On the Relationship Between Input Sparsity and Noise Robustness in Hierarchical Temporal Memory Spatial Pooler

Damir Dobric
University of Plymouth
Drake Circus
PlymouthDevon PL4
+441752 600600
ddobric@daenet.de

Andreas Pech
Frankfurt University oAS
Nibelungenplatz 1
60318 Frankfurt am Main
+49 69 15330
pech@fb2.fra-uas.de

Bogdan Ghita
University of Plymouth
Drake Circus
Portland Square B334
+441752 600600
bogdan.ghita@plymouth.ac.uk

Thomas Wennekers
University of Plymouth
Drake Circus
Plymouth Devon PL4
+441752 600600
thomas.wennekers@plymouth.ac.uk

ABSTRACT
Hierarchical Temporal Memory - Spatial Pooler is a cortical learning algorithm inspired by the biological functioning of the neocortex. It is responsible for the sparse encoding of spatial patterns used as an input for further processing inside of a Hierarchical Temporal Memory (HTM). During the learning process, the Spatial Pooler groups spatially similar inputs into the same sparse distributed representation (SDR) memorized as a set of active mini-columns. The role of SDR generated by the learning process of the Spatial Pooler is to provide an input for learning of sequences inside of the HTM. One of the features of the Spatial Pooler is also the robustness to noise in the input. This paper summarizes the work in progress, which analyses the relationship between the encoding of the input pattern and the robustness of the memorized pattern against noise. In this work many synthetic input patterns with different sparsity were used to set the hypothesis, which claims that SP robustness against the noise in the input depends on the sparsity of the input. To validate the hypothesis, many random input vectors with a large portion of noise were generated. Then the change of SDR output was compared with the change of input by the given portion of noise. It was shown that the SDR output change is very small in comparison to change of the input by adding of a large portion of the noise in the input. By adding of a significant portion of the noise to the input, the learned output remains almost unchanged. This indicates a great robustness to noise. Experiments show that the robustness against noise of the Spatial Pooler directly depends on the sparsity of the input pattern. Preliminary tests suggest implementation of a boosting mechanism of the input to improve the robustness against noise.

CCS Concepts
• **Theory of Computation → Theory and algorithms for application domains → Machine learning theory → Unsupervised learning and clustering**

ESSE 2020, November 6–8, 2020, Rome, Italy

ACM ISBN 978-1-4503-7762-1/20/11…$15.00

https://doi.org/10.1145/3393822.3432317

Keywords
Artificial intelligence, temporal memory, HTM, machine learning, neocortex, noise robustness, spatial pooler

1. INTRODUCTION
Hierarchical Temporal Memory (HTM)-Spatial Pooler (SP) is a Learning Algorithm inspired by biological processes in the neocortex. It is responsible for the sparse encoding of spatial patterns typically used as an input for further processing inside of Hierarchical Temporal Memory (HTM). HTM is known as a theoretical model that incorporates many algorithmic properties of the neocortex [1]. One of the main modules of an HTM is the Spatial Pooler, which is responsible for continuous encoding of sensory input streams and other sources of information, from different regions and layers of the neocortex [1]. Processing of information in the neurons inside the HTM is sparsely encoded as in biological neuronal circuits [2]. HTM in a nutshell, uses Hebbian learning rules [3] on binary sparse arrays represented as sequences of integers (0/1). One of the most important features of a Spatial Pooler is its capability to encode streams of zeros and ones as Sparse Distributed Representations (SDR). Spatial Pooler groups similar spatial patterns represented as active neurons into highly

sparse representations of cortical mini columns [4]. SDRs produced by Spatial Pooler typically engage a small percentage (i.e.: 2%) of simulated columns only. The SP algorithm is inspired by encoding of the spatial input pattern as found in neocortex [1,2]. It is extremely robust on a potential damage of certain neurons, which connect mini columns [5] and can efficiently remember spatially similar patterns. A further, in this context more important capability of a Spatial Pooler is the robustness against noise, as documented in earlier work [1,5].

In this study, we measure the sensitivity of the SP output representations against input noise. Adding more and more noise to the input, trained outputs for specified inputs keep stable for large fractions of input noise. This result demonstrates that a trained SP is robust against noise in contrast to an untrained SP. In some previous work [6] it was found that Spatial Pooler can be successfully used in frameworks to improve the robustness against noise. However, it remains unclear how this capability is related to the input pattern itself. The aim of the experiments is to test the hypothesis that the results depend on the sparseness of the spatial input. This work aims to investigate how encoding of input patterns influences robustness of SDRs against noise when using a Spatial Pooler. Existing work do not suggest how the input should be encoded to achieve higher robustness by passing the input through

SP and getting output without or with less noise than input. Knowing this is essential to easier implement reliable real-world applications based on Spatial Pooler, which are robust to noisy input. Assuming that sparseness is defined as the number of active neurons divided by the number of available bits [1], experiments in this work show that a higher number of non-zero bits in the representation of an input spatial pattern leads to better robustness against noise.

In section 2 the hypothesis is discussed, which refers to the influence of the number of non-zero bits in the input vector on the robustness against noise. Furthermore, the methodology and the experiments used to validate the thesis are described.

Figure 1. Digits "7" and "8", 28x28 pixels obtained from the MNIST database. These images were originally used to measure noise robustness of trained and untrained Spatial Pooler. Image 8 and image 7 have sparseness of 17% and 6% respectively. Sparseness is defined as a fraction of non-zeros in the total number of bits.

Section 3 describes the results of the experiments presented in this paper. It is explained how the noise robustness depends on the sparseness of the input, which indicates that input patterns can be amplified (increased) to achieve a better memorizing of the reference pattern and to produce a more stable representation of SDR with a very high noise level.

2. METHODS

The Spatial Pooler (SP) performs fast memorizing of spatial patterns by leveraging a sparse encoding [7] and has a good robustness against noise. This was shown by measuring the sensitivity of the SP to a varying amount of input noise [1]. For each input, a subset of the active input bits is randomly flipped to inactive bits, and of inactive input bits to active bits. By following observations in [1], in this work robustness was measured by varying the sparsity of the input vector while training the Spatial Pooler instance. To illustrate the influence of sparsity, first, two images of 28x28 pixels were taken from the MNIST database. The images were binarized to 0-1 vectors and used as input during the training process.

In all experiments a set of input vectors were presented to the Spatial Pooler for few cycles (iterations). In every cycle Spatial

Pooler learns the spatial pattern and generates a set of active columns. Resulting active columns represent the SDR code of the learned input. The SDR code (active columns) is stable if it does not change over time by presenting the same input to the Spatial Pooler.

All the experiments in this paper were executed using the HTM implementation in .NET Core [8]. To be sure that results presented in this paper are not a framework related, experiments have also been compared to results produced by Python and JAVA HTM implementation [9] as shown in the earlier work [1]. For every image, the same instance of Spatial Pooler was used with the parameters shown in Table 1.

Spatial Pooler can operate in two modes, depending on the use of inhibition in the model [1], which influences how encoding works. The first mode is global inhibition, which calculates the overlap between input and columns across the whole image representation.

Table 1. Spatial Pooler parameters used in the experiments. Parameters shown in this table are commonly used in most HTM experiments. However, DUTY_CYCLE relevant parameters are chosen to prevent Spatial Pooler from early boosting. When Spatial Pooler starts column boosting, it will "forget" learned pattern. To avoid "forgetting" of learned patterns column boosting is deactivated during experiments. For more detailed information about the meaning of all parameters please see [8].

Parameters	Value
POTENTIAL_RADIUS	-1 (all inputs are connected to every column)
POTENTIAL_PCT	1
LOCAL_AREA_DENSITY	-1
NUM_ACTIVE_COLUMNS_PER_ INH_AREA	0,02*64*64 (2% of columns)
STIMULUS_THRESHOLD	0,5
SYN_PERM_INACTIVE_DEC	0,00
SYN_PERM_ACTIVE_INC	0,01
SYN_PERM_CONNECTED	0,1
MIN_PCT_OVERLAP_DUTY_CYCLES	0,001
MIN_PCT_ACTIVE_DUTY_CYCLES	0,001
DUTY_CYCLE_PERIOD	100
MAX_BOOST	10

In contrast, the local inhibition (the second mode) takes slices of the image (input vector) by splitting it to a neighbourhood area of each input bit and then calculating the overlap locally for each area. Results of all experiments presented in this paper were observed by using the global inhibition. The Spatial Pooler tracks column activity during the learning process. If some columns are not sufficiently participating in the learning process, they will be stimulated to achieve uniform participation of all columns. This process is referred to as boosting and it can be influenced by using various parameters. Unfortunately, boosting can have a negative influence on the learning process as, once it is activated, SP can enter an unstable state and forget the learned patterns. Once SP enters unstable state due to the boosting mechanism, the SP is no longer able to classify learned patterns for the next few cycles. For this reason, in all experiments the boosting of columns was minimized by using various DUTY_CYCLE parameters, as shown in the Table 1. This approach makes sure that, eventual boosting will not erase the spatial patterns memorized during the learning process.

The whole learning process was set to take ten iterations steps, which ensure that the Spatial Pooler may entered stable representation of both images in active columns set.

Given the parameters listed in Table 1 and associated input vectors, the Spatial Pooler enters a stable state in usually 2-3 steps. Ten iteration steps were chosen to ascertain this.

Various levels of noise (5%, 10%, 15%,..,100%) were added to the original reference image. Adding noise consists of random flipping of zeros to non-zeros and non-zeros to zeros for the given percentage of all pixels in the image. For every noise level, the existing instance of Spatial Pooler was trained with the noise-deformed representations of the reference input.

After every training step, the input distance and output distance between vectors were calculated. The distance in this context is defined as a measure *d*, which quantifies a similarity between compared vectors.

$$o_k = \sum_{i=0}^{N} |a_0(i) \circ a_k(i)|_0 \quad (1)$$

$$l_k = \sum_{i=0}^{N} a_k(i) \; | o_k \le l_k \le N \quad (2)$$

$$d_k = \frac{l_0 - (l_0 - o_k)}{l_0} = \frac{o_k}{l_0} \quad (3)$$

The input and output vectors with noise were compared with the respective reference input and output vectors without noise. Finally, the input and output distances were compared and the relation between them was used as a measure to determine resistance against noise. The model is robust against noise if a smaller output distance is observed for the same or higher input distance. In other words, for relatively significant fractions of noise in the input, the output keeps almost unchanged.

As defined in equation (1), first the overlap o_k between vectors and the reference vector is calculated. It is defined as a number of overlapping bits at the same position i between reference vector (input or output) and input *k*.

$a_O$is a reference vector (input or output) without noise and a_k is a vector (input or output) with specific noise level defined by noise level index *k*. Both compared vectors must have the same length N. Every input vector with noise is bitwise compared with a reference input by using the L0-norm, which defines the total number of non-zero elements of the vector at the same position (elements are binary values). The overlap o_k between two vectors is calculated as the number of non-zero bits at corresponding position.

Value l_k in equation (2) represents the number of non-zero bits of the *k*-th noisy vector. For *k=0* equation (2) represents the number of non-zero bits of the reference vector without noise. The distance d_k in equation (3) defines the measure used in this experiment, which shows how similar is vector a_k to a reference vector a_0. It is a fraction of number of non-zero bits of the reference vector subtracted by number of different bits at positions of non-zero bits of the reference vector and total number of non-zero bits of the reference vector. This measure is referred as a *distance* and it is used in all experiments described later in this paper.

Individual experiments traverse the list of predefined or randomly generated input vectors. For every vector in the list of input vectors, the Spatial Pooler is trained to remember the spatial pattern of the input vector. The training is repeated 5 times, to ensure that the pattern is fully learned. In this context, the pattern is considered as learned, if the set of active columns calculated by the Spatial Pooler algorithm does not change if the already learned (trained) input vector appears again. By following this definition, the pattern is not fully learned if the output of Spatial Pooler for the same input vector produces active columns which change from cycle to cycle.

Note that between input vector and Spatial Pooler no further encoding is used. Usually, when working with the Spatial Pooler, using an encoder like Scalar Encoder, Date Encoder or similar is required. In experiments described in this paper, input vectors are raw images or random vectors used without any additional encoding.

The pseudo code in *Algorithm 1* shows how the training was implemented. The function *NoiseTest* gets as an argument the list of input vectors. Every input vector is trained with 6 noise levels. The first noise level is zero-level, which represents the input vector without noise. The next five levels (1-5) adds 5% more of noise each to the original (reference) input vector (5%, 10%, 15%, 20% and 25%). The Spatial Pooler is trained with the reference input without noise (second argument of the *train* function is *true*).

For every result, the distance d_k is calculated by equation (3) between the column output for a given noise level and the column output for the same input vector without noise (reference output).

Algorithm 1 - Spatial Pooler training

```
Function NoiseTest(InputVectors)
 for i=0 to len(inputVectors) do
  for j=0 to 5 do
   if j=0 then
     output[i,0] ←train(inputVectors[i],true)
   else
    noiseVector ←addNoise(inputVectors[i], j*5/100)
    output[i,j] ←train(noiseVector,false)
   end if
   d ←calcDistance(output[i,0], output[i,j])
   writeResult(i,noiseLevel,d)
  end for
end for
End Function

Function Train(inputVector)
  for i=0 to 10 do
   activeCols ←sp.compute(inputVector, true)
  end for
End Function

Function AddNoise(inputVector, noisePct)
  //Flip pct percent of zero bits to non-zero and vs. versa
  return vectorWithNoise.
End Function
```

3. RESULTS

In the first experiment, many MNIST images were tested. The robustness was calculated for different digits. As an example, Figure 2 shows the distance of the trained Spatial Pooler instance for two randomly selected MNIST images. The distance is calculated as described by equations (1,2 and 3). According to the orange line, Image "7" (sparseness 6%) holds a stable set of active columns until approximately 30% of noise. This means that when the SP presents the input with the high noise level, it still outputs almost the same output as it has learned for the same input without noise.

Similarly, image "8" (sparseness 17%) is stable until about 40% of noise. It can be assumed that image "8" is more resistant to noise than image "7".

Figure 2. The blue line shows the calculated distances between input and noised input after adding noise. Orange lines show the distances between active columns encoded by SP for the reference input (without noise) and the output columns encoded by SP for a given noised input vector.

To validate this hypothesis, two synthetic box images with 1024x1024 pixels were first created (see Figure 3) and used to visually represent how the SDR code (output of SP) changes as a function of the noise in the input vector.

As next, a set of random reference input vectors was created with the fixed number of non-zero bits. Finally, it was analysed how the output changes depending on the input by adding more and more noise to the reference input for each of the input vectors in the random set (Figures 6 and 7).

To visually present the input and output in dependence on the noise, the Spatial Pooler was trained with two reference images without noise (Figure 3). Non-zero bits are displayed in greyscale and zero-bits in yellow. The first image referred to as "Box1", contains 240 and second image "Box2" contains 110 non-zero bits. Zero bits are represented yellow. This corresponds to a sparsity of 0.24 and 0.11 respectively. Box2 is sparser than Box1.

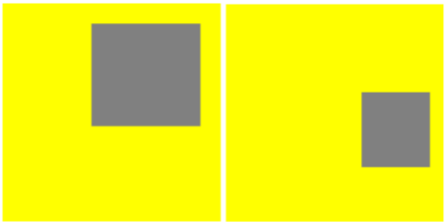

Figure 3. Two boxes 1024x1024 pixels (in yellow) with different sparsity (0.24 on the left and 0.11 on the right) have been used for training.

For both images in the experiments, noise levels of 5, 10, 15, 20,..100 percent were applied to the reference image. Figure 4 shows the results of five different portions of the noise for the two boxes defined in Figure 3.

Every image-pair in Figure 4 contains representations of the input with noise (left) and the resulting SDR (right).

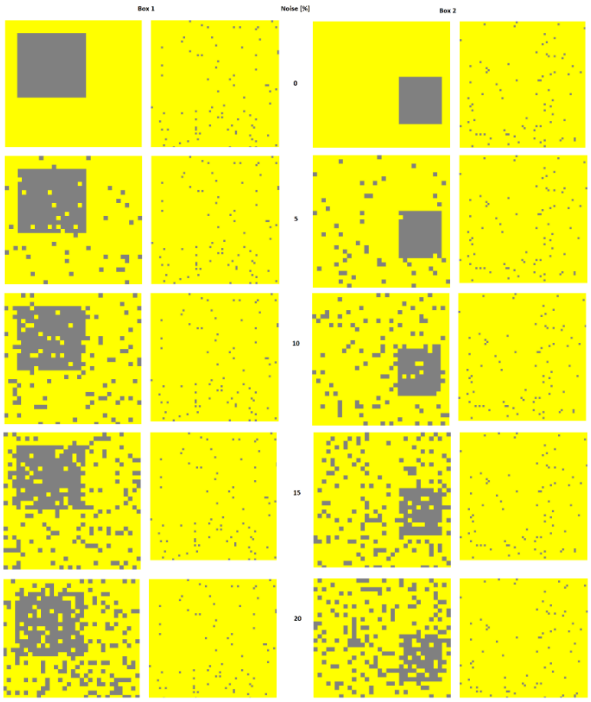

Figure 4. Sparse representation of Box1 (image pairs on left) and Box2 (image pairs on right) together with the noised inputs. Each image represents the input vector with specific noise percentage (left side of each pair) and their sparse representation of active columns (right side of the pair).

The output of the Spatial Pooler is the set of active columns as calculated by Spatial Pooler for the given input. The input with 0% noise represents the reference input. This input is followed by inputs with 5%, 10%, 15% and 20% of noise subsequently injected in the reference image.

Figure 4 (Box1 left and Box left) clearly shows that the original input was massively destroyed by the addition of a large amount of noise. Sliding through the SDR representation of the two boxes in Figure 4 (box 1 on the right and box 2 on the right), one can see that the SDR remains stable.

To validate this, the input distance and the output distance were calculated using equation (3), as shown in Figure 5. As expected, the input distance (blue line) changes linearly by adding the noise

to the input. Interestingly, the output distance (orange line) remains stable as more and more noise is added. By following the results from Figure 5, the Box1 is still recognized with the output distance of near to 100% for a given input distance of approx. 40% (Figure 4 on left).

Results for Box2 look slightly different. Image with Box2 gets well recognized for inputs with approx. up to 25% of noise.

That could mean the Box1 with the higher number of nonzero bits shows better results than Box2. Results shown in figures 2, 3 and 4 indicate that more non-zero bits in the input vector lead to more resistant sparse representations of active columns (output).

To validate this hypothesis the Spatial Pooler was trained with thousand randomly generated reference input-vectors without noise.

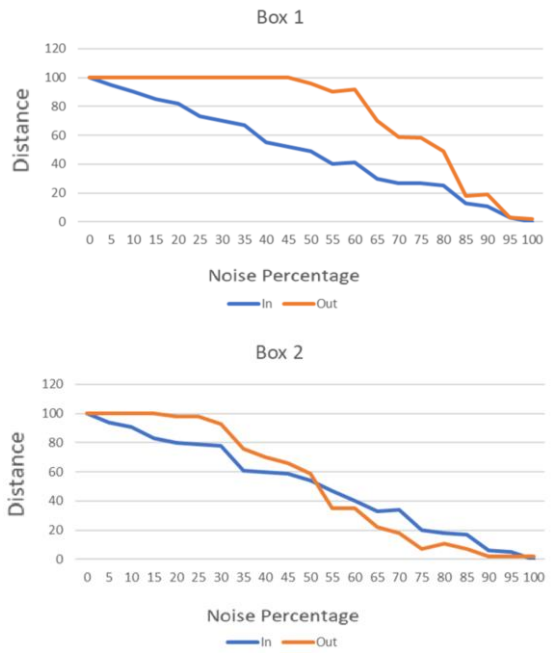

Figure 5. Noise robustness of two boxes stimuli. Blue line shows distance between noised input vectors and reference input vector without noise. Orange line shows the distance between output of trained Spatial Pooler of noised input and output of reference input.

After the Spatial Pooler has learned these random input-vectors (reference inputs), for every random input a set of random noise-vectors with the noise was created by varying the percentage of non-zero bits.

The set of input vectors was shared in four groups with 10, 20, 30, and 40 percent of non-zero bits. For this set of vectors, the Spatial Pooler was trained with the *Algorithm 1*. The training was done with activated learning and then with disabled learning. This can be configured by using the second Boolean argument of the calculation method in the *Train* function. This is the standard capability of the Spatial Pooler, which, when learning disabled, uses internal inhibition mechanisms in neurons, but does not increase the permanence values (usually referred to as "width" in neural networks). This is also called inferring (prediction) mode. In this mode, a new input is not learned, but encoded to SDR. The input and output overlap shown in Figures 6 and 7 are calculated as the average value over the set of noisy vectors with the noise 1-100% for every of input vectors inside of four groups. The overlap shown in Figures 6 and 7 is the function that is inversely proportional to the distance function, as shown in Equations 1, 2 and 3. For example, zero noise corresponds to 100% overlap etc.

Figures 6 and 7 show results with disabled and enabled learning, respectively for groups of 10, 20, 30 and 40 percent of non-zero bits. Using Spatial Pooler with disabled learning does not memorize the patterns and is not resistant against noise. As shown in figure 6 the output overlap changes even faster than the input overlap, especially at lower noise levels, which are more applicable in practice.

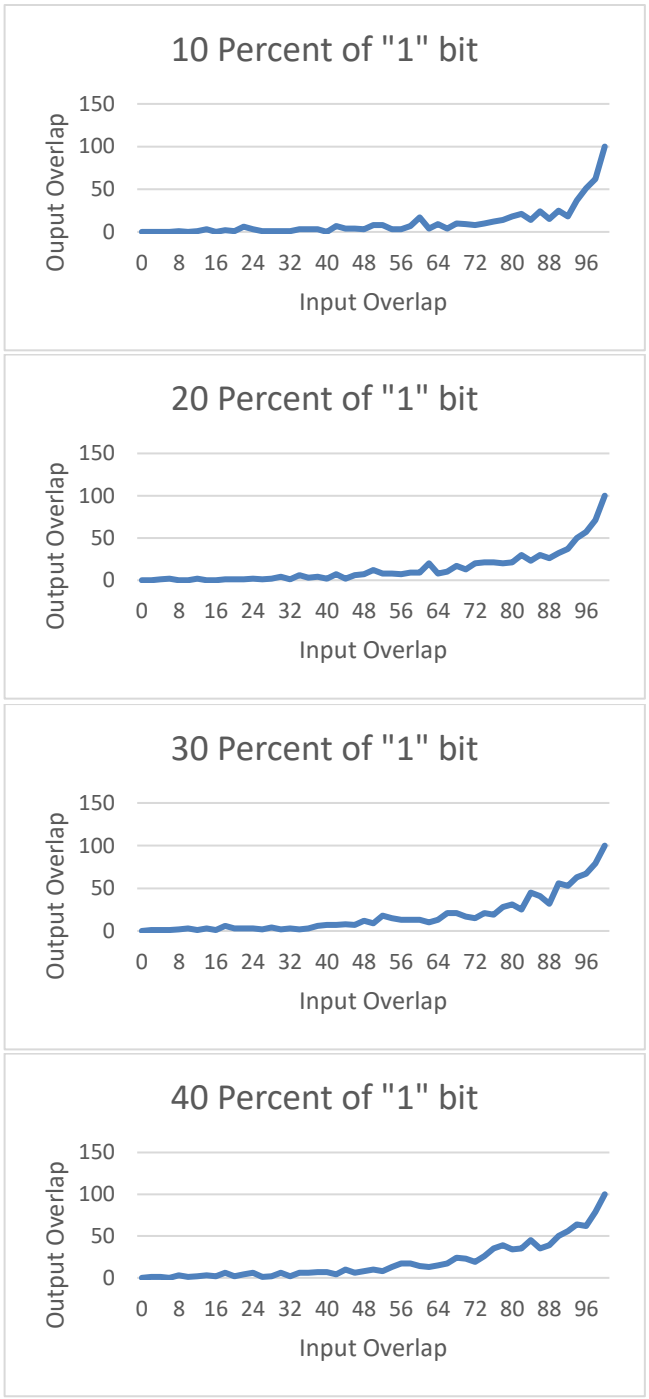

Figure 6. Relation between the input and output overlap for the input with noise by a given percentage of non-zero bits in the input vector. Results are shown for SP with disabled learning, also known as inference mode.

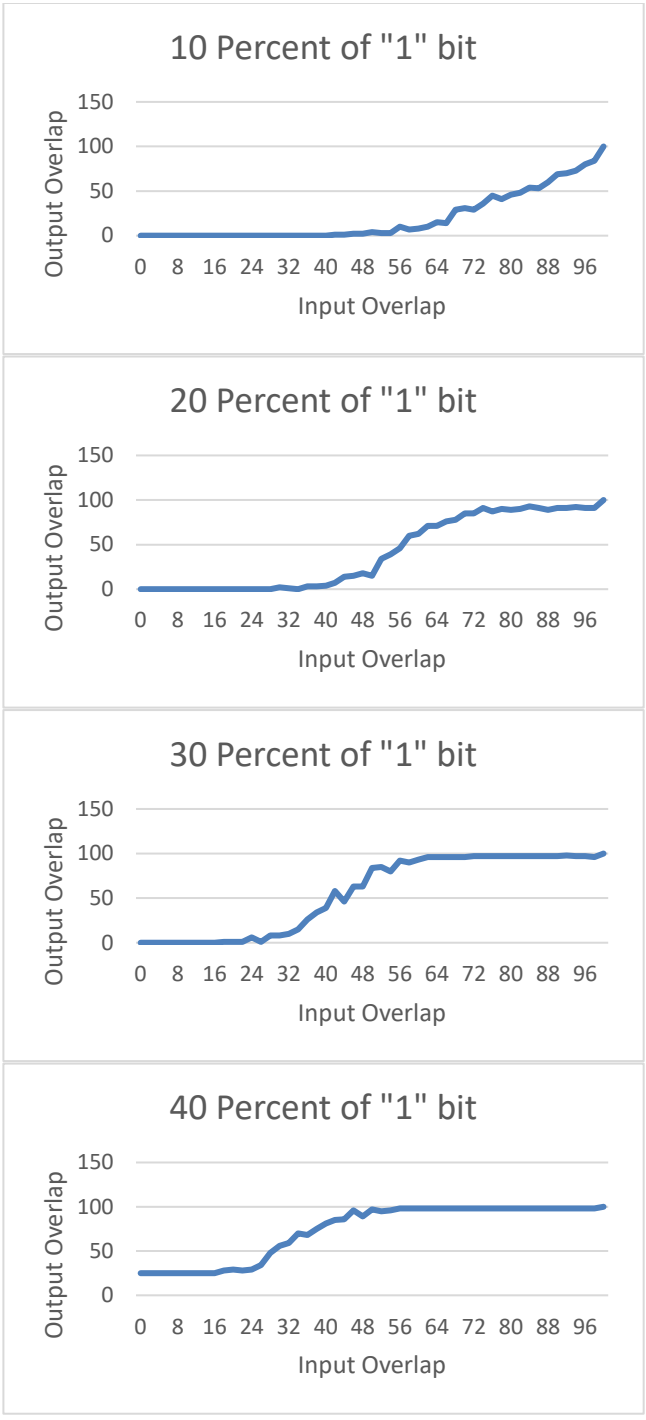

Figure 7. Relation between the input and output overlap for the input with noise by a given percentage of non-zero bits in the input vector. Results are shown for SP with enabled learning.

It is obvious, the Spatial Pooler does not show good resistance when used for images with less than 10% non-zero bits in conjunction with learning enabled (first figure). In this case, even slightly deforming an input immediately deforms the output, which is not a wanted result.

By comparing results with enabled learning (Figure 7) and disabled learning (Figure 6), it can be concluded that the Spatial Pooler keeps stable SDR output with enabled learning.

With enabled learning and a higher number of non-zero bits input, the distance between the output of the vector with noise applied and a reference output with zero-noise keeps almost constant for relatively wide range of noise. This indicates that the Spatial Pooler "remembers" the reference values even under noisy conditions. For example, by increasing the number of non-zero bits by 20% (second diagram in Figure 7), the output keeps stable with more than 90% of output distance by almost 25% of noise in the input. With 30% of non-zero bits, the Spatial Pooler will maintain a virtually unchanged output for images with noise of up to 50%. This is a good indicator for the robustness of the Spatial Pooler, and it shows that an increased number of non-zero bits in the input leads to a higher robustness.

Unfortunately, higher number of non-zero bits also decreases sparsity and reduces the learning capacity [5, 7] of the Hierarchical Temporal Memory. If the higher number of mini-columns is required to increase a capacity the distributed and parallel version of Spatial Pooler can be used [12]. Given these recognised problems, the Spatial Pooler can be used in various real-world applications [10, 11] to generate an SDR-encoded representation of the reference (ideal) output from the input with noise

The signal with added noise is presented to the trained Spatial Pooler in the inferring mode, which computes the reference output for a given input. Finally, a mapping component is needed (not further discussed in this paper), which maps the SDR of the output the reference input without noise. The following pseudo code demonstrates how to remove the noise.

Algorithm 2 - Remove Noise

Function *RemoveNoise(InputVectorWithNoise)*

inferringMode ← *true*
output ← *sp.compute(inputWithNoise, inferingMode)*

referenceInput ← *mapToInput(output)*

return *referenceInput*

End Function

4. CONCLUSION

Results presented in this paper show that the number of non-zero bits in input vectors, that encode a specific spatial pattern, has a massive influence on the noise robustness of the Spatial Pooler. To successfully remember some spatial pattern and achieve better noise robustness, a higher number of non-zero bits should be used when encoding an input. However, an increase of the number of non-zero bits will also lead to a decrease in the overall capacity of the memory and higher probability of false positives. This is a trade-off, which needs to be considered when building applications.

In the case of encoding of inputs by encoders (i.e. scalar encoder, category encoder, etc.), as proposed by the HTM framework, encoders can easily be adapted to boost (increase) encoding values. However, if using images or similar spatial patterns, other technique must be used. To achieve better results for such kind of inputs, it is proposed to implement a new spatial boosting input layer or a pattern boosting encoder. Note that this is not related to

the existing column boosting algorithm, which has a different purpose [7]. The proposed spatial boosting input layer should receive the encoded input from an encoder (by keeping the same semantical meaning of an input), increase the number of non-zeros by boosting neighbourhood area around existing non-zero sensory input cells before passing it to the Spatial Pooler. Finally, this component should also make sure that all semantically different patterns have a similar sparseness, because different sparseness of inputs will lead to different robustness against noise.

5. REFERENCES

[1] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," Frontiers in Computational Neurosince, vol. , no. 111, p. 15, 2017.

[2] Luca A. Finelli, Seth Haney, Maxim Bazhenov, Mark Stopfer, Terrence J. Sejnowski, "Synaptic Learning Rules and Sparse Coding in a Model Sensory System," PLOS Computational Biology, 2007.

[3] Wulfram Gerstner, Werner M. Kistler, "Mathematical formulations of Hebbian learning," Gerstner, W., Kistler, W. Mathematical formulations of Hebbian learning. Biol Cybern 87, 404–415 (2002).

[4] KAAS JH. Evolution of Columns, Modules, and Domains in the Neocortex of Primates. In: National Academy of Sciences; Striedter GF, Avise JC, Ayala FJ, editors. In the Light of Evolution: Volume VI: Brain and Behavior. Washington (DC): National Academies Press (US); 2013 Jan 25. 7. Available from: https://www.ncbi.nlm.nih.gov/books/NBK207177/

[5] L. S. Subutai Ahmad, "How Can We Be So Dense? The Robustness of Highly Sparse Representations," CoRR, vol. abs/1903.11257, p. 11, 2019.

[6] Maciej Wielgosz, Marcin Pietroń, Kazimierz Wiatr, "Using Spatial Pooler of Hierarchical Temporal Memory for object classification in noisy video streams," Frontiers in Neural Circuits, vol. 10, p. 13, 2016.

[7] Jeff Hawkins, Subutai Ahmad, "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex," Frontiers in Neural Circuits, vol. 10, p. 13, 2016.

[8] Damir Dobric, NeoCortexAPi, the implementaiton of HTM in .NET Core, GitHub: https://github.com/ddobric/neocortexapi, 2018

[9] Numenta, the platform for intelligent computing, GitHub: https://github.com/numenta/nupic

[10] Fallas-Moya F., Torres-Rojas F. (2018) Object Recognition Using Hierarchical Temporal Memory. In: Brito-Loeza C., Espinosa-Romero A. (eds) Intelligent Computing Systems. ISICS 2018. Communications in Computer and Information Science, vol 820. Springer, Cham

[11] Bonhoff, Gerod M., 1st Lt, USAF, Using Hierarchical Temporal Memory for Detecting Anomalous Network Activity, March 2008.

[12] Damir Dobric, Andreas Pech, Bogdan Ghita, Thomas Wennekers, "The Parallel HTM Spatial Pooler with Actor Model," in International Conference on Artificial Intelligence and Soft Computing (AIS 2020), Helsinki, 2020.