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

Investigation of Hierarchical Temporal Memory Spatial Pooler's Noise Robustness and Specificity

Sang Nguyen
phuocsangnguyen97@gmail.com

Duy Nguyen
ngthanhduy7@gmail.com

***Abstract*— The Thousand Brains Theory of Intelligence is a new and rising approach to understand human intelligence. The theory attempts to explain the fundamental principles behind human intelligence through many discovered biological evidences and logical reasoning. This theory lays the foundation for Hierarchical Temporal Memory (HTM) - an AI framework, which has many applications in practice. In this paper's HTM model, building block of a basic HTM structure comprises of an Encoder, a Spatial Pooler and a Temporal Memory. This fundamental component has two prominent features: noise robustness and prediction. The Spatial Pooler is mostly responsible for noise handling function of the complete structure. This paper provides some experimental data and comments about the reliability of the Spatial Pooler's noise handling function. Specifically, the test input in this paper is a simple data set, the level of noise robustness is measured by the similarity between outputs of the Spatial Pooler when it takes the original or the additive Gaussian noisy data sets as inputs, provided that it is only trained with the original data set. Additionally, the specificity, defined in this paper as the ability to differentiate between two different inputs, is also discussed.**

Keywords—Noise robustness, additive Gaussian noise, input differentiation, Spatial Pooler, HTM performance.

I. Introduction

The Thousand Brains Theory of Intelligence (TBTI) was introduced by Jeff Hawkins in his book "On Intelligence" [1]. This is a new biological theory, whose goal is to unify different available scientific discoveries in order to come up with a sensible and rigorous explanation for the unique intelligence of human.

Hierarchical Temporal Memory (HTM) is the result of the TBTI [2]. This AI framework, as described in [3], is different from the others, because it is "biologically constrained", meaning it is heavily inspired and based on details of how the brain actually works. However, this does not mean HTM tries to recreate the real structure of human brain, it omits some details that are not relevant to the fundamental principles of intelligence's operation [3]. Therefore, the building block of basic HTM structure is concrete and rigorous, containing three main components: Encoder, Spatial Pooler and Temporal Memory.

The main function of the Encoder is to translate different types of perception inputs into a final unified representation, which could be understood by the Spatial Pooler [4]. The Spatial Pooler then processes and picks up important semantic meanings of the translated inputs in order to ensure a robust representation of these inputs [5] by means of Sparse Distributed Representation [6]. Finally, the Temporal Memory learns and tries to predict the pattern of the sequential inputs [7].

Obviously, the Spatial Pooler plays an important role in the overall operation of HTM structure. It is, therefore, necessary to investigate the reliability of the Spatial Pooler module. The content here is based on some previously published results [8], in which, the effect of the input sparsity on noise robustness of the Spatial Pooler is investigated with discrete, separate test inputs. This paper, on the other hand, concerns with Spatial Pooler's reliability with respect to a complete set of input. Specifically, the Spatial Pooler is trained with a simple noise-free data set. After that, it is used to infer some noisy versions of the data set. The outputs of the Spatial Pooler in these two phases is then compared and investigated.

The main goal is to investigate the noise robustness of the module within a practical scenario. Additionally, *the specificity* (*defined in this paper as the ability to differentiate inputs with subtle differences*) of the Spatial Pooler is also researched in the same context.

This paper includes description of experimental procedure to accomplish the above mentioned goals, the experimental data and discussion on these results.

II. METHODS

A. Overview

All experiments in this paper are conducted with .NET® implementation of the Scalar Encoder and the Spatial Pooler in [9].

In this experiment, the tested HTM module includes two components: Scalar Encoder and Spatial Pooler. This module as a whole is fed two different types of scalar input data sets:

- In "training mode", the input is a noise-free data set. The Spatial Pooler "learns" this input. The number of learning iteration is 20.
- In "testing mode", the inputs are noisy (additive Gaussian noise) versions of the original data set. The Spatial Pooler does "not learn" from this input, only infer the result basing on what it already learned in "training mode".

The outputs of the Spatial Pooler are in the form of bit arrays. The Spatial Pooler's outputs with respect to every sample of the original input data set and the corresponding sample from the noisy input data set are compared in terms of similarity percentage (percentage of the same active bits), which is named in this paper "Hamming distance". These values are then averaged over all samples. The final result is that each pair of original and noisy input data sets is characterized by a "similarity" value. The higher this value is, the more similar the two (noise-free and noisy) outputs of Spatial Pooler are, and the more robust the Spatial Pooler is. One hundred percent "similarity" means the two bit arrays are exactly the same, zero percent "similarity" means they are totally different. In this paper, "similarity" higher than 90% means sufficient robustness.

The "specificity" of the trained Spatial Pooler is measured from noise-free Spatial Pooler's output. The "Hamming distance" between outputs of trained Spatial Pooler with respect to two consecutively incremental input values regarding amplitude is measured. This value implies the ability of the Spatial Pooler to differentiate two different meaningful inputs from the training data set. Ideally, for every consecutively-incremental-input-values pair from the training data set, this values should be less than 100% and higher than 70-90%. This makes sure that, while being robust to noise, the Spatial Pooler could still be specific enough to differentiate input values with subtle differences at the same time.

Different levels of noisy input are tested to deduce their effect on the Spatial Pooler's performance.

B. *Creating input file*

All input files are saved in comma-separated value (CSV) file format.

The following periodic function is used to generate original input data set:

$$f(x) = 10 \cdot \cos(0.01\pi \cdot x) \cdot \cos(0.05\pi \cdot x) \qquad (1)$$

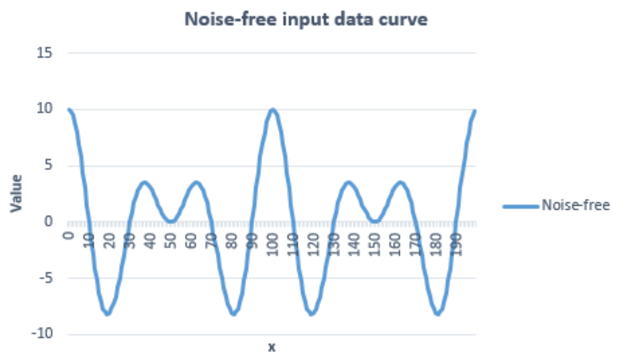

Figure 1. The 200 samples from the original input data set

A Microsoft (MS) Excel® file is created with data generated from this function (the data are sampled from this function at integral values of x). There are 200 samples created. The sampled values are then rounded to one decimal place. This file is used as the original input data set.

Table 1 shows all the possible values from the noise-free input data set and the value next to them in decreasing order. The "specificity" is investigated from the trained Spatial Pooler's outputs with respect to these input data pairs (Figure 9).

Table 1. Consecutively-decremental-input-values pairs regarding amplitude from noise-free input set

Pair number	First value	Second Value	Difference
1	10	9.9	0.1
2	9.9	9.5	0.4
3	9.5	8.9	0.6
⋮	⋮	⋮	⋮
44	-8	-8.1	0.1
44	-8.1	-8.2	0.1

(Only some values are presented here, the full table could be reproduced from Equation (1))

To generate the noisy input file from the original data file, MS Excel® statistic functions are used:

NORM.INV(RAND(), m, s)

This function generates a Gaussian random variable with mean *m* and standard deviation *s* ($\mathcal{N}(m, s)$). For each value of standard deviation *s*, this random variable is then added to every sample of the original input data set and rounded to one decimal place. The resulted files are the corrupted input data sets at different levels of noise.

In this paper, the noise level is defined to be the ratio between standard deviation s and input resolution.

The reason why it is necessary to define the noise level this way is that this value can capture the relative amount of value fluctuation in the noisy input data. When comparing two system with different input solution, small noise will have more obvious effect on the one with finer resolution.

Figure 2 and Figure 3 show deviations between original input data set and noisy input data sets with additive Gaussian noise variable, whose mean is 0 and standard deviation is 0.2, respectively, 2. Different standard deviations *s* result in different levels of distortion.

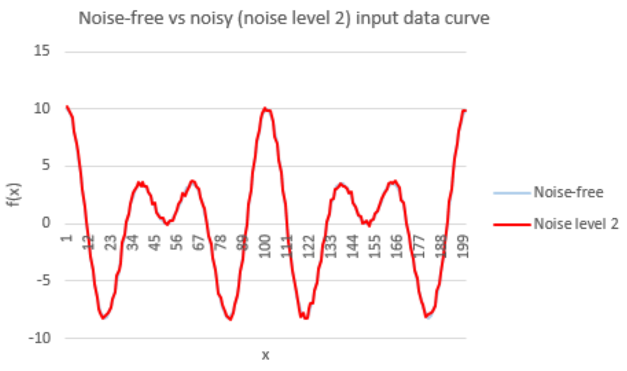

Figure 2.Comparison between original and noisy (noise level 2) input data set

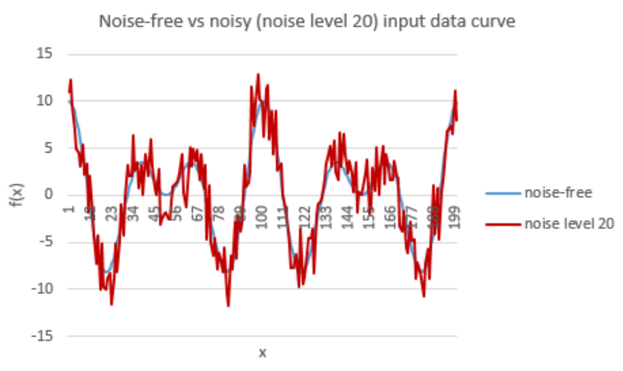

Figure 3. Comparison between original and noisy (noise level 20) input data set

C. *Scalar Encoder's Parameter*

Table 2 Scalar Encoder's parameters

Parameter	Value
W	65
N	465
MinVal	-20.0
MaxVal	20.0
Periodic	false
ClipInput	true
Offset	108

(All other parameters are set to default values as in [9])

According to data in Table 2 and the Scalar Encoder's implementation in [9], it could be calculated that the *resolution* of the Scalar Encoder in this paper is 0.1.

D. *Spatial Pooler's Parameter*

Table 3 Spatial Pooler's parameters

Parameter	Value
inputDimensions	465
comlumnsDimension	2048
potentialRadius	-1
potentialPct	1
globalInhibition	true
numActiveColumnsPerInhArea	0.02*2048 (2%)
stimulusThreshold	0.5
synPermInactiveDec	0.008
synPermActiveInc	0.01
synPermConnected	0.1
dutyCyclePeriod	100
maxBoost	10

(All other parameters are set to default values as in [9])

E. *Comparison method*

A measure of similarity, which is somewhat similar to "Hamming distance", is utilized in this experiment. This measuring function is already implemented in "NeoCortexApi.Akka" solution in [9]:

```
public static double GetHammingDistance(int[] originArray, int[] comparingArray, bool countNoneZerosOnly = false)
```

Code snippet 1. Function to calculate "Hamming distance" from [9]

Specifically, this function takes two arrays as inputs and returns percentage of identical elements between these two arrays as output. In this paper, this percentage measure is named "Hamming distance" (however, it should not be confused with the mathematical Hamming distance). In our experiments, the arrays to be compared are bit arrays and only active bits are taken into account (`countNoneZerosOnly = true`).

Outputs of a trained Spatial Pooler (with respect to noise-free and noisy input data set) are compared using this function. To be specific, for each pair of (original and noisy) input data sets, the "Hamming distances" between Spatial Pooler's outputs at respective samples are measured. These values are then summed up and averaged across the whole samples range. The result is the averaged "Hamming distance" across all samples, which is called "similarity" in this paper.

Additionally, this function is also used to measure the "specificity" of the trained Spatial Pooler. To be more specific, the "Hamming distance" between outputs of trained Spatial Pooler with respect to every two consecutively incremental input values from the original data set are measured.

III. RESULT AND DISCUSSION

A. *Spatial Pooler's robustness and specificity against additive Gaussian noise*

1) *Experimental Data*

Following the procedure in the "Methods" section with noise-level-2 input data set, the following result is obtained. Figure 4 shows "Hamming distance" between original and corrupted output data set at each sample. Each data point is the "Hamming distance" between outputs of the trained Spatial Pooler in two cases (when the Spatial Pooler takes the noise-free input sample as input and when the Spatial Pooler takes the corresponding noise-level-2 input sample as input).

Similarly, Figure 5 shows the result for noise-level-20 input data set.

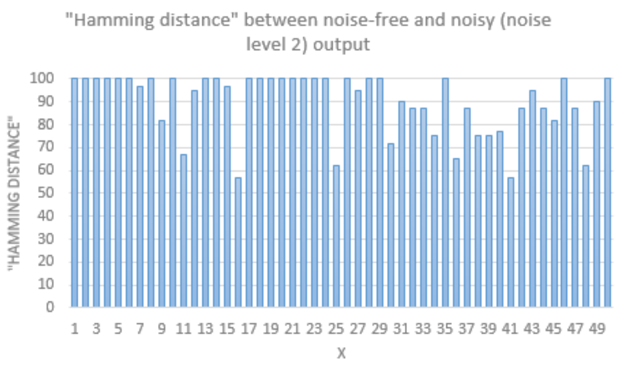

Figure 4. "Hamming distance" between noise-free and noisy (noise level 2) Spatial Pooler's output (for clarity, only data for the first 50 samples are shown)

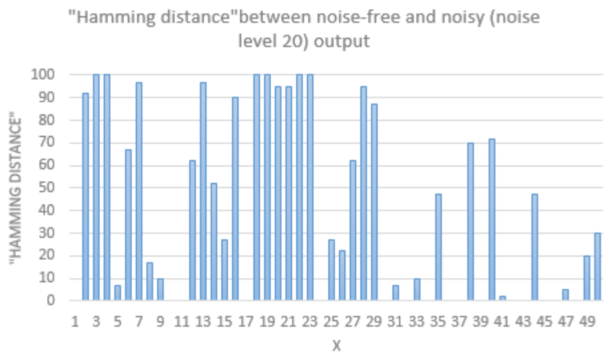

Figure 5. "Hamming distance" between noise-free and noisy (noise level 20) Spatial Pooler's output (for clarity, only data for the first 50 samples are shown)

Figure 6 and Figure 7 are the experimental data, which show the average similarity in outputs of the Spatial Pooler with respect to noise-free input and various levels of noisy inputs.

For example, in Figure 6, looking at the data point where "level of input distortion (standard deviation s) / input resolution" (noise level) is equal 2, the value is approximately 90%. This value is just the average of the "Hamming distances" from all data point in Figure 4 (actually there are 200 data points, however, for clarity, only 50 data points are shown in Figure 4).

Simply speaking, the "similarity" value in Figure 6 and Figure 7 is a way to measure how similar the outputs of the trained Spatial Pooler with respect to the noise-free and the corresponding noisy (noise level is shown on horizontal axis) input are.

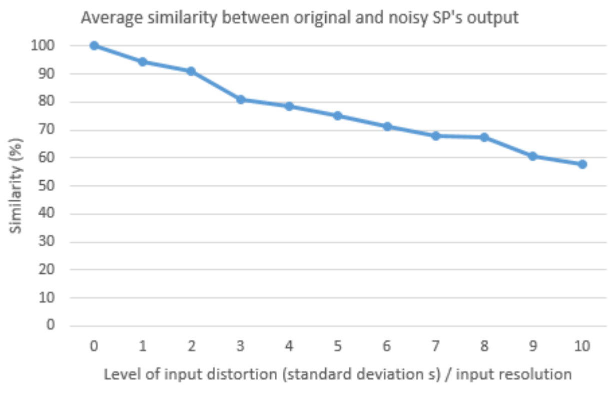

Figure 6. Average similarity between noise-free and noisy output of Spatial Pooler with noise level from 1 to 10 (noise level from 1 to 10)

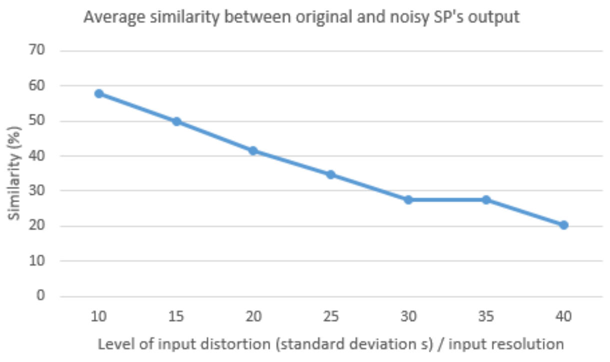

Figure 7. Average similarity between noise-free and noisy output of Spatial Pooler with noise level from 10 to 40 (noise level from 10 to 40)

Figure 8 presents the "Hamming distance" between outputs of trained Spatial Pooler with respect to two consecutively incremental input values regarding amplitude from the training data set. For clarity, only difference between consecutively-incremental-input-values pairs are used to label the data point. For the purpose of this report, the information about the values of the two corresponding inputs is not necessary and is omitted.

For example, looking at the first data point in Figure 8, it is shown that the outputs of the trained Spatial Pooler are exactly the same when it takes two input values, between whom the difference is 0.1, as input. It means that although the two inputs are semantically different, the Spatial Pooler still recognized them as only one meaningful input. Looking at the second data point, the input difference is also 0.1. However, for this pair of inputs, the Spatial Pooler could still recognize these two inputs as different ones.

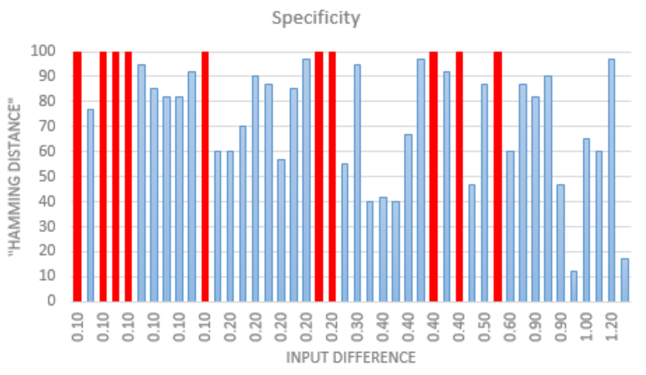

Figure 8. "Hamming distance" between outputs of trained Spatial Pooler with respect to two consecutively incremental input values regarding amplitude from the training data set. Columns in red are data points with "Hamming distance" equal 100%. (Here only input difference is used to label data, refer to Table 1. To find out exactly which input value pair a certain data point belongs to, recreate the experiment with the described settings and procedure in this paper.)

2) *Discussion*

As can be seen from Figure 6, the "similarity" between original and noisy output of the Spatial Pooler is higher than 90% when the noise level is less than or equal to 2. Following this paper's criteria, this means that the Spatial Pooler, with the mentioned settings, are robust against additive Gaussian noisy inputs with standard deviation equal or less than two times the input resolution. However, from visual inspection of Figure 2, the amount of noise, when the noise level is equal to two, is insignificant. This means, the Spatial Pooler, with its common settings, is only robust against relatively low level of noise.

In Figure 7, the experiment cannot be carried out for higher levels of input distortion because some samples of the noisy input sets may lie outside the range of the Scalar Encoder and be clipped to minimum or maximum value. This may make the experiment's result less meaningful.

From Figure 8, it could be observed that, for some consecutively-incremental-input-value pairs from the original input data set, their outputs from the trained Spatial Pooler could be exactly the same. This means that, at these input value pairs, the Spatial Pooler is unable to differentiate them. Most of these confusion occur when the difference between input values is 0.1. However, it is possible that even some pairs with greater difference could also be subjected to these confusion. In general, about 75% input value pairs could be distinguished by the trained Spatial Pooler to some extent.

Depending on applications, we could accordingly modify the parameters to adjust the robustness of the Spatial Pooler. However, it should be noted that there is a trade-off between robustness and specificity. The more robust the Spatial Pooler is, the less specificity it may have. This trade-off between robustness and specificity could have a great impact on the performance of the Temporal Memory and, consequentially, on the whole HTM basic module as well. These considerations should be taken care of when designing an HTM system.

3) Additional test

This time, to test every decimal deviation in between integer numbers within a range, two linear data sets was used. The training set is for the Spatial Pooler learning process that comprises of only integer numbers from -20 to 20 with the step of 1. The testing set comprises of first order decimal numbers with the same range from -20 to 20 but with the step of 0.1. The decimal numbers are treated as noises (similar to the additive Gaussian noise) in this case and were not put in the learning process.

The training set is encoded and then learned for 200 times to create SPD. After that, SP must use the training set which includes the decimal numbers to predict the SPD patterns but without prior knowledge of the decimal numbers. The similarities of SPD patterns are, again, calculated using "Hamming distances" that takes into account of only non-zero bits between each 0.1 decimal step and the integer number. For example, the hamming distance from 6.9 and 7.1 to 7 or 6.3 and 7.7 to 7. This is done for every numbers and averaged out. This was the result:

Avg 0.1 step: 87.46052631578948

Avg 0.2 step: 78.14473684210526

Avg 0.3 step: 70.8157894736842

Avg 0.4 step: 59.73684210526316

Avg 0.5 step: 36.276315789473685

Avg 0.6 step: 14.473684210526315

Avg 0.7 step: 8.5

Avg 0.8 step: 8.25

Avg 0.9 step: 7.723684210526316

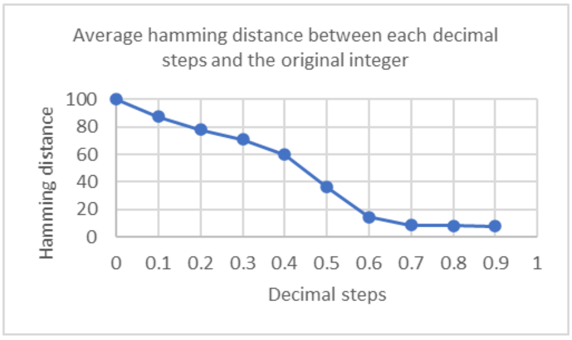

Figure 10. Average "Hamming distance" between each 0.1 decimal step and the integer number.

Even though the data have not been put into temporal memory, from the gathered result, the patterns are most similar up to 0.3 step and from 0.5 up, the hamming distance decreases rapidly. With that said, if noise changes the input value for more than 30% from the original data, the SP may create patterns that can be very different from the original data. But even if the similarity between 7.9 and 7.0 is only 7.72%, 7.9 is more similar to 8.0 (87.46%) which is the next data point. Meaning even with no prior knowledge, SP can predict a number that falls between two known data points such as 7.9 is similar but still different to 8.0 or 7.0 and not similar to 20.0 or 100.0 or something else.

IV. CONCLUSION

From the experimental data and the following discussion, it could be concluded that, under the effect of additive Gaussian noise, the output of the Spatial Pooler (with common settings) with respect to noisy input (whose noise-variable's-standard-deviation-to-input-resolution ratio is less than 2) could attain up to 90 percent similarity compared to that of original output. Regarding specificity, about 75% of two neighboring input values (regarding amplitude) could be differentiated by the Spatial Pooler.

The second test can still reflect the result above. With the 0.1 decimal step from the original data (consecutively-incremental-input-value pairs) results in average of 87% similarity (specificity) and shows that an unknown data can be predicted to be similar but still different to its nearby known data points.

The above result could serve as some reference while developing simple HTM system. However, more researches should be conducted to investigate more thoroughly the relation between noise robustness and specificity of HTM Spatial Pooler as well as its effect on the performance of the whole HTM system. Such works would certainly help to construct more reliable HTM systems with better performance.

References

[1] J. Hawkins and S. Blakeslee, *On intelligence*. New York: Times Books/Henry Holt, 2008.

[2] J. Hawkins and C. Maver, "Introduction Chapter," in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[3] J. Hawkins and C. Maver, "HTM Overview Chapter," in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[4] S. Purdy, "Encoding Data for HTM Systems," in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[5] S. Ahmad, M. Taylor, and Y. Cui, “Spatial Pooling Algorithm Details,” in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[6] A. Lavin, S. Ahmad, and J. Hawkins, “Sparse Distributed Representations,” in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[7] S. Ahmad and M. Lewis, “Temporal Memory Algorithm Details,” in *Biological and Machine Intelligence*, Release 0.4., Numenta, 2016.

[8] D. Dobric, “Influence of input sparsity to Hierarchical Temporal Memory Spatial Pooler noise robustness,” 2019.

[9] D. Dobric, “NeoCortexApi,” 2019. [Online]. Available: https://github.com/ddobric/neocortexapi/.