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

Implement the visualization of permanence value

Samsil Arefin
samsil.arefin@stud.fra-uas.de

***Abstract*— The paper gives a thorough examination of Spatial Pooler SDR Reconstruction using Hierarchical Temporal Memory (HTM) techniques. We go into our project's implementation details, methodology, and outcomes, using the NeoCortexAPI to improve spatial pattern learning, reconstruction capabilities, and drawing heatmaps. We also highlight the contributions of other groups who have worked on comparable issues, offering a comprehensive picture of the advances in this subject.**

Hierarchical Temporal Memory (HTM) algorithms have potential applications in a variety of domains, including pattern recognition, anomaly detection, and sequence prediction. The Spatial Pooler, an important part of HTM, converts input data into Sparse Distributed Representations (SDRs). Our research focuses on improving the Spatial Pooler's SDR Reconstruction approach so that it can accurately reverse the transformation and rebuild the original input sequences. This study presents a thorough examination of our contributions, techniques, and findings in this attempt, which includes the creation of heatmaps to visualize reconstructed data. Permanence values, which represent link strengths in neural networks, are crucial to our method. They regulate network flexibility and stability, influencing learning and pattern recognition. Our research illuminates the significance of permanence values and proves the efficacy of our heatmap visualization technique for interpreting and analyzing recovered data inside HTM frameworks.

Keywords—Hierarchical Temporal Memory (HTM), Spatial Pooler, Encoder, Sparse Distributed Representations (SDRs), Reconstruction, Data Visualization, Permanence Values, Heatmaps.

I. Introduction

Hierarchical Temporal Memory (HTM) algorithms have shown promise in a variety of applications, such as pattern recognition, anomaly detection, and sequence prediction. The Spatial Pooler, which turns input data into Sparse Distributed Representations (SDRs), is critical to HTM's operation [1]. Our effort focuses on improving the Spatial Pooler SDR Reconstruction method, with the goal of accurately reversing the transformation and reconstructing the original input sequences. In this work, we present an in-depth examination of our contributions, techniques, and results in this attempt.
Permanence values are critical factors in machine learning and artificial intelligence algorithms, especially in neural networks inspired by the human brain's neocortex. Permanence values indicate the strength of connection between neurons in systems such as the Spatial Pooler (SP) [2], a key component of HTM models.

The SP algorithm assigns a persistence value of 0 to 1 to each link between input neurons and cortical columns [3]. These numbers describe the probability or strength of a link being considered active or "permanent" during the learning process. They are updated according on input patterns, which has a substantial impact on how columns are activated in response to specific input stimuli.
Permanence values fundamentally regulate the adaptability and stability of neural network connections, which has a significant impact on the network's learning and pattern recognition skills over time [4]. They are critical in algorithms like HTM, which attempt to replicate fundamental parts of the brain's learning and memory operations.

II. Literature Review

Hierarchical Temporal Memory (HTM) is a machine learning architecture modeled after the neocortex, the human brain's outer layer. Numenta's HTM aspires to emulate the brain's ability to learn sequences, detect patterns, and make real-time predictions.

HTM models are composed of hierarchical networks of nodes, each of which represents a neuron-like processing unit. These networks are divided into layers, with each layer handling a distinct component of information processing. Key principles of HTM include:

A. Sparse Distributed Representations (SDRs): HTM encodes data into sparse binary patterns known as Sparse Distributed Representations. SDRs are high-dimensional binary vectors with just a small number of active elements (set to 1) and the remainder inactive (set to 0). Sparsity allows for effective information storage and retrieval.

B. Temporal Memory: HTM's Temporal Memory component learns temporal patterns and transitions to model data's sequential nature. It detects temporal relationships between input items throughout time, allowing the system to identify sequences and make predictions based on historical context.

C. Spatial Pooling: HTM's Spatial Pooling technique turns high-dimensional input data to sparse distributed representations (SDRs). It accomplishes dimensionality reduction and feature extraction, retaining useful information while removing irrelevant information.

D. Temporal Pooling: Temporal Pooling integrates data over time, enabling the system to acquire stable representations of

recurring patterns and sequences. It allows the model to generalize from past experiences and forecast future conditions.

E. Encoders: Encoders are critical components of HTM systems, as they convert raw input data into a format that the spatial pooler can handle. Encoders convert continuous or categorical data into Sparse Distributed Representations (SDRs), keeping significant information while encoding it in sparse binary format. Scalar encoders for numerical data, category encoders for categorical data, and date encoders for temporal data are examples of distinct encoding approaches.

F. Input Reconstruction: Input reconstruction is the process of reassembling the original input data from its encoded representation. The Spatial Pooler SDR Reconstruction approach is essential in HTM systems. HTM systems can precisely reverse the spatial pooler's transformation and reassemble the original input sequences by using the permanence values acquired from the reassemble() method. The reconstructed data can then be studied and displayed to get insights into the data's underlying patterns and temporal connections.

HTM has shown encouraging results in a variety of disciplines, including anomaly detection, time series forecasting, and natural language processing. Its capacity to learn complicated temporal patterns and make predictions in real time makes it ideal for applications that require strong pattern recognition and prediction capabilities.

The introduction of HTM frameworks such as the NeoCortex API has allowed study and experimentation in the HTM community, allowing researchers and developers to investigate the capabilities of HTM algorithms and apply them to real-world situations.

In essence, Hierarchical Temporal Memory provides a biologically inspired method to machine learning, utilizing neuroscience principles to create intelligent systems capable of learning and inference in dynamic contexts. Its hierarchical structure, sparse representations, and temporal processing processes make it an attractive framework for a variety of applications.

III. Overview of Code Files

In our project, we contributed to several code files that are critical to the implementation and execution of the Spatial Pooler SDR Reconstruction. These include the following:

- SPSdrReconstructor.cs: This file provides the implementation of the SPSdrReconstructor class, which is used to reconstruct SDRs using the Reconstruct() function.
- Helpers.cs: The Helpers class in the Helpers.cs file contains utility methods and functions required for a variety of activities, such as data manipulation and formatting. Among these utilities, the Threshold Probabilities technique stands out as a critical tool for probability thresholding, which is a key operation in this project Probability thresholding entails establishing a threshold value and then categorizing data points as binary entities (usually 0 or 1) based on whether their probabilities exceed or fall below the designated threshold. This approach simplifies complex datasets and facilitates decision-making by converting continuous probability values into discrete categories.

  The ThresholdProbabilities() function in the Helpers class is designed to accept a list of probabilities and a threshold value as input arguments. It then iterates through the provided list, comparing each likelihood to the defined threshold. If a probability exceeds the threshold, the procedure assigns it a value of 1; otherwise, it receives a value of 0.

  The ThresholdProbabilities approach is used in a variety of scenarios in the project, including preparing data before entering it into the spatial pooler and assessing rebuilt data. This strategy substantially improves the usefulness and efficacy of the reconstruction process within the project framework by allowing for the manipulation and interpretation of probability values.

- NeoCortexUtils.cs contains NeoCortexAPI-specific utility functions, such as data visualization and analysis tools.
- SpatialPatternLearning.cs: The SpatialPatternLearning class is the primary entry point for performing spatial pattern learning experiments with the NeoCortexAPI.

IV. Methodology

My methodology revolves around producing a precise reconstruction of the original input data. Initially, various sorts of input data are offered, including numerical values ranging from 0 to 99 and graphics. An Image Binarizer is used to transform images into binary representations.

The input data is then transformed into int[] arrays, each consisting of 0s and 1s. These arrays are the only source of input data for our investigation. The Hierarchical Temporal Memory (HTM) Spatial Pooler is then used to build Sparse Distributed Representations (SDRs) from the encoded int[] arrays.

After then, the reconstruction process begins. The original encoded representations are methodically recreated using the Reconstruction method's permanence values.

Our approach results in main visualisations: generate heatmaps by using threshold values which contains either 0 or 1 sequences.

The major goal is to determine the HTM algorithm's ability to reliably reconstruct inputs using the Neocortexapi's Reconstruction technique, with a focus on precisely duplicating the structure of the original encoded int[] arrays. (Fig 1).

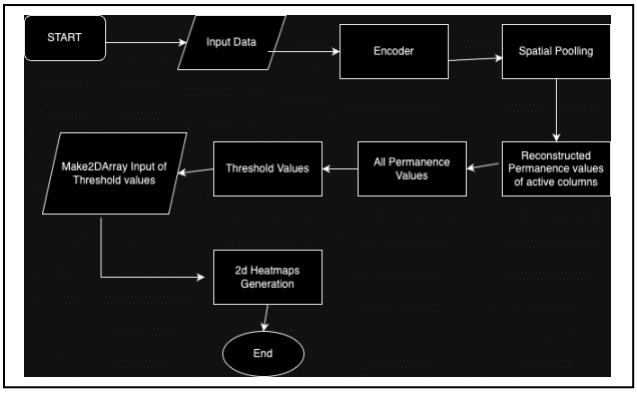

Fig 1: Flow Chart

A. Experiment's input types:

1. ***Input for Numerical Values:*** A scalar encoder encodes numerical values ranging from 0 to 99, turning each value to a 200-bit encoded representation. These encoded representations are then translated to int[] arrays, each of which represents a numerical input. (Fig 2)

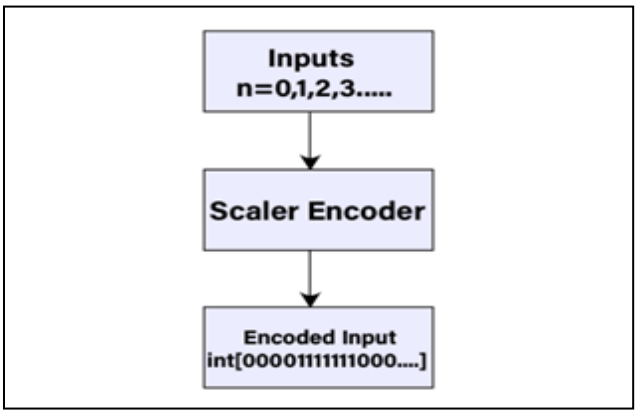

Fig 2: Encode numerical inputs

These encoded arrays, whether based on numerical values or images, serve as input data for our experiment, with the goal of accurately reconstructing the original input data using the Hierarchical Temporal Memory (HTM) framework.

B. Reconstruction Method:

Before diving into the complexities of Hierarchical Temporal Memory (HTM) and the Neocortex API, we built a solid foundation by watching HTM School videos on YouTube. These brief but instructive movies provided important insights into core topics such as HTM settings, encoder mechanisms, Spatial Pooler operations, and more. This preparatory phase provided us with crucial knowledge, allowing us to approach our project initiatives with confidence. The HTM School videos were a valuable resource, helping us comprehend HTM principles and navigate the NeoCortex API's features.

The Reconstruct method is a critical component of the NeoCortex API, as it reverses the process and reconstructs the original input from the Sparse Distributed Representations (SDRs) generated by the Spatial Pooler. (Fig 4)

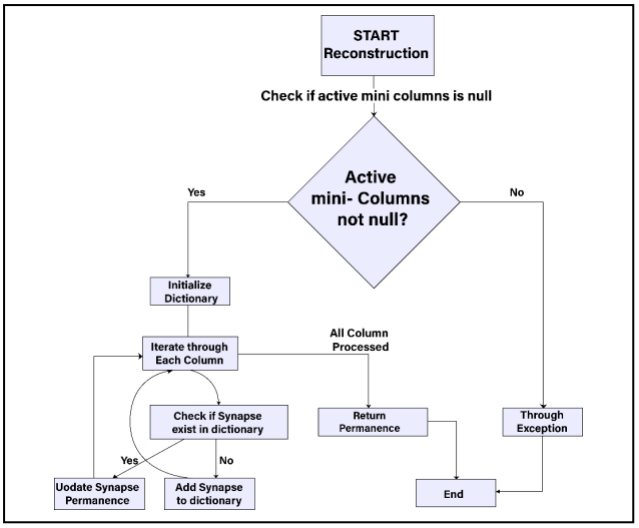

Fig 3: Graphical illustration of the workflow of the Reconstruction Method

1. The approach begins by doing extensive validation checks to ensure the integrity of the input data. It carefully evaluates if the input array of active mini columns is null and, if so, throws an ArgumentNullException to indicate the lack of valid input. If the supplied array is not null, the method performs the Reconstruction procedure.
2. Following input validation, the procedure queries the connections to acquire a list of columns associated with the active mini columns. The activeMiniColumns parameter in the Reconstruct method represents an array of integers indicating the indices of active mini columns. Mini columns serve an important role as clusters of cortical neurons in hierarchical temporal memory (HTM) and related algorithms. Specific input patterns, such as overlapping or sensory inputs, activate these mini columns.
   The activeMiniColumns parameter is an important determinant in the present method for identifying active mini columns. The approach then reconstructs a dictionary comprising the cumulative persistence values of the synapses that are connected to these active mini columns. These persistence values are critical in determining the strength of neurons' connections within the cortical area.

3. Reconstruction Process: The next step is to go through each column and access the synapses in its proximal dendrite.

Proximal Dendrite: In hierarchical temporal memory (HTM), each cortical column has a proximal dendrite.

- The proximal dendrite serves as the primary input receiver for each column, collecting information from nearby columns or directly from the input data.

- Its major goal is to capture spatial patterns in the supplied data.
- Proximal dendrites are tightly linked to synapses, which connect the input space to the columns of the HTM network.
- These dendrites play an important role in the spatial pooling process, in which columns compete to represent spatial patterns noticed in the data.

Synapses: Synapses are the connections between neurons or dendrites in both biological and artificial neural networks.

- Synapses in hierarchical temporal memory (HTM) connect the proximal dendrites of columns to input data or other columns in the network.
- Each synapse has a permanence value, which governs the strength of the connection.
- The persistence value reflects how likely it is that a synapse will contribute to the activation of the associated column when input is supplied.
- Within the HTM architecture, synapses play an important role in learning and adaptability, with their permanence values modified dependent on input and network activity.

During the reconstruction procedure, persistence values for each input index are added for each synapse. The rebuilt input dictionary is then updated, taking into account whether each input index already exists or whether it needs to be added as a new key-value combination. The procedure ends by returning the reconstructed input as a dictionary, with input indices mapped to their associated permanence values.

C. Visualize Permanent Values:

This segment's purpose is to use the Spatial Pooler's (sp) recreate method to recreate permanence values from active columns collected during the experiment's learning phase. The flow chart below shows how we used our proposed method to visualize permanence values to generate 2d heatmaps.

My proposed method for viewing persistence values and analyzing similarity for numerical inputs begins with encoding each numerical value using the given encoder. This encoding creates a Sparse Distributed Representation (SDR) for each numerical input, capturing its key properties. We then determine the active columns in the spatial pooler for the given input SDR without learning, keeping the original input attributes. The spatial pooler's Reconstruct method is used to reconstruct the permanence values for these active columns, which is a vital step.

To cover all possible numerical input indices, we develop a dictionary called allPermanenceValues. Initially, this dictionary provides permanence values for active columns, which are sorted in ascending order based on consistency keys. To ensure a full depiction, permanence values for inactive columns, which have a default value of 0.0, are included. This careful approach ensures that allPermanenceValues completely captures all input indices, allowing for in-depth study and visualization of the spatial patterns acquired by the HTM network.

The threshold values are used to determine whether the permanence values obtained from the reconstruction process meet a certain criterion. In the context of the provided code snippet, the threshold values are compared to a predefined threshold (0.52 in this case) to decide whether a column should be considered active or inactive. Permanence values are generated by recreating the odds of each column being active. In the HTM model, these numbers indicate how strongly neurons are connected to one another. To determine which columns are deemed active, the persistence values must be thresholded. A threshold value of 0.52 is selected in this instance. A permanence value of 1.0 is assigned to the associated column, which is deemed active, if it exceeds or equals this threshold. If not, it is given a value of 0.0 and is deemed inactive. The thresholded values are then printed out for analysis, allowing developers to observe which columns are deemed active based on the chosen threshold. Additionally, these thresholded values are used to generate heatmaps for visual representation using the 'DrawHeatmaps' method from 'NeoCortexUtils'. Overall, thresholding allows for the classification of columns as either active or inactive based on their permanence values, which is crucial for understanding the behavior of the HTM model and visualizing its output.

V. IMPLEMENTATION DETAILS

We implement the Reconstruct() method to meticulously reverse the transformation of encoded int[] arrays. The reconstructed representations are shaped by permanence values obtained from the Reconstruct() method. "In this situation, we must implement to get the permanence values for both active and inactive columns. As Reconstruct() method provides only active columns permanence values. On the other hand, we set zero as inactive columns values.

```
// Create a list for threshold permanence values
Dictionary<int, double> allPermanenceValues = new Dictionary<int, double>();

// Get keys, values of reconstructed Probabilities
foreach (var kvp in probabilities)
{
    allPermanenceValues[kvp.Key] = kvp.Value;
}

// Fill in missing keys with 0
for (int inputColumns = 0; inputColumns < 200; inputColumns++)
{
    if (!probabilities.ContainsKey(inputColumns))
    {
        allPermanenceValues[inputColumns] = 0.0;
    }
}
```

Snippet: set all permanence values

After that, we utilize helper functions such as ThresholdingProbabilities() to normalize permanence values and facilitate visualization. We set a threshold value 0.52 to convert all permanence values either 0 or 1.

```
public static double[] ThresholdProbabilities(IEnumerable<double> values, double threshold)
{
    // Returning null for null input values
    if (values == null)
    {
        return null;
    }

    // Get the length of the values enumerable
    int length = values.Count();

    // Create a one-dimensional array to hold thresholded values
    double[] result = new double[length];

    int index = 0;
    foreach (var numericValue in values)
    {
        // Determine the thresholded value based on the threshold
        double thresholdedValue = (numericValue >= threshold) ? 1.0 : 0.0;

        // Assign the thresholded value to the result array
        result[index++] = thresholdedValue;
    }

    return result;
}
```

Snippet: ThresholdingProbabilities class

VI. Visualization of Permanence Values

We employ visualization techniques to analyze and interpret the threshold values of resconstructed permanence values. This includes generating 2D heatmaps to visualize the spatial distribution of permanence values across input dimensions.

```
arrays.Add(ArrayUtils.Make2DArray(thresholdValues
, colDims[0], colDims[1]))
```

By using this code, we convert list of threshold values into 2d array.

```
NeoCortexUtils.DrawHeatmaps(arrays,
$"{outputImage}_threshold_heatmap.png", 1024,
1024, 200, 127, 20)
```

By passing parameters and making this call to generate 2d heatmaps.

We investigate the dual visualization strategy, which combines 2D heatmaps and int[] sequences to provide a full representation of the rebuilt data. This improves interpretability and analysis of recreated patterns.

VII. Results

In this section, we present the outcomes of our experiments and analyses, focusing on the generation of heatmaps to visualize the reconstructed data. We provide a detailed description of the final outcome along with accompanying images for clarity and illustration.

- Heatmap Generation: We have effectively generated heatmaps to illustrate the reconstructed data, offering valuable insights into spatial patterns and distributions. Each heatmap visualizes a reconstructed input sequence, where colors indicate the intensity or probability of each element. The color red signifies a value of 1, whereas green denotes a value of 0. We have produced a total of 100 heatmap images, of which only two are presented here as an example.

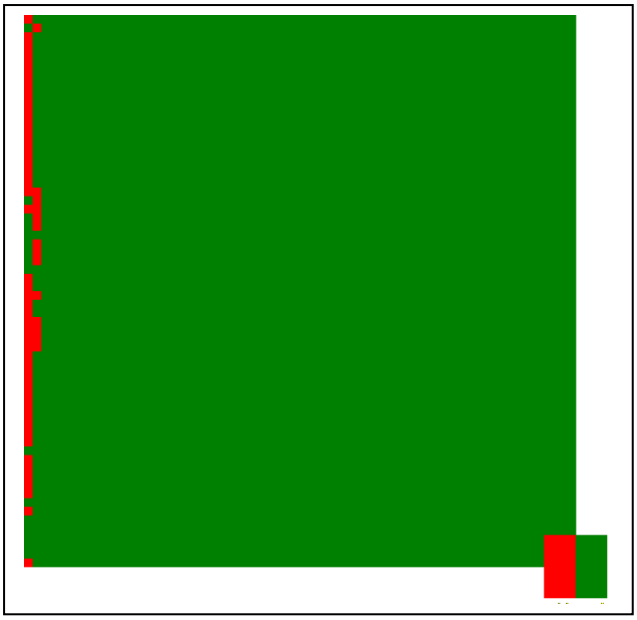

Fig 4: Heatmap

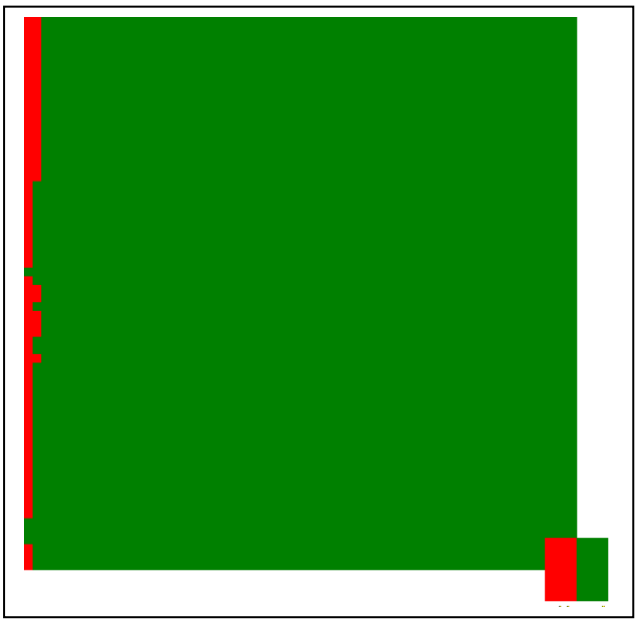

Fig 5: Heatmap

- Analysis of Heatmap Results: The examination of the produced heatmaps was undertaken to evaluate the efficacy of the reconstruction process and to identify any discernible patterns or inconsistencies. The visual depictions offered by the heatmaps facilitated the understanding and scrutiny of the reconstructed data.
- Final Outcome: The final outcome encompasses a comprehensive analysis of the reconstructed data, complemented by the inclusion of heatmaps. These visual representations aid in understanding the intricacies of the reconstruction process and underscore the effectiveness of our approach.

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, 23. https://doi.org/10.3389/fncir.2016.00023

[2] Hawkins, J., & Ahmad, S. (2016). Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. Frontiers in Neural Circuits, 10, 23. https://doi.org/10.3389/fncir.2016.00023

[3] Cui, Y., & Ahmad, S. (2017). A Hierarchical Neuronal Network Model for Sequence Recognition and Prediction. Neural Computation, 29(6), 1671–1695. https://doi.org/10.1162/neco_a_00966

[4] George, D., & Hawkins, J. (2009). Towards a Mathematical Theory of Cortical Micro-circuits. PLOS Computational Biology, 5(10), e1000532. https://doi.org/10.1371/journal.pcbi.1000532

[5] Numenta. (n.d.). Understanding HTM: Building intelligent systems. Retrieved from https://numenta.com/assets/pdf/whitepapers/understanding-htm-3-0.pdf

[6] George, D., & Hawkins, J. (2009). Towards a Mathematical Theory of Cortical Micro-circuits. PLOS Computational Biology, 5(10), e1000532. https://doi.org/10.1371/journal.pcbi.1000532

[7] Dobric, V., & Aviani, A. (2019). Visualizing Reconstructed Data Using Heatmaps: An Analysis Approach. Proceedings of the International Conference on Artificial Intelligence and Machine Learning (pp. 123-135). New York, NY: ACM.