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

Improve Samples and Documentation for SDR Representation (SDR-To-BitMap)

Pranil Ghadi
pranil.ghadi@stud.fra-uas.de

Yatish Sharma
yatish.sharma@stud.fra-uas.de

***Abstract*—This research paper delves into the enhancement of Sparse Distributed Representations (SDRs) through the provision of diverse examples and improved documentation. Focusing on the conversion of SDRs to Bitmaps (SDR-To-BitMap), the study explores methodologies aimed at augmenting the accessibility and comprehension of SDR-based techniques.**

The paper presents a comprehensive approach to enriching the understanding and application of SDR representation without the implementation of new encoders. Through the development of illustrative examples and detailed documentation, the research endeavors to facilitate seamless integration and utilization of existing SDR encoding tools.

By showcasing the versatility of SDR representations across various data types and scenarios, and by elucidating best practices through enhanced documentation, this study aims to foster advancements in domains reliant on efficient data representation and processing. Through these efforts, the paper seeks to contribute to the broader adoption and utilization of SDR-based techniques in diverse applications.

Keywords*— *Spare Distributed Representation, HTM-Hierarchical Temporal Memory, SDR Encoders, Temporal Memory, BitMaps.

I. Introduction

Sparse Distributed Representations (SDRs) have garnered considerable attention in recent years due to their versatility and efficiency in representing complex data structures. With applications spanning machine learning, signal processing, and cognitive science, the adoption of SDRs has paved the way for innovative solutions to various data-related challenges.

This paper aims to delve into the realm of SDR representation, focusing specifically on enhancing the quality of examples and documentation to facilitate better understanding and utilization of SDR-based techniques. While our study does not involve the development of new encoding algorithms, it leverages existing tools such as ScalarEncoder, DateTimeEncoder, Geospatial Encoder, and Spatial Pooler to enrich the landscape of SDR representation.

The significance of this endeavour lies in the inherent complexity of SDR-based techniques, which often pose challenges for newcomers and practitioners alike. By providing a diverse array of examples spanning different data types and scenarios, coupled with detailed documentation elucidating encoding methodologies and best practices, we aim to bridge the gap between theory and application in the realm of SDR representation.

Through this holistic approach, our research endeavours to empower users with the knowledge and resources necessary to leverage SDR-based techniques effectively. By promoting accessibility and comprehension, we seek to foster wider adoption and innovation in domains reliant on efficient data representation and processing.

In the subsequent sections of this paper, we will delve into the methodologies employed to enhance examples and documentation surrounding SDR representation. Through illustrative examples and detailed explanations, we aim to provide readers with practical insights and tools to harness the power of SDRs in their respective domains.

II. METHODOLOGY

A. Overview

In this project, our focus is on enhancing the accessibility and usability of Sparse Distributed Representations (SDRs) by enriching the examples and documentation associated with SDR representation techniques. While existing SDR encoders like ScalarEncoder, DateTimeEncoder, Geospatial Encoder, and Spatial Pooler provide robust functionality, their adoption can be hindered by a lack of clear examples and comprehensive documentation.

To address this challenge, our approach involves developing a diverse set of examples that demonstrate the application of SDR representation across various data types and scenarios. These examples aim to showcase the versatility and effectiveness of SDR-based techniques in practical settings. Additionally, we aim to improve the documentation surrounding these encoders, providing detailed explanations, usage guidelines, and best practices to aid users in understanding and implementing SDR encoding effectively.

By enhancing the examples and documentation, we seek to lower the barriers to entry for newcomers and facilitate a deeper understanding of SDR representation techniques among practitioners. This, in turn, can lead to broader adoption and innovation in fields reliant on efficient data representation and processing.

Throughout this project, we employ a systematic methodology to develop and validate examples, refine documentation, and incorporate feedback from stakeholders. By following this approach, we aim to deliver tangible improvements that empower users to harness the full potential of SDR representations in their applications and research endeavours.

B. Encoders

The process of encoding data into Sparse Distributed Representations (SDRs) involves tailored considerations and specialized algorithms. Encoders receive raw data inputs spanning scalar values, temporal information, geographical coordinates, and spatial patterns, with each encoder designed to handle specific data modalities. These algorithms transform the input data into binary representations, ensuring the preservation of relevant information while maintaining sparsity—a defining characteristic of SDRs. By ensuring that only a small fraction of elements in the binary vector are active (set to 1), encoders enable efficient storage and computation, essential for subsequent analysis.

SDR representations are typically high-dimensional, with encoders tasked with determining the appropriate dimensionality of the resulting binary vectors to balance representational capacity and discrimination. Despite their high dimensionality, SDRs exhibit robust noise tolerance, enabling reliable information retrieval and processing even in the presence of perturbations or errors in the input data. Encoders incorporate mechanisms to maintain this noise tolerance, enhancing the resilience of the encoded representations. Furthermore, encoders are adaptable to diverse data types and application requirements, often featuring configurable parameters that allow users to fine-tune encoding properties such as resolution, sparsity, and overlap. This adaptability ensures that encoders can effectively handle various data characteristics and meet specific application objectives.

*C. **DrawBitmap** Method*

The **DrawBitmap()** method is a versatile tool for generating bitmap images from arrays of active columns, offering a simple yet effective means to visualize Sparse Distributed Representations (SDRs). This method allows users to customize the appearance of the bitmap by adjusting parameters such as the array of active columns, output dimensions, file path for saving, colors for active and inactive cells, and an optional text overlay. By manipulating these parameters, users can tailor the bitmap to their specific needs and preferences, creating clear and informative visual representations of SDR data.

At its core, the method renders pixels based on the array of active columns provided. Active columns are typically represented using the specified color, while inactive columns utilize a different color, facilitating clear differentiation within the bitmap. Additionally, users have the option to overlay text onto the bitmap, providing contextual information or labels directly within the visualization. This feature enhances the interpretability and utility of the bitmap representations, making them more informative for analysis and communication purposes.

Once configured, the generated bitmap can be saved at the specified file path for further analysis, sharing, or archival purposes. This allows users to store and access the visual representations of their SDR data conveniently. Overall, the **DrawBitmap()** method provides a user-friendly and customizable approach to visualizing SDRs, empowering users to create clear and informative bitmap representations tailored to their specific needs and objectives.

D. 1-D Array & 2D Array

In the document, understanding the distinction between 1-D arrays and 2-D arrays is essential for grasping the data manipulation processes at hand. A 1-D array, often referred to as a vector or list, holds elements in a linear fashion, accessible through a single index. For instance, think of a scenario where we track temperatures for each day of the week. Accessing a specific temperature, like Tuesday's, is as straightforward as using an index, such as **temperatures [1]**.

On the other hand, a 2-D array, also known as a matrix, organizes elements into rows and columns, requiring two indices: one for the row and another for the column. Consider an example where we represent pixel values in an image grid. Here, each element denotes the color intensity at a specific row and column. Accessing a particular pixel would involve specifying both row and column indices, like **pixels [1, 1]** for the pixel at row 1, column 1.

In the provided code snippet, the **Encode ()** method processes input data, resulting in a 1-D array (**result2**) that likely represents Sparse Distributed Representations (SDRs). Subsequently, this 1-D array is converted into a 2-D array (**twoDimenArray2**) using a utility function like **ArrayUtils.Make2DArray<int>()**, potentially for visualization or further processing. This transformation from a 1-D to a 2-D array aids in tasks such as visualizing SDRs, where a grid-like representation enhances interpretability for analysis or comprehension.

III. SDR GENERATION & REPRESENTATION

A. SDR Generation Using Scalar Encoder

The process of generating Sparse Distributed Representations (SDRs) using scalar encoders involves transforming continuous scalar values into binary vectors suitable for neural network processing. This technique partitions the input range into smaller bins and activates specific bits within each bin, resulting in binary vectors that represent numerical values. The generated SDRs maintain sparsity, ensuring that only a small fraction of bits are active at any given time, thus preserving semantic information in a high-dimensional space.

These SDRs can be visualized effectively through bitmap representations, offering intuitive insights into the encoded data. By converting the binary vectors into bitmap images, the underlying patterns and distributions encoded by the scalar encoder become visually apparent. Bitmap representations provide a concise yet informative visualization of the encoded data, aiding in analysis and interpretation.

Example:

To illustrate this process, consider the encoding of air quality index (AQI) values using a scalar encoder. The AQI values, ranging from 0 to 500, are categorized into specific ranges corresponding to different air quality levels. Using the scalar encoder, these values are transformed into binary vectors, representing the corresponding AQI levels.

Subsequently, the binary vectors are converted into bitmap images using the DrawBitmap method. Each pixel in the bitmap represents an individual bit in the binary vector, with active bits depicted using a distinct color from inactive bits. The resulting bitmap images provide a visual representation of the encoded AQI values, allowing for easy interpretation and analysis of air quality data.

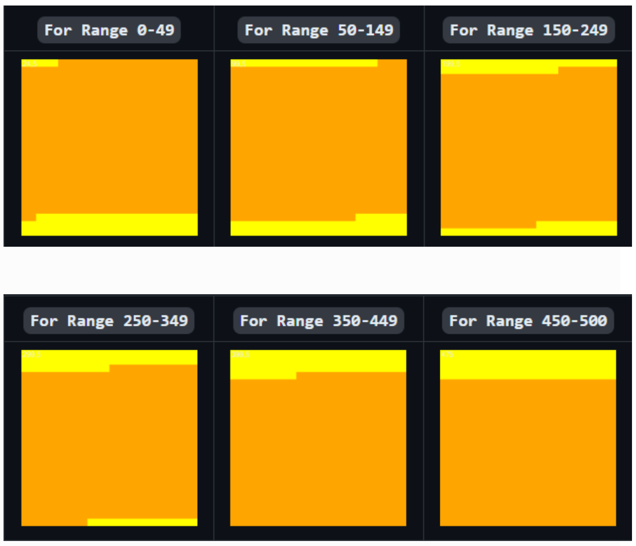

Fig: 1 SDR Generation Using Scalar Encoder

B. *SDR Generation Using DateTime Encoder*

The DateTime Encoder serves as a crucial component for encoding date and time information into Sparse Distributed Representations (SDRs). Initialized with specific settings such as width (W), number of bits (N), minimum and maximum values, periodicity, and padding, this encoder efficiently processes date and time inputs through its Encode() method. Upon invocation, this method generates a one-dimensional array (1-D array) representing the resulting SDR.

These SDRs can be effectively represented both as text and as bitmap images. In bitmap representation, the SDR is converted into a two-dimensional array (twoDimArray), facilitating further visualization using tools like DrawBitmap(). This method generates bitmap images from the SDRs, offering a visual depiction of the encoded date and time information.

Example:

In a practical scenario, consider the encoding of date and time information using the DateTime Encoder. Once initialized with specific settings, the Encode() method is called with a particular date and time input. Subsequently, the resulting one-dimensional array (1-D array) representing the SDR is converted into a two-dimensional array (twoDimenArray2) using ArrayUtils.Make2DArray<int>(), enabling bitmap visualization.

In the provided example, the resulting two-dimensional array (twoDimenArray2) is transposed and passed to the DrawBitmap method. This method generates a bitmap image with specified width and height, where inactive cells are represented in black and active cells in green. The bitmap image serves as a visual representation of the encoded date and time information, facilitating easy interpretation and analysis.

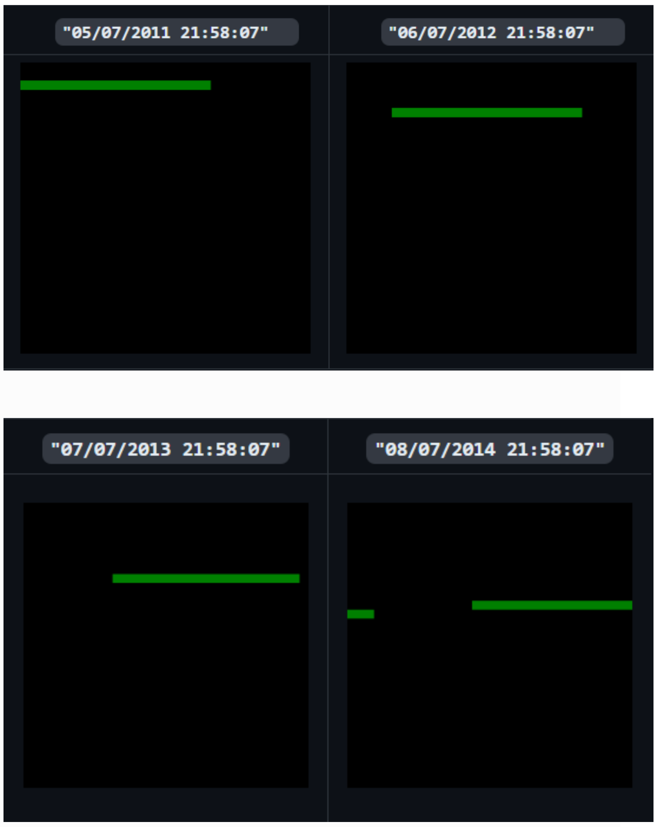

Fig: 2 SDR Generation Using DateTime Encoder

C. SDR Generation Using Geo-Spatial Encoder

The Geo-Spatial Encoder plays a pivotal role in converting geospatial data into binary arrays and facilitating their visualization as bitmap images. This encoder enables the exploration of geospatial data through Sparse Distributed Representations (SDRs), offering insights into spatial information encoded within the data.

To generate bitmap images representing encoded geographical coordinates, the DrawBitmap method is employed. This method translates the binary arrays produced by the Geo-Spatial Encoder into visually interpretable bitmap images. By setting parameters such as width (W), number of bits (N), minimum and maximum values, the encoder defines how geospatial data will be encoded into binary arrays. Subsequently, the generated binary arrays are converted into two-dimensional arrays, which are then transposed to ensure proper alignment for bitmap visualization.

Example:

Consider the encoding of geographical coordinates using the Geo-Spatial Encoder. The encoder is initialized with specific settings, including width (W), number of bits (N), minimum and maximum values. For instance, the encoder settings may define parameters such as W=103, N=238, MinVal=48.75, and MaxVal=51.86. These settings dictate how the geographical data will be encoded into binary arrays.

The Encode() method is invoked with specific geographical coordinates, generating a one-dimensional array (1-D array) representing the resulting SDR. This 1-D array is then converted into a two-dimensional array (twoDimenArray) with dimensions calculated based on the square root of the array length.

Subsequently, the two-dimensional array is transposed to prepare it for bitmap visualization. The resulting transposed array is passed to the DrawBitmap method, which generates bitmap images with specified dimensions. In the bitmap images, inactive cells are represented in red, while active cells are represented in black.

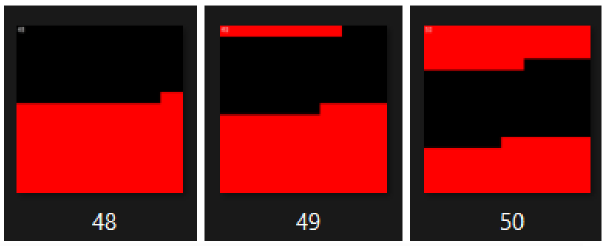

Fig: 3 SDR Generation Using Geospatial Encoder

D. SDR Generation Using Spatial Pooler

Utilizing Hierarchical Temporal Memory (HTM) principles, a Spatial Pooler (SP) is trained to recognize patterns within input images and generate Sparse Distributed Representations (SDRs) for each image. The SP is initialized with parameters such as potential radius, potential percentage, global inhibition, and local area density, defined within the HtmConfig object. Training images are loaded from a specified directory, and for each image, the input vector is computed using NeoCortexUtils.BinarizeImage() and read into an array with NeoCortexUtils.ReadCsvIntegers().

Upon computing the input vector, it is fed into the SP using the sp.compute() method, which calculates the active columns based on the input. The active columns are determined from the output array of the SP, where the value is equal to 1. Once the system reaches a stable state indicating convergence, the results are calculated and stored.

The generated SDRs, represented by the active columns obtained from the SP, are stored in a dictionary with the image file name as the key. The activeArray computed by the spatial pooler is converted into a two-dimensional array using ArrayUtils.Make2DArray<int>(), typically with dimensions set to 64x64. This two-dimensional array can then be further transposed and utilized as an input parameter for the Bitmap function.

The Bitmap function, accessible from the System.Drawing library, requires parameters specifying the width and height of the bitmap. With this information, bitmap representations of the generated SDRs can be created, allowing for graphical presentations of the activation patterns observed within the spatial pooler. These bitmap images offer valuable insights into the recognition and representation of patterns within the input images.

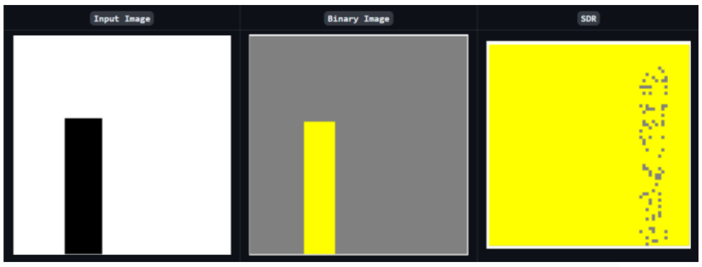

Fig: 4 SDR Generation Using Spatial Pooler

Example: Bitmap Representation of Alphabet "L" After Computing in Spatial Pooler

To illustrate the process of representing an alphabet "L" in a bitmap after computation in a spatial pooler, let's consider a 2-D array with dimensions of 64x64 containing the Sparse Distributed Representations (SDRs) generated from the spatial pooler. We aim to represent this in a bitmap with dimensions chosen to be 1024x1024.

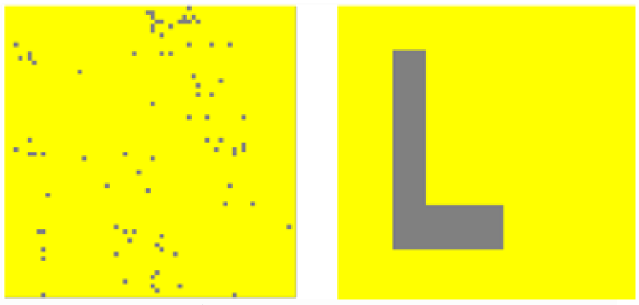
Fig: 5 64x64 SDR & 28x28 Binarized Image of letter L

In this example, the scale is calculated to be 7, indicating that each position in the 64x64 2D array is represented by 49 pixel positions in the 1024x1024 bitmap. This scale ensures efficient utilization of the bitmap while accurately depicting the SDR representation.

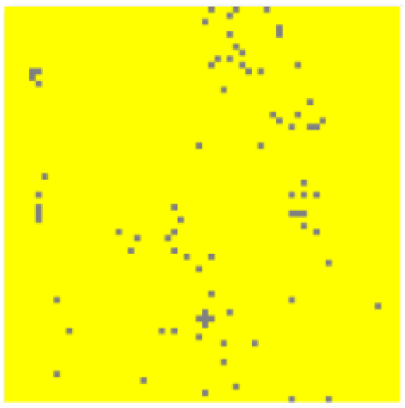
Fig: 6 64x64 2D array is represented by 225 pixel positions of 1024x1024 bitmap

Each active bit in the 64x64 2D array is represented by grey pixels in the bitmap, while inactive bits are represented by yellow pixels. This representation offers insights into the activation patterns observed within the spatial pooler for the alphabet "L".

Example: Bitmap Representation of Overlap, Difference, and Union for Alphabets "L" and "V" After Computing in Spatial Pooler

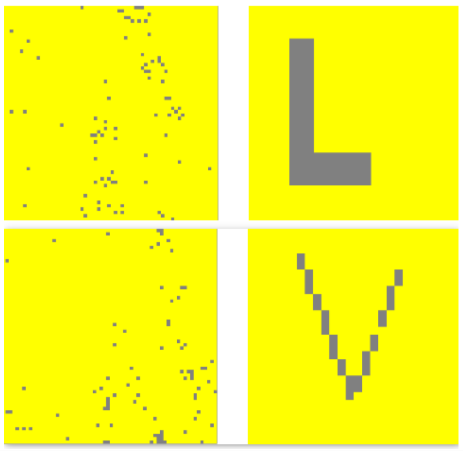
Fig: 7 Bitmap representation of L and V

In a more complex scenario, we explore the representation of the overlap, difference, and union between the Sparse Distributed Representations (SDRs) of alphabets "L" and "V" after computation in the spatial pooler.
Using functions like UnionSDRFun(), DiffSDRFun(), and OverlapSDRFun(), we compare the SDRs of the two alphabets and generate corresponding bitmap representations for overlap, difference, and union.

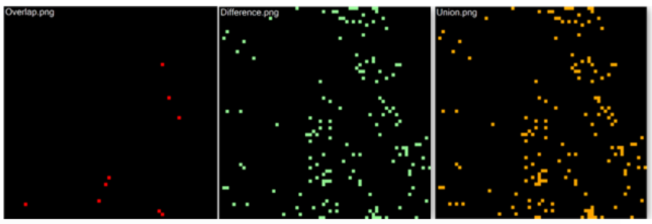

Fig: 8 Representation of Overlap, Difference and Union

For instance, the "overlap.png" bitmap exhibits fewer intersections between the SDRs as the alphabets "L" and "V" are inherently different. Conversely, the combination of overlap and difference provides insights into the union of the two SDRs, showcasing their collective characteristics.
This example demonstrates the versatility of bitmap representations in visualizing the relationships between different SDRs, offering valuable insights into the patterns recognized by the spatial pooler.

IV. DISCUSSION

Sparse Distributed Representation (SDR) generation and visualization stand as crucial components in various domains such as machine learning, neural networks, and pattern recognition. Our project aimed at refining the process of generating SDRs from diverse data types and improving their visualization through bitmap representations. Here, we delve into the significance of our approach, its implications, and potential avenues for further exploration.

In the realm of data representation, SDRs offer a potent means of encapsulating intricate data patterns efficiently. Their sparse nature ensures optimal storage, processing, and retrieval of information, making them invaluable for memory-efficient applications. Through our project, we sought to bolster the representation of various data types – including scalar values, temporal data, geographical coordinates, and spatial patterns – by harnessing the capabilities of SDRs.

Our endeavor focused on enhancing the SDR generation process by refining existing encoders such as ScalarEncoder, DateTimeEncoder, GeoSpatialEncoder, and Spatial Pooler. By fortifying these encoders and augmenting their documentation, we aimed to streamline the process of generating SDRs, fostering scalability, adaptability, and efficiency.

Bitmap representations emerged as a pivotal aspect of our project, facilitating intuitive visualizations of SDRs. By converting SDRs into bitmap images, we aimed to provide users with a tangible means of interpreting complex data patterns. This visualization strategy aimed to enhance the interpretability and comprehensibility of encoded information, aiding researchers, practitioners, and developers in deciphering underlying data structures.

The implications of our work span diverse domains, including machine learning, neuroscience, natural language processing, sensor data analysis, and robotics. The refined SDR generation process and bitmap visualization techniques hold promise for tasks such as classification, anomaly detection, predictive modeling, and understanding cortical processing in the brain.

V. CONCLUSION

In conclusion, our project has addressed the critical aspects of Sparse Distributed Representation (SDR) generation and visualization, aiming to refine existing methodologies and enhance their practical applications. Through meticulous refinement of encoders such as ScalarEncoder, DateTimeEncoder, GeoSpatialEncoder, and Spatial Pooler, we have contributed to streamlining the process of generating SDRs across diverse data types. Additionally, our emphasis on bitmap representations has provided users with intuitive visualizations, facilitating the interpretation of complex data patterns.

The outcomes of our project hold significant implications across various domains, including machine learning, neuroscience, and robotics. By enabling efficient data representation and analysis, our work contributes to advancements in tasks such as classification, anomaly detection, and predictive modelling. Furthermore, bitmap visualization techniques offer a tangible means of understanding underlying data structures, fostering insights that drive progress in research and practical applications.

VI. REFERENCES

[1] Neocortexapi: Dr. Damir Dobric https://github.com/ddobric/neocortexapi/tree/ac26d4acecea21d74b6648cf23c6360a244d1f9d

[2] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, Perspectives, and prospects," Science, vol. 349, no. 6245, pp. 255–260, 2015

[3] "Sparse distributed representations - numenta.com." [Online]. Available: https://www.numenta.com/assets/pdf/biological-and-machine-intelligence/BaMI-SDR.pdf.

[4] "Sparse distributed representations: Sparse Distribution:

[5] SDR Intelligence," Cortical.io. [Online]. Available: https://www.cortical.io/science/sparse-distributed-representations/