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

Investigation of Sequence Learning of SP/TM Layer (Cells Per Column)

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*— The ability to recognize and predict temporal sequences of sensory inputs is vital for survival in natural environments. Based on the number of known properties of cortical neurons, hierarchical temporal memory (HTM) has been recently proposed as a theoretical framework for sequence learning in the neo cortex. In this paper, we analyze the sequence learning behavior of spatial pooler and temporal memory layer in dependence on learning parameter-Cells per Column. We have demonstrated how changing the number of cells per column improvised the learning process for different input sequences. The results show the proposed model is able to learn a sequence of data by keeping the number of cells beyond a certain value depending upon the complexity of input sequence.**

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

I. INTRODUCTION

Hierarchical temporal memory (HTM) is a neuromorphic machine learning algorithm that emulated the performance of the human brain neocortex operations. The HTM architecture consists of a spatial pooler (SP) and a temporal memory (TM) and is characterized by sparsity, modularity, and hierarchy respectively. The SP performs sparse distributed representation of the input data, while the TM is responsible for the learning process. The imitation of human brain functionality makes HTM a core-adjustable algorithm applicable to various operations such as, object categorization, pattern discovery, and data classification. HTM works with the assumption that everything the neo cortex does is based on memory and recall of sequences of patterns. [1]

The main objective of this paper is to analyze an important HTM parameter namely *cells per column*, by working on simple sequence and complex sequence of inputs and observe the influences on 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.

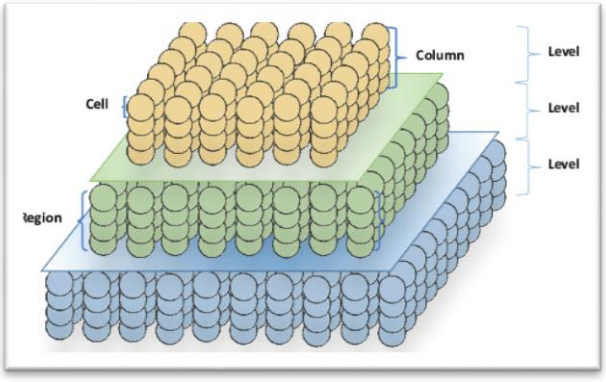


Fig. 1. A representation of three layers **of HTM** arranged hierarchically.

A. *Encoder*

In this part, the data taken from as input is encoded semantically as a sparse distributed representation (SDR). These arrays are then passed through the spatial pooler making them into a fixed size. This is done by Hebbian learning, by activating the inactive cells. It is also important to obtain similar SDR'S from similar inputs. 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 the 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 there are enough number of active synapses connected to active bits in the input, 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 three stages namely initialization, overlap, inhibition, and learning. [1]

C. Temporal Memory Model

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. [2]

III. METHODOLOGY

The major inspiration of this experiment is to study the required number of learning cycles to learn the input sequence in dependence on the number of cells per column. Below are the parameters defined while executing this experiment in order to achieve the conclusion.

- **Width (W)**

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 (N)**

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.

- **Max and Min values**

An encoder parameter, which defines the input range. The input sequence must be within that range. Minimum value and Maximum values are termed as *MinVal* and *MaxVal*

- **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*

- **Radius**

Radius is defined as the number of active bits (W) divided by the Input bits to encode a value (N) into the range of input.
Mathematically, (W/N).I where I is the range. We have set the value of Radius to -1, to neglect its effect.

- **Periodic**

The SDR's for an input value should repeat itself after a particular internal or not is defined by the Periodic parameter. Setting it false means, it is non periodic.

In order to study the effect of cells per column, we vary the number of cells in a single mini column by keeping all other parameters constant as shown below. This specifically allows us to study the behavior of learning process based on cells per column.

```
int inputBits = 200;
{ "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 });
p.Set(KEY.CELLS_PER_COLUMN, C);
```

Initially the number of cells per column is set to 1 and changed to 2, 3, 4, 5, 7, 10, 11, 15, and 20 respectively to observe how it affects the number of cycles required to learn different input sequences. The input sequences are:

1) **Input = [0 1 2 3 4 5 6 7 8 9]**
2) **Input = [1 2 2 3 4 4 5 6 6 7 8]**
3) **Input = [1 2 3 1 2 4]**
4) **Input = [1 2 3 4 1 2 3 5]**
5) **Input = [1 2 3 4 5 1 2 3 4 6]**

The experiment used is CellsPerColumnExperiment.cs. The basis of this experiment is SimpleSequenceExperiment.cs in which the encoder and spatial pooler are initially added to layer1.

```
CortexLayer<object, object> layer1 = new
CortexLayer<object, object>("L1");
region0.AddLayer(layer1);
layer1.HtmModules.Add("encoder", encoder);
layer1.HtmModules.Add("sp", sp1);
```

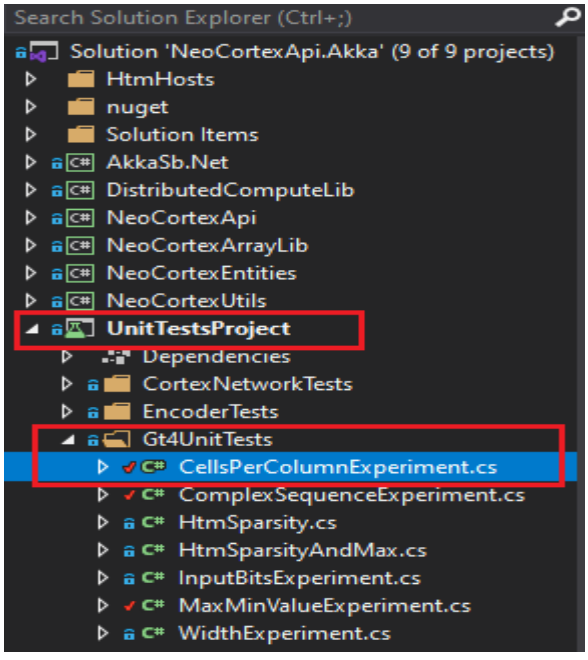
And the above mentioned parameters are then passed. As it is considered as a newborn learning stage, the SDR's of individual inputs are introduced to the algorithm. However, at this point of time there is no TM attached to the layer and as a result these inputs are incomprehensible. Later, the instance of TM is added to layer1.

```
layer1.HtmModules.Add("tm", tm1);
```

Now the Spatial pooler and the temporal memory are trained together and these inputs will be clear and reasonable enough for the algorithm to start making predictions of the future input values as discussed earlier. The code is run for 400 cycles. In each cycle the input sequence is taken as input and the algorithm tries to predict the next values in the sequence. It takes few cycles for the algorithm to start predicting the actual sequence. In each cycle there is an input value and the algorithm predicts a value that would be the next input to come, if the predicted values and the next input values in a cycle are the same, it means that the algorithm has learned that particular sequence. For an input sequence from 0 to 9, the algorithm tries to predict all the succeeding values and we have recorded the minimum cycles that the algorithm took to predict the 100% correct sequence respectively.

IV. C# Code

The unit test used in the project can be found under the solution "NeoCortexapi.Akka"in HTM folder, where there is a section named "UnitTestProject" which contains a folder Gt4UnitTests that includes "CellsPerColumnExperiment.cs".



This unit test is modified version of "SimpleSequenceExperiment", which is specifically made to perform this experiment. We have used DataRow to pass parameters to the function.

```
    [DataRow(1, 30)]
    [DataRow(2, 30)]
    [DataRow(3, 30)] …
public void CellsPerColumn(int C,int loop))
{
}
```

The first column represents the number of cells and the second column is the number of times the whole experiment will run. A single experiment consist of 400 cycles.

For every cell value, we are running the experiment 30 times referred to as “loop” in the code to achieve accurate results.

V. RESULTS

The following graphs are obtained for different input sequences respectively and are plot between Cycles Required to Learn and Cells Per Column.

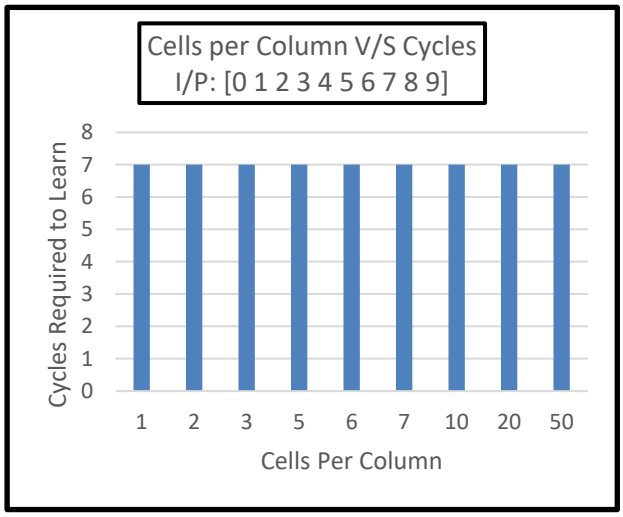
Cycles Required to Learn

It is the minimum number of cycles for the algorithm required to learn the given input sequence.

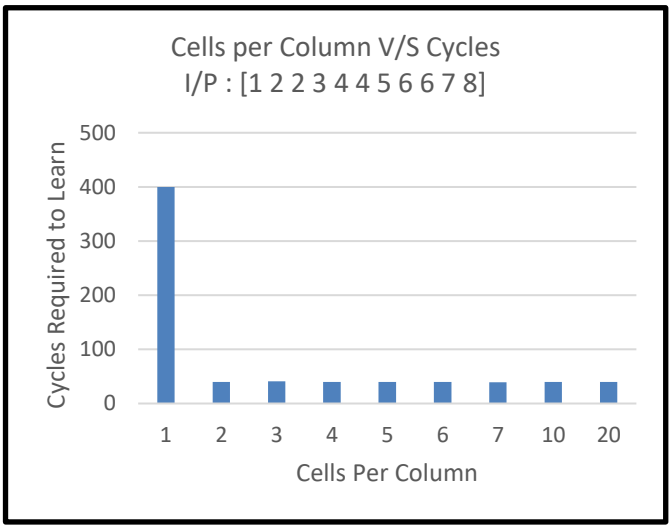
Cells per Column

By changing the number of cells we try to observe the effect on cycles taken by algorithm to learn the input sequence.

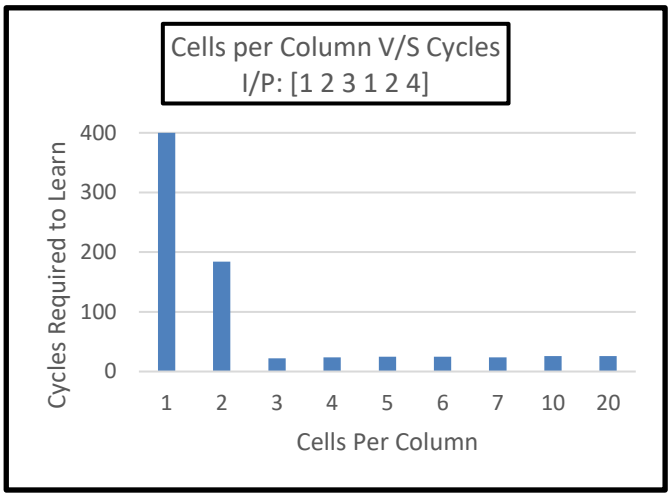
Case 1. For Input Sequence [0 1 2 3 4 5 6 7 8 9



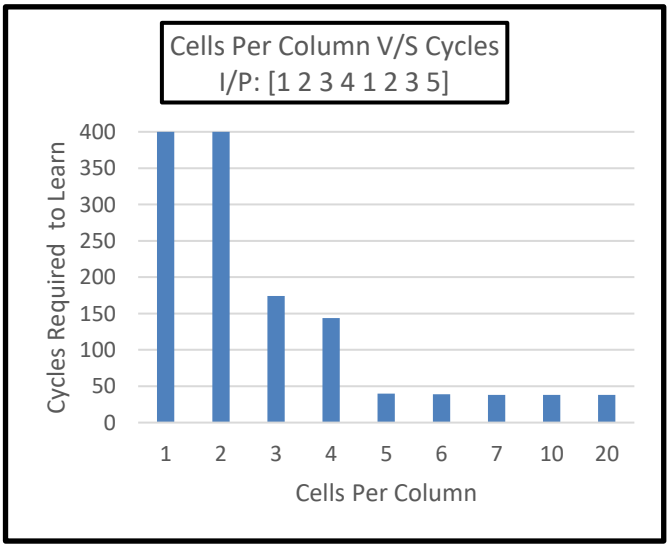
Case 2. For Input Sequence [1 2 2 3 4 4 5 6 6 7 8]



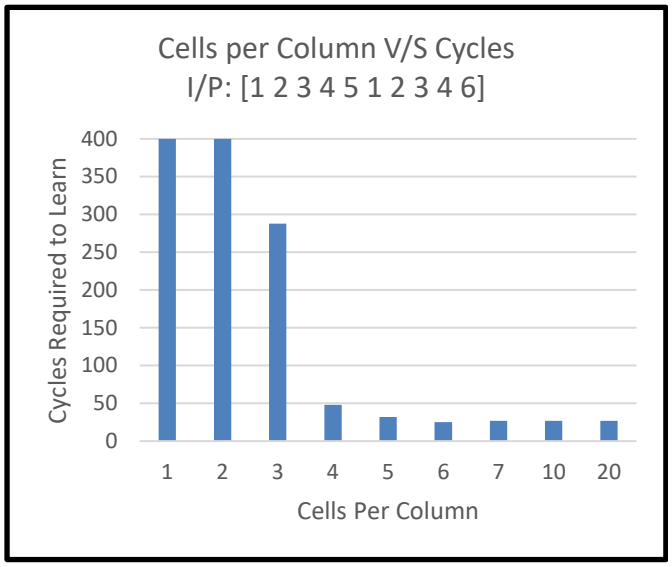
Case 3. For Input Sequence [1 2 3 1 2 4]



Case 4. For Input Sequence [1 2 3 4 1 2 3 5]



Case 5. For Input Sequence [1 2 3 4 5 1 2 3 4 6]



VI. CONCLUSION

Based on the above results, it can be seen that if there is 1 cell in a column, in means that the algorithm can predict the next number on the basis of just the current input if the sequence is of distinct numbers like in case1.

However, if you have a complex sequence like [1 2 3 1 2 4], where number 2 precedes two numbers, 3 and 4, when there is 2 as input, the algorithm has two choices either predict 3 or 4. If you have 1 cell in column, means, the algorithm must predict on the basis of just current value, it cannot predict the correct value. If there are 3 cells in a column, it means algorithm can see 2 values back and the current value to make a prediction and by seeing the last two values and the current value, it will be able to correctly predict the next value. The number of cells per column has to be varied depending upon the complexity of the input sequence

REFERENCES

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

[3] 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.