CS786A Project Report: Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory

Shreyash Ravi (17807683) Khyathi Vagolu (190432) Deepankur Kansal (180226)

Abstract

In this paper report, we review properties of Sparse Distributed Representations (SDRs) in the context of their applications in Hierarchical Temporal Memory algorithms and perform a python implementation of SDRs and their properties.
Every region of the cortex encodes information using sparse activity patterns. Sparse Distributed Representations (SDRs) are cortical representations that encode information throughout the cortex. Mathematically, given a population of neurons, an SDR represents their instantaneous activity as sparse n-dimensional vector of binary components.

Introduction

Sparse distributed representations encode information throughout the cortex and have a variety of functions given that they are sparse and the encoding is distributed across a set of neurons. In this project, we explore SDRs and their applications in Hierarchical Temporal Memory algorithms. The main properties of SDRs include uniqueness, subsampling, vector classification, and unions. We implement code to demonstrate the robustness to noise, scalability, etc. of the SDRs as proposed in the paper[1]. We also illustrate spacial pooling and temporal memory to see how SDRs are involved in HTM algorithms. All these operations are implemented efficiently, with the complexity being linear in time in the number of ON bits in the SDRs.

Implementation details

In this module, we have laid the basic framework for a computational model of SDRs which can be used to extend and perform further experiments in the future. Our code structure is as follows.

1. **sdr.py**: contains the main class called SDR which includes 2 ways to initialise it, either from a vector directly or from an index list of position of ON bits. It also contains various methods and properties inherent to any SDR.

2. **sdr_utils.py**: includes utility functions of SDRs such as generating a random SDR, finding the probability of false positives, etc.

3. **spatial_pooling.py**: Implements the spatial pooling algorithm, and generates an overlap curve graph to compare with the one provided in the original paper.

4. **temporal_memory.py**: Implements temporal memory algorithm to calculate the active state in time step t, and predict the next states.

5. **union_graph.py**: Critically analyses the effect of an increasing number of vectors in the Union vector.

6. **property_graph.py**: Includes code to plot the variation of probability of false positives with noise.

Link to the GitHub Repository: https://github.com/drakari7/SDR-HTM-CogSci.

Mathematical properties of SDRs

This section describes some properties of SDRs, such as probability of mismatches, robustness, subsampling, classification of vectors, and unions. The properties demonstrate how SDRs can be used a memory storing device.

Definition

An SDR is an n-dimensional binary vector $x = [b_0, b_1, ..b_n]$ with a small percentage of ON bits. w_x is the number of ON bits in vector i.e., $w_x = ||x||_1$.

Overlap

Overlap score or similarity between two SDRs is the number of ON bits in the same locations in both the SDRs. It can be computed as the dot product of the two vectors.

$$overlap(x, y) = x \cdot y \tag{1}$$

Matching

A match is when the overlap between two vectors is greater than some threshold θ. This is useful for maintaining robustness to noise. Typically, θ is greater than $w/2$.

$$match(x, y) \equiv overlap(x, y) \geq \theta \tag{2}$$

Here $\theta \leq w_x, w_y$.

Uniqueness

The number of unique SDRs for a given n, w pair is given by:

$$\binom{n}{w} = \frac{n!}{w!(n-w)!} \tag{3}$$

The probability that two SDRs with the same n, w pair are identical is

$$P(x = y) = 1/\binom{n}{w} \tag{4}$$

This probability decreases rapidly as w increases and is essentially zero in sparse vectors. Hence, it is highly likely for two randomly encoded SDRs to be unique.

Overlap Set

The overlap set of x with respect to b is $\Omega(n, w, b)$, defined as the set of vectors of size n with w bits on, that have exactly b bits of overlap with x. The size of this set is given by:

$$|\Omega_x(n, w, b)| = \binom{w_x}{b} \times \binom{n - w_x}{w - b} \tag{5}$$

Here $b \leq w, w_x$.

Inexact Matching

Lowering θ decreases the sensitivity and increases the overall noise robustness of the system at the cost of more false positives. But with appropriate n, w values, SDRs can have great noise robustness with a very small number of false positives. With $\binom{n}{w}$ total patterns, the probability of a false positive is:

$$fp_w^n(\theta) = \frac{\sum_{b=\theta}^{w} |\Omega_x(n, w, b)|}{\binom{n}{w}} \tag{6}$$

As the first term dominates by an order of magnitude or more, this can be approximated as:

$$fp_w^n(\theta) \approx \frac{|\Omega_x(n, w, \theta)|}{\binom{n}{w}} \tag{7}$$

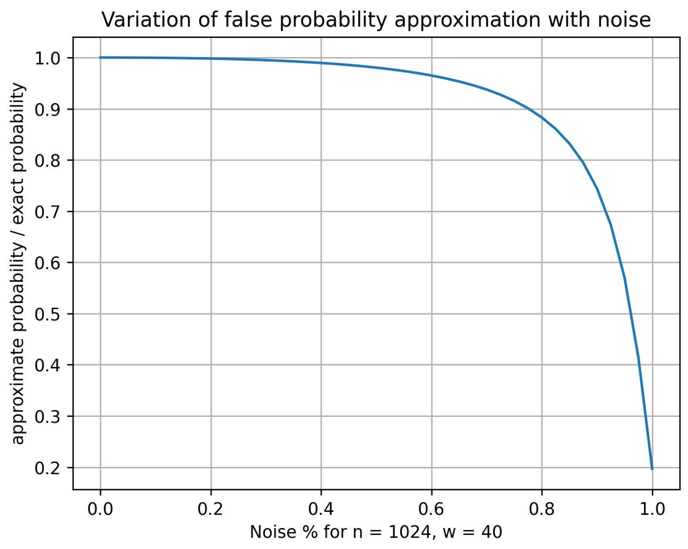


Figure 1: Validity of approximation for inexact matching

Subsampling

SDRs allow us to conveniently recognize large patterns only by comparing a subset of the ON bits in the large vector. But as the size of the subset decreases, the number of false positives increases although at a slow rate.

Let x be an SDR an x' a subsampled version of x. Now, the overlap set with respect to x' and a random vector y with $b \leq w_{x'}$ and $w_{x'} \leq w_y$ is:

$$|\Omega_{x'}(n, w_y, b)| = \binom{w_{x'}}{b} \times \binom{n - w_{x'}}{w_y - b} \tag{8}$$

And given a threshold $\theta \leq w_x$, the probability of a false positive between x' and the random vector y is:

$$fp^n_{w_y}(\theta) = \frac{\sum_{b=\theta}^{w_{x'}} |\Omega_{x'}(n, w_y, b)|}{\binom{n}{w_y}} \tag{9}$$

These equations differ from the above equations only by the vectors being compared.

Classifying SDRs

Suppose we have a set of M SDRs that are unique with respect to matching, we classify a new vector y as part of set M if it matches any one of the SDRs from set M. To see how reliably we can classify a vector, we do the following: since all vectors in set M are unique with respect to matching, the probability of getting a false positive is bounded by:

$$fp_x(t) \leq \sum_{i=0}^{M} fp^n_{w_{x_i}}(t) \tag{10}$$

When all the vectors in M have the same w, the above equation becomes:

$$fp_x(\theta) \leq M fp^n_w(\theta) \tag{11}$$

Union Property

The union property of SDRs enables us to store a set of M vectors by simply taking the OR of the vectors resulting in a new vector X. Now, to determine if a new SDR y is a member of the set, we compute $match(X, y)$. Thus, a fixed size vector can store and operate on a dynamic list.

$$y \in M \equiv match(X, y) = 1 \tag{12}$$

As the number of SDRs increases, the union vector gets saturated with ON bits to a point where it becomes useless (resulting is a great number of false positive matches).
Exact matching: With $\theta = w$, the probability that a given bit is zero in a set of M vectors is:

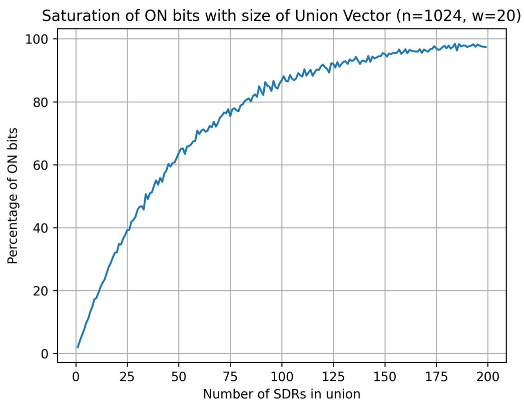
$$p_0 = (1 - \frac{w}{n})^M \tag{13}$$

Now, the probability of a false positive where all w bits in y (a random vector) are ON is:

$$p_{fp} = (1 - p_0)^w = (1 - (1 - \frac{w}{n})^M)^w \tag{14}$$

Note: there is a mistake in this equation in the original paper.

Here we present our findings on the rate of saturation of SDR Union v/s the expected theoretical graph presented in the paper. Note that since we ran our simulation on different parameters n and w, the exact values are different, however, the shape of the curve still remains the same.



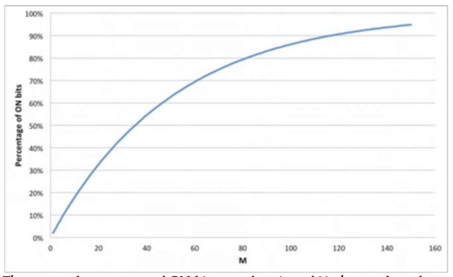


Figure 2: The expected percentage of ON bits as a function of M, the number of vectors in the union set. In this case, $N = 2048$ and $w = 40$, i.e. $s = 2\%$.

Figure 2: Union saturation curve plotted by us vs expected theoretical curve

Computational efficiency

All of the above proposed operations and functions on SDRs are linear in time in the number of ON bits. This is reflected in our implementation as we store an SDR with 2 characteristics, n, the size of the vector, and index_list, a set storing the position of the ON bits.

SDRs and HTMs

In this section, we look at SDRs in the context of the HTM neuron model. Specifically, we explore the Spatial Pooling (SP) and Temporal Memory (TM) algorithms described in the paper and implement the same in python.
The following are constants used in the code:
N : size of input vector
C : number of columns
k : the number of columns active after spatial inhibition
L : number of cells per column
S : number of segments per cell

Spatial Pooling

The Spatial Pooling algorithm takes a binary vector of length N as input such that the vector is usually sparse and each column of the vector represents proximal segments in a cell. A binary $N \times C$ matrix (doesn't have to be sparse but usually is in practice) is used to represent the set of connected synapses in the spatial pool. The vector and matrix are multiplied, resulting in a vector of overlap counts. Next, the indices corresponding to the top k overlaps are determined as the winners in the inhibition step and are made equal to one and the rest are made zero, resulting in a binary vector of size $1 \times C$.
This algorithm examines the overlap between a randomly initialised matrix and the binary input vector (the SDR) and returns an SDR in which the indices corresponding to the top k columns form the ON bits. This SDR is then used as an input in Temporal Memory (TM) algorithms.

The overlap curve: Let X be a set of random binary vectors with size n and w_x ON bits. The probability that a new random vector y matches with exactly one vector from set X is:

$$p(overlap(x,y) = b) = \frac{|\Omega_y(n, w_x, b)|}{\binom{n}{w_x}} \tag{15}$$

The expected number of columns with b bits of overlap is $|X| = C$. So the overlap for each column after sorting would result in the overlap curve. From the curve, we observe that the sharper the drop off after k, the more robustness in the system. Thus, the robustness to noise can be increased by making the overlap curve sharper.

We have produced the overlap curve for the data with n = 1024, and w = 30, and compared it with the original paper's predicted overlap curve. We can see that the results are similar. We again provide experimental proof of expected theoretical results.

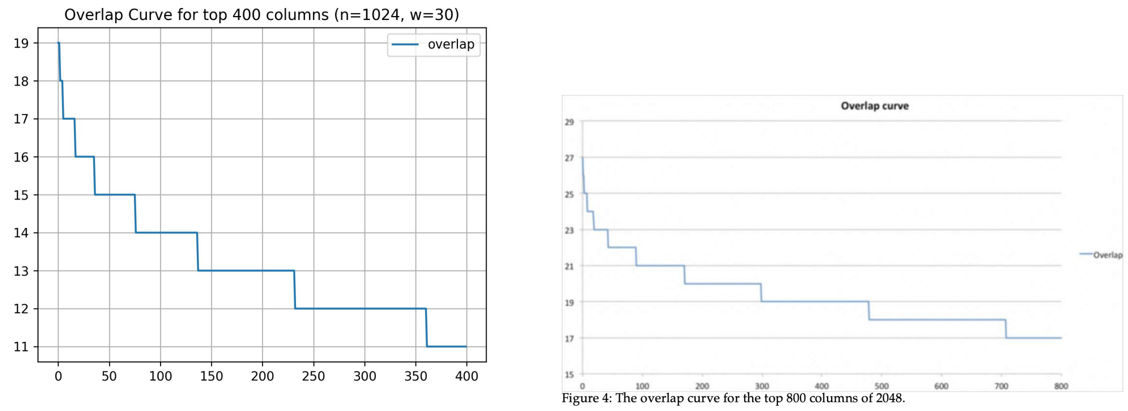


Figure 3: Overlap curve from our experiment vs predicted overlap curve using probabilistic analysis

Temporal Memory

The TM algorithm can be explained in two phases:
Phase 1: In this phase, the current active temporal states (at time step t) are calculated following Spatial Pooling. The cells in each column of the predicted state and the SP output SDR are examined. The ON cells in the columns corresponding to the ON bits in the SDR stay ON, representing the active temporal state. In other words, row-by-row element-wise multiplication is done between the $1 \times C$ SDR and the $L \times C$ predicted state, resulting in an $L \times C$ active state, all of which are sparse.
Phase 2: In this phase we calculate the predictions for the next time step. We take the current active state, and do classification with the neighbouring cells. Cells are modeled in the form of a list of segments, where each segment is an LxC SDR. On obtaining the match scores, we can then predict the state for the next time step.
We can perform predictions for multiple cells by taking their unions, and the robustness of the union property allows us to do this. In this way, for the same input, different cells can be ON and provide different contexts. The number of contexts is truly large, for typical values, in the order 1.33×10^{36}. We can also calculate the number of cells we can predict for till it becomes too saturated and the result becomes useless.

Conclusion

In this part, authors should conclude the significance of the study, emphasize its value and state expectation on future studies that may need to be carried out. In details, it may include summary of key findings, strengths and limitations of the study, controversies raised by this study, and future research directions, etc.

Acknowledgement

We would like to express our gratitude to our professor, Prof. Nisheeth Srivastava for giving us the opportunity to work on this project as part of the Computational Cognitive Science course and guiding us throughout the process.

References

1. Subutai Ahmad, J. H. (2015). Properties of sparse distributed representations and their application to hierarchical temporal memory. *Numenta*