Implement the new Spatial Learning Experiment

1st Mandar Gokul Kale
mandar.kale@stud.fra-uas.de

2nd Anushruthpal Keshavathi Jayapal
anushruthpal.keshavathi-jayapal
@stud.fra-uas.de

3rd Lavanya Suresh
lavanya.suresh@stud.fra-uas.de

Abstract*—Hierarchical Temporal Memory (HTM) is a machine learning algorithm that attempts to replicate the functioning of human neocortex. The HTM architecture consists of Spatial Pooler (SP) that transforms input data in to Sparse Distributed Representation (SDR) and Temporal Memory (TM) which is responsible for the learning process. Precisely, the input data are fed to the encoder that converts them to binary values that is received by the SP which then generates SDR values. Hierarchical Temporal Memory (HTM) systems rely on efficient spatial pooling mechanisms to accurately encode and process input data. In this project, we focus on implementing a new Spatial Learning experiment for optimizing the performance of the HTM Spatial Pooler by fine-tuning key parameters and addressing specific challenges encountered during the learning phase. After thorough debug and analysis, issues such as slow activation of mini-columns during the learning phase has been identified and rectified, significantly enhancing the system functionality. A dictionary is utilized for storing the values of Sparse Distributed Representations (SDRs). For better representation of output, two significant parameters are added such as N- total iterations the SDR of the input was not changed and stable cycles. This paper also involves configuring parameters to achieve consistent system stability and implementing appropriate exit conditions. The effectiveness of the methods is demonstrated through empirical data, showcasing substantial improvements in efficiency metrics.

Keywords*—Machine Learning, Neocortex, Hierarchical Temporal Memory (HTM), Spatial Pooler, Sparse Distributed Representation (SDR)

I. INTRODUCTION

For decades, providing a computer with the ability to learn has been diligently pursued as one of the most important goals in the computer sciences. Major advances in the field have led through development of numerous machine learning methods and technologies, resulting in successful extraction and storing of useful knowledge from various sorts of input data. Neural computing is a hot topic in the field of artificial intelligence and machine learning. The mammalian brain is a complex set of structures from which the neocortex is the most recently evolved region. The neocortex, a thin and intricately folded layer of grey matter overlaying the brain's older structures plays vital part in conscious behavior, high-level thinking and learning. Human brain continuously receives vast information about the external world through peripheral sensors that transform changes in light, sound and skin deformations into millions of spike trains. Every neuron must interpret a constant set of time-varying inputs by forming synaptic connections to a portion of presynaptic neurons. The collective activation pattern of populations of neurons contributes to our perception and behavior. Cognitive functions such as perception and language are mostly dependent on neocortical processing. Understanding this could potentially prompt the development of a completely new generation of intelligent agents able not only to learn, but also to adapt easily to, and cope with challenging environments [1].

Hierarchical temporal memory (HTM) is a kind of machine learning technology that simulates the organization and mechanism of cerebral cortex cells as well as the information processing pipeline of the human brain. HTM is a neural network model designed to emulate the functionality of neocortex. It is particularly suitable for sequence learning and have achieved promising results in various application field, such as anomaly detection, traffic flow prediction and stock forecasting [2]. Compared with the conventional machine learning methods, HTM tends to propose a general theory of intelligence rather than being designed to solve specific types of problems. In order to capture spatial information and learn temporal dependencies from sequential data, HTM uses two core learning algorithms, Spatial Pooler and temporal memory.

In HTM, an Encoder serves a crucial role by preparing input data before it enters the HTM network. It transforms raw data like numbers, text, or images into binary vectors. The Spatial Pooling (SP) algorithm then takes over, creating an internal representation of the data pattern in the form of Sparse Distributed Representations (SDRs). These SDRs are based on columns and cells. To learn these internal representations, each column establishes numerous connections with the input space through a proximal dendrite segment. In SP, the input data at each time step is depicted by a group of active columns. However, identifying these active columns involves traversing all columns and conducting numerous comparison operations.

Each column in SP forms synaptic connections, namely synapses, with the input space. The synapse has a permanence value indicating the strength of the connection with the input bits. If the permanence exceeds a given threshold, the synapse is viewed as connected (solid line), otherwise, it is a potential synapse (dotted line). The columns whose permanence value is greater than their neighbors are activated, and the set of active columns constructs the SDR for the current input (red columns in Figure 1).

The Spatial Pooler algorithm employs inhibition and boosting mechanisms. Inhibition ensures that only a small number of columns become active, promoting a sparse representation, while boosting encourages underutilized columns to contribute, enhancing the system's ability to learn and recognize patterns in input data. Intuitively, the data representation should gradually converge to a stabilized state and this stability is crucial for the model performance because it affects the subsequent learning of temporal dependencies and the final prediction accuracy.

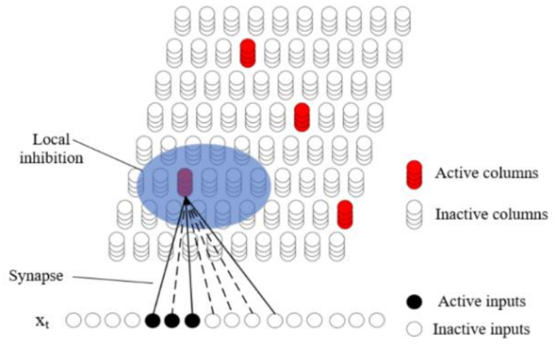


Figure 1 :Spatial Pooler Structure of Hierarchical Temporal Memory.

II. LITERATURE REVIEW

The aim of this section is to understand the functioning of HTM-Spatial Pooler and the process of generating SDRs which forms the basis of this paper, exploring various avenues to achieve a more efficient system.

Structured into regions comprising cell columns, the HTM algorithm operates with each column forming a computational unit where Spatial Pooling (SP) functions. SP emulates sparse neuron activation through inhibition, wherein columns inhibit neighboring columns within a defined radius from becoming active. Synapses connect cells within columns to input bits, with permanence values determining full connections based on a threshold. [3]

There are some challenges being discussed in conventional HTM one of which is, the Spatial Pooler algorithm needs significant amount of time to generate the internal representation of input data. [2]

One of the properties of the SP is its ability to form fixed-sparsity representations of input stimuli. This characteristic ensures that all input patterns can be equally detected by downstream neurons. Variability in sparsity levels may lead to increased false positives for dense patterns and false negatives for sparse ones, emphasizing the importance of fixed sparsity for reliable recognition. An optimal neural system should effectively utilize all available resources to encode input information. This principle underscores the importance of ensuring that each neuron responds to a fraction of inputs, promoting a diverse representation of the input space. The boosting mechanism within the SP framework serves to achieve this goal. [4]

In dynamic environments, neural systems must adapt to evolving input statistics. Here, it is been emphasized the importance of flexibility and plasticity, particularly in applications with continuously changing data streams. The SP's capacity to adjust synaptic connections in response to shifting input distributions aligns with this requirement. Neural systems must exhibit resilience to faults and damage, ensuring continued functionality despite perturbations. It highlights the brain's remarkable ability to recover and adapt following injury. Similarly, the SP should maintain operational integrity in the event of system faults, such as loss of input or output neurons. [4]

It is shown that with careful initialization of parameters the SP maps distinct inputs into distinct SDR's as indicated by the column activity. Assuming appropriate parameter selection, we make observations about the learning dynamics for both non-overlapping and overlapping inputs. [5]

As a cortical algorithm, HTM offers a novel model for hardware implementation of neocortical functions, proven feasible through image processing and recognition applications. Various digital and analog designs have been proposed, showcasing implementations for SP alone capable of learning and classifying diverse datasets. [3]

The Spatial Pooler algorithm was extended with the new homeostatic plasticity component that controls the boosting and deactivates it after entering the stable state. Results show that learned SDRs remain stable during the lifetime of the SP. [6]

III. METHODOLOGY

The project Implementation of new Spatial Learning experiment is developed using C# .Net Core in Microsoft Visual Studio 2022 IDE (Integrated Development Environment) and is illustrated to understand the functioning of SP learning. This experiment is inspired by HTM theory which encompasses various components, including the Scalar Encoder, Spatial Pooler, and Homeostatic Plasticity Controller as shown in Figure 2 , each playing a vital role in emulating the functionalities of the neocortex and enabling continuous learning of different patterns and adaptive behavior in systems.

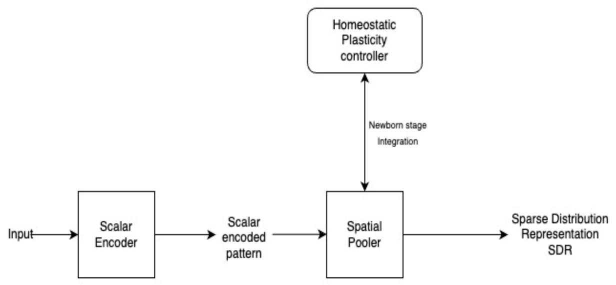

Homeostatic Plasticity controller
Newborn stage Integration
Input
Scalar Encoder
Scalar encoded pattern
Spatial Pooler
Sparse Distribution Representation SDR

Figure 2: Block diagram representing the components of Spatial learning experiment.

A. *Scalar Encoder*

In this project, a Scalar Encoder is utilized, a fundamental encoding technique within Hierarchical Temporal Memory (HTM) systems. Specifically designed to transform continuous input values into binary representations suitable for HTM processing, the Scalar Encoder plays a crucial role in our project. Prior to entering the SP, we encode a sequence consisting of values ranging from 0 to 100. Each input value is encoded into a 200-bit representation, with 15 non-zero bits allocated to represent individual values.

B. *Spatial Pooler*

The Spatial Pooler (SP) is a crucial algorithm within the Hierarchical Temporal Memory (HTM) framework designed to learn spatial patterns inspired by the neo-cortex. It is designed to learn the pattern in a few iteration steps and to generate the Sparse Distributed Representation (SDR) of the input. This SDR serves as a compact and efficient representation of the input data, capturing its essential spatial features. Through the process of spatial pooling, the SP identifies and activates a subset of neurons, known as mini-columns, based on the input's spatial patterns, thus facilitating further processing within the HTM network. The parameters in Table 1 were carefully chosen based on prior research and empirical experimentation to ensure optimal performance of the SP within our specific experimental setup.

Table 1: Set of parameters shown in the table that are used in experiments with the Spatial Pooler.

Spatial Pooler configurations	
Parameters	Values
CellsPerColumn	10
MaxBoost	10.0
DutyCyclePeriod	1000
MinPctOverlapDutyCycles	0.001
GlobalInhibition	False
NumActiveColumnsPerInhArea	20.48 (0.02 *1024)
PotentialRadius	30 (0.15 *200)
LocalAreaDensity	-1
ActivationThreshold	10
MaxSynapsesPerSegment	10 (0.01 *1024)
Random	ThreadSafeRandom (42)
StimulusThreshold	10
inputBits	200
numColumns	1024

Some of the parameters that influence the boosting in SP are discussed below

MaxBoost: This is a boosting parameter that ensures Homeostatic Plasticity effect. Boosting is a mechanism used to adjust the permanence values of synapses to promote the activation of underutilized columns in a SP. However, excessive boosting can lead to instability in the system by causing rapid and unpredictable changes in synaptic connections, potentially disrupting the learned patterns and impairing the system's ability to recognize and predict sequences.

By controlling the maximum boost allowed, the "maxBoost" parameter helps maintain stability in the system by preventing overly aggressive adjustments to synaptic connections. This parameter sets a limit on the magnitude of boosting, thereby restraining the degree of plasticity in the network. Fine-tuning the "maxBoost" parameter allows for balancing between adaptability and stability in the Homeostatic Plasticity Controller, ensuring that synaptic connections are sufficiently flexible to learn new patterns while avoiding destabilizing changes that may compromise the system's overall performance.

minPctOverlapCycles: This parameter represents a fractional value, typically ranging between 0 and 1.0. It acts as a threshold to enforce a minimal frequency for column activation during learning iterations. Periodically, each column evaluates the overlap duty cycles of neighboring columns within its inhibition radius, setting its own internal minimal acceptable duty cycle based on this analysis. If a column's overlap duty cycle falls below this computed threshold, it triggers a mechanism where all of its permanence values are boosted, allowing cells to explore new inputs in response to changing patterns or competition from other columns. This dynamic adjustment process ensures adaptability and efficiency in learning spatial patterns.

DutyCyclePeriod: This parameter determines the time period over which the activity of columns is measured. It represents the duration, in time steps, over which the average activation level of each column is calculated. This parameter helps regulate the stability of column activations by defining how often the activity of columns is updated and assessed. Adjusting this parameter can influence the responsiveness of the SP to changes in input patterns and contribute to maintaining a balanced level of column activity over time. Higher values make it take longer to respond to changes in boost, whereas shorter values make it more unstable and likely to oscillate.

C. *Column Inhibition*

Inhibition is the process of deactivating the fraction of mini-columns that tend to be active in the current learning cycle. This ensures that the encoding of learned patterns becomes sparse and not dense. Two distinct forms of inhibition exist: global and local inhibition. Global inhibition functions by deactivating active mini-columns across the entire mini-column space. Conversely, local inhibition targets the deactivation of columns within a locally defined neighborhood.

The local inhibition algorithm, a fundamental mechanism within the Hierarchical Temporal Memory (HTM) framework is employed in the spatial learning experiment. This algorithm plays a crucial role in processing sensory inputs by selectively activating mini-columns within a specified neighborhood.

The local inhibition algorithm scans through all mini-columns within the HTM area and exclusively triggers mini-columns located within the neighborhood of the current column. The getColumnNeighborhood function identifies all mini-columns within the vicinity of the presently computed mini-column. These selected columns fulfill the criteria of having an overlap greater than the predefined stimulus threshold, as well as surpassing the overlap of the currently computed column.

Figure 3 shows an example of local inhibition. The shaded circle indicates the sliding window, representing the local inhibition area, with the black dot denoting the calculating mini-column positioned at its center. The size of this local inhibition area is determined by the inhibition radius. When the stimulus threshold value is set to 4, all mini-columns within this window are highlighted if their overlap exceeds 4, including the overlap of the calculating mini-column marked in orange.

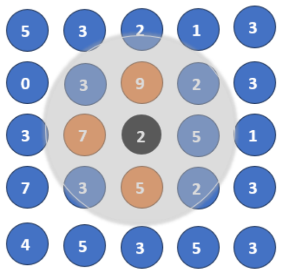


Figure 3: Illustration of Local Inhibition Algorithm with Neighborhood Selection.

D. *Homeostatic Plasticity Controller*

The Homeostatic Plasticity Controller (HPC) extends the default Spatial Pooler algorithm within the Hierarchical Temporal Memory framework. Its primary function is to initiate the SP in a newborn stage at the onset of the learning process. During this phase, the SP initiates boosting mechanisms to stimulate mini-columns, allowing for initial instability in the learning process. After a predefined number of iterations, the HPC deactivates boosting and awaits the SP to achieve stability. This approach enables the SP to converge to a stable state, enhancing the quality of learning and facilitating reliable solutions. Additionally, applications can be notified about the SP's state changes, improving adaptability and performance in various contexts.

Below discussed are the parameters that impact the Homeostatic plasticity effect in the SP.

numOfCyclesToWaitOnChange: It determines the number of cycles the SP should wait after a change in input before updating its internal state. This parameter set to 100 in this experiment, allows the SP to delay processing new input until stable input patterns emerge. By waiting for a specified number of cycles, the SP can avoid reacting to transient changes or noise in the input data, thus promoting stability in the representation. This approach helps ensure that the SP captures consistent and meaningful patterns over time, rather than responding to temporary fluctuations.

RequiredSimilarityThreshold: It sets a threshold for how closely a potential synapse's permanence value must match the active input bit's state (on or off) for the synapse to be considered connected. In this Experiment, this threshold is set to 10. If the similarity between the permanence value and the input bit exceeds this threshold, the synapse is strengthened, influencing the column's activation. Otherwise, the synapse remains unchanged or may weaken over time if it falls below the threshold. A higher threshold may lead to more selective synapse formation, requiring input patterns to closely match learned representations for columns to become active. Conversely, a lower threshold may allow for more generalization, with columns becoming active in response to a broader range of input patterns.

E. *Generation of SDR*

The Spatial Pooler algorithm contains three steps: initialization, column activation and synaptic permanence learning.

Let $E_t = [e_{1t}, e_{2t}, \dots, e_{nt}]$ denote the encoded binary input for x_t at time t. Here, $e_{it} \in \{0,1\}$ represents the i^{th} element of the encoding at time t. The vector E_t satisfies $\|E_t\|_1 = \omega$, where ω represents the number of "1" bits in E_t.

In the initialization stage, it sets the main parameters of HTM and establishes the synaptic connections and initializes their permanence values on the proximal dendritic segment. Later, in column activation, HTM calculates the overlap value for each column and the columns whose overlap value are among the top k values will be activated. The overlap value depends on the number of active connected proximal synapses within the column's receptive field. The overlap O_{C_i} is calculated as per equation 1:

$$\boldsymbol{O_{C_i}} = \left(\sum_{j=0}^{n} \boldsymbol{e_i^j} \times \hat{s}_i^j\right) \times \boldsymbol{b_i} \qquad ---(1)$$

where $\hat{s}_i^j \in \{0,\ 1\}$ is the connected synapse on C_i whose, permanence value exceeds the given threshold, C_i is column i. b_i is the boosting factor determined by the frequency of column activation.

In synaptic permanence learning within HTM, the Hebbian learning rule is employed. This rule adjusts the permanence values on active columns. Synaptic connections to active bits are strengthened by increasing their permanence value, while connections to inactive bits are weakened by decreasing their permanence.

After the generation of SDRs, it becomes essential to evaluate their similarity with previously generated representations. To accomplish this, Jaccard similarity algorithm is implemented. The SP transforms every input pattern into a Sparse SDR, which is essentially a set of active mini-column indices. denoted as Sk for the given pattern in iteration k. In each learning step of the same pattern, the similarity between the SDRs at step k and step k+1 is computed as per equation 2.

$$\boldsymbol{J(S_k, S_{k+1})} = \frac{|\boldsymbol{S_k} \cap \boldsymbol{S_{k+1}}|}{|\boldsymbol{S_k} \cup \boldsymbol{S_{k+1}}|} \qquad ---(2)$$

According to the Jaccard similarity concept, the similarity, denoted as J, is computed as the ratio of the intersection of the sets of active mini-columns in the SDRs generated at steps k and k+1 (denoted as S_k and S_{k+1} respectively) to the total number of unique active mini-columns observed across the two comparison steps. This means it evaluates the commonality between the sets (overlap) relative to the total unique elements across both sets, ensuring that unique features in each set are not disregarded. By incorporating both shared characteristics and distinct features, the Jaccard method provides a more nuanced understanding of similarity, making it suitable for comparing SDRs generated at different steps where both shared patterns and unique variations are relevant. The stability of the SP is determined by whether the resulting SDR for a specific pattern remains consistent over its entire duration. When this occurs, the similarity between all SDRs associated with the same pattern is 100%.

IV. IMPLEMENTATION

A. *Investigation of Slow activation of second half of dataset on the learning of Spatial Pooler*

It was observed that in the learning phase of SP (i.e., First 40 cycles), the Mini-columns for input > 50 are not activated. The CalcMiniColumnOverlap method calculates the overlap of a column, which is the number of synapses connected to active input bits within that column. The input is a binary feature vector of size 200 bits. The synaptic map is built by randomly choosing values ranging from 1 to 200 for each row without replacement. This process signifies the random selection of features from the binary feature vector. Overlap values that are lower than the 'stimulus threshold' are ignored.

In the learning phase of the SP, boosting and inhibition mechanisms together promote the emergence of sparse distributed representations that capture the statistical structure of input patterns. Boosting reinforces weak representations, while inhibition enforces competition and sparsity, collectively contributing to the robust encoding and recognition of patterns in complex data streams. Later the AdaptSynapses method Adapts the permanence values of the synapses based on the input vector, and the chosen columns' permanence values are increased for synapses connected to input bits that are turned on and decreased for synapses connected to input bits that are turned off.

The BoostColsWithLowOverlap method raises the permanence values of synapses associated with columns that exhibit insufficient overlap levels. Such columns are identified by having an overlap duty cycle (activation frequency) that drops too much below those of similar columns. As a result, the permanence values of synapses connected to these columns are boosted.

In the previous version of the SP's `SetPermanences` method, an incorrect indexing approach was identified where the loop index `i` was directly utilized to access the `connectedCounts` matrix. This approach implied a straightforward sequential mapping of values from the `perms` array to positions in the `connectedCounts` matrix as depicted in Figure 4 , resulting in no overlaps with the input vector, which proved to be inaccurate.

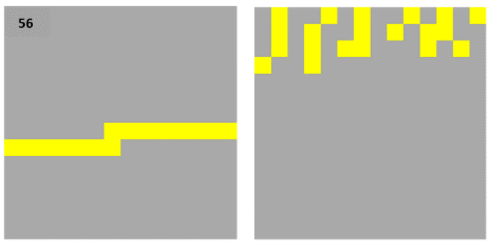


Figure 4 : Representation of input vector labelled as "56"(left) and the corresponding matrix displaying connected counts(right), which results in no overlaps.

To address this issue, a new 'SetUpdatedPermanences' method has been implemented to utilize the `inputIndexes` array for indexing. This adjustment ensures a more targeted and accurate mapping of values from `perms` to `connectedCounts`, as it considers specific indices provided in `inputIndexes`. By incorporating `inputIndexes[i]` as the index for accessing the `connectedCounts` matrix, as depicted in Figure 5, results in overlaps with the input vector. This revised method facilitates a more flexible and customized mapping, aligning with the intended functionality of the SP. This enhancement improves the overall accuracy and effectiveness of the SP's permanence setting mechanism, contributing to its reliability and performance within the project.

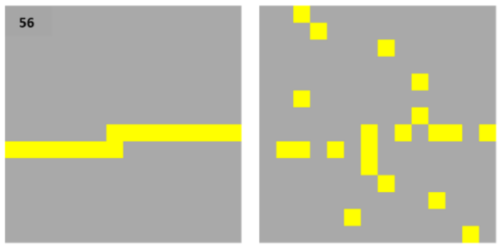


Figure 5 : Representation of input vector labelled as "56"(left) and the corresponding matrix displaying connected counts(right), which results in overlaps.

B. *Stability Analysis*

The experiments conducted involved varying configurations of parameters, specifically focusing on "numOfCyclesToWaitOnChange" and "MaxBoost" values. Throughout all experiments, Global Inhibition was consistently set to False, while other parameters like "MinPctOverlapDutyCycles" and "DutyCyclePeriod" remained fixed at 0.001 and 1000, respectively. The focus of the experiments was to analyze how the system's behavior varied in terms of stability when different parameter combinations were used. For each set of parameters, the experiments recorded various metrics, including the cycle when stability was first achieved, occurrences of unstable phases, and the final stability attainment.

Table 2 :Effects of MaxBoost and numOfCyclesToWaitOnChange on Stability.

numOf Cycles ToWait On Change	Max Boost	1st Stable Cycle	Stability Fluctuations			Final Stable cycle
			1st unstable cycle	2nd stable cycle	2nd unstable cycle	
50	5	308	345	395	484	534
	10	309	345	-	-	395
75	5	333	345	420	484	559
	10	334	345	-	-	420
100	5	445	484	-	-	584
	10	445	-	-	-	445

Table 2 shows the analysis of the results that reveals several key findings such as follows: The parameter "MaxBoost", which controls the boosting factor in the learning algorithm, exhibited a notable influence on stability. Higher values of "MaxBoost" contribute to enhanced stability by reducing the likelihood of the system entering an unstable state. The parameter “numOfCyclesToWaitOnChange”, representing the number of cycles to wait for stability after a change, also influenced stability dynamics. lower values of this parameter, causing more frequent status alterations, would be undesirable. Conversely, higher the values, despite resulting in a longer time for stabilization, would be preferable as they promote system stability. With MaxBoost value as 10 and numOfCyclesToWaitOnChange as 100, the system enters Stable state at cycle 445 and therefore no fluctuations in stability are observed. The exact cycle counts for stability attainment varied across different parameter configurations, suggesting a nuanced relationship between parameter settings and system stability.

This experiment provided valuable insights into the dynamics of the stability of the system. The findings underscored the importance of parameter tuning and highlighted the complex interplay between parameters and system behavior.

C. *Dictionary*

To efficiently handle data across multiple cycles, a simple dictionary structure is implemented. This dictionary allows us to associate each input with its corresponding SDR values over successive cycles. The input, represented by a scalar value serves as the key for the implemented SDR dictionary. Within this dictionary, the values consist of lists of integers, representing SDRs generated by the SP. During the learning phase, the SP generates diverse SDRs for identical inputs initially, but over time, it stabilizes these representations. These SDRs, arrayed with active elements, capture the essential features of the input.

The unique keys in the dictionary ensure that each input's SDR values are stored distinctly, facilitating independent access and manipulation. Its flexibility enables easy addition, updating, and deletion of entries, making it ideal for real-time applications where data manipulation is paramount. This approach not only provides a convenient means of organizing and accessing our data but also ensures the integrity and autonomy of each input's representation throughout the processing pipeline.

D. *Exiting upon Stability achievement*

The stability of the system is primarily influenced by two parameters: numOfCyclesToWaitOnChange and requiredSimilarityThreshold. To determine when the system has reached stability, a condition (if isInStableState) is implemented. Once this condition is set to true, the experiment checks for stability for an additional 100 cycles. Here, a counter “StableCyclesElapsed” has been initialized to increment or reset its value based on whether the system is stable or unstable, respectively. The condition (if StableCyclesElapsed == LoopBreakerThreshold) verifies if the counter has reached the specified threshold (number of stable cycles, in this case 100) to exit the experiment.

V. RESULTS

Below are the outcomes from the implementation of a newly designed spatial learning experiment:

Addressing slow activation of mini-columns: As discussed in preceding sections, the SP undergoes a learning process to decipher input patterns and produce Sparse Distributed Representations (SDRs). The introduction of the new “SetUpdatedPermanences” method successfully addressed the indexing issue identified within the Spatial Pooler algorithm. Through accurate indexing, the SP now demonstrates improved performance in generating SDRs for all the inputs in the learning phase.

To visualize the impact of the implemented solution, Figure 6 (left) depicts the representation of SDRs before the implementation of the “SetUpdatedPermanences” method. Noticeably, mini-columns associated with inputs greater than 50 exhibit no overlaps resulting in no SDRs, indicative of the indexing issue affecting the SP's learning phase.

After the implementation of the new “SetUpdatedPermanences” method, Figure 6 (right) showcases the revised representation of SDRs. In contrast to the previous scenario, mini-columns now exhibit overlaps, particularly for inputs exceeding 50. This improvement highlights the efficacy of the implemented solution in addressing the indexing issue and enhancing the SP's learning capabilities.

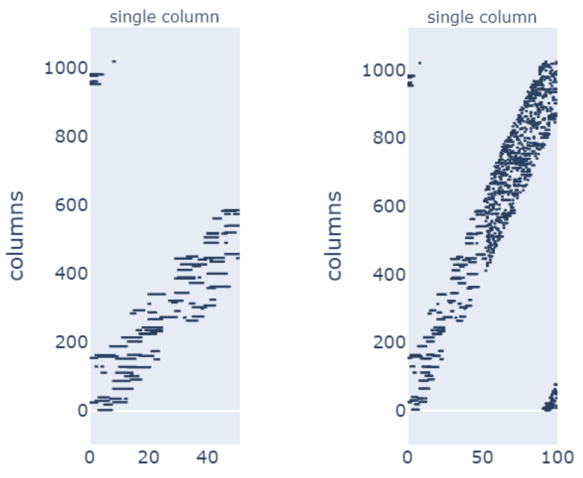


Figure 6: Representation of SDRs generated for only input 0 to 50 due to incorrect indexing(left) and SDRs generated for input 0 to 99 with proper indexing (right). The horizontal axis shows the index of the input. The vertical axis shows the SDR. Every blue dot represents the active mini column.

Ensuring consistent stability: In the current Spatial Learning experiment, it has been demonstrated that once the system achieves a stable state, no further changes are observed in the SDRs which results in stability of the system. This is illustrated graphically in Figure 7, where the stability remains constant after the variable “isInStableState” is set to true. The experiment is designed to exit after confirming the absence of SDR changes for a specified number of iterations, in this case, 100 cycles. This ensures that the system maintains its stable state without any fluctuations, thus validating the effectiveness of the implemented solution.

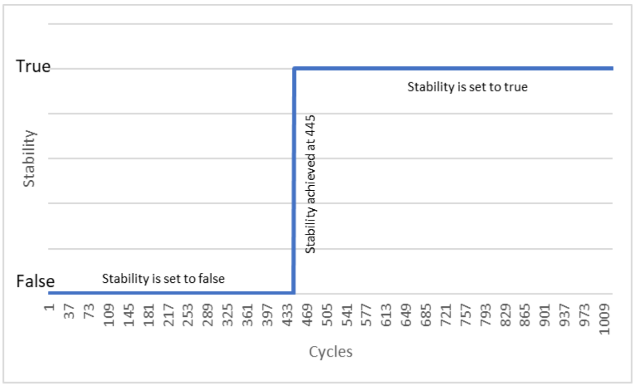
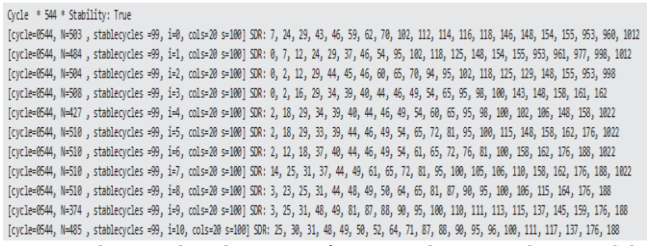


Figure 7 :Stability trends over 1000 cycles with Stable State reached at cycle 445 and constant behavior thereafter.

Improved output visualization: All input patterns enter the stable state at different iteration steps. To understand the behavior of individual input patterns, an additional parameter has been introduced denoted by "N", representing the number of iterations during which the SDR for the input remained unchanged or stable. Once the system attains the stable state, the following 100 stable cycles are printed. Also, the SDRs for input 0 from the initial cycle to the last one are displayed for visual comparison, allowing for analysis and providing deeper insights into the stability of the SP.

As depicted in the Figure 8, at cycle 544, the SP processed input index 0, generating SDR with 20 active mini-columns. This SDR remained stable for 503 iterations represented by "N", with a perfect similarity score of 100. Moreover, there were 99 consecutive stable cycles leading up to the current one, suggesting a period of relative stability in the system's behavior. Overall, this output provides insight into the SP's processing of input data, showcasing its ability to generate stable and informative representations through sparse activation of mini-columns.



```
Cycle  * 544 * Stability: True
[cycle=0544, N=503 , stablecycles =99, i=0, cols=20 s=100] SDR: 7, 24, 29, 43, 46, 59, 62, 70, 102, 112, 114, 116, 118, 146, 148, 154, 155, 953, 960, 1012
[cycle=0544, N=484 , stablecycles =99, i=1, cols=20 s=100] SDR: 0, 7, 12, 24, 29, 37, 46, 54, 95, 102, 118, 125, 148, 154, 155, 953, 961, 977, 998, 1012
[cycle=0544, N=504 , stablecycles =99, i=2, cols=20 s=100] SDR: 0, 2, 12, 29, 44, 45, 46, 60, 65, 70, 94, 95, 102, 118, 125, 129, 148, 155, 953, 998
[cycle=0544, N=508 , stablecycles =99, i=3, cols=20 s=100] SDR: 0, 2, 16, 29, 34, 39, 40, 44, 46, 49, 54, 65, 95, 98, 100, 143, 148, 158, 161, 162
[cycle=0544, N=427 , stablecycles =99, i=4, cols=20 s=100] SDR: 2, 18, 29, 34, 39, 40, 44, 46, 49, 54, 60, 65, 95, 98, 100, 102, 106, 148, 158, 1022
[cycle=0544, N=510 , stablecycles =99, i=5, cols=20 s=100] SDR: 2, 18, 29, 33, 39, 44, 46, 49, 54, 65, 72, 81, 95, 100, 115, 148, 158, 162, 176, 1022
[cycle=0544, N=510 , stablecycles =99, i=6, cols=20 s=100] SDR: 2, 12, 18, 37, 40, 44, 46, 49, 54, 61, 65, 72, 76, 81, 100, 158, 162, 176, 188, 1022
[cycle=0544, N=510 , stablecycles =99, i=7, cols=20 s=100] SDR: 14, 25, 31, 37, 44, 49, 61, 65, 72, 81, 95, 100, 105, 106, 110, 158, 162, 176, 188, 1022
[cycle=0544, N=510 , stablecycles =99, i=8, cols=20 s=100] SDR: 3, 23, 25, 31, 44, 48, 49, 50, 64, 65, 81, 87, 90, 95, 100, 106, 115, 164, 176, 188
[cycle=0544, N=374 , stablecycles =99, i=9, cols=20 s=100] SDR: 3, 25, 31, 48, 49, 81, 87, 88, 90, 95, 100, 110, 111, 113, 115, 137, 145, 159, 176, 188
[cycle=0544, N=485 , stablecycles =99, i=10, cols=20 s=100] SDR: 25, 30, 31, 48, 49, 50, 52, 64, 71, 87, 88, 90, 95, 96, 100, 111, 117, 137, 176, 188
```

Figure 8: The Spatial Pooler's output for input index 0 at cycle 544 and the resulting SDR.

Figure 9 (left) depicts that initially, during cycle 0, SDRs of inputs are less active, reflecting the system's early learning phase. As the process progresses towards the last cycle, there's an increase in active columns within the SDRs, as shown in Figure 9 (right) indicating greater stability. This suggests that the system has learned and adapted to the input data over time, resulting in more robust and refined representations. Furthermore, the stability achieved by the SP in later cycles showcases the equilibrium in the distribution of active mini-columns across subsequent input presentations.

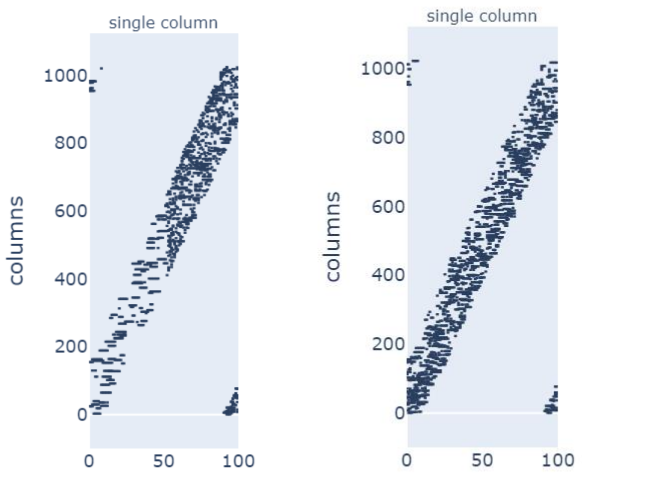
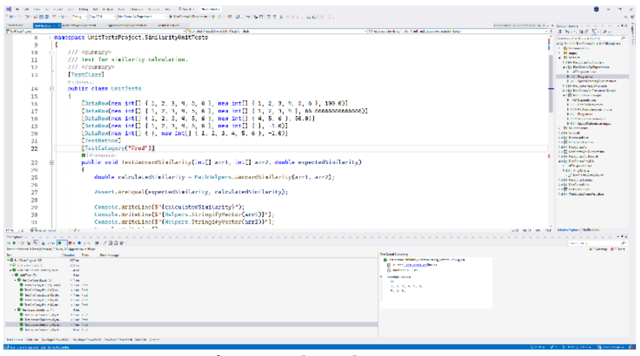


Figure 9 : Representation of SDRs for all inputs at cycle 0 (left) and last stable cycle 544 (right) The horizontal axis shows the index of the input. The vertical axis shows the SDR. Every blue dot represents the active mini column.

Unit Testing: The unit tests successfully validate the functionality of both methods under various conditions. For the "AreArraysEqual" method, tests cover cases where arrays are identical, have different elements, or are null. Similarly, for the "JaccardSimilarity" method, tests evaluate scenarios with arrays of different lengths, identical arrays, and arrays with partial overlap, and arrays with full overlap. The tests ensure that the methods perform as expected, providing accurate results across diverse input data. Figure **10** represents the Unit Testing result for Jaccard Similarity.



```
namespace UnitTestsProject.SimilarityUnitTests
{
    /// <summary>
    /// Test for similarity calculation.
    /// </summary>
    [TestClass]
    public class UnitTests
    {
        [DataRow(new int[] { 1, 2, 3, 4, 5, 6 }, new int[] { 1, 2, 3, 4, 5, 6 }, 100.0)]
        [DataRow(new int[] { 1, 2, 3, 4, 5, 6 }, new int[] { 1, 2, 3, 4 }, 66.6666666666666)]
        [DataRow(new int[] { 1, 2, 3, 4, 5, 6 }, new int[] { 4, 5, 6 }, 50.0)]
        [DataRow(new int[] { 1, 2, 3, 4, 5, 6 }, new int[] { }, -1.0)]
        [DataRow(new int[] { }, new int[] { 1, 2, 3, 4, 5, 6 }, -1.0)]
        [TestMethod]
        [TestCategory("Prod")]
        public void TestJaccardSimilarity(int[] arr1, int[] arr2, double expectedSimilarity)
        {
            double calculatedSimilarity = MathHelpers.JaccardSimilarity(arr1, arr2);

            Assert.AreEqual(expectedSimilarity, calculatedSimilarity);

            Console.WriteLine($"{calculatedSimilarity}");
            Console.WriteLine($"{Helpers.StringifyVector(arr1)}");
            Console.WriteLine($"{Helpers.StringifyVector(arr2)}");
```

Figure 10: Unit Testing for Jaccard Similarity.

VI. CONCLUSION

The new Spatial Learning experiment is the improved version of the Spatial Pattern Learning Experiment. The transition from the old 1000-cycle methodology to the new approach, which terminates once stability is confirmed and sustained for an additional 100 cycles, is a notable efficiency improvement. This change conserves computational power by stopping unnecessary runs after stability, saving valuable time in time-sensitive research projects. The investigation of slow activation of dataset in the learning phase of SP has been addressed and resolved with a change in indexing that proved to have made the system more effective. Jaccard similarity is implemented to compare SDRs, effectively capturing both shared patterns and unique variations, providing a nuanced understanding of similarity. Certain essential parameters were fine-tuned to ensure the system retains stability without any disturbance in the future data. The output has been demonstrated appropriately showcasing relevant parameters. Overall, this experiment proves to have enhanced the efficiency of the earlier version of the experiment.

VII. References

[1] R. B. M. D. M. Daniel E. Padilla, "Performance of a Hierarchical Temporal Memory Network in Noisy Sequence Learning," *Institute for Telecommunications Research,University of South Australia,* pp. 45-47, 2013.

[2] Z. W. T. C. L. L. J. J. Y. C. Z. L. Dejiao Niu1, "A New Spatial Pooler Algorithm Based on Heterogeneous Hash Group," in *International Joint Conference on Neural Networks (IJCNN),* 2023.

[3] T. I. P. J. Olga Krestinskaya, "Hierarchical Temporal Memory Features with Memristor Logic Circuits for Pattern Recognition," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS,* vol. 37, pp. 1143-1145, 2018.

[4] S. A. J. H. Yuwei Cui, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," in *Frontiers in Computer Neuroscience,* 2017.

[5] L. X. K. R. W. I. Mackenzie Leake, "Effect of Spatial Pooler Initialization on Column Activity in Hierarchical Temporal Memory," in *29th AAAI Conference on Artificial Intelligence*.

[6] A. P. B. G. T. W. Damir Dobric, "Improved HTM Spatial Pooler with Homeostatic Plasticity Control," in *10th International Conference on Pattern Recognition Applications and Methods*, 2021.