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

Implementation of Category Encoder in HTM

Muhammad Basaier
Muhammad.basaier@stud.fra-uas.de

Syed Faizan Khursheed
syed.khursheed@stud.fra-uas.de

Syed Muhammad Rah
Muhammad.syed@stud.fra-uas.de

Abstract— **HTM (Hierarchical Temporal Memory) can be defined as memory of sequences. This theory hypothesize that every excitatory neuron in the neocortex is learning transitions of patterns and that the majority of synapses on every neuron are dedicated to learning these transitions. Temporal Memory is therefore the substrate upon which all neocortical functions are built. HTM starts with the assumption that everything the neocortex does is based on memory and recall of sequences of patterns. There are two major techniques of HTM which are category and scalar encoders. Based on the technique we have implemented a category encoder which is the fundamental block of HTM. Category Encoder encodes the string into a Sparse Distributed Representations (SDR) on the basis of bits we have to distinguish elements in the list. Array of output binary bits can be used in HTM system.[1]**

Keywords— HTM, SDR, Category Encoder

I. INTRODUCTION

HTM in our machine intelligence is a learning algorithm that can store and recall spatial and temporal patterns. It is well suited for prediction, detection and classification. In this modern era there are still various tasks that a human being finds it easy to do, but the machines are unable to do at the present time. Tasks such as understanding spoken language, recognize and manipulate the objects by using navigation, graphical pattern recognition are easy for humans. . In humans, these abilities are largely performed by way of the neocortex. HTM serves the ensurement of building machines that are adjacent or exceed human level performance for many cognitive tasks. An encoder converts a value to a sparse distributed representation (SDR).

Brain uses Sparse Distributed Representations (SDRs) to process information. One of the most remarkable observations about the neocortex is that no matter where we look, the activity of neurons is sparse, meaning only a small percentage of them are spiking at a point in time. Sparse distributed representations (SDRs) models this property and are a key component of HTM theory. SDR is extremely noise resistant. If you introduce noise of high magnitude SDR will match the object which is very similar. We will realize this same attribute for human memory. Try to recall a name of a colleague from your school and try remembering the face. It is . SDR can be sub-sampled without losing much information. If your SDR has 15 ones and 1000 zeroes than remove 5 ones from this SDR, still SDR has ability to use the subsampled SDR to match with new SDRs. [2] possible that you might not remember the face clearly. Despite of it if 100 random pictures which include your colleague's image are showed, you will definitely locate your colleague with high accuracy.

Encoders
Encoders are used to represent the real world data in a form that machine can understand it. The data is represented in Binary form so that computer can understand it.
There are two types of encoder

1-) Scalar Encoder : This type of encoder is used to encode numbers.

2-) Category Encoder : This type of encoder is used to encode strings (words).
In this project we have worked on category encoder.

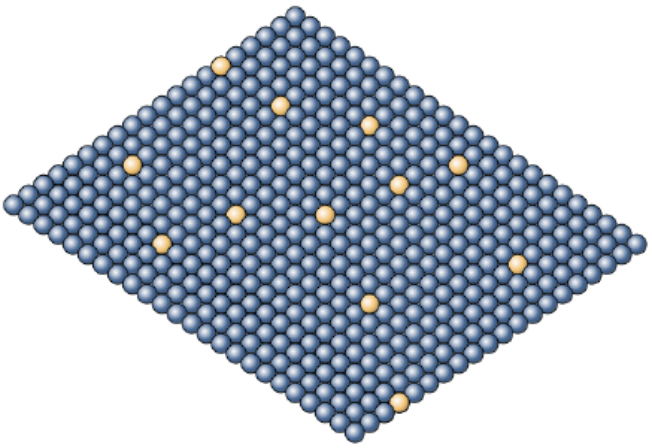


Fig.1 Sparse Distributed Representation of Data

II. METHODS

In Category encoder, the input can be pictures or words. In our case, one of our input is a category list which includes different brands of cars etc. It encodes the list of categories that are not related to each other, which means they will have completely different SDR. We will describe different parameters which we have to used in this project.

Parameters

- Width (W)

The number of bits that are on in an array (number of 1's). In this algorithm the value of width is 3. So three numbers of binary bits are 1's. Below mention code, condition is implemented if the values of width is less than 1 the code will automatically assign the value 1 else the value is same as input.

```
if (settings.TryGetValue("W", out object
width) && (int)width > 0)
     {
         this.W = (int)width;
```

```
                this.N = this.W *
category.Length;

// Number of bits in array
     }
else
     {
         this.W = 1;
     }
```

• Radius (R)

Radius in encoder reflects the number of overlapping bits. If there are no overlapping bits then the value of radius is one. In category encoder the overlapping bits is zero so the value of radius is taken as one.

this.Radius = 1;

```
 // Radius is fixed in category encoder
```

• Range

In this project, range is the number of elements in the category list. If we have four numbers of elements in the list so the value of the range is 4. Example list groceries { "Milk", "Sugar", "Bread", "Egg" }

```
this.Range = category.Length;
 // this gives the length of category list
```

• Length of array (N)

This is the total length of the array. If the value of width and range is known then we can multiply both elements to get length of array.

```
this.N = this.W * category.Length;
 // Number of bits in array
```

In Category encoder, there are two parameters i.e. width (W) and the range. For category encoder implementation is done by using the following formula:

n = Range* Width

In the project 'n' represents the total number of bits which are to be encoded, whereas width represents all the number of binary bits which are 1's, Range represents the total number of elements in the category list. As we have added the number of elements the length of the encoded array increases because it depends upon the range. In category encoder the value of radius should be fixed to 1. The overlapping bits are not required bits. Here is an example of the output of our category list.

Category List	Output
Porsche	111000000000000000000000
Audi	000111000000000000000000
Opel	000000111000000000000000
BMW	000000000111000000000000
Mercedes	000000000000111000000000
Ford	000000000000000111000000
Jeep	000000000000000000111000
MINI	000000000000000000000111

Table 1 Category list and its output

The Table above shows the output with respect to given input. When we increase the number of elements of the category list the output bits increases. Here are 8 elements in the category list so there are 24 bits in the output.

Block diagram of encoder

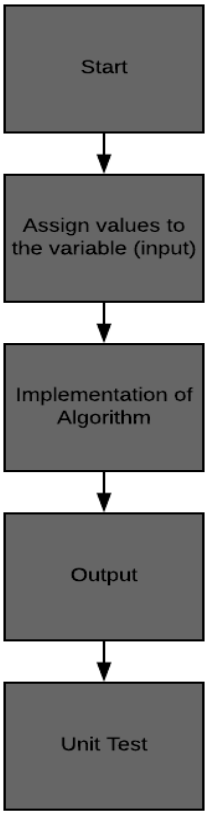


Fig.2 Block diagram of encoder

III. RESULTS

Below mention code is used for assigning the values to the variable.

```
Public      CategoryEncoder(String[]      category,
Dictionary<string,      object>      settings)      :
base(settings)
        {
            this.Radius=1;
// Radius is fixed in category encoder

this.Range = category.Length;
 // this gives the length of category list

{
       this.W = (int)width;
      this.N = this.W *category.Length;
// Number of bits in array
            }
```

```
        else
        {
            this.W = 1;
        }
```

{

encoderSettings	Count = 2
[0]	{[W, 3]}
[1]	{[Radius, 1]}

Fig.3 encoder settings

The figure shows the default settings which are implemented as width=3 and radius=1.

Reference to the C# Code

The unit test used in the project can be found under the solution "NeoCortexApi.Akka" in HTM folder. There is a section named UnitTestsProject which has all the Unit Tests we worked on for our project. The Unit Tests are highlighted in figure below. In our project we have made total 8 unit test projects.

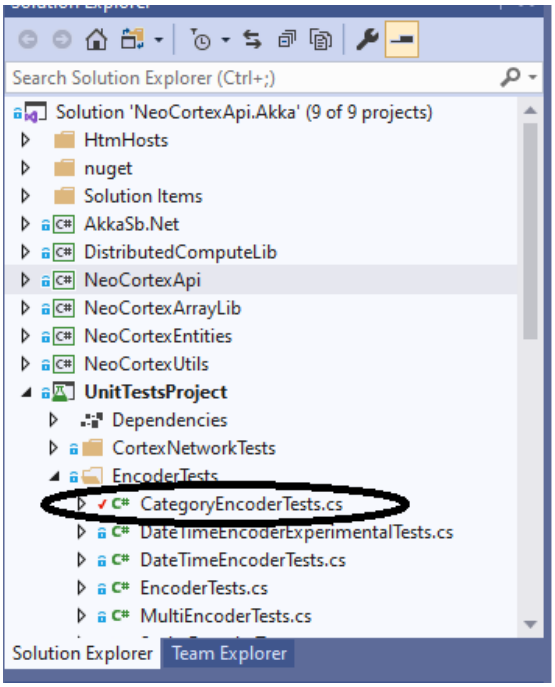


Fig. 4 CategoryEncoderTests file

We would like to give a short brief of our test cases.

```
 [TestMethod]
        public void
TestCategoryTestEncoderDefaultConstructor()
        {
 CategoryEncoder categoryEncoder = new
CategoryEncoder();
Assert.IsNotNull(categoryEncoder);
```

Here a empty constructor is used but the parameters are passed due to inheritance. The the output is not null.

```
[TestMethod]
        public void
TestCategoryEncoderConstructor()

CategoryEncoder categoryEncoder = new
CategoryEncoder(categoryArray, settings);
Assert.IsNotNull(categoryEncoder);
```

Here the constructor has the parameters defined as a string and the other is dictionary. So the output is not null.

```
[TestMethod]
 public void
TestCategoryEncoderWithSelfMadeEncoderSet

Assert.AreEqual("hello", encoder["Name"]);
 }
```

We change here value of Name property. Here encodersettings is in a form of dictionary and getDefaultSettings is passing the values to the dictionary.

Here name is the key and hello is the value. The output is been compared over here with a self-given output and the output from variable encoder.

```
[TestMethod]
 public                                  void
TestCategoryEncoderIfDefaultSettingsAreInitial
ised()
 {

Assert.AreEqual(encoderSettings["W"],
encoder["W"]);
Assert.AreEqual(encoderSettings["Radius"],
encoder["Radius"]);
```

Here the same thing is happening as the above test case but the difference is that we donot provide any self given output. In this experiment we are comparing the values of W and Radius of the default settings with the values passed from the encoder. If the value of W from the default settings is equal to the value of W passed from the encoder, the test will pass. The test category encoder if default settings are initialized is applied here.

Inputs

In this project we have used two input sets. The first set is a list of array which contains item of groceries like milk, sugar, bread and egg. The second set is a list of array which contains different brands of cars like Porsche, Audi, Opel, BMW, Mercedes, Ford, Jeep and Mini.

Set 1

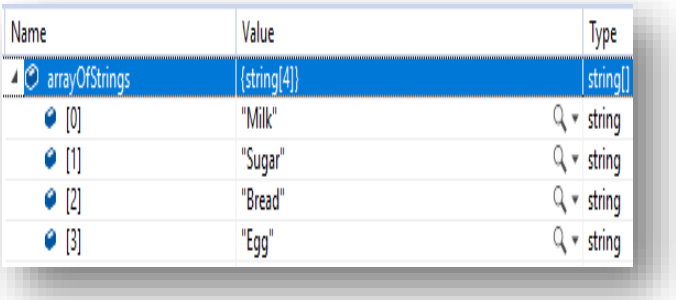
var arrayOfStrings = new string[] { "Porsche", "Audi", "Opel", "BMW", "Mercedes", "Ford", "Jeep", "MINI" };

Name	Value	Type
arrayOfStrings	{string[8]}	string[]
[0]	"Porsche"	string
[1]	"Audi"	string
[2]	"Opel"	string
[3]	"BMW"	string
[4]	"Mercedes"	string
[5]	"Ford"	string
[6]	"Jeep"	string
[7]	"MINI"	string

Fig. 5 Category List used in encoder

Set 2

```
var arrayOfStrings = new string[] { "Milk",
"Sugar", "Bread", "Egg"
```



Name	Value	Type
arrayOfStrings	{string[4]}	string[]
[0]	"Milk"	string
[1]	"Sugar"	string
[2]	"Bread"	string
[3]	"Egg"	string

Fig. 6 Category List used in encoder

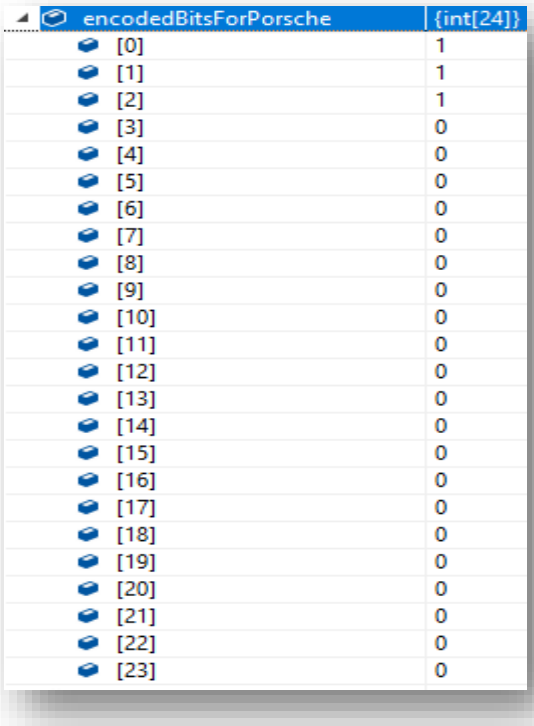
Category Encoder implemented in such a way that the categories are never mixed with each other which means that the there is no overlapping high bits and also the bits are non-periodic. In this project the value of width is 3 so that we only have three high bits (1's) and the remaning are low bits (0's) in output array. Figure 5 shows that the output values with respect to given input which is one of the element in the category list so the three bits are ones while remaining bits are zero.

```
int[] encodedBits = { 1, 1, 1 };
  var arrayOfStrings = new string[] {
"Pakistan" };
```

Here the length of array is 3

```
int[] encodedBitsForMercedes = { 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0 };
var arrayOfStrings = new string[] {
"Porsche", "Audi","Opel", "BMW", "Mercedes",
"Ford", "Jeep", "MINI" };
```

Here the length of array is 24



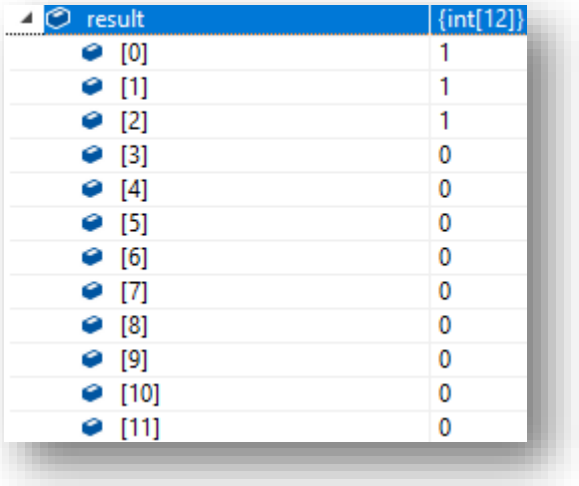
encodedBitsForPorsche	{int[24]}
[0]	1
[1]	1
[2]	1
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0
[10]	0
[11]	0
[12]	0
[13]	0
[14]	0
[15]	0
[16]	0
[17]	0
[18]	0
[19]	0
[20]	0
[21]	0
[22]	0
[23]	0

Fig. 7 Output array with respect to input

```
int[] encodedBits = { 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0 };

var arrayOfStrings = new string[] { "Milk", "Sugar",
"Bread", "Egg" };
```

Here the length of array is 12



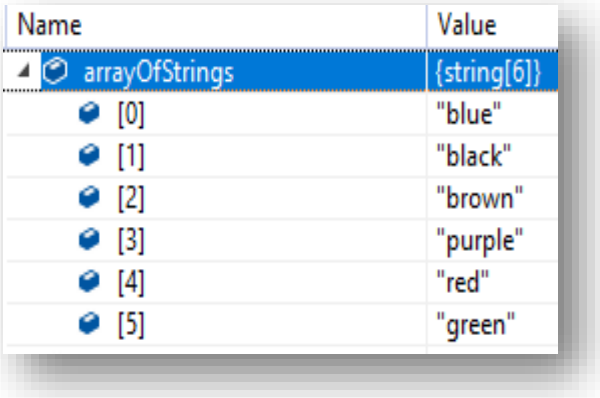
result	{int[12]}
[0]	1
[1]	1
[2]	1
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0
[10]	0
[11]	0

Fig. 8 Output array with respect to input

Explaining how Category Encoder works

We have a taken 6 different colours as input which means the size of string array is 6.

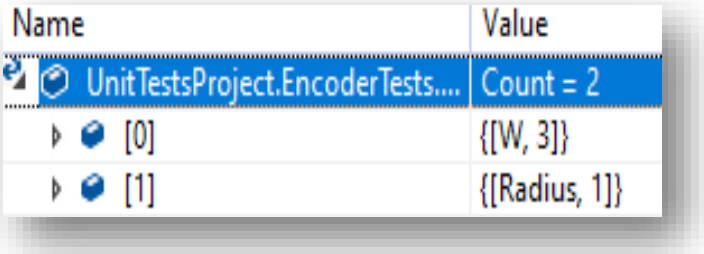
```
Vararrayofstrings=new string[]
{"blue,black,brown,purple,red,green" };
```



Name	Value
arrayOfStrings	{string[6]}
[0]	"blue"
[1]	"black"
[2]	"brown"
[3]	"purple"
[4]	"red"
[5]	"green"

Fig. 9 Category List used in encoder

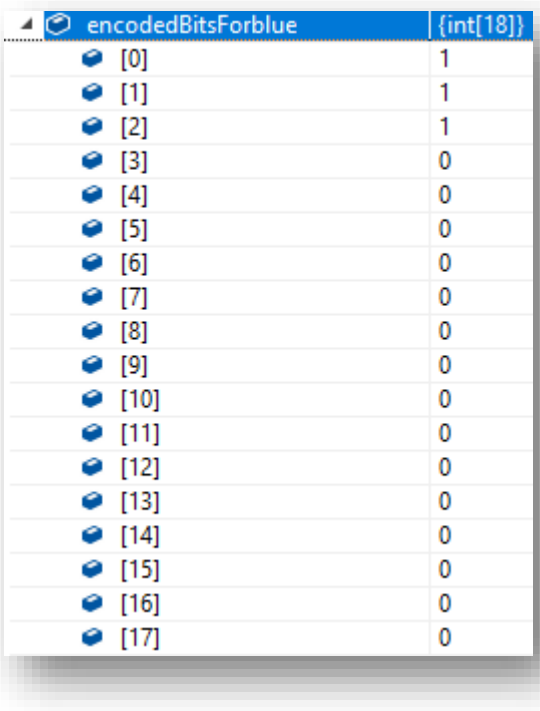
We have taken default settings which are width set to 3 and radius is set to 1, this means the encoded bit array should be of 24 bits in size.



Name	Value
UnitTestsProject.EncoderTests....	Count = 2
[0]	{[W, 3]}
[1]	{[Radius, 1]}

Fig. 10 default settings

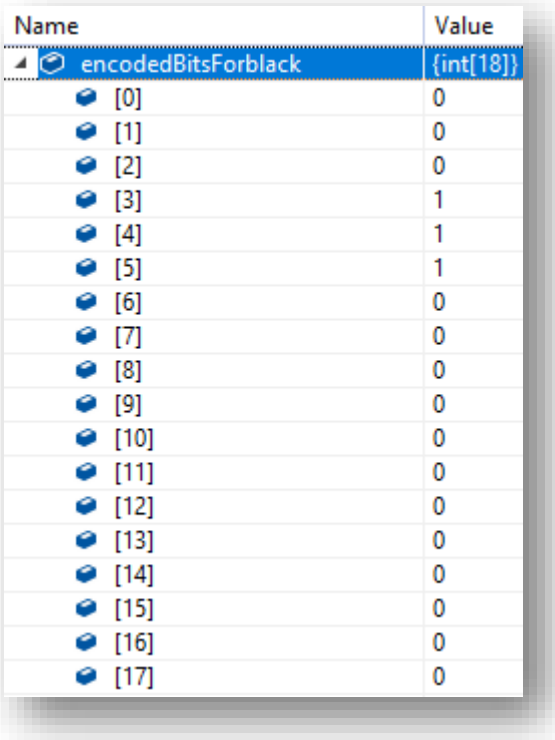
```
encodedBitsForblue= { 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```



encodedBitsForblue	{int[18]}
[0]	1
[1]	1
[2]	1
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0
[8]	0
[9]	0
[10]	0
[11]	0
[12]	0
[13]	0
[14]	0
[15]	0
[16]	0
[17]	0

Fig. 11 Output array with respect to input

```
encodedBitsForblack = { 0, 0, 0, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
```

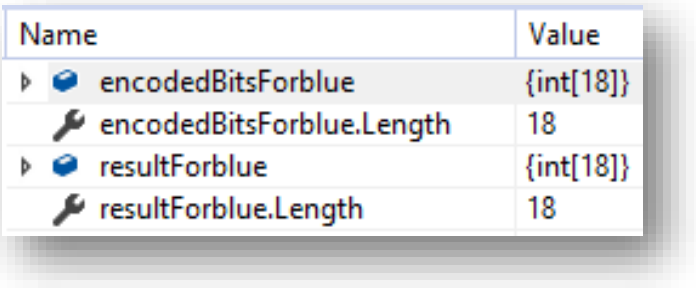


Name	Value
encodedBitsForblack	{int[18]}
[0]	0
[1]	0
[2]	0
[3]	1
[4]	1
[5]	1
[6]	0
[7]	0
[8]	0
[9]	0
[10]	0
[11]	0
[12]	0
[13]	0
[14]	0
[15]	0
[16]	0
[17]	0

Fig. 12 Output array with respect to input

In our input list blue in on the 0 position so the first 3 bits are set as ones and the rest are zeroes, same goes for black it has 1 position in our input so so the next three bits are ones which are 3-5 bits position and the rest are zeroes. As you can see each colour represents three bits as our width is set to 3. There is no overlapping of bits which means if first three bits are one for blue these bits cannot be ones for black.

Further in our test case we have checked our value length of the encoded bits for blue which should be 18. The length we get for encodedbitsforblue and resultforblue should be same.



Name	Value
encodedBitsForblue	{int[18]}
encodedBitsForblue.Length	18
resultForblue	{int[18]}
resultForblue.Length	18

Fig. 13 result

Limitations

- The value of radius is fixed if we have to change the value of radius the encoder cannot perform correct encoding and also give some errors.

- The encoder output is depends upon the length of the category list if we have to increase the number of elements in category list the number of output bits also increase.

- In the formula 1 we have used the parameter width. If the value of the width is zero the program cannot be run correctly and give errors.

Further work

- In future one can work to make it more efficient so that it can work on all the parameter and also reduce the length of output array to avoid the usage of memory.

- More work is to make the decoder so efficient that it can be used for communication purpose because the noise cannot disturb. Due to this encoder and decoder we can cope with noise effect easily.

- The encoder is implementing in such a way that we can properly save the data into the data base so that it can use easily and also save from third party like hackers.

Conclusion

The main purpose of this project is to implement the algorithm of Category Encoder for Hierarchical Temporal Memory. The values are encodes into the 1's and 0's according to the implemented algorithm. The input data is list of category which is encoded. At the end we have to verify the results with the help of unit test. Unit test is very helpful tool to verify different behavioral aspects of the system under test. The above figures shows that our project is perform operation according to the requirement, unit test also verify the results with previously recorded results. Taking everything into account, the composed calculation satisfies all the necessity and detail of the Category Encoder for HTM.

Links

1-) https://medium.com/@rockingrichie1994/understanding-hierarchal-temporal-memory-f6a1be38e07e

2-) https://numenta.com/neuroscience-research/sparse-distributed-representations/