*Improve the Scalar Encoder of NeocortexApi*

Bishal Gautam  
bishal.gautam@stud.fra-uas.de

*Abstract*—Neocortexapi is library implemented on .Net framework that tries to mimic what is happening in the neocortex region of brain. It is an implementation of Hierarchical Temporal Memory (HTM). HTM is the biologically inspired machine intelligence technology which tries to mimic the neocortex region of brain in terms of architecture and processing. In the neocortex, data is inputted in the form of Sparse Distributed Representations (SDRs). The SDRs are generated in the form of active pulse through millions of axons from sensory organs and from other cortical regions. Encoder is a part of neocortexapi that acts as sensory organs. Just like sensory organs which converts sensations to pulses and send that pulse to the neocortex region of brain for processing, likewise encoder encodes any input to the form of SDRs, which are then processed by the neocortexapi. In this paper, one of the encoders, Scalar Encoder’s will be described and improved. Scalar Encoder is responsible for encoding numeric or floating-point values in a series of 0’s and contiguous block of 1’s. Current version of Scalar Encoder is tested and improved by choosing appropriate value for parameters and by improving the interface and current logic.

Keywords—sparse distributed representations, scalar encoder, neocortex, NeocortexApi, .NET6, APIs, HTM

# Introduction

Massive amount of sensory information is processed continuously in the neocortex region. Neocortex never stops processing the information and still work flawlessly. All the information is processed as a series of active bits passed down from the sensory organs to the neurons through the synaptic connections. This information is represented in neuron in the form of active pulse for a time instant. The brain, especially neocortex, can hold huge amount of information from life to death. This is the possible because of sparse representation of neurons for a particular input. In sparse representation, large array of bits is inactive(0’s) and a very few are active(1’s). Each bit carries some meaning. Two SDRs having more than a few overlaps would represent that two SDRs are semantically similar. Encoder in neocortexapi corresponds to sensory organs. Encoder encodes any input in the form of spare distributed representations.

Scalar Encoder is a type of encoding technique in HTM system that encodes scalar values such as numeric or floating-point value in the series of 0’s and 1’s. In this type of encoder, output has 0’s with adjacent block of 1’s. The goal of this paper is to ameliorate the encoding of scalar values. Enhancement is done on preexisting scalar encoder by setting appropriate parameters and by adding various functionalities.

# Materials and methods

## Encoder Overview

First step of using neocortexapi is to transform the data to a form (SDRs) so that it can be digested for further processing (i.e. by Spatial Pooler). Encoder is responsible for converting the data to a SDRs representation. There are various types of encoders such as Geospatial encoder, Image encoder, Scalar Encoder, DateTime Encoder, etc. Geospatial encoder takes geometric coordinates values and convert them to SDRs. Two objects can be said to be close to each other geographically if their SDRs overlap over some threshold. Similarly in Scalar Encoder, two values SDRs would be similar if they are equal or close to each other.



Fig. 1. Scalar Encoding of scalar values with total number of bits 12 and 3 number of active bits

Two encodings are said to be similar if they have more overlapping active bits. As shown in figure 1, we can see that the encoding of 1 and 2 are nearly similar to each other. Likewise, the encoding of 6 and 7 are nearly similar to each other. But the encoding of 1 and 7 is totally different than the encoding of 1 and 2. This is due to the fact that 1 and 2 has less difference than 1 and 7 and hence 1 and 2 are more similar both semantically and with SDRs overlap.

There are some important aspects that need to be considered when encoding data. They are described as [1]:

* For data which are semantically similar, encoding should result in SDRs with overlapping active bits
* Encoding should be deterministic, meaning that same input should always produce the same SDR as output.
* The output should have the same dimensionality (total number of bits) for all inputs.
* The output of encoding should have similar sparsity for all input. Output should have enough one-bits to handle noise and subsampling.

## Parameters and Terms for Scalar Encoder

Before encoding scalar values, scalar encoder is initialized with parameter dictionary. The parameters which the encoder depends on can be explained as:

* N: Total number of bits to represent the input data
* W: Number of active bits (1’s) to represent the input data. W should be odd to avoid centering problem.
* MinVal: Minimum Value that the SDR can represent without clipping
* MaxVal: Maximum Value that the SDR can represent without clipping. For periodic input, input data to encode should be strictly less than MaxVal
* Periodic: Boolean Value, if true, representation is repeated after certain interval
* ClipInput: If true, value less than the minimum has SDR similar to minimum value and values more than the maximum has SDR similar to maximum value for periodic data
* Resolution: For a particular value of resolution, input separated by more than or equal to resolution would have different representations. For e.g., if resolution = 0.5, then 4 and 5 will have different encoding but 4.1 and 4.4 could have similar encoding.
* Radius: For a particular value of radius, input separated by the radius would have completely different representations with non-overlapping active bits.
* Name: It is an optional string that will become a part of description

N, Resolution and Radius are three mutually exclusive parameters. Exactly one of N, Radius or Resolution should be given for the input space, rest of the two should be “0” or not set on the input space. Rest of the two parameters will be calculated internally based on the parameters supplied to the scalar encoder. Total Buckets represent the total number of spaces that can represent distinct SDRs. For example, for a particular setting if the bucket size is 14 then the encoder can encode 14 distinct SDRs. The equation to calculate the Bucket size for an encoder is:

Total Buckets = *(N-W+1)* (1)

# Test Cases with results

Current version of Scalar encoder has some defects, in this part we will be discussion the pitfalls of currently implemented scalar encoder.

After finding the pitfalls, we will be experimenting to find the correct solution with a way to improve current version of scalar encoder.

## Test Case generating the bucket size problem

In the first experiment, we are tried to regenerate the error of bucket size problem with the predefined parameters. For the experiment, we have:

1. N = 10
2. W = 3
3. MinVal = 0
4. MaxVal = 9
5. Periodic = False
6. ClipInput = True

The output with the above defined parameter is shown as :

Text

Description automatically generated with medium confidence

Fig.2. Output of Scalar Encoder with similar SDRs

Since periodic is true, the total number of buckets that our configuration can represent is = *(N-W+1)* = (10-3+1) = 8 for a Resolution of 1. We need to represent data from 0 to 9, which in total should be 10 different representations. Because the number of available bucket (8) is less than the required bucket (10), this causes an overlap in SDR of input space. In the above case, overlap occurs for the input of 2 and 3.

## Test Case generating the bucket size problem – Second

In the test case A, the overlapping was clearly due to the low number of bucket size. So, for our second experiment, we increased the value of N to 11, we have:

1. N = 11
2. W = 3
3. MinVal = 0
4. MaxVal = 9
5. Periodic = False
6. ClipInput = True

The output with the above defined parameter is shown as:

A picture containing text

Description automatically generated

Fig. 3. Output of Scalar Encoder for the given parameter with similar SDRs

The total number of buckets that our configuration can represent is = *(N – W + 1)* = (11-3+1) = 9. Since we need to represent data from 0 to 9, which is in total 10. The number of available bucket (9) is less that the required bucket (10). This cause in an overlap in SDR of input space. In our case, 4 and 5 input overlapped.

## Test Case for the solution of the given predefined parameter

We experimented by increasing the value of N again so as to increase the number of bucket size. The parameters value is:

1. N = 12
2. W = 3
3. MinVal = 0
4. MaxVal = 9
5. Periodic = False
6. ClipInput = True

The output with the above defined parameter is shown as:

Text

Description automatically generated

Fig. 4. Output of the Scalar Encoder after increasing N for appropriate bucket size

The total number of buckets that our configuration can represent is = *(N - W + 1)* = (12 – 3 + 1) = 10. Since we need to represent data from 0 to 9, which is 10 different SDRs representations. The number of available buckets is equal to the number of required buckets. This causes a discrete SDR representation of different input.

## Test Case for the Solution of Non-Distinct Encodings

When the value of N is used as input to set encoder settings (Radius and Resolution values are zero or not set), we found out that some values of N were too low that resulted in similar SDRs for distinct inputs. In order to avoid the problem, argument exception check had been implemented. This makes sure that either inputs would have distinct encoding or argument exception is thrown in case the present setting of encoder could result a non-distinct encoding for the given input space.

*requiredN* = *(int)Math.Ceiling(W + MaxVal – MinVal)* (2.a.)

*requiredN* = *(int)Math.Ceiling(MaxVal – MinVal)* (2.b.)

If the encoder settings are set for periodic input, i.e., Periodic is True in encoder setting then equation 2.b. is used as comparison for required value of N. If the encoding should be done for non-periodic inputs, i.e., Periodic is False in encoder setting then equation 2.a. is used as comparison with the present value of N. If present value of N is less than requiredN then argument exception is thrown.

When the value of Resolution is used as input to set encoder settings (i.e., Radius and N is zero), we found out that some values of Resolution were too high that resulted in similar SDRs for different inputs. In order to avoid the problem of similar encoding for different input, proper argument exception check had been implemented. This makes sure that when the value of Resolution is more than required (i.e., 1), argument exception would be thrown. If the value of Resolution is proper, distinct encoding is outputted for the given input space.

When the value of Radius is used as input to set encoder settings (i.e., Resolution and N is zero), we found out that some values of Radius were too high that resulted in similar SDRs for different inputs. Argument exception check had been implemented for Radius so that either distinct encoding is output for the given input space or argument exception is thrown if the inputted settings could result in non-distinct encoding for the given input space.

*Resolution = Radius / W* (3)

Equation 3 is used to calculate the value of Resolution for a given Radius. Then check is implemented with the calculated value of Resolution. If Resolution is less than or equal to 1, distinct encoding is done by the Encoder, else argument exception is thrown.

## Test Case generating the OverflowException

Instead of relying on N for encoding, this time we choose different values of Resolution for Encoding.

1. Resolution = 1 to 10
2. W = 3
3. MinVal = 0
4. MaxVal = 9
5. Periodic = False
6. ClipInput = True

While using the above parameters for encoding, we observed System.OverflowException. This is shown in the figure 5.a.

Text

Description automatically generated with medium confidence

Fig. 5.a. Output showing Exception for a given set of parameters

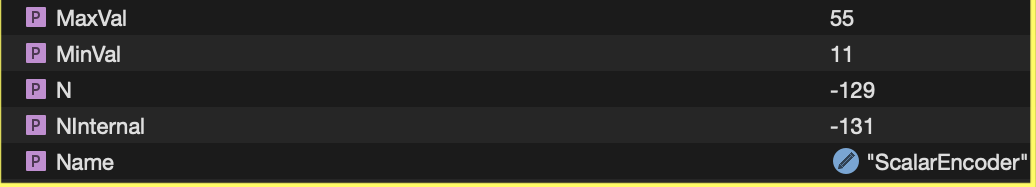


Fig 5.b. Values of N when the Exception occurred

After observing the values of local variable using breakpoints, we found out that the value of N to be negative as shown in figure 5.b. This should be the reason for the overflow exception.

## Test Case generating IndexOutOfRangeException

For this experiment, we chose Radius instead of N or Resolution for encoding.

Clearly as viewed on the figure 6, at certain value of Radius, IndexOutOfRangeException occurred. Even so, we can view the encodings at different input space.

Text

Description automatically generated

Fig. 6. IndexOutOfRangeException for some input space

## Test Case explaining the OverFlowException

As a comment on test case E, we worked with various values of Resolution (without setting N or Radius). After experimenting and through debugging we found that System.OverFlowException occurred because the default Radius configuration was negative (-1), which was set inside the EncoderBase.cs file. This resulted in negative value of N. Since the output array was initialized using the equation 4 as:

Output = *int[N]* (4)

This resulted in OverFlowException.

Setting the default value of Radius to 0, which was previously -1 inside the Initialize method of EncoderBase.cs file, N was no longer negative and OverFlowException was solved. After solving this issue, IndexOutOfRangeException was observed in some input space.

## Test Case explaining IndexOutOfRangeException

In the test case H, we solved the problem of OverFlow Exception. After that we need to solve the problem of IndexOutOfRangeException. By experimenting with various inputs, we find out that the exception was due to the result of minbin or maxbin value, which denotes the starting position and ending position of the active bits.

We first find the centerbin and on adding padding to the left and right of the centerbin we get the values of minbin and maxbin. minbin represent the index position to start the encoding and maxbin represent the index position of the end of encoding for a value of N. The values of parameters used for encoding for this test case is:

1. Resolution = 1 to 10
2. W = 3
3. MinVal = 13
4. MaxVal = 29
5. Periodic = False
6. ClipInput = True

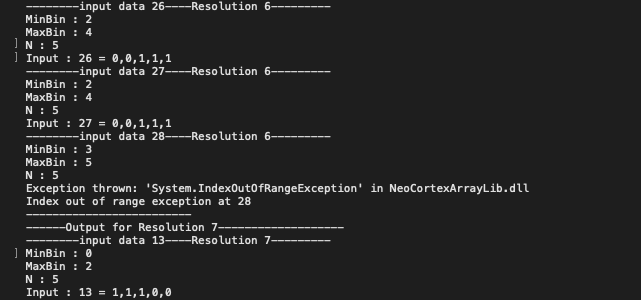


Fig. 7. Experiment showing IndexOutOfRangeException

Clearly in figure 7, in case of exception, minbin is 3 and maxbin equals 5. The value of N is 5 for this experiment. Minbin and maxbin represent the index position and index position starts at 0. This means that our encoder tries to set the data from N[3] to N[5]. The value of N is 5, because of this we can only go up to N[4] as indexing starts at 0 in c-sharp and most other programming language. This resulted in IndexOutofRangeException.

## Test Case Solving IndexOutOfRangeException

Experimenting with different values and by using breakpoints, we found out that there might be issue in the value of center bin or N. Center bin value after experimenting seems to be fine, logic seemed okay. So, we then looked for the value of N. Because of low value of N, the error seems to persist in some input space. To solve this, we changed the value of N in line 157 of ScalarEncoder.cs.

*N = (int)(nFloat)* (5)

*N = (int)Math.Round(nFloat)* (6)

The value of N was changed from equation 5 to equation 6. This makes sure that the decimal values are rounded to nearest whole number. This increased the value of N for some input space.

For the input parameters of test case H, the problem seems to be solved.

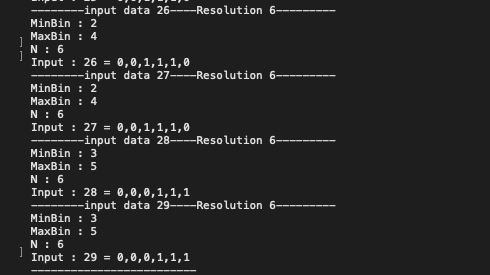


Fig. 8.a. Experiment solving IndexOutOfRangeException for the given parameter

But still the problem seems to be unsolved for some different input space. The parameters which resulted IndexOutOfRangeException were:

1. Resolution = 1 to 10
2. W = 3
3. MinVal = 0
4. MaxVal = 9
5. Periodic = False
6. ClipInput = True

Figure 8.b. shows the output of the above parameters.

Text



Fig. 8.b. IndexOutOfRangeException still persists

After working with different settings, we found out that changing the value of N in line 157 of ScalarEncoder.cs to equation 7 solved the issue of IndexOutOfRange Exception.

*N = (int)Math.Ceiling(nFloat)* (7)

Text

Description automatically generated

Fig. 8.c. IndexOutOfRangeException solved for any input space

Using Math.Ceiling increased the value of N in case of any decimal value of nFloat. This resulted in increase in the number of positions for output bits. Due to this, calculated maxbin was always less than or equal to N.

## Test Case Implementing Exception Handlers

For appropriately handling the Exception and to make sure that N, Radius and Resolution are mutually exclusive, ArgumentException check had been implemented. For a proper implementation of Exception handler, the default value of Resolution is changed to 0 inside the initialize method of EncoderBase.cs file. This facilitates proper exception when the input setting of encoder is inappropriate for encoding.

## Test Case making Scalar Encoder callable using command line argument

In improved version of the ScalarEncoder, it can take input from both the dictionary settings and command line arguments. To work with the command line arguments, arguments passed on the Main function of our program can be directly passed to the ScalarEncoder. Because of function overloading, method that processes the command line arguments will be called instead of the method that takes dictionary as input for encoding.

1. The command line arguments can be set on visual studio by going to the properties of our current solution and adding arguments.
2. Command line arguments can also be passed directly from the console and the program can be run as shown below:

Listing 1. Command line code to encode the given input

dotnet run --project path-to-project-file --n 14 --w 3 --minval 1 --maxval 8 --periodic false --clipinput true

In our case the path to the project file would be:

*neocortexapi/source/ScalarEncoderImproved/ScalarEncoderImproved.csproj*

User can create their own project and implement Scalar Encoder inside. If the user in the same folder as project file, i.e., inside their project folder where .csproj file is present, then user do not need to specify the project option while encoding using command line arguments.

A picture containing text

Description automatically generated

Fig. 9. Output of using command line for encoding

Proper exception handling is implemented if the supplied argument is not in a correct order or format (example shown in figure 9).

## Test Case Implementing the Unit Tests

Proper tests case had been implemented that makes sure that distinct encoding is done by the improved version of scalar encoder. If no distinct encoding can be done, exception is given by the program. With proper and leading exception, user can easily debug and set appropriate parameter for distinct encoding.

# Conclusion

Working with the scalar encoder, we found various exceptions and errors. We implemented proper logic and corrected the exception handling.

If the given encoder settings resulted in non-distinct encoding, then exception is thrown to the user with a helpful message. This helps user to debug the program to generate a distinct encoding. If the user wishes to get a distinct encoding, they can also update the value of parameter from the intuitive terminal by selection either yes or no. We found proper default values for the parameter so that proper check could be implemented inside the program. We implemented proper exception handling so that N, Radius and Resolution are always mutually exclusive.

Implementation of how N is calculated is changed. This makes sure that there are required number of total bits for scalar encoder for distinct encoding of given input space.

# References

1. Purdy, S., “Encoding Data for HTM Systems,” <https://arxiv.org/abs/1602.05925>, 2016.
2. Hawkings, J. & Ahmad , S., “Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex”, <https://arxiv.org/abs/1511.00083v2>, 2015