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

ML20/21-5.8. Implement SDR Representation Samples

Junaid Khalid
junaid.khalid@stud.fra-uas.de

Abbad Zafar
abbad.zafar@stud.fra-uas.de

Sabeen Javaid
sabeen.javaid@stud.fra-uas.de

Farrukh Shafique
farrukh.sahfique@stud.fra-uas.de

***Abstract*— The main objective of the project is to describe how to represent Sparse Distributed Representations (SDRs). In this report We are representing SDRs as Indices and bitmaps. We are also representing the spatial pooler SDR learning using Heatmaps. There are multiple ways to generate SDRs, we have described how an input image or any scalar value is converted into SDRs using multiple examples. In this paper we are focusing on representing and creating SDRs using encoders, spatial pooler (SP) and Temporal Memory (TM). Encoders are used to convert the data input into binary array and is a part of Hierarchical Temporal Memory (HTM). Spatial pooler (SP) and temporal memory (TM) utilizes HTM parameters to generate SDRs using a HomeostaticPlasticityController. In this paper, we have discussed and described different test cases using the learning/memorizing capabilities of the spatial pooler (SP) and temporal memory (TM) with different learning sequences.**

Keywords—HTM, spatial pooler, encoder, SDR, temporal memory, heatmap, bitmap.

I. INTRODUCTION

Hierarchical Temporal Memory Cortical Algorithm is divided into three different parts which includes an encoder that converts the user input into a binary stream with n size also known as input vector/array of SP. The 2nd part is Spatial Pooler which can accept an input vector and create an SDR of that specific vector. The 3rd part includes Temporal Memory which can also refer to the Cognitive capacity of the system and the main objective of TM is to learn any sequence of SDRs which are given as input. Combination of these three parts completes a cortical sub layer which is enough to memorize multiple sequences but, like a brain, the memory is limited. The main challenge of HTM is building and controlling these SDRs.

The encoders are utilized to change over the information input into SDR in HTM frameworks. The encoders moreover choose which bits should be ones and which bits should be zeros within the output for any given input esteem in such a way that it captures the vital semantic characteristics of information. Subsequently, in case the inputs are alike, at that point output is more likely to have profoundly covering SDRs. In the HTM Cortical Learning Algorithm (CLA) Structure, an uncommon frame of input information is required to be prepared, which is called Sparse Distributed Representation SDR (a huge cluster of 0's and 1's bits). The on bits "1" demonstrate the dynamic neurons, while off bits "0" compare to inactive neurons [1]. Encoder takes the human justifiable information as input and changes over it into Machine readable format. Spatial Pooler at that point forms this input advance, actuates a few columns as per its algorithm and produces sparse binary vectors. Spatial pooler's output at that point acts as input for Temporal memory, where the dynamic columns from spatial pooler activate their cells. Temporal memory observes and learns the SDRs design from Spatial pooler and predicts another esteem for the relative arrangements on basis of its past information [2]. HTM CLA at that point gets vectors of dynamic cells from Temporal Memory as input and performs advanced cognitive capacities like prediction/inference and learning.

II. METHODOLOGY

A. Overview

The motivation behind HTM CLA is to let the machine perform progressed functionalities and cognitive tasks the same as the human brain. This time-based prediction system is planned to perform future information expectation based on learning and memorizing the information already fed into it. For HTM to memorize and anticipate, it needs input from the user.

The input can be within the form of a scalar value, date time string or a picture. The encoder is at that point utilized to change over the input into SDR which can be learned by HTM classifier. Depending on the encoder, the value can be anything i.e., scalar value, time, day in a week, geo-location etc. The SDR is regularly a cluster in '1' and '0'.

The Spatial Pooler is an algorithm inside of HTM that's able to learn spatial patterns. The Spatial Pooler regularly gets a cluster of bits as an input and changes over it into the SDR. Input of the Spatial Pooler is as a rule the array changed over by some encoder or bits of an image.

Another Portion of HTM CLA is Temporal Memory. The input of Temporal Memory is the SDR output produced by Spatial Pooler. Temporal memory is an algorithm which learns arrangements of Sparse Distributed Representations (SDRs) shaped by the Spatial Pooling algorithm and makes expectations of what another input SDR will be [3].

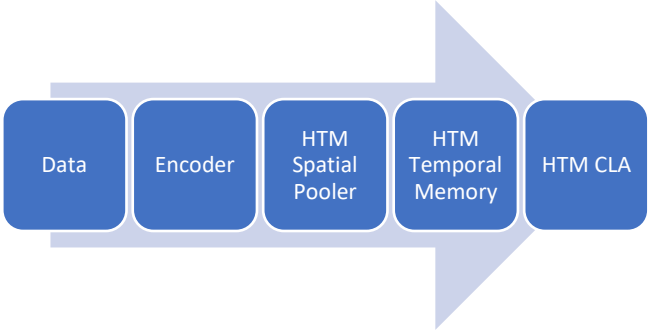


Fig.1. Flowchart of HTM CLA

B. Encoders

In this Context we have utilized distinctive encoders to encode the given input image/scalar value into a binary array which can be advanced passed to the spatial pooler for learning the SDRs of these inputs. Scalar Encoder could be a sort of encoder which binaries any scalar value given as input.

Encoding real-world information into SDRs may be an exceptionally vital preparation to get it in HTM. Semantic meaning within the input information must be encoded into a binary representation [4]. Encoders have to begin with the parameters which are initialized with predefined settings using its constructor. Encoder settings are made as a dictionary consisting of properties and their values. Properties consist of following:

1) Choose the range of values i.e., MinVal and MaxVal.
2) Range = MaxVal – MinVal.
3) The number of bits that are set to encode a single value the 'width' of output signal 'W' should be chosen.
4) Total number of bits in the output 'N' used for representation should be selected.
5) The resolution should be calculated based on periodic or non-periodic parameters.
6) Creating the encoded representation by beginning with N unset bits and then setting the W consecutive bits to active from index i.

Following is the code snippet illustrates how to create encoder settings:

```
{EncoderName} encoder = new {EncoderName}(new
Dictionary<string, object>()
            {
                { "W", 25},
                { "N", (int)0},
                { "Radius", (double)2.5},
                { "MinVal", (double)1},
                { "MaxVal", (double)50},
                { "Periodic", false},
                { "Name", "e.g Scalar Encoder"},
                { "ClipInput", false},
            });
```

After loading encoder settings into the encoder, the method 'encoder.Encode()' is invoked to start the encoding process and return an array of '0's and '1's which can be interpreted using 'StringifyVector'.

```
var result = encoder.Encode(input);
Debug.WriteLine(NeoCortexApi.Helpers.StringifyVe
ctor(result));
```

We can also represent SDR as Indices by returning the index number where SDRs are 1. Below code is used to get SDR as Indices

```
Debug.WriteLine(NeoCortexApi.Helpers.StringifyVe
ctor(ArrayUtils.IndexWhere(result, k => k ==
1)));
```

We can also represent SDR as bitmap, which is the graphical representation of SDRs. Below method is used to generate bitmap image.

```
NeoCortexUtils.DrawBitmap(twoDimArray, 1024,
1024, $"{outFolder}\\{input}.png", Color.Yellow,
Color.Black, text: input.ToString());
```

We can also represent SDR bitmap in form of Overlap, Union and Intersection using method "OverlapArraFun()" ,"UnionArraFun()" and "DrawIntersections()". Shown below:

```
Overlaparray = OverlapArraFun(sdrs[h], sdrs[w]);
int[,] twoDimenArray2 =
ArrayUtils.Make2DArray<int>(Overlaparray,
(int)Math.Sqrt(Overlaparray.Length),
(int)Math.Sqrt(Overlaparray.Length));
```

```
            int[,] twoDimArray1 =
ArrayUtils.Transpose(twoDimenArray2);

NeoCortexUtils.DrawBitmap(twoDimArray1, 1024,
1024,
$"{folder}\\Overlap_Union\\Overlap_{h}_{w}.png",
Color.PaleGreen, Color.Red, text:
$"Overlap_{h}_{w}.png");

Unionarray = UnionArraFun(sdrs[h], sdrs[w]);
int[,] twoDimenArray4 =
ArrayUtils.Make2DArray<int>(Unionarray,
(int)Math.Sqrt(Unionarray.Length),
(int)Math.Sqrt(Unionarray.Length));
            int[,] twoDimArray3 =
ArrayUtils.Transpose(twoDimenArray4);

NeoCortexUtils.DrawBitmap(twoDimArray3, 1024,
1024,
$"{folder}\\Overlap_Union\\Union_{h}_{w}.png",
Color.PaleGreen, Color.Green, text:
$"Overlap_{h}_{w}.png");

NeoCortexUtils.DrawIntersections(twoDimArray3,
twoDimArray1, 100,
$"{folder}\\Overlap_Union\\Intersection_{h}_{w}.
png", Color.Black, Color.Gray, text:
$"Intersection_{h}_{w}.png");
```

Fig.2. Graphical Representation of Overlap, Intersection and Union

Encoder is chosen according to the type of the inputs. There are some encoders available are such as:

- Scalar Encoder
- Datetime Encoder
- Boolean Encoder
- Category Encoder
- Geo-Spatial Encoder

In the below part we will discuss three encoders:

1) <u>Scalar Encoder:</u> This type of encoder is used to encode numbers.
2) <u>Datetime Encoder:</u> This type of encoder is used to encode Date and time.
3) <u>Category Encoder:</u> This type of encoder is used to encode strings.

1) *Scaler Encoder Example:*

For instance, our inputs are scalar values '10' and '20'. We need to encoder this input using scaler encoder. Following is the code:

```
int[] inputs = new inputs[] { 10, 20};
ScalarEncoder encoder = new
ScalarEncoder(new Dictionary<string,
object>()
 {
   { "W", 3},
   { "N", 100},
   { "MinVal", (double)0},
   { "MaxVal", (double)99},
   { "Periodic", true},
   { "Name", "Scalar Sequence"},
   { "ClipInput", true},
 });
```

W is the bit population used to represent a single input value. **N** is the total number of bits the encoder will use to encode the input. Encoder must also be intelligent enough to code the closest sorted scalar values with similar binary arrays, so that we have the correct information to be passed on to the next phase.

Below mentioned code shows how to encode input into a binary array:

```
var result = encoder.Encode(input);
Debug.WriteLine(input);
Debug.WriteLine(NeoCortexApi.Helpers.StringifyV
ector(result));
Debug.WriteLine("SDR As Indices");
Debug.WriteLine(NeoCortexApi.Helpers.StringifyV
ector(ArrayUtils.IndexWhere(result, k => k ==
1)));
```

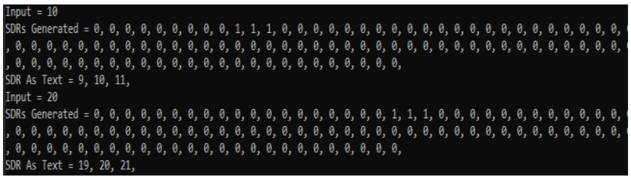


Fig.3. Scalar encoder SDRs and Indices representation

Below mentioned code snippets is used to generate bitmap. Bitmap output is shown in fig.4:

```
int[,] twoDimenArray =
ArrayUtils.Make2DArray<int>(result,
(int)Math.Sqrt(result.Length),
(int)Math.Sqrt(result.Length));
                var twoDimArray =
ArrayUtils.Transpose(twoDimenArray);
```

```
NeoCortexUtils.DrawBitmap(twoDimArray, 1024,
1024, $"{outFolder}\\{input}.png",
Color.Yellow, Color.Black, text:
input.ToString());
```

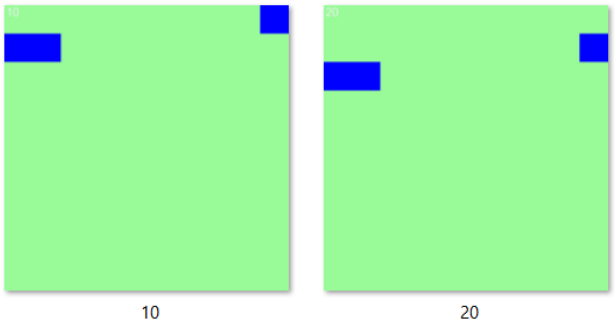


Fig.4. Scalar encoder Bitmap Representation

Fig.5. Scalar encoder Overlap, Union and Intersection graphical representation

2) Datetime Encoder Example:

Let's suppose Datetime encoder's input are "06/04/2020 01:28:07", "08/01/2017 11:27:07"Following is code for Datetime encoder:

```
Object[] inputs = new inputs[] { "05/02/2020
22:58:06", "08/01/2017 11:27:07" };
var now = DateTimeOffset.Now;

Dictionary<string, Dictionary<string, object>>
encoderSettings = new Dictionary<string,
Dictionary<string, object>>();
```

```
encoderSettings.Add("DateTimeEncoder", new
Dictionary<string, object>()
                    {
    { "W", 7},
    { "N", 20},
    { "MinVal", now.AddYears(-10)},
    { "MaxVal", now},
    { "Periodic", false},
    { "Name", "DateTimeEncoder"},
    { "ClipInput", false},
    { "Padding", 5},
   });

   var encoder = new
DateTimeEncoder(encoderSettings,
DateTimeEncoder.Precision.Days);


Console.WriteLine($"Input = {input}");
Console.WriteLine($"SDRs Generated =
{NeoCortexApi.Helpers.StringifyVector(result
)}");
Console.WriteLine($"SDR As Indices =
{NeoCortexApi.Helpers.StringifyVector(ArrayU
tils.IndexWhere(result, k => k == 1))}");
```

Input "06/04/2020 01:28:07", "08/01/2017 11:27:07" are shown as indices below and their intersection, union and overlap are shown in fig.7:

```
Input = 05/02/2020 22:58:06
SDRs Generated = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
SDR As Text = 12, 13, 14, 15, 16, 17, 18,
Input = 08/01/2017 11:27:07
SDRs Generated = 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
SDR As Text = 8, 9, 10, 11, 12, 13, 14,
```

Fig.6. Datetime encoder SDRs and Indices Representation

Fig.7a. Datetime encoder bitmap representation

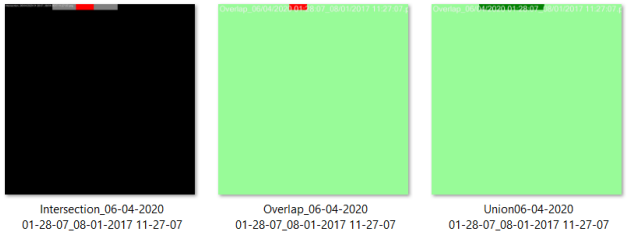


Fig.7b. Datetime encoder Overlap, Union and Intersection graphical representation

3) Category Encoder Example:

Let's take the Category encoder's inputs as "Doctor" and "Engineer". The code for this Encoder will be as follows:

```
Dictionary<string, object> encoderSettings3 =
getDefaultSettings();
var arrayOfStrings = new string[] { "Doctor",
"Engineer"};
static Dictionary<string, object>
getDefaultSettings(){
 Dictionary<String, Object> encoderSettings3 =
new Dictionary<string, object>();
 encoderSettings3.Add("W", 3);
 encoderSettings3.Add("Radius", (double)1);
 return encoderSettings3;
}
Console.WriteLine($"Input = {input}");
Console.WriteLine($"SDRs Generated =
{NeoCortexApi.Helpers.StringifyVector(result
)}");
Console.WriteLine($"SDR As Indices =
{NeoCortexApi.Helpers.StringifyVector(ArrayU
tils.IndexWhere(result, k => k == 1))}");
```

Input as "Doctor" and "Engineer". are shown as Indices below and their intersection, union and overlap are shown in fig.9:

```
Input = Doctor
SDRs Generated = 1, 1, 1, 0, 0, 0,
SDR As Text = 0, 1, 2,
Input = Engineer
SDRs Generated = 0, 0, 0, 1, 1, 1,
SDR As Text = 3, 4, 5,
```

Fig.8. Category encoder SDRs and Indices Representation

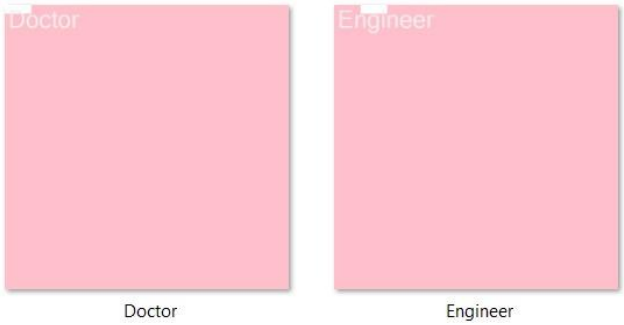


Fig.9a. Category encoder bitmap graphical representation

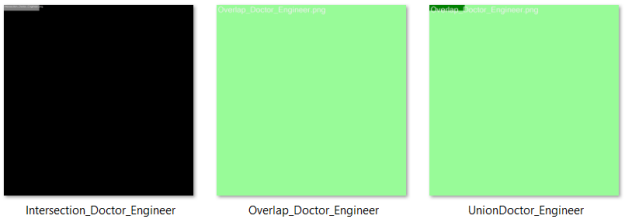


Fig.9. Category encoder Overlap, Union and Intersection graphical representation

C. Spatial Pooler

Spatial Pooler is the next phase in HTM learning of every input user provide. It is used to learn the SDR of the given input binary array which is also the output of the Encoder. Spatial Pooler in combination with Encoder can also be referred to as the recording of the input. The main idea for the Spatial Pooler SDR learning is to generate SDRs of the inputs. Once stable SDRs are learned, if the same input is given again, it will check the similarity with already learned SDRs and then generate a similar SDR of the current matching input. In this way it can differentiate between inputs if it is a new input or a previously known one. We will train the Spatial Pooler by providing different images as input and it will learn the SDRs for each input. During the learning phase, the SDR is in an unstable state. We will use **HomeostaticPlasticityController** to track the state of the SDR during learning. Once the stable state is reached, the learning process will be terminated.

Parameters used for Spatial Pooler learning of the Image are as follows:

```
HtmConfig htmConfig = new HtmConfig(new int[] {
imgSize, imgSize }, new int[] { 64, 64 })
{
 PotentialRadius = 10,
 PotentialPct = 1,
 GlobalInhibition = true,
 LocalAreaDensity = -1.0,
 NumActiveColumnsPerInhArea = 0.02 *
numOfCols,
 StimulusThreshold = 0.0,
 SynPermInactiveDec = 0.008,
 SynPermActiveInc = 0.05,
 SynPermConnected = 0.10,
 MinPctOverlapDutyCycles = 1.0,
 MinPctActiveDutyCycles = 0.001,
 DutyCyclePeriod = 100,
 MaxBoost = 10.0,
 RandomGenSeed = 42,
 Random = new ThreadSafeRandom(42)
 };
```

'**NumActiveColumsPerInhArea** = 0.02 * numOfCols.' This parameter only activates two percent of the total number of columns in the SDR to represent a given binary array. Once the SDR is learned, the total number of active columns will be 2%.

'**PotentialRadius** = 10.' These are the number of connections between the columns and the sensory input bits. It is that each column will decide its weight of the connection by looking at 10 sensory bits.

```
NeoCortexUtils.DrawBitmap(twoDimArray,1024,
1024, $"{outFolder}\\{input}.png",
Color.PaleGreen, Color.Blue, text:
input.ToString());
```

The **DrawBitmap** in NeoCortexUtils is a function which we use to create the graphical representation of the input arrays. It will take the array and plot it in a plane so that the array can be represented as Graphics. Further you can also set the colors of the indices where there is "1" and "0" respectively. For an example you can see the image below:

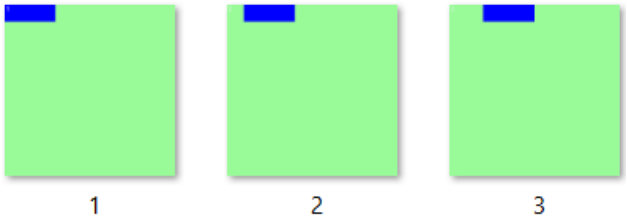


Fig.10. Bitmap example

```
public int[] UnionArraFun1(int[] arr1, int[]
arr2)
{
  int[] nw = arr1;
  int[] old = arr2;
  int[] un = new int[100];
  for (int i = 0; i < arr1.Length; i++)
   {
     if ((nw[i] == 0 && old[i] == 1) ||
(nw[i] == 1 && old[i] == 0) || (nw[i] == 1 &&
old[i] == 1))
      {
       un[i] = 1;
      }
      else
      {
       un[i] = 0;
      }
   }
 return un;
}
```

Union: "`UnionArraFun1()`" it is method we defined that returns a binary array which contains the combination of all the firing bits of the two-comparing array. The idea behind is to observe the learned vectors together and check how many numbers of firing bits are there in comparison. As input, it will take two arrays. Example is pasted below:

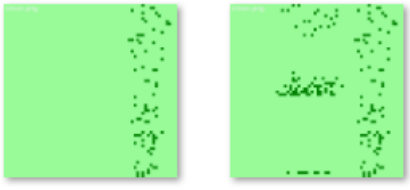


Fig11. Union example

```
public int[] OverlapArraFun1(int[] arr1, int[] arr2)
{
 int[] nw = arr1;
 int[] old = arr2;
  int[] ovrlap = new int[500];
  for (int i = 0; i < arr1.Length; i++)
   {
    if (nw[i] == 1 && old[i] == 1)
    {
     ovrlap[i] = 1;
    }
    else
    {
     ovrlap[i] = 0;
    }
   }
 return ovrlap;
 }
```

Overlaps: "OverlapArraFun1()" it is method we defined that returns binary vector which shows the overlapping between the two binary arrays getting compared. While comparing we check for the active bits of the two arrays are on the same indices or not. Those which are on the same indices will be returned.

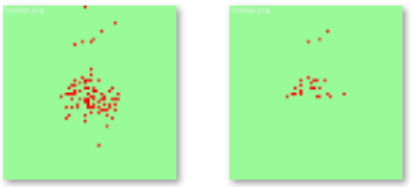


Fig.12. Overlapping example

```
NeoCortexUtils.DrawIntersections(twoDimArray3,
twoDimArray1, 100,
$"{folder}\\Overlap_Union\\Intersection_{h.ToS
tring().Replace("/", "-").Replace(":", "-
")}_{w.ToString().Replace("/", "-
").Replace(":", "-")}.png", Color.Black,
Color.Gray, text:
$"Intersection_{h}_{w}.png");
```

Intersection: "DrawIntersections()" it is method we defined to the check the similarity between two arrays. It takes two arrays as input and generate a bitmap image of intersection between them. The intersection columns are shown in red. Example is shown below:

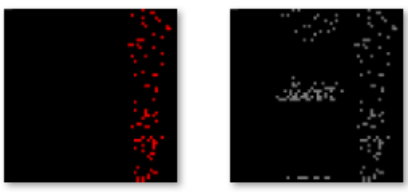


Fig.13. Intersection example

```
string inputBinaryImageFile =
NeoCortexUtils.BinarizeImage($"{trainingImage}"
, imgSize, testName);
```

BinarizeImage() is the predefined function in NeoCortexUtils class used to binarize the given image of any size into a binary array. It is a type of encoder which takes an image as input and returns a Binary Bit array. Encoders are categorized based on their input; the output of every encoder will always be a binary array.

Heatmaps:

During the learning process of any SDR in Spatial Pooler there are multiple SDRs which are created which are known as Unstable SDRs. But each SDR has 2% of the whole number of Columns active. To keep track of every SDR during learning we can use heatmaps overlaps which will overlap each unstable SDR until the Stable SDR is achieved. During learning in Spatial Pooler there is a Permanence Threshold **θp** which should be achieved in order to turn on the Column in the final SDR of the input. After the learning process.

```
NeoCortexUtils.DrawHeatmaps(overlapArrays,
$"{outputImage}_overlap.png", 1024, 1024, red,
red, green);
```

DrawHeatmaps is a predefined function in the NeoCortexUtils class which can generate the graphical representation of the two-dimensional array you provide as input. 2nd argument is the file path to save the image. 3rd and 4th argument are used to set the size of the image in pixels, and the last three arguments are the thresholds you can set for showing hotspots in a heatmaps. Example of heatmap is shown below:

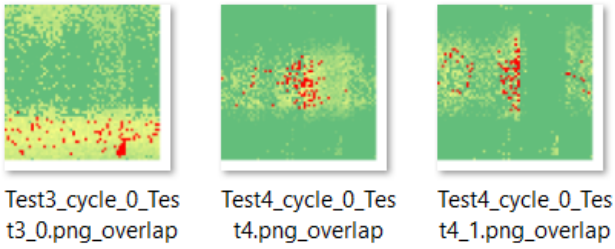


Fig.14. Heatmap example

D. Temporal Memory

Temporal memory is an algorithm which learns sequences of Sparse Distributed Representations (SDRs) formed by the Spatial Pooling algorithm, and makes predictions of what the next input SDR will be [3]

After the SDRs are created by SP, it is then passed to Temporal memory for learning. The HtmConfig parameters are the same as that of spatial pooler and Temporal Memory is then initialized. It is added to the Cortex Layer after the new born cycle is completed.

```
TemporalMemory tm = new TemporalMemory();4
tm.Init(mem);
layer1.HtmModules.Add("tm", tm);
```

"Compute()" function is again used to compute the results in the form of Cell SDR. . An example SDR produced by Temporal memory algorithm is presented in the following image:

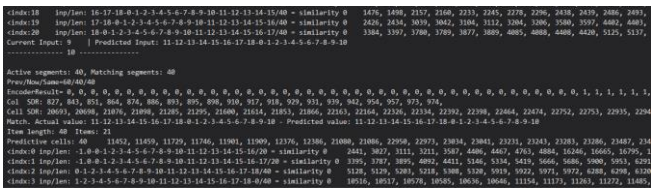


Fig.15. Temporal Memory Algorithm

III. TEST CASES WITH RESULT

1. Test Case to encode Random Number using Scaler Encoder.

This Unit test is performed, to test how scaler encoder is used to encode different scalar inputs. In this test, we have used different inputs such as `1,2,3,4,5,6.`

Parameter setting done for this encoding scheme is:

1) This MinVal is 0 and the MaxVal 99.
2) The number of bits that are set to encode a single value the 'width' of output signal 'W' used for representation is 3.
3) Total number of bits in the output 'N' used for representation is 100.
4) We are choosing the value of N=100 and W=3 to get the desired output shown below:

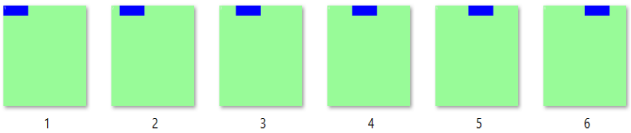


Fig.16. Expected Output of Scaler Encoder

```
Dictionary<string, Dictionary<string,
object>> encoderSettings = new
Dictionary<string, Dictionary<string,
object>>();
```

```
encoderSettings.Add("DateTimeEncoder", new
Dictionary<string, object>()
{
   { "W", 7},
   { "N", 20},
   { "MinVal", now.AddYears(-10)},
   { "MaxVal", now},
   { "Periodic", false},
   { "Name", "DateTimeEncoder"},
   { "ClipInput", false},
   { "Padding", 5},
});

var encoder = new
DateTimeEncoder(encoderSettings,
DateTimeEncoder.Precision.Days);
```

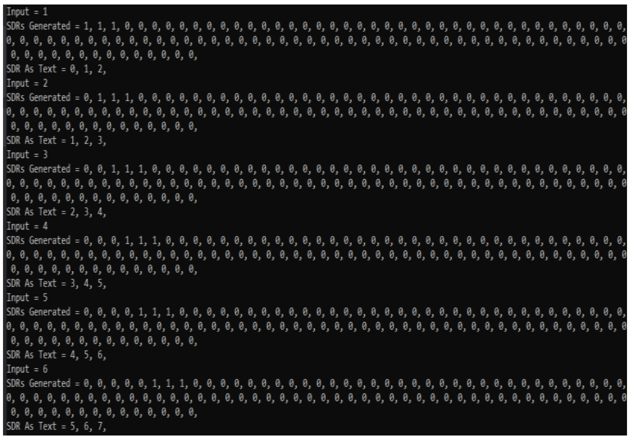


Fig.17. Indices Representation of SDR

```
Overlaparray = OverlapArraFun(sdrs[h],
sdrs[w]);
NeoCortexUtils.DrawBitmap(twoDimArray1,
1024, 1024,
$"{folder}\\Overlap_Union\\Overlap_{h}_{w}.p
ng", Color.PaleGreen, Color.Red, text:
$"Overlap_{h}_{w}.png");

Unionarray = UnionArraFun(sdrs[h], sdrs[w]);
NeoCortexUtils.DrawBitmap(twoDimArray3,
1024, 1024,
$"{folder}\\Overlap_Union\\Union_{h}_{w}.png
", Color.PaleGreen, Color.Green, text:
$"Overlap_{h}_{w}.png");

NeoCortexUtils.DrawIntersections(twoDimArray
3, twoDimArray1, 100,
$"{folder}\\Overlap_Union\\Intersection_{h}_
{w}.png", Color.Black, Color.Gray, text:
$"Intersection_{h}_{w}.png");
```

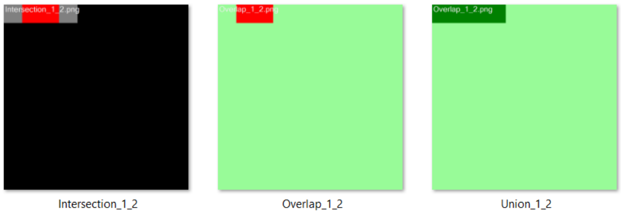


Fig.18. Scalar encoder Overlap, Union and Intersection graphical representation

We use the Bitmap technique to display the output in 2D Bitmap format after all the inputs have been encoded. We run the program after setting all the parameter values, the output images will be captured and stored in a folder, demonstrating how the shifting occurs for each input. The SDR can also be represented as indices which means we can mention only indexes where SDR is “1” and overlap, intersection and union are also represented graphically.

2. Test Case to encode Date and time using Datetime Encoder.

Another unit test is performed, to test how we can encode Date and time using Datetime Encoder. In this test, we have used multiple Date Time values such as `"05/02/2020 22:58:06"`, `"06/04/2020 01:28:07"`, `"07/09/2019 21:15:07"`, `"08/01/2017 11:27:07"`. The encoder parameters setting is as follows:

The bitmap image created using Datetime encoder is as follows:

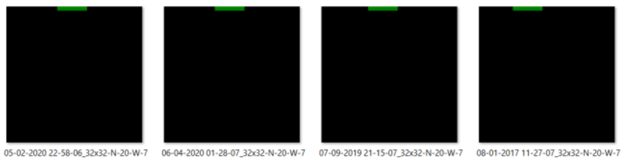


Fig.19. Bitmap Representation of SDR using Datetime Encoder

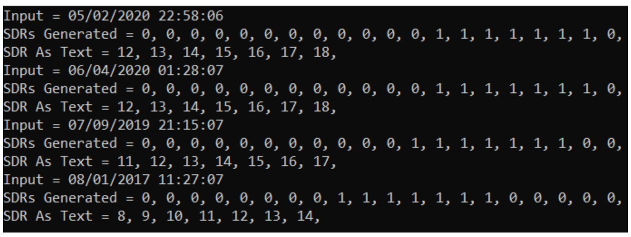


Fig.20. Indices Representation of SDR using Datetime Encoder

```
NeoCortexUtils.DrawBitmap(twoDimArray1,
1024, 1024,
$"{folder}\\Overlap_Union\\Overlap_{h.ToStri
ng().Replace("/", "-").Replace(":", "-
")}_{w.ToString().Replace("/", "-
").Replace(":", "-")}.png", Color.PaleGreen,
Color.Red, text: $"Overlap_{h}_{w}.png");
Unionarray = UnionArraFun1(sdr1, sdr2);
int[,] twoDimenArray4 =
ArrayUtils.Make2DArray<int>(Unionarray, 32,
32);
int[,] twoDimArray3 =
ArrayUtils.Transpose(twoDimenArray4);

NeoCortexUtils.DrawBitmap(twoDimArray3,
1024, 1024,
$"{folder}\\Overlap_Union\\Union{h.ToString(
).Replace("/", "-").Replace(":", "-
")}_{w.ToString().Replace("/", "-
").Replace(":", "-")}.png", Color.PaleGreen,
Color.Green, text: $"Overlap_{h}_{w}.png");

NeoCortexUtils.DrawIntersections(twoDimArray
3, twoDimArray1, 100,
$"{folder}\\Overlap_Union\\Intersection_{h.T
oString().Replace("/", "-").Replace(":", "-
")}_{w.ToString().Replace("/", "-
").Replace(":", "-")}.png", Color.Black,
Color.Gray, text:
$"Intersection_{h}_{w}.png");
```

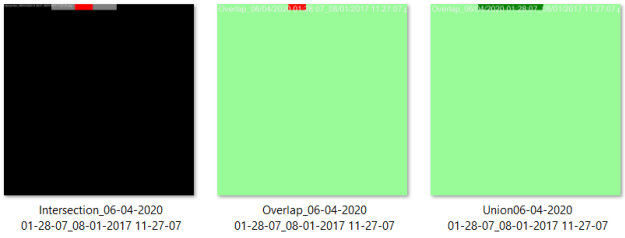


Fig.21. Datetime encoder Overlap, Union and Intersection graphical representation

3. Test Case to encode Array of Colors using Category Encoder.

Another unit test is performed using Category Encoder. This type of encoder is used to encode strings (words). For this test, we have used an array of string (colors) such as “"Doctor", "Engineer", "Lawyer", "Scientist" and encoded it using a Category encoder.

Following parameters are defined in settings of category encoder:

```
Dictionary<String, Object> encoderSettings = new
Dictionary<string, object>();
encoderSettings.Add(“W”, 3);
encoderSettings.Add(“Radius”, (double)1);
```

Following Bitmaps are generated using category encoder:

Fig.22. Bitmap Representation of SDR using Category Encoder

Following SDRs are generated using category encoder:

```
Input = Doctor
SDRs Generated = 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
SDR As Text = 0, 1, 2,
Input = Engineer
SDRs Generated = 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
SDR As Text = 3, 4, 5,
Input = Lawyer
SDRs Generated = 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
SDR As Text = 6, 7, 8,
Input = Scientist
SDRs Generated = 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
SDR As Text = 9, 10, 11,
```

Fig.23. Indices Representation of SDR using Category Encoder

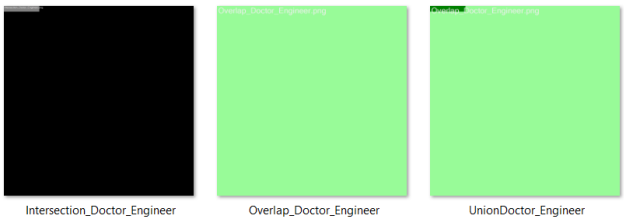


Fig.24. Category encoder Overlap, Union and Intersection graphical representation

4. Test for learning a pattern of input using Spatial Pooler.

In this test spatial pooler is used to learn SDR of scalar value 0 to 10.

Following are the HTM configuration parameters used in this experiment:

```
HtmConfig cfg = new HtmConfig(new int[] {
inputBits }, new int[] { numColumns })
{
CellsPerColumn = 10,
MaxBoost = maxBoost,
DutyCyclePeriod = 100,
MinPctOverlapDutyCycles = minOctOverlapCycles,
```

```
GlobalInhibition = true,
NumActiveColumnsPerInhArea = 0.02 * numColumns,
PotentialRadius = (int)(0.8 * inputBits),
LocalAreaDensity = -1,
ActivationThreshold = 10,
MaxSynapsesPerSegment = (int)(0.02 *
numColumns),
Random = new ThreadSafeRandom(42)
};
```

Scaler encoder is used to encode the inputs of numbers. The encoder parameters are defined below:

```
Dictionary<string, object> settings = new
Dictionary<string, object>()
            {
                { "W", 15},
                { "N", inputBits},
                { "Radius", -1.0},
                { "MinVal", 0.0},
                 { "Periodic", false},
                { "Name", "scalar"},
                { "ClipInput", false},
                { "MaxVal", max}
            };
```

First encoder and spatial pooler are initialized, and a cortex layer is created and after that encoder and spatial pooler are added to that layer. Active Columns are SDR's generated using spatial pooler and Compute function will be used for this purpose. A small result part of this unit test is attached below:

Fig.25. SDR using Spatial Pooler

5. Unit Test Similarity using Overlap Images:

This test will demonstrate how to generate Image as input into a binary array then binary into SDR and then after that how to build the overlap heatmap. A brief Introduction of this experiment is shown in the below image:

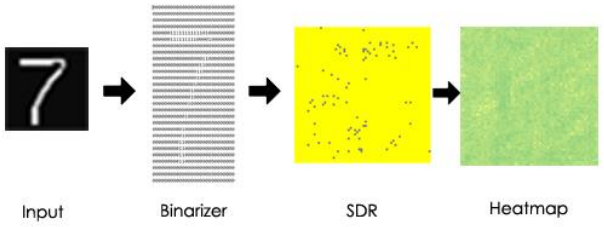


Fig.28. Unit test flowgraph

Overlap Bitmaps Using Scalar Encoder:

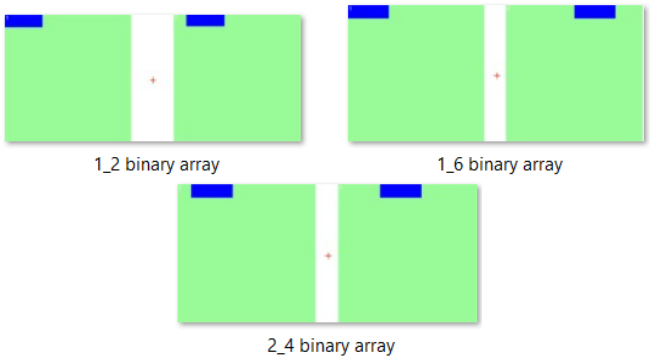


Fig.29. Binary arrays

We have encoded a sequence of scalar values from 1 to 6. The bitmaps generated are given in the fig.29. The blue line is showing the number of ON bits while the Green is showing the number of Off bits. Now we will see the overlaps and unions they make while placing one above the other.

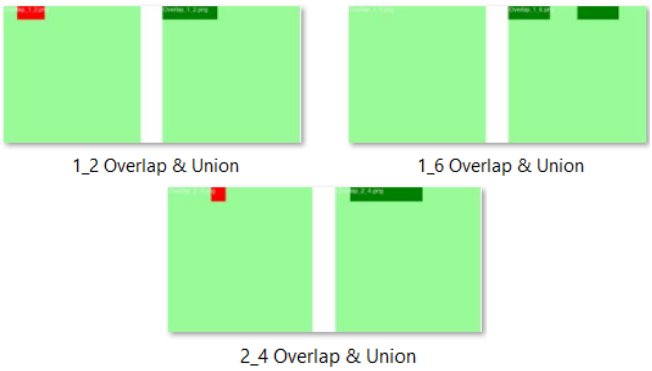


Fig.30. Overlaps & Unions

In fig.30., bits which are overlapping over each other are represented as red while the bits which are union with each other are shown in green. In the 1st picture we observed that the overlap bits are greater in number than the other comparison we made. Input must be sorted before giving to an encoder. Comparison between 1 and 2 has shown more overlap as they are adjacent values in a sequence we know as counting. The overlap between 3 and 5 is smaller as compared to 1st overlap is because there is another value in between which is 4 which was also encoded. The last Overlap between 1 and 6 has no overlap because you can see in the union image that their encoded bits are placed far from each other. Note Encoder must be intelligent enough to encode

adjacent values in a sequence with bit arrays that have a greater overlapping part.

6. Overlap Bitmaps using Spatial Pooler:

Test No 1:

Input 7_Line Image

Fig.31. Input Images

In the fig.31., we have two input images which are not at all similar with each other and are separated by a yellow line. Let's see how their binaries image and there stable SDR looks like:

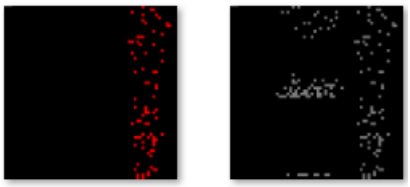
Line Number

Fig.32. SDR & Binary arrays

In fig.32., here there are two bitmap images as output both are the graphical representation of an array. The left side image represents SDRs which is learned after multiple iterations in Spatial Pooler learning mechanism after the array became stable it was then passed to the bitmap which gives a two-dimensional image of that SDR. At the right-side image is the bit array that we get as output from the encoder. In SDRs, we observed that the arrangement of active columns is shown in an irregular manner; this is why it is known as Sparse Distributed Representation. For example, if we pass the SDR array of "7" to the system it will know that is the image "7" written on it. Machine understands every frame you give as SDRs.

SDR Overlap & Union 7 SDR Overlap & Union 7_Line

Fig.33. Overlaps & Unions



Intersection_1 Intersection_2

Fig.34. Intersection images

During this experiment we have also observed how much similarity we can see if you compare the similar images and unsimilar images. In the fig.34., "Intesection_1" shows the intersection between the SDR of two "line_image.png" the you can see that all the columns are showing in red because they are overlapping each other in the intersection image. Now "Intersection_2" represents the comparison between "Image_7.png" and "Image_line.png". All the columns are shown in gray because no column is overlapping between these two input images.

Test No 2:

In this test we will take input "image_8.png" and "image_7.png" and do the similarity test. Input is given below:

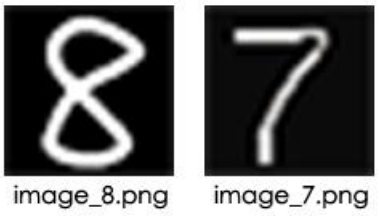


image_8.png image_7.png

Fig.35. Input Images

In this example we will compare Image_8.png and Image_7.png and see their comparison between the SDRs of Image_8.png and Image_7.png using methods Union, Overlap and Intersection. The binary array and the SDR are shown below

SDR_Binary Array of Image_7 SDR_Binary Array of Image_8

Fig.36. SDR & Binary arrays

Once you have created both overlap and union array now you can generate the intersection image between the two SDRs. We modified the NeoCortexUtils.DrawBitmap() code and created a method NeoCortexUtils.DrawIntersections() which will take Union and overlapping array as input and generate intersection bitmap as output of two comparing SDRs:

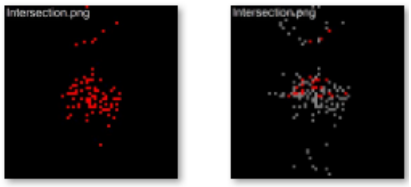


Fig.37. Intersection images

In the fig.37., "Intesection_1" shows the intersection between the SDR of two "Image_8.png" the you can see that all the columns are showing in red because they are overlapping each other in the intersection image. Now "Intersection_2" represents the comparison between "Image_7.png" and "Image_8.png". Some of the columns are shown in gray because some columns are in red which shows overlapping between these two input images.

Image With/Without Transpose:

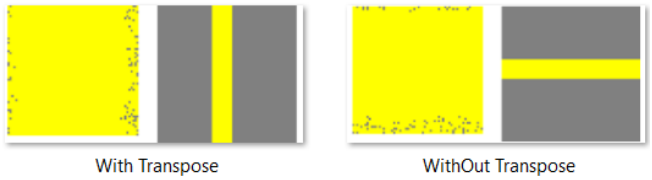


Fig.38. Transpose of SDR & Binary arrays

In this Experiment, we have plotted the active area and the SDR array with transpose shown on left and without transpose shown on right. In Fig.38. it is clearly observed that while taking a transpose rows and columns are interchanged with each other.

Overlap Heatmaps:

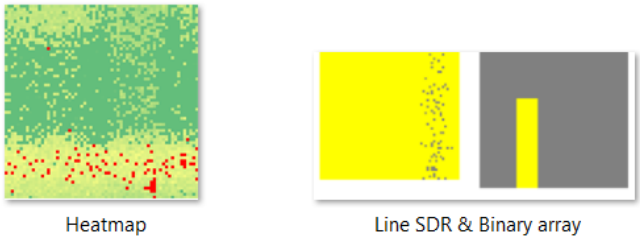


Fig.39. Heatmap Image during SDR learning

In this experiment, we observed the heatmaps and what the information we can fetch form it. I have used the same input which is a vertical line. In Spatial Pooler, during the learning of any Input there are multiple SDRs which are generated when the spatial pooler is trying to remember the image frame we passed as input. Heatmap will keep track of every SDR during the learning; those SDRs are known as Unstable SDRs and at a certain point in time a SDR will turn ON 2% of the columns as active columns. There is a threshold **θp** which we set in the parameters of HTM known as Permanence Threshold. What we observed in this experiment is that when 2% of the columns cross the given threshold it stops the process of learning and enters into the stable state. The columns which are above those thresholds are shown as red on the heatmap. While the yellow color represents the column, which are around 50% of the maximum value a column achieved during learning and green show which are not used at all during learning of this image.

IV. Conclusion

In this project we have compares different scalar value as well as different and same inputs images the results are given below:

When we are encoding a sequence of scalar values first the data need to be sorted because the encoder will create the similarity between the values which are passed adjacent to each other. Let suppose if the date is in {1,4,2,5,6} It must be arranged in proper order {1,2,4,5,6}.

While checking the similarity between two SDR we can use the overlap & union function in order to see the overlapping columns. If we find any similarity between the SDRs there will surely be similarity between the inputs that have those SDRs. If the same input is again passed during SP learning. it can distinguish between either that input is already learned or new based on its comparison with already learned SDRs.

V. References

[1] S. Ahmad, M. Lewis, "Temporal memory algorithm", Technical Report Version 0.5, 2017, Numenta Inc

[2] Temporal Memory Algorithm https://numenta.com/resources/biological-and-machine-intelligence/temporal-memory-algorithm/

[3] HTM School https://numenta.org/htm-school/

[4] ddobric/neocortexapi https://github.com/ddobric/neocortexapi

[5] ML 19/20-5.10 Validate Memorizing Capabilities of Spatial Pooler Dipanjan Saha, Pradosh Kumar Panda, Rina Yadav https://github.com/ddobric/neocortexapi/blob/master/NeoCortexApi/Documentation/Experiments/ML-19-20_20-5.10_ValdatingMemorizingCapabilitesOfSpatialPooler