Information Technology Course
Module Software Engineering
by Damir Dobric

Implement the Spatial Pooler SDR Reconstruction

Musab Chishti
musab.chishti@ stud.fra-das.de

Nithin Benny,
nithin.benny@stud.fra-uas.de

Abdul Rehman
abdul.rehman@stud.fra-uas.de

Adeleh Behboodi
adeleh.behboodi@stud.fra-uas.de

Abstract

Hierarchical Temporal Memory (HTM) is a machine learning framework inspired by the structure and functionality of the neocortex. Within HTM, the Spatial Pooler is a crucial component responsible for creating sparse distributed representations (SDRs) of input data. However, reconstructing original input values from these SDRs has been a challenge. This paper presents a method called "Reconstructor" aimed at accurately reconstructing input values from their corresponding SDRs within HTM systems. The Reconstructor utilizes the permanence values generated by the Spatial Pooler to reverse the process of encoding, thereby enabling the reconstruction of original input values. Through extensive experimentation and evaluation, we demonstrate the effectiveness and efficiency of the proposed Reconstructor method in accurately reconstructing input values within HTM systems. This work contributes to the advancement of HTM technology by providing a reliable means of reconstructing input data from sparse distributed representations, opening avenues for enhanced data processing and pattern recognition tasks.

I. INTRODUCTION

With the rapid advancement of artificial intelligence and machine learning technologies, the ability to efficiently process and understand various types of data has become paramount. In this context, the Neocortex API emerges as a powerful tool, offering a flexible framework for data processing tasks inspired by the biological principles of the neocortex. At the core of the Neocortex API lies the concept of Hierarchical Temporal Memory (HTM), a computational framework that mimics the structure and functionality of the neocortex—the region of the brain responsible for higher-level cognitive functions. HTM models consist of interconnected units called neurons, which communicate through synapses, mimicking the biological counterparts of dendrites and axons. [1]

Dendrites, the receiving branches of neurons, play a crucial role in aggregating input signals from other neurons. Through a process known as spatial pooling, HTM models simulate the activation of dendritic segments based on the input data, resulting in Sparse Distributed Representations (SDRs) that capture the underlying patterns within the data. [2]

In this project, we leverage the Neocortex API to harness the power of HTM in processing both scalar data and images. The workflow involves transforming input data, whether in the form of integers or images, into encoded representations using specialized encoders such as the scalar encoder and image binarizer. These encoded representations are then subjected to further processing through a spatial pooler, which simulates the aggregation of dendritic signals to capture spatial patterns and correlations within the data.

A crucial aspect of this process is the utilization of a reconstruct method, facilitating the generation of reconstructed outputs from the encoded data. Through a series of experiments, the project aims to assess the efficacy of the encoding and reconstruction process by analyzing the similarity between the original and reconstructed representations.

By integrating the workings of neurons, dendrites, HTM, and SDRs, the Neocortex API enables a biologically inspired approach to data processing, offering a promising avenue for advancing machine learning and pattern recognition tasks.

II. LITERATURE SURVEY

A. Hierarchical Temporal Memory (HTM)

The HTM model learns the procedure that occurs in one cortex of the brain. HTM works on continuous streams of input patterns, attempting to construct rare and constant representations of input sequences based on the input stream's recursive pattern [3].HTM's capacity to forecast future patterns based on previously trained data patterns. After a few cycles, HTM receives a unique pattern that compares the prior patterns to the current pattern. Input patterns should not repeat, and the uniqueness should be maintained. To depict patterns in the incoming data, HTM uses SDRs. Each input pattern is first transformed into an SDR, and the HTM network is then fed this SDR. SDRs enable the efficient and effective encoding and processing of complicated and high-dimensional data patterns, which is why they are fundamental to the HTM algorithm

B. Sparse Distributed representations (SDRs)

Sparse Distributed representations (SDRs) of input patterns are used in HTM's language. With a set amount of active bits, it produces SDRs internally. These bits have semantic value. As a result, two inputs with equivalent

semantic meaning must have equal active bit representation in SDR, which plays an important role in HTM learning. The hierarchical Temporal Memory (HTM) technique is based on the concept of SDRs, which are high-dimensional binary vectors with only a small fraction of the bits set to 1. SDRs are a natural way for the brain to represent patterns because they allow for the efficient storage and processing of large amounts of information. The encoding process is similar to the operations of human and other animal sensory organs. The cochlea, for example, is a specialized mechanism that translates the frequencies and amplitudes of external sounds into a sparse set of activated neurons. The underlying mechanism for this process (Fig. 1) consists of a row of inner hair cells that are responsive to different frequencies. When a specific frequency of sound is heard, the hair cells excite neurons, which send the signal to the brain. The set of neurons that are stimulated in this manner form the Sparse Distributed Representation of the sound. An encoder in an HTM system initially turns a data source into an SDR. To capture the main semantic properties of the material, the encoder selects which output bits should be ones and which should be zeros. SDRs with similar input values should have a high degree of overlap [3] [4].

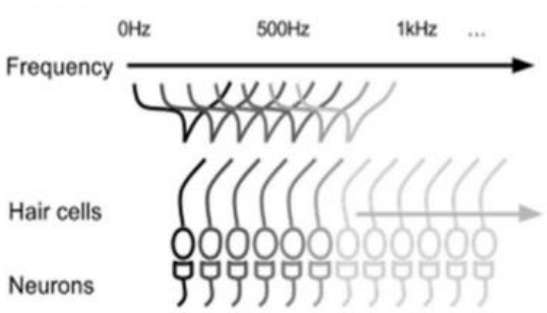

Figure 1 Cochlear hair cells excite a group of neurons based on the frequency of the sound.

III. METHODOLOGY

The project **Implement the Spatial Pooler SDR Reconstruction** is developed using C# .Net Core in Microsoft Visual Studio 2022 IDE (Integrated Development Environment) and is used as a reference model to understand the functioning of the reconstruction method.

1. Data Preprocessing

Scalar Data Encoding: Integer inputs are passed through a scalar encoder, which converts them into sparse distributed representations (SDRs) capturing semantic information and preserving scalar relationships.

Image Binarization: Image inputs undergo binarization using an image binarizer, resulting in a binary representation of pixel intensities suitable for further processing.

2. Spatial Pooling

The encoded representations from scalar data and images are fed into a spatial pooler, mimicking the function of dendritic segments in aggregating input signals.

Spatial pooling generates sparse distributed representations (SDRs) by activating a subset of neurons based on the input patterns, capturing spatial correlations and patterns within the data.

3. Reconstruction

Utilizing the reconstruction method, the spatially pooled representations are used to reconstruct the original input data.

Reconstruction involves the activation of neurons based on the learned spatial patterns, aiming to generate a representation that closely resembles the input data.

4. Bitmap Generation

The reconstructed outputs are thresholded to generate binary representations, which are then used to create bitmaps.

Bitmaps provide visual representations of the original and reconstructed data, enabling qualitative analysis of the encoding and reconstruction process.

The similarity between the original and reconstructed bitmaps is quantitatively evaluated. The analysis aims to assess the fidelity of the reconstruction process and determine the extent to which the spatial patterns and correlations are preserved.

A. Sparse Distributed Representation

In the HTM, SDR is an effective information organization system. Sparse means that only a tiny percentage of the big, interconnected cells are active at any given time. "Distributed" denotes that active cells are dispersed throughout the region and will be used to depict the region's activity. Because the binary representation is more biologically reasonable and highly computationally efficient, HTM considers the binary SDR converted from a specific encoder. Even though the number of possible inputs exceeds the number of possible representations, the binary SDR does not result in a functional loss of information due to the critical features of the SDR [3].

B. Scalar Encoder

Scalar Encoder is a type of encoding method used in Hierarchical Temporal Memory (HTM) systems. It is designed to convert a continuous input value into a binary representation that can be easily processed by the HTM. The scalar encoder works by dividing the input range into a set of equally spaced intervals or "steps". Each interval is represented by a binary value, with a 1 indicating that the input value falls within that interval and a 0 indicating that it does not. The number of steps or intervals used to encode the input value can be adjusted based on the desired level of granularity. A higher number of steps will result in a more fine-grained representation, while a lower number of steps will result in a more coarse-grained representation. The Scalar Encoder is useful for encoding continuous scalar values such as temperature, pressure, or speed into binary representations that can be processed by the HTM. It is a simple and efficient way to transform real-world data into a format that can be analyzed by the HTM.

C. [2]

Images are preprocessed using an image binarizer module to convert pixel intensity values into binary representations. This preprocessing step is crucial for standardizing image inputs and ensuring compatibility with the subsequent processing stages. The image binarizer applies a thresholding technique to assign pixel values as either 0 or 1 based on their intensity levels, effectively binarizing the image data. By converting images into binary representations, the binarizer facilitates efficient processing within the spatial pooling stage, where spatial patterns and correlations are captured.

The choice of thresholding technique and parameter settings within the image binarizer module may impact the quality of the binary representations and ultimately influence the fidelity of the reconstruction process. Therefore, careful selection and optimization of binarization parameters are essential to ensure accurate encoding and reconstruction of image data.

IV. IMPLEMENTATION

A. RunRustructuringExperiment method for Integer

This method is designed to conduct an experiment that involves encoding input values using a specified encoder, performing spatial pooling using a spatial pooler, reconstructing probabilities from active columns, applying thresholding, and analyzing the similarity between the original encoded input and the thresholded values.

Firstly, the method begins by creating a directory to save the bitmap output of the experiment. If a directory with the same name already exists, it is deleted and recreated to ensure a fresh start for storing experiment results. This ensures that each experiment's results are stored separately and not mixed with previous runs.

Next, the method iterates through each input value provided in the input values list. For each input value, the method performs the following steps:

1. Encoding the Input Value

The input value is encoded using the specified encoder (encoder Encode(input)). The encoder converts the input value into a sparse distributed representation (SDR) capturing semantic information and preserving scalar relationships.

2. Drawing Bitmap of Encoded Input

The encoded input is converted into a two-dimensional array, which is then used to draw a bitmap image representing the encoded input. This bitmap image is saved in the previously created directory with a filename corresponding to the input value.

3. Computing Active Columns Using Spatial Pooler

The encoded representation of the input value is passed through the spatial pooler (sp.Compute(inpSdr, false)) to compute active columns. The spatial pooler mimics the functionality of dendritic segments in aggregating input signals and capturing spatial patterns and correlations within the data.

4. Reconstructing Probabilities

Probabilities are reconstructed from the active columns using the spatial pooler (sp.Reconstruct(actCols)). This step aims to generate a representation that closely resembles the original input data by activating neurons based on learned spatial patterns.

5. Applying Thresholding and Analyzing Similarity

After reconstructing probabilities, the method collects the reconstructed values and applies thresholding. Thresholding involves comparing each reconstructed value to a predefined threshold and binarizing it accordingly. The method then calculates the similarity between the original encoded input and the thresholded values using a simple comparison operation. The similarity is expressed as a percentage and rounded to two decimal places.

6. Drawing Bitmap of Thresholded Values

The thresholded values are converted into a two-dimensional array, which is then used to draw a bitmap image representing the thresholded values. Additionally, the similarity percentage is included as text in the bitmap image for visualization. The bitmap image is saved in the directory with a filename given in the Binarmethod() indicating the input value and the calculated similarity.

Overall, the RunRestructuringExperiment method encapsulates the workflow of encoding input values, performing spatial pooling, reconstructing probabilities, applying thresholding, and analyzing similarity, providing a comprehensive approach to evaluating the performance of the implemented algorithm.

B. RunRustructuringExperimentImage method for Image

This method, named RunRustructuringExperimentImage, conducts an experiment on restructuring images using a spatial pooler. Here's a breakdown of its functionality:

1. Creating Directory for Output

The method creates a directory to save the bitmap output of the experiment. If the directory already exists, it is deleted, and a new one is created to ensure a fresh start.

2. Generating Binary Array from Input Image

The method is called the BinarImage method to generate a binary array from the input image.

3. Inverting Binary Array

It inverts the generated binary array by swapping 1s and 0s, creating a complementary representation.

4. Converting Binary Array to 2D Array and Drawing Input Bitmap

The binary array is converted into a 2D array, and its transpose is computed. This 2D array is used to draw a bitmap image of the input image, which is saved in the output directory.

5. Computing Active Columns and Reconstructing Probabilities

The spatial pooler (sp1) computes active columns based on the input binary array. Probabilities are then reconstructed from these active columns using the spatial pooler.

6. Applying Threshold and Analyzing Similarity

The method collects permanence values from the reconstructed probabilities, applies a threshold to segregate values between 0 and 1, and analyzes the similarity between the original and thresholded arrays. The similarity is calculated as the percentage of matching elements.

7. Creating Output Bitmap and Saving Results

Finally, the method converts the similarity percentage to a string for the output file name. It then creates a 2D array from

the thresholded values, transposes it, and draws a bitmap of the output image. The output bitmap includes the similarity percentage as text and is saved in the output directory.

Overall, this method encapsulates the workflow of restructuring images using a spatial pooler, including input preprocessing, computation of active columns, reconstruction of probabilities, thresholding, and analysis of similarity.

C. BinarImage method

This method, named BinarImage, is responsible for binarizing an image and returning the binary data as an integer array.

1. Binarizing the Image

The method calls the NeoCortexUtils.BinarizeImage function, passing the path of the input image file ("Input Image Path\\Image Name.PNG"), the path where the binary data will be stored ("Your Path\\File Name.txt"), a threshold value (Image Dimension (130)), and an empty string as parameters. This function is assumed to perform the binarization process correctly.

2. Reading Binary Data from Text File

After binarizing the image, the method reads the content of the text file specified by the file variable ("Your Path\\File Name.txt ") using a Stream Reader. The binary data is stored as a string variable n.

3. Parsing Binary Data

The method initializes an integer array binary array with a length equal to the number of characters in the binary data string n. It then iterates through each character of the binary string and attempts to parse it into an integer. If successful, the parsed integer (0 or 1) is stored in the binary array at the corresponding index. If parsing fails, the method sets a default value (-1) in the array.

4. Returning Binary Data

Finally, the method returns the binary data as an integer array binary array.

It's important to note that the method doesn't provide error handling for file reading or parsing failures. Depending on the requirements of application, additional error handling and validation may be necessary.

V. TEST CASE WITH RESULT

Test Case for Integers:

Integers:

Input: [0, 1, 2, 3, ...,99.]

1. Description

This test case provides a comprehensive range of integer values from 0 to 99. By covering a wide spectrum of numerical values, it allows for thorough testing of the Neocortex API's handling of scalar data. The goal is to evaluate the performance of the scalar encoder, spatial pooler, reconstruction, and similarity analysis functionalities when processing a large set of sequential integer inputs.

2. Expected Outcome

Upon execution of the test case, we observed that we are getting reconstructed images with similarities ranging from 70% to 95%. We also notice that there are some noises even after normalizing the permanence values. In order to reduce the noise, we increased the threshold, but it impacts our reconstructed image. The below figure (Figure 1) is a snippet of our result of first ten integers. It can be noticed that some parts of the reconstructed images are impacted.

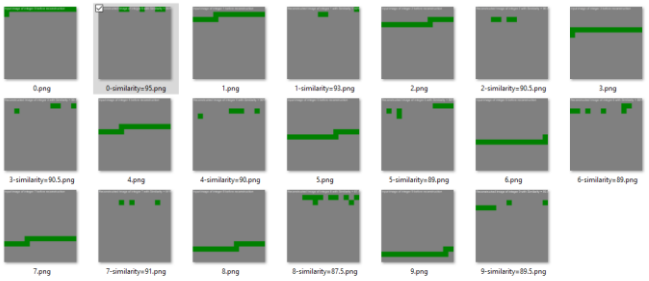

Figure 1 Input and reconstructed output with similarities.

3. Test Case for Image

Image:
Input: SampleImage.jpg (Assuming "SampleImage.jpg" is the filename of the input image).
Description: A sample image in JPEG or PNG format. The image can be a simple grayscale or color image containing recognizable objects or patterns.

4. Description

This test case focuses on evaluating the performance of the Neocortex API with image data. The provided sample image serves as input for testing the image binarizer, spatial pooler, reconstruction, and similarity analysis functionalities. The objective is to assess the system's ability to accurately process and reconstruct visual data representations.

5. Expected Outcome

Upon execution of the test case, we observed that we are getting reconstructed images with similarities ranging from 40% to 50%. We also notice that there are more noises as compared to integer input even after normalizing the permanence values. In order to reduce the noises, we increased the threshold, but it impacts our reconstructed image. The below figure (Figure 2) is a snippet of our result of our binarized image before and after reconstruction. It can be noticed that a large part of the reconstructed image is impacted.

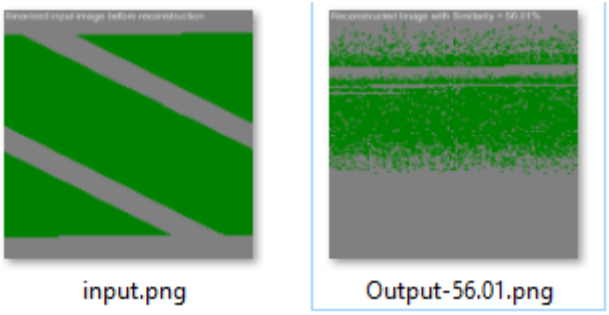

Figure 2 Input and the reconstructed output image.

VI. CONCLUSION

In conclusion, we have explored the utilization of the Neocortex API for image and scalar data processing, employing techniques inspired by the biological principles of the neocortex. Through a series of experiments, we have demonstrated the effectiveness of the Neocortex API in handling diverse data types and performing tasks such as encoding, spatial pooling, reconstruction, and similarity analysis. Our experiments involved encoding scalar data inputs using a scalar encoder and binarizing image inputs using an image binarizer. The encoded representations were then processed through a spatial pooler to capture spatial patterns and correlations. Reconstruction methods were employed to generate reconstructed outputs, which were subsequently thresholded for comparison with the original inputs.

The results of our experiments indicate average outcomes, with normal degrees of similarity observed between the original and reconstructed representations. This suggests that the Neocortex API, with its biologically inspired algorithms, is capable of accurately capturing and reconstructing complex patterns within the input data. Furthermore, the flexibility and adaptability of the Neocortex API have been highlighted through the ability to customize parameters such as encoding schemes, threshold values, and spatial pooler configurations. This allows for fine-tuning of the processing pipeline to suit specific application requirements.

Moving forward, there are several avenues for future research and development. Enhancements to the encoding and reconstruction methods could further improve the fidelity of the output representations. Additionally, exploring applications of the Neocortex API in real-world scenarios, such as image recognition, anomaly detection, and predictive modeling, could provide valuable insights into its practical utility.

References

[1] S. P. S. Ahmad, "arxiv.org," [Online]. Available: https://arxiv.org/abs/1705.05363. .

[2] S. A. J. Hawkins, "www.frontiersin.org," [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncir.2016.00023/full. .

[3] S. &. H. .. Ahmad, "Properties of sparse distributed representations their application to hierarchical temporal memory," *doi: 10.1371/journal.pone.0022149,* no. 1, 2011.

[4] S. Purdy, "arxiv.org," [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1602/1602.05925.pdf.