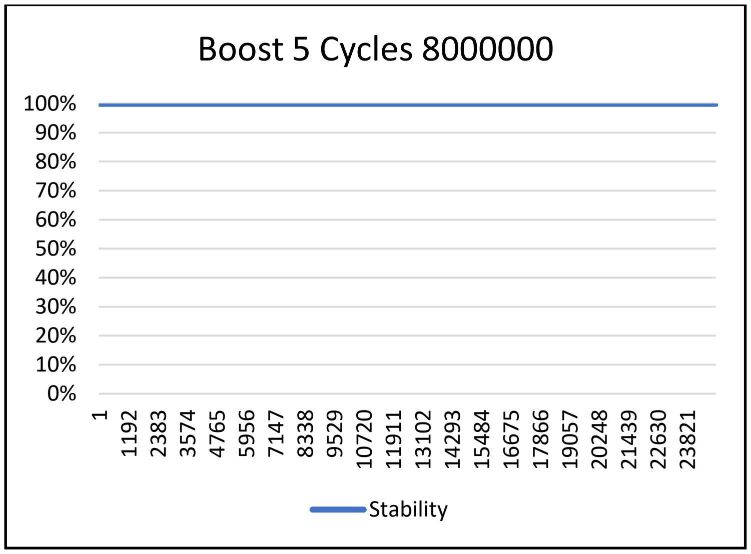












These graphical representations are the representation of data which is fetched from SpatialPooler_Stability_Experiment. We ran this program many times by setting up the different values of Max boost from 10 to 1 and Duty Cycles from 10000 to 40000000 We concluded that there is inverse relation of Max boost and Duty Cycle. The higher value of boost the lower the stability is but in fact if we reduce boost but increasing Duty cycle while the remaining parameters are same, so the stability also increase not 100% but approx. 90 to 95%. We added three graphs at boost value 10, 5, and 1. The only 100% stability can achieve at MaxBoost = 1. But the next three graphs showing the better result at high boost value because of the tuning of duty cycles. Sometimes, we ran program at duty cycles 8000000 while boost set to 5 so stability reached to 100% not always. Moreover, there is one more parameter which play a vital role for making it stable is "BUMPUP". It disables all the weak columns (Columns in which average or all the cells are not active). As this unit test takes more than 102 minutes to execute for all input series from 0 to 10 but, if we reduced no of columns from 2048 to 1000 or even 500 but keeping no of cell per column same so, it reduces the time of execution however, the stability still at 100%.

VI. CONCLUSION

Based on the above experiments, it can be said that various parameters of encoder, spatial pooler and temporal memory have effects on learning. We have observed how the parameters like input bits (N), Width (W), sparsity, Cells per column, number of mini columns in the neo cortex effect the learning by changing the values of these parameters and comparing them with the minimum number of cycles required to learn the input sequence. We have tested these parameters on simple sequence and on complex sequence and results are documented above.

VII. REFERENCE

[1] Hawkins, J., & Ahmad, S. (2016), Why neurons have thousands of synapses, a theory of sequence memory in neocortex. Frontiers in neural circuits, 10.

[2] Zyarah, A. M., & Kudithipudi, D. (2019), Neuromorphic architecture for the hierarchical temporal memory. IEEE Transactions on Emerging Topics in Computational Intelligence, 3(1), 4-14.

[3] S. Ahmad, M. Lewis, (2017), "Temporal memory algorithm", Technical Report Version 0.5, Numenta Inc.