

are employed to encode discrete data such as words, colors, or symbols.

III. METHODOLOGY

Most of the time in this experiment was spent on understanding and experimenting on Multisequence Learning. The Multisequence Learning is based on HTM Classifier taken from the NeoCortexApi [1].

In this algorithm, the connections of taken as per HTM Configuration. The Spatial Pooler creates a sparse representation of encoded data, while the Temporal Memory is used to remember the sparse representation. Additionally, there is a Cortex Layer and an HTM Classifier that receives the trained model and predicts the outcome.

Following is the pseudo code for Multisequence Learning:

01. *Get HTM Config and initialize memory of Connections.*
02. *Initialize HTM Classifier and Cortex Layer*
03. *Initialize HomeostaticPlasticityController*
04. *Initialize memory for Spatial Pooler and Temporal Memory*
05. *Add Spatial Pooler memory to Cortex Layer*
 - 05.01 *Compute the SDR of all encoded segment for multi-sequences using Spatial Pooler*
 - 05.02 *Continue for maximum number of cycles.*
06. *Add Temporal Memory to Cortex Layer*
 - 06.01 *Compute the SDR as Compute Cycle and get Active Cells*
 - 06.02 *Learn the Label with Active Cells*
 - 06.03 *Get the input predicted values and update the last predicted value depending upon the similarity.*
 - 06.04 *Reset the Temporal Memory*
 - 06.05 *Continue all above steps for sequences of multi-sequences for maximum cycles.*
07. *Get the trained Cortex Layer and HTM Classifier*

A. Identifying use of MaxNewSynapseCount

After going through the project, we found that MaxNewSynapseCount is being stored in HTMConfig and used in TemporalMemory (TM). In TM, ActivatePredictedColumn() and BurstColumn() which in the end calls GrowSynapses() depending on the condition of Active Segment and Matching Segments.

B. Logger in Temporal Memory

Added logger string in TemporalMemory.cs and log the calls for GrowSynapses() and by which condition it is called.

C. Spatial Pooler

The Spatial Pooler is responsible for creating a sparse distributed representation of the input data, which is then fed into the Temporal Memory component for further processing.

The Spatial Pooler works by creating a set of columns, each of which represents a potential input feature [3]. The input data is then mapped onto these columns, with each input feature activating a subset of the columns. The Spatial Pooler then selects a small subset of the active columns to represent the input data, using a process known as inhibition.

The inhibition process involves selecting a fixed percentage of the most highly activated columns and suppressing the activation of all other columns. This results in a sparse distributed representation, where only a small subset of the columns is active at any given time.

D. Temporal Memory

The temporal memory component of HTM is responsible for learning and predicting sequences in time-varying input data.

Temporal Memory works by maintaining a set of cells that represent patterns of activity in the input data. These cells are connected to each other through synapses, which can strengthen or weaken over time based on the input data patterns. As the input data sequences are presented to the temporal memory, the cells become active in a sparse and distributed manner, forming a representation of the input sequence.

When a new input sequence is presented to the temporal memory, it compares the sequence to the patterns it has learned and predicts the next likely pattern based on the sequence's context.

E. Encoder

An encoder is a technique that converts unprocessed input data into a sparsely dispersed depiction, which can be handled by the Spatial Pooler component [4]. The encoder's task is to transform continuous input data into a collection of distinct categories, forming a representation that is resistant to noise and can adapt to novel patterns.

In HTM, various encoder types are available, which include scalar encoders, category encoders, and datetime encoders. Among these, scalar encoders are the most frequently employed type and are utilized to encode continuous data like temperature values, sensor readings, and audio signals. On the other hand, category encoders

```

public List<String> Logger { get; set; }
public int countLogger { get; set; }
:           :           :
:           :           :
//in GrowSynapses()
this.Logger.Add($"GrowSynapses(): calledBy: {calledBy},
requiredNewSynapses: { requiredNewSynapses },
numMissingSynapses: {numMissingSynapses}");
this.countLogger++;
//use above variables as per each cycle is changed

```

Code block 1: Adding and updating logger variable

C. Report

The Report is a data model which is updated per cycle of learning loop and saves sequence name with logs from temporal memory and accuracy per cycle.

D. Analysis

In Analysis data model we create a extracted model out of Report data model which stores the cycle sequence name and number of call made by which condition and number of synapses increased in that cycle.

IV. RESULTS

This experiment has been tried to run as many times as possible and find our optimal configuration and trace the changes.

The reports and analysis is store in base path of app domain (AppDomain.CurrentDomain.BaseDirectory) which has reports and analysis directory having all the outputs.

Furthermore,

1. MaxNewSynapseCount value changes in ActivatePredictedColumn() if we have active column and active segment and the positive difference in MaxNewSynapseCount and active potential synapse is addition of synapse
2. MaxNewSynapseCount value changes in BurstColumn() if we have active column and no active segment and the positive difference in MaxNewSynapseCount and last active potential synapse is addition of synapse
3. The accuracy of each cycle changes (goes up usually) even if we do not add any new synapses, but it does not assure that adding synapses with increase the accuracy.

A. Based of Reports data model

The report shows all the calls made for calling GrowSynapses() and the number of synapses added:

```

----- Start of Cycle: 210 -----
Cycle: 210, Sequence: S1, Accuracy: 0
GrowSynapses(): calledBy: BurstColumnWithMatchingSegments, requiredNewSynapses: 2, numMissingSynapses: 2, input: 4
GrowSynapses(): calledBy: BurstColumnWithMatchingSegments, requiredNewSynapses: 1, numMissingSynapses: 1, input: 4
GrowSynapses(): calledBy: BurstColumnWithoutMatchingSegments, requiredNewSynapses: 2, numMissingSynapses: 2, input: 5
GrowSynapses(): calledBy: BurstColumnWithoutMatchingSegments, requiredNewSynapses: 1, numMissingSynapses: 1, input: 5
----- End of Cycle: 210 -----
----- Start of Cycle: 211 -----
Cycle: 211, Sequence: S1, Accuracy: 22.222222222222
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 2, numMissingSynapses: 2, input: 4
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 1, numMissingSynapses: 1, input: 4
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 4, numMissingSynapses: 4, input: 5
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 3, numMissingSynapses: 3, input: 5
----- End of Cycle: 211 -----
----- Start of Cycle: 212 -----
Cycle: 212, Sequence: S1, Accuracy: 77.777777777778
GrowSynapses(): calledBy: BurstColumnWithMatchingSegments, requiredNewSynapses: 7, numMissingSynapses: 7, input: 4
GrowSynapses(): calledBy: BurstColumnWithMatchingSegments, requiredNewSynapses: 7, numMissingSynapses: 7, input: 4
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 2, numMissingSynapses: 2, input: 5
GrowSynapses(): calledBy: ActivatePredictedColumn, requiredNewSynapses: 1, numMissingSynapses: 1, input: 5
----- End of Cycle: 212 -----

```

Figure 2: Slice from the report generated

1. calledBy: ActivatePredictedColumn

It means that there were some Active Segments and even though there were some Active Segments, new synapses can be added that connect previously active cells with the segment in ActivatePredictedColumn(). If the difference b/w MaxNewSynapseCount and last potential active cells is positive, we grow that many synapses.

2. calledBy: BurstColumnWithMatchingSegments

It means that there were no Active Segments and we need to active random cell in BurstColumn(). We adapt segment and check for matching segment and if the positive difference b/w MaxNewSynapseCount and last potential active synapse is positive we grow that many synapses.

3. calledBy: BurstColumnWithoutMatchingSegments

It means that there were no Active Segments and we need to active random cell in BurstColumn(). We adapt segment and if no matching segment then we calculate the minimum b/w MaxNewSynapseCount and previous winner cell count to grow the synapses

B. Based of Analysis data model

We save number of calls made by ActivatePredictedColumn or BurstColumnWithMatchingSegments or BurstColumnWithoutMatchingSegments and the number of synapses increased for all sequence per cycle. The following image shows visual representation of change happening for a sequence in all its cycles. These are saved as CSV files.

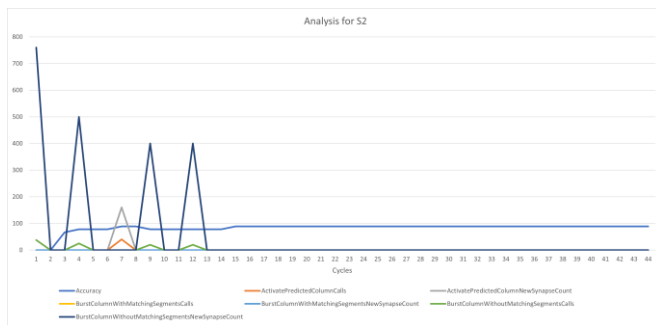


Figure 3: Analysis of a sequence

V. DISCUSSION

It is always seen that synapse are added in initial cycles of the learning later of the count goes down. Also if the algorithm is learning a new sequence if adds some new synapse but not as much as it added when it started learning in very first cycle. There is no guarantee that increasing in synapse count will increase the accuracy of upcoming cycle.

Also, it is nature of the temporal memory that after couple of cycle if has already extracted the pattern.

VI. REFERENCES

- [1] "NeoCortexApi : <https://github.com/ddobric/neocortexapi>".
- [2] Jeff Hawkins, "On Intelligence" (2004)
- [3] Subutai Ahmad and Jeff Hawkins, "A Review of Hierarchical Temporal Memory", 2016
- [4] Matthew K. Graham, William C. Gross, Subutai Ahmad, and Jeff Hawkins, "Real-Time Anomaly Detection in Streaming Sensor Data Using Hierarchical Temporal Memory", 2019