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

Investigation of Hierarchical Temporal Memory Spatial Pooler's Noise Robustness against Gaussian noise

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 level of noise robustness is measured by the similarity between outputs of the Spatial Pooler when it takes the original data set and then the additive Gaussian noisy data sets as inputs, provided that it is only trained with the original data set.**

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 first 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 are then compared and investigated.

The main goal is to investigate the noise robustness of the Spatial Pooler within a practical scenario. The results in this paper is intended to serve as an auxiliary reference data when designing robust HTM systems.

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

II. Main Idea

The purpose of this paper is to provide the readers with quantitative result on the effect of Gaussian noise on Spatial Pooler's output (the Spatial Pooler is assumed to have the settings as specified in this paper).

Roughly speaking, the Spatial Pooler is trained with a noise-free data set. After that, it is used to infer a corrupted version of this data set. The outputs between these two cases are compared to observe the capability of noise robustness of the Spatial Pooler.

III. 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.

The function of the Scalar Encoder is auxiliary: to encode scalar values into proper bit arrays. The function of Spatial Pooler is the focus of this paper: noise robustness. When the Spatial Pooler is said to "be trained" or to "be used to infer a data set", it means that the data sets are first encoded by the Scalar Encoder (with the specified settings in this paper) and then fed to the Spatial Pooler.

The experiment includes 2 phases:

- Training: The Spatial Pooler is trained with a simple noise-free data set.
- Testing: The trained Spatial Pooler is used to infer noisy versions of the noise-free data set.

The outputs of the Spatial Pooler from these two phases are then compared. From that, the influence of Gaussian noise on the Spatial Pooler output could be quantitatively observed.

B. *Scalar Encoder's Parameter*

Table 1. Scalar Encoder's Settings

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.

C. *Spatial Pooler's Parameter*

Table 2. Spatial Pooler's parameters

Parameter	Value
inputDimensions	445
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])

The number of learning iteration is 20.

D. *Generating noise-free data set*

The following periodic function is used to generate noise-free input data set:

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

The above function will be sampled at integer-values of *x* from 0 to 199 to generate 200 samples. These values are saved in CSV format and is used as input value.

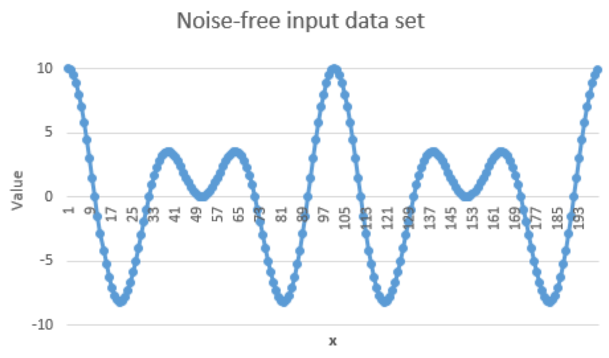


Figure 1. Noise-free data set

E. *Generating noisy data sets*

1) *Noise type*

Gaussian noise is utilized in this paper. The random variable, which is used to create noise in the experiment, follows Gaussian distribution.

Consequently, the random variable can be characterized by *mean* and *standard deviation. Mean* is the value that the random variable would most likely obtain. *Standard deviation* specifies the range around *mean*, within which the random variable would obtain the value with probability 68%. Roughly speaking, *standard deviation* shows how much the random variable fluctuates around the *mean* value.

For example, if a random variable follows Gaussian distribution with *mean* 0 and *standard deviation* 1, this random variable would most likely obtain value 0 and it would obtain values in the interval from -1 to 1 for 68% of the times.

2) *Generating noisy data sets from noise-free data set*

To generate the above mentioned type of random variable, the following function from Microsoft (MS) Excel® is used:

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

This function helps to generate a random variable with *mean m* and *standard deviation s*. This random variable is added to every sample in the noise-free data set. The resulting data set is the noisy version of the noise-free data set.

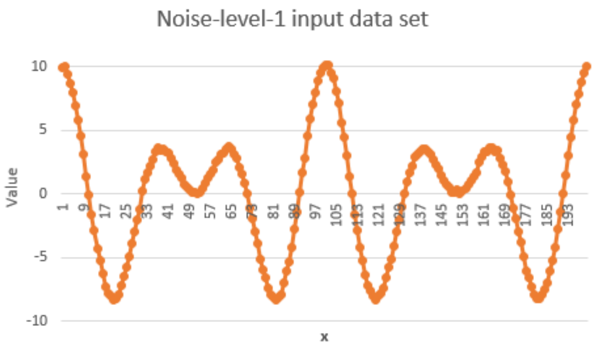
Different values of *s* are used to create different level of noisy data set.

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

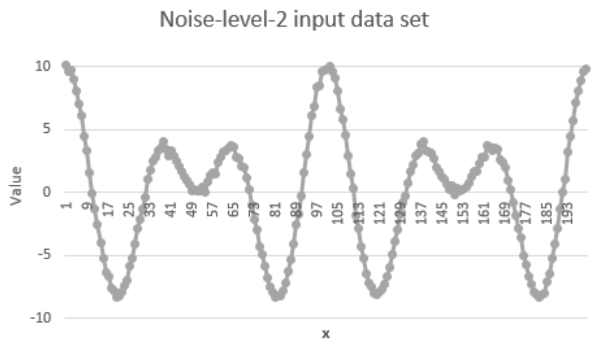
The reason why it is necessary to define the *noise level* this way is that this value can capture the relative amount of value fluctuation in the noisy input data. Systems with finer solution tend to be more susceptible to small noise.

In this paper's experiment, there are in total 10 noisy input data sets. Their *noise level* vary from 1 to 10.

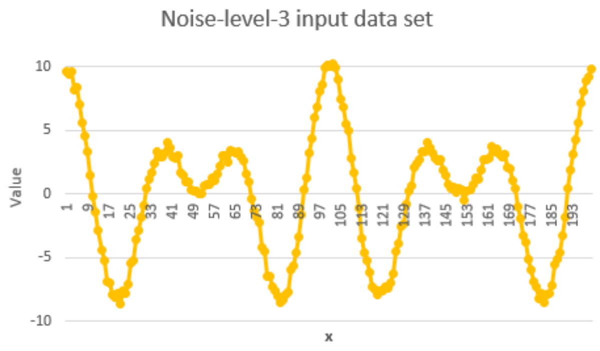
Figure 2 shows some noisy input data sets with additive Gaussian noise variable, whose mean is 0 and standard deviation is 0.1, 0.2 and 0.3 respectively. Different standard deviations *s* result in different levels of noise. The *noise level* of the data set in Figure 2 is 1, 2 and 3.



a)



b)



c)

Figure 2. Noisy input data sets. Only 3 out of 10 utilized noisy sets are shown. All 10 input data sets are available at [10].

F. Comparison

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)
```

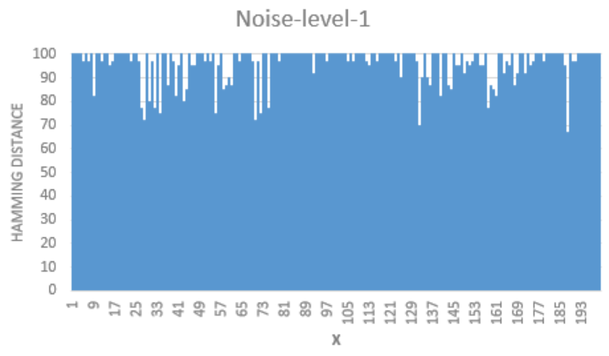
Code snippet 1. Function to calculate "Hamming distance" from [9]

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`).

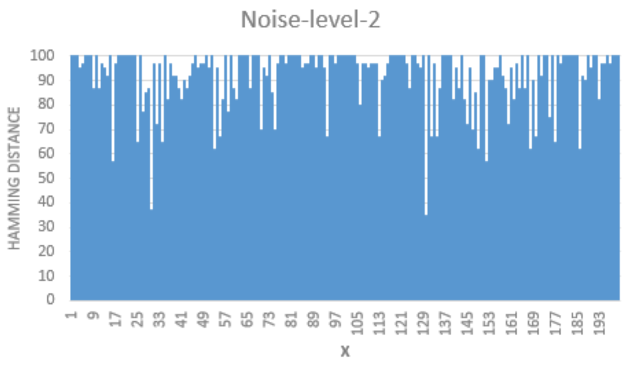
IV. Result and Discussion

A. Comparison between noise-free and noisy Spatial Pooler's outputs

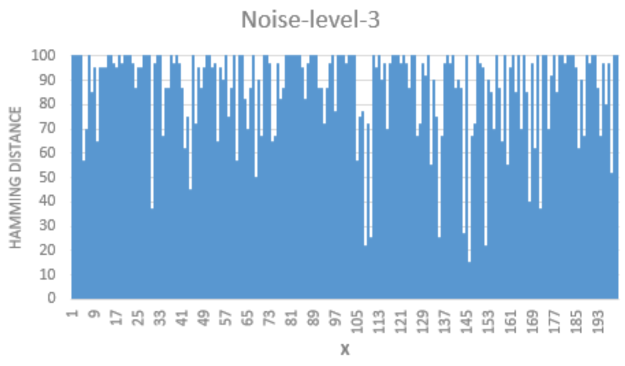
Figure 3 shows the "Hamming distance" between Spatial Pooler's outputs for noise-free input (depicted in Figure 1) and different level of noisy input (depicted in Figure 2).



a)



b)



c)

Figure 3. Comparison between Spatial Pooler's outputs from noise-free and some levels of noisy input data sets. Only data for 3 out of 10 utilized noisy sets are shown. Data for all 10 input data sets are available at [10]. ***("x" in these figures is the ordinal number of the input pair, whose outputs are being compared at this data point. For example, in Figure 3a, at data point "x" = 1, the value of this data point is the "Hamming distance" between Spatial Pooler's outputs from 2 input***

values, one of which is the value at "x" = 1 from the noise-free input data set (Figure 1) and one of which is the value at "x" = 1 from the noise-level-1 input data set.)

As expected, the similarity in Spatial Pooler's outputs decreases as the level of noise increases. To quantitatively summarize the data, the "Hamming distance" for each noise-level are averaged over the range of "x". The result is shown in Figure 4 for *noise level* in the range from 1 to 10.

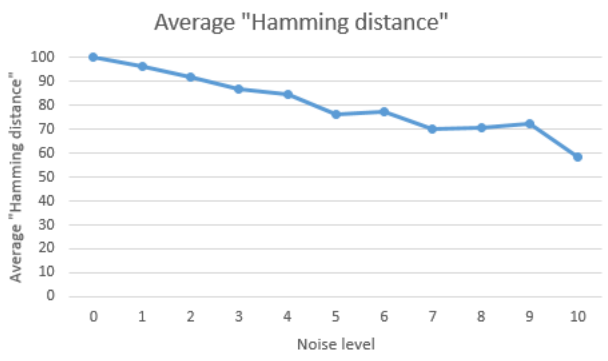


Figure 4. Averaged "Hamming distance" over the whole range of "x" for different level of noise when the outputs of the Spatial Pooler from noise-free and noisy inputs are compared

From Figure 4, it could be observed that the similarity between Spatial Pooler's noise-free and noisy outputs could attain at least 80% when the *noise level* is 4 or lower.

This means that the Spatial Pooler could guarantee satisfied noise robustness against Gaussian noise, whose *standard deviation* is 4 times as high as the resolution of the HTM system's Scalar Encoder.

For *noise level* higher than 4, the Spatial Pooler can no longer keep a similar output as noise-free one. This is understandable, because higher *noise level* means more drastic fluctuation, which may lead to the problem that some input values may fluctuate too much and takes on the value of another meaningful input (input from noise-free data set, which is used to train the Spatial Pooler). This make the Spatial Pooler unable to recognize whether such input is a corrupted value or not.

In general, the Spatial Pooler with the specified common settings is practically robust against moderate level of Gaussian noise.

B. 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

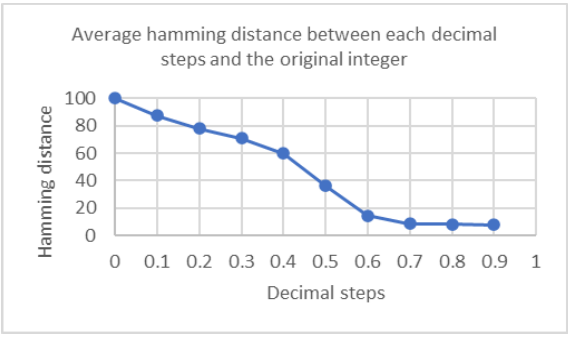


Figure 10. Average "Hamming distance" between each 0.1 decimal step and the integer number.

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.

V. 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 4) could attain at least to 80 percent similarity compared to that of original output.

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.

In conclusion, the Spatial Pooler with the specified common settings is satisfactorily robust against Gaussian noise and, meanwhile, still be able to distinguish meaningful inputs from each other.

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/.

[10] S. Nguyen, D. Nguyen, "se-cloud-2019-2020" 2020. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-cloud-2019-2020/tree/VNGroup/Source/MyProject