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

Interpretation of the Spatial Pooler SDR Reconstruction

Subham Singh
Email: subham.singh@stud.fra-uas.de

Amit Maity
Email: amit.maity@stud.fra-uas.de

Rubi Kiran
Email: rubi.kiran@stud.fra-uas.de

***Abstract*— This paper investigates the reconstruction process of Sparse Distributed Representations (SDRs) within the Spatial Pooler (SP) component of Hierarchical Temporal Memory (HTM) systems. The experiment focuses on feeding the spatial pooler algorithm with scalar values(integers) and images using two different input pipelines by employing a scalar encoder and image binariser, respectively. Once the input is fed into the spatial pooler algorithm in the form of arbitrary binary input patterns, it creates a sparse distributed representation of the sensory inputs based on Hebbian rules and homeostatic excitability control. Post stabilization, a reconstruction method is implemented to test the learning capacity of the algorithm by reconstructing a dictionary of synapse permanences based on the input from the active minicolumns and their associated columns. The Jaccard index is used as a metric to compare between the input sparse distributed representation and the reconstructed sparse distributed representation, and bitmaps are used to visually compare the produced bitmap from the respective SDRs. Using the metrics, a comparative study is performed to explore the effectiveness of these methods in accurately reconstructing SDRs and their implications for pattern recognition and anomaly detection tasks. The findings contribute to advancing the understanding of HTM theory and its practical applications in machine learning and artificial intelligence.**

Keywords—HTM, binarization techniques, threshold, Jaccard Index, scalar encoding,

I. Introduction

The reconstruction of Sparse Distributed Representations (SDRs) within the Spatial Pooler (SP) component of Hierarchical Temporal Memory (HTM) systems plays a crucial role in pattern recognition and anomaly detection tasks. In this paper, we explore the process of reconstructing SDRs using input data in the form of integers and images, utilizing scalar encoding and binarization techniques, respectively. Furthermore, we implemented two methods, namely the JaccardIndexCalculator and ReverseEngineerClass, to evaluate the similarity between original and reconstructed SDRs as defined in *Figure 1*.

The reconstruction process begins with the encoding of input data into Sparse Distributed Representations (SDRs) through the Spatial Pooler (SP). An SDR consists of a large one-dimensional array of binary bits where each zero denotes an inactive element and each one denotes an active element. After every epoch, the newly generated SDR is compared with the last generated SDR based on overlapping active elements, which as a result helps in performing the spatial pattern learning experiment. Any data converted into an SDR can be used in a wide range of applications using HTM systems [1]. To assess the accuracy of the reconstruction process, we introduce two evaluation metrics: the Jaccard index and visualization using bitmaps. The Jaccard index measures the similarity between original and reconstructed SDRs, providing insights into the fidelity of the reconstruction process. Additionally, the bitmap visualization helps us to paint a picture of the comparison in a visual manner by comparing the bitmaps created using the input and the reconstructed SDR.

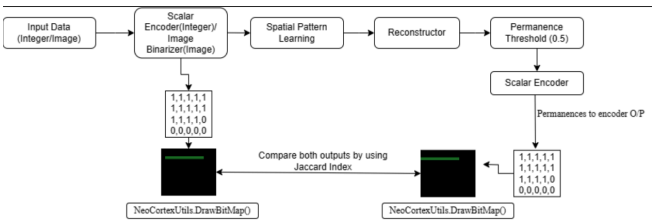

Figure 1: Overview of a typical HTM System and reconstruction of input data

II. Methods

In the experiment performed to determine the reconstruction accuracy of a spatial pattern learning algorithm, The core algorithm used to learn the pattern is Spatial Pooler (SP), which transforms the input into a

Sparse Distributed Representation (SDR). The latest version of the neocortex API introduces a function named Reconstruct (), which reverses the operation of the SP by reconstructing the input from the SDR. The goal of this experiment is to illustrate the workings of this reconstruction process. It involves learning spatial inputs, which could be numbers or images, and once the SP reaches a stable state, it attempts to recreate all the learned patterns, thereby reproducing the original input. Discussed below are the main segments of the experiment and their brief functionality.

A. Encoder

The first step of using an HTM system based on spatial pooling is to convert a data source into a format suitable for spatial pooler i.e SDR. The encoder is responsible for determining which output bits should be ones, and which should be zeros, for a given input value in such a way as to capture the important semantic characteristics of the data [1]. For integer input, a scalar encoder is employed to convert numerical values into sparse binary representations. Similarly, for image input, binarization techniques are utilized to convert pixel values into binary representations suitable for processing by the Spatial Pooler.

The *Figure 2* demonstrates the binarization process applied to images from the MNIST dataset:

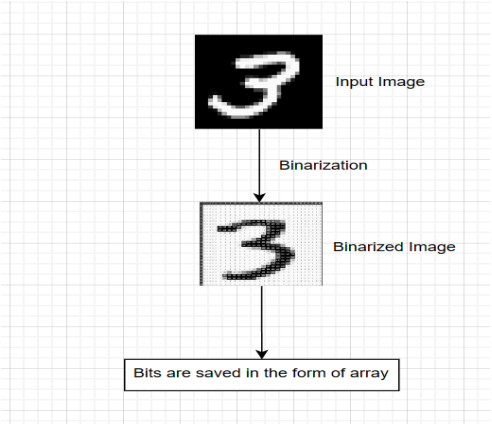

Figure 2: Image Binarization Process

B. Spatial Pooler

Once the scalar data/image is encoded into a suitable format for the spatial pooler the output from the encoder is fed into the Spatial Pooler. The Spatial Pooler operates by continuously converting input patterns into Sparse Distributed Representations (SDRs). In the Hierarchical Temporal Memory (HTM) framework, the temporal memory component learns temporal sequences based on these SDRs and generates predictions for future inputs. Essentially, the spatial pooler organizes or clusters data in the spatial dimension, where each incoming pattern is matched against a database of existing patterns during the learning process. The SP learns a one-to-one mapping of the input SDR space to another sparse binary representation of minicolumns [2]. A minicolumn can be thought of as a column of cells in the SP region. The Spatial Pooler consists of several minicolumns where each minicolumn is connected to different bits in the input space.

C. Sparse Distributed Representations

The spatial pooler algorithm produces Sparse distributed representations which are widespread across the neocortex. (SDRs) are expansive binary arrays characterized by a small fraction of bits set to 1, with the majority remaining at 0. This mirrors the operation of the neocortex, where neuron activations are sparsely distributed. In SDRs, the binary values of 1 and 0 can be likened to active and inactive neurons, respectively, aligning with the functioning of the human brain. Typically, SDRs comprise thousands of bits, with only about 1% to 2% being active. While individual bits lack specific labels or values, they convey semantic meanings collectively [3].

To make sure that the SDRs are reproduced expectedly, the HTM model is configured in predefined parameters as in *Table 1* such that the model can attempt different kinds of data and tasks.

Table 1: Values of HTM parameters

Parameters	Values
CellsPerColumn	10
MaxBoost	5
DutyCyclePeriod	100
MinPctOverlapDutyCycles	1
GlobalInhibition	FALSE
NumActiveColumnsPerInhArea	0.02*1024
PotentialRadius	0.15*200
LocalAreaDensity	-1
ActivationThreshold	10
MaxSynapsesPerSegment	0.01*1024
Random	42
Stimulus Threshold	10

D. Reconstruction

This method is a part of the spatial pooler class which returns a dictionary containing the input indices as keys and their corresponding aggregated permanence values as values. Which indicates the reconstructed input pattern based on the active columns in the spatial pooler.

E. Similarity Calculation of SDRs

Upon completion of the training process, the active columns of the input vectors are collected and passed to the Reconstruct () method to generate a reconstructed Sparse Distributed Representation (SDR), which is expected to be identical to the Input SDR. The similarity between these two SDRs is calculated both visually and mathematically.

Once the reconstructed SDR is obtained, the Jaccard Index [4] is computed to measure the similarity between the original input SDR and the reconstructed one. The Jaccard Index is a measure of similarity between two sets, is calculated as the size of the intersection divided by the size of the union of the sets as shown in *Figure 3*. Both the original input SDR and the reconstructed SDR are visualized as bitmaps using the BitmapVisualizer.GenerateAndDrawBitmap method. The Jaccard Index and the similarity percentage between the two bitmaps are subsequently calculated and displayed.

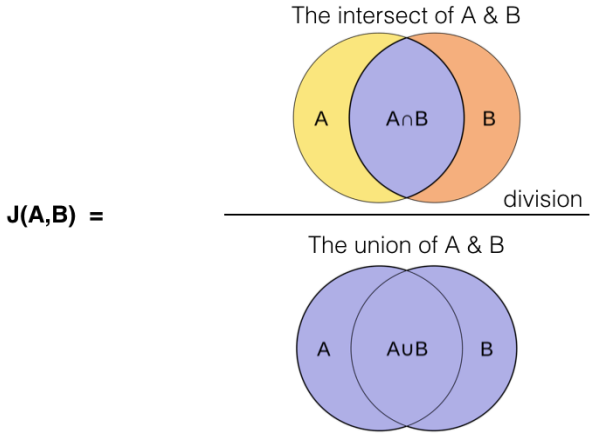

Figure 3: Mathematical expression for Jaccard Index

III. EXPERIMENT

In this section, we describe two different experimental cases conducted on random integer inputs and images of handwritten digits obtained from the MNIST dataset. The experiments are performed to learn the spatial pattern from the input and then reconstruct the input from the resultant SDR [5]. Subsequently, we perform additional experiments to assess the similarities and differences visually and mathematically between the original input and the reconstructed input.

Learning Phase:

For Integer Input: During the training phase, a range of random integer inputs were injected. The initial experiments were conducted using the default parameters specified in the htmconfig.json file. The Spatial Pooler was trained with each input over several iterations. The training process continued until the Spatial Pooler achieved a stable state, a process monitored by the HomeostaticPlasticityController (HPC) class. The primary function of the HPC is to reset the Spatial Pooler to its initial state at the onset of the learning process. During this phase, boosting is highly active, leading to an unstable state of the Spatial Pooler. Once the Sparse Distributed Representation (SDR) for each input stabilizes, the HPC triggers an event indicating that the Spatial Pooler has reached a stable state. During the learning process, the number of cycles taken by the spatial pooler to become stable was between A to B depending on the range of input integers. After several iterations, the Spatial Pooler enters the STABLE state.

For Image Input: During the training process, as depicted in *Figure 4,* we utilized a single MNIST image [6] with dimensions of 28x28 pixels. The initial experiments were conducted using the default parameters from the htmconfig.json file. The training of the image was carried out in several steps, as specified by the iteration steps argument. The image was converted into a binarized form that is a one-dimensional array and then transformed into a one-dimensional list. This list was then converted into a double numerical value using the class "BinarySDRConverter" which was then passed on to the Spatial pattern learning method to train the spatial pooler. The training continued until the spatial pooler reached a stable state, a process overseen by the HomeostaticPlasticityController (HPC) class. The objective of the HPC is to reset the Spatial Pooler to an initial state at the beginning of the learning process. At this stage, boosting is highly active, but the spatial pooler is unstable. Once the SDR generated for each input stabilizes, the HPC triggers an event to signal that the spatial pooler has achieved stability.

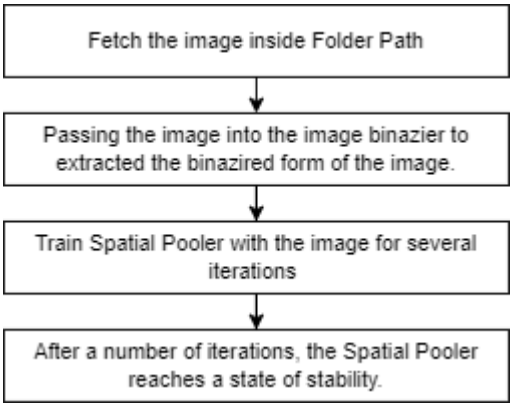

Figure 4: Spatial Pattern Learning Image Training Process Flow

Reconstruction Phase

In the post-learning phase, the active columns from the learned Spatial Pattern are extracted. The reconstruction phase as illustrated in *Figure 5,* employs the `Reconstruct ()` method present in the spatial pooler class, which accepts these active columns as input and retrieves a list of columns corresponding to the active mini-columns. For each column retrieved, the sum of permanence is calculated for each input index across all synapses. Permanence represents the strength of the connection between a column's synapse and the corresponding input index. Finally, the permanence values aggregate into a dictionary where the input index serves as the key, and the aggregated permanence serves as the value. This dictionary represents the reconstructed input pattern based on the active columns in the spatial pooler [7].

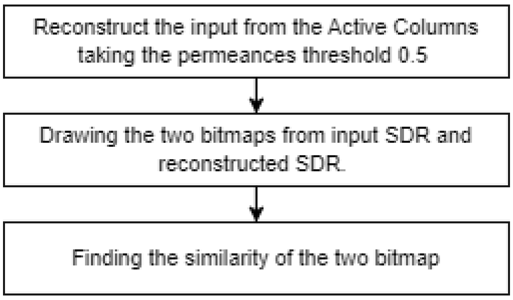

Figure 5: Overview of Reconstruction of Input and find the similarity.

Permanence Threshold

The output from the reconstruction phase is passed as an input to the ReverseEngineerInput method. This further iterates over all possible input indices and calculates the average permanence for synapses connected to each index making sure if it exceeds a certain threshold value, the permanence value is stored in a list of potential inputs. The average of the potential input values gets returned as the reconstructed input.

IV. Results

In the course of our experimental study, we incorporated a diverse set of 100, 300, and 500 random integer inputs and specifically those representing the digits 0, 1, and 3, from the MNIST dataset. These images were systematically introduced into each iteration of the training process, which subsequently involved the reconstruction of the image. Upon completion of the reconstruction phase, a thorough analysis was conducted across all experiments to assess the degree of similarity between the bitmaps of the original and reconstructed images. This comparative analysis served as a pivotal aspect of our research, offering valuable insights into the efficacy of our training and reconstruction methodology.

1. *Case 1:*

In our study, we conducted an analysis on a diverse range of random integer values, specifically within the range of 100 to 1000. A list of these values, referred to as 'inputValues', was generated and subsequently passed to the 'Spatial Pattern Learning' process. During this process, the spatial pooler was trained, yielding a Sparse Distributed Representation (SDR) for each integer. Upon receipt of the resultant SDR, we collected the active columns of the Spatial Pooler, which were then utilized to reconstruct the given input, henceforth referred to as 'reconstructedSdr'(in *Figure 6*).

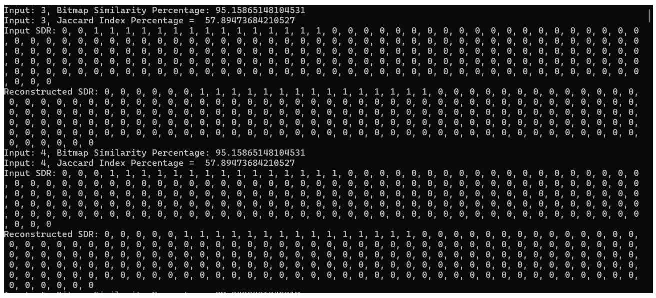

Figure 6: Representation of Input and Reconstructed SDR

Subsequently, two separate bitmaps were generated using these two SDRs, namely 'InputSdr' and 'reconstructedSdr'. The comparison of these bitmaps was facilitated using the BitmapComparator class. Following the generation and preservation of the input and reconstructed bitmaps, the BitmapComparator.Compare method was invoked to evaluate the visual similarity between these two bitmaps. The similarity percentage, as returned by the BitmapComparator.Compare method, was then documented, and utilized for further analysis. This comparative study in *Table 2* served as a critical component of our research, providing valuable insights into the visual similarities between the original and reconstructed bitmaps.

Table 2: Comparison chart for different sets of integer values

Random integer	Cycle	Avg Similarity	Avg Jaccard Index
100	363	84.99	0.046
300	357	85.08	0.042
500	177	85.2	0.04
700	198	86.23	0.039
1000	180	85.2	0.036

For the calculation of the Jaccard index, the CalculateJaccardIndex method was invoked with the input and reconstructed binary arrays (inpSdr and reconstructedSdr), and the result was stored in jaccardIndex. For bitmap comparison, the Compare method from BitmapComparator was invoked with the input and reconstructed bitmap paths (outputPath and similarityOutputPath), and the resulting similarity percentage was stored in similarityPercentage. These values (jaccardIndex and similarityPercentage) were then printed(in *Figure 7*) or utilized for further analysis as required.

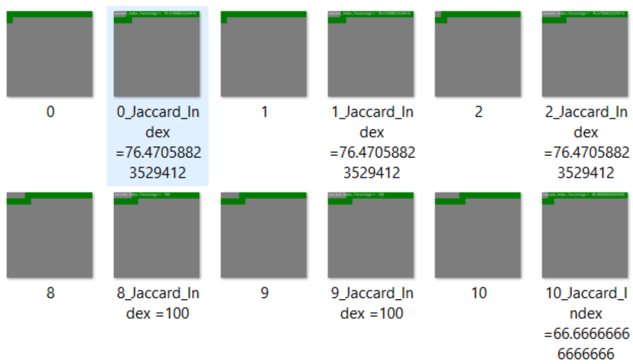

Figure 7: Bitmap of Input Data and Reconstructed Image

2. *Case 2:*

In this case, a subset of the MNIST dataset, specifically images of digits 0, 1, 3, and 9, is utilized. Each image, in PNG format, has dimensions of 28x28. The learning process is performed individually for each image. To initiate the learning phase, the image is binarized and transformed into a one-dimensional list. This list is then converted into a double, stored in 'double convertedValue', which represents the input image. This value is subsequently passed into the Spatial Pooler (SP) for training to learn the Spatial Pattern and generate a resultant Sparse Distributed Representation (SDR).

Upon completion of SP training, the RunRustructuringExperiment method is invoked to reconstruct an input SDR. An input SDR (inpSdr) as demostrated (*Figure 8)* is generated using a binarization process (ImgBinarizer.Imgo()). The SP computes the active columns for this input SDR using sp.Compute(inpSdr, false). The probabilities of each column being active are obtained using sp.Reconstruct(actCols).

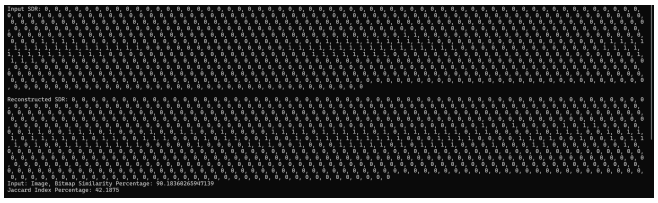

Figure 8:Representation of Input and Reconstructed SDR of Image

The probabilities obtained from the reconstruction are compared against a threshold (in this case, 8) to determine which columns should be active in the reconstructed SDR. If the probability of a column being active is greater than or equal to the threshold, the corresponding index in the reconstructed input SDR is set to 1; otherwise, it remains 0. Once the reconstructed SDR is obtained, the Jaccard Index is calculated to measure the similarity between the original input SDR and the reconstructed one. The Jaccard Index is a measure of similarity between two sets and is calculated as the size of the intersection divided by the size of the union of the sets. A comparative analysis has been done between different types of handwritten digits taken from MNIST dataset as observed in Table 3:

Table 3: Comparison chart for different MNIST Dataset

MNIST Dataset	Cycle	Similarity	Jaccard Index
"0"	60	78.52	33.6
"1"	60	90.33	44.8
"3"	60	80.29	23.5
"5"	60	89.16	29.6

Both the original input SDR and the reconstructed SDR are visualized as bitmaps using the BitmapVisualizer.Generate And DrawBitmap method(in *Figure 9*). The Jaccard Index and the similarity percentage between the two bitmaps are calculated and displayed.

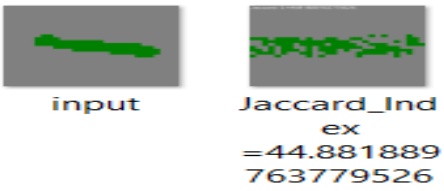

Figure 9: Bitmap of Input Image and Reconstructed Image

V. Discussion

We recognize that our experiment has certain limitations and boundaries that need to be overcome in order to enhance the program's robustness. In future development, we aim to address several key areas. Firstly, we are exploring how to efficiently process batches of images rather than handling them one at a time. Additionally, we plan to implement logic that works effectively with higher resolution and color images. Secondly, we are working on improving the training and reconstruction of images to achieve more accurate reconstructions. Notably, there are specific challenges related to training the Spatial Pooler, especially when dealing with previously binarized input.

VI. References

[1] J. H. &. S. Ahmad, " "how do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites,"," *neurons and cognition (q-bio.nc); artificial intelligence (cs.ai),* vol. V2, May 2016.

[2] S. A. J. H. Yuwei Cui, ""The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding,"," *Front. Comput. Neurosci., vol. 11| https://doi.org/10.3389/fncom.2017.00111, no. 29 Nov, 2017. ,* vol. 11, 2017.

[3] S. K. Mohanavelu, " "Online Anomaly Detection on Streaming Log Data",," *" https://research.tue.nl/en/studentTheses/online-anomaly-detection-on-streaming-log-data-using-hierarchical",* 28 september 2022.

[4] M. Z. I. Sam Fletcher, "Comparing sets of patterns with the Jaccard index," *Australasian Journal of Information Systems 22,* no. DOI: 10.3127/ajis.v22i0.1538, March 2018.

[5] D. Dobric, "neocortexapi," GitHub, [Online]. Available: https://github.com/ddobric/neocortexapi.

[6] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research," *IEEE Signal Processing Magazine,* vol. 29, no. 6, November 2012.

[7] anonymous, "How can I reconstruct the input using an SDR?," Numenta, [Online]. Available: https://discourse.numenta.org/t/how-can-i-reconstruct-the-input-using-an-sdr/3142 .