Information Technology Course Module Individual Project
by Dr. Andreas Pech / Damir Dobric

Grid Cell Encoder

Sabin Bir Bajracharya
sabin.bajracharya@gmail.com

***Abstract*— Grid cells produce internal coordinate maps that allow an animal to navigate from one location to another. Each grid cell forms a unique pattern of coordinates, which is shifted with respect to the coordinates formed by other nearby grid cells. A group of grid cell module are little SDRs which when combine together produces a bigger SDR that represents a unique location in space that can map high dimensional space.**

***Keywords*—Grid cell, Grid cell module, SDR, Grid cell Encoder**

I. Introduction

One of the most fundamental and important abilities in the animal kingdom, and also for humans, is navigation in the environment, or traveling from one location to another. The navigation system in the brain is in charge of representing the location in the environment and orienting the body so that movement from one place to another is successful. This mental image of the surroundings is frequently referred to as a cognitive map. An animal needs to construct an internal "cognitive map" of the surrounding world in order to successfully navigate. This is carried out through a particular system in the brain that consists of multiple different brain areas and different cell types, each with a special function in navigation. One of such special system of nerve cells in the brain is called grid cells.

The entorhinal cortex (Fig. 1, purple area), a deep brain region close to the hippocampus, is where the grid cell system is found in the centre of the brain, a little below the ear level. There are numerous places in the environment where a grid cell becomes active (Fig. 1). The results show that these places create a symmetric, incredibly precise crystal-like structure, with equilateral triangles connecting the centers of nearby places. These places, or coordinates, come together to create a hexagonal (six-sided polygon) grid.

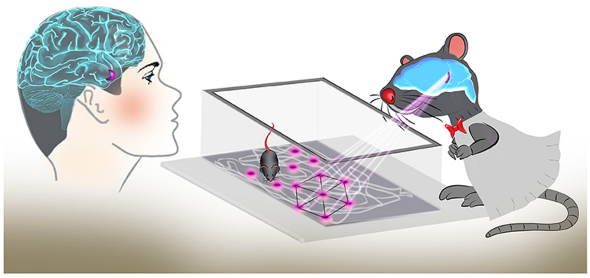

Fig 1: Grid cells firing at multiple location forming a hexagonal grid [2]

Each grid cell creates its own set of coordinates that are displaced in relation to the coordinates created by other grid cells nearby. In this manner, grid patterns are "filled" throughout the entire area (Fig. 2 (A)). Because each grid cell is active in several areas and forms a grid, you cannot locate the animal using just one grid cell. However, due to the shifting of the animal's location across different grid cells and the different grid scales (Fig. 2(C)), it is possible to characterize the animal's current location with extreme precision by utilizing the overlapping grids of numerous cells[3]. These grid patterns serve as an internal map of coordinates in the brain and can also be used for measuring the distance between different points in space, a critical requirement for navigation (Fig 2 (B)).

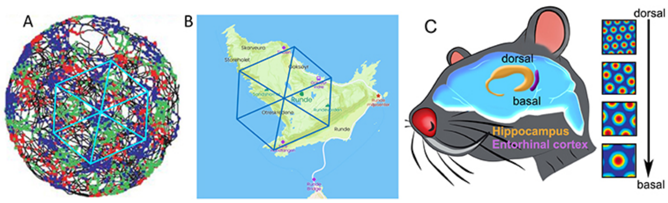

Fig 2: Grid cell coordinate mapping the environment [2].

Grid cells produce internal coordinate maps that allow an animal to navigate from one location to another. The grid cells work in unison with place cells and with other cell types, such as head direction cells and speed cells. This navigation system also integrates information from the senses, to calibrate the internal maps with the environment. This whole navigation system in the brain allows us to perform complex navigation tasks in a smooth and seamless manner.

A grid cell module is simply a group of grid cells sharing the same projection properties onto space. Using many grid cells, we can cover a patch of space and use it to tile over a larger space.

In figure 3, the overlay on the bottom left shows all 16 grid cells within the module, and which one is currently active in response to an object’s location (red dot) in space. One grid cell module can tell us a lot more about an object’s location in space than a lone grid cell. The grid cell module can show one must be within one of many locations moving across the space. But it cannot decipher an exact location. So one grid cell module is not enough to uniquely map space. But when we use many grid cell modules together, we can map a virtually infinite amount of space.

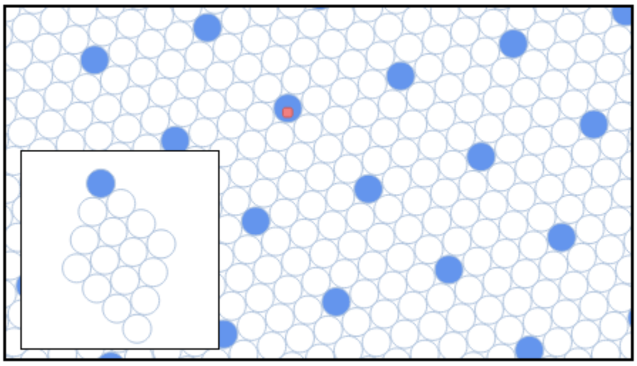

Figure 3: Grid cell modules containing 16 Grid Cells [2]

When grid cell modules work together, they can uniquely map space by combining their representations. Each cell in a grid cell module is like a bit in a Sparse Distributed Representation (SDR). As shown in Figure 4, this space is being mapped with 6 grid cell modules each with 16 cells (96 grid cells).

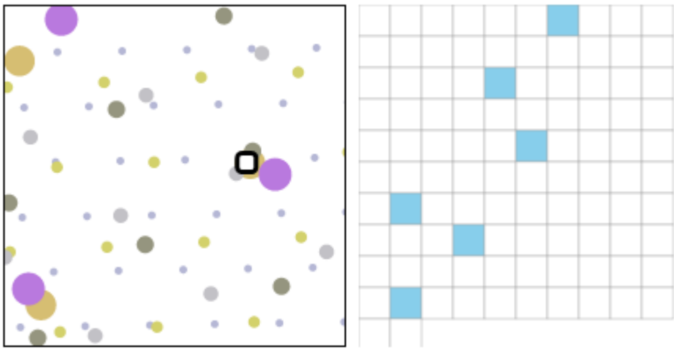

Figure 4: Grid Cell Modules as SDR [2]

Grid cell module activations are SDRs (Figure 5), and can be used to represent semantic location information in many ways in the brain. Many grid cell modules' SDR representations can be used to create unique binary representations of space.

II.METHODS

A. GridCell Encoder constructor

```
public GridCellEncoder(int size, double sparsity = 0.15)
{
    this.size = size;
    this.sparsity = sparsity;

    for (var i = 0; i < 5; i++)
    {
        periods[i] = 6 * Math.Pow(Math.Pow(2, 0.5), i);
    }

    var partitions = np.linspace(0, size, periods.Length + 1);
    for (var i = 0; i < partitions.size - 1; i++)
    {
        this.partitions.Add(new int[2] {
            (int) ((float) partitions[i].GetValue()), (int)
((float)partitions[i+1].GetValue()),
        });
    }

    // Assign each module a random offset and orientation.
    var rng = np.random.RandomState(23);
    offsets_ = rng.uniform(0, periods.Max() * 9, new int[]
{ size, 2 });

    foreach (var period in periods)
    {
        var angle = rng.uniform(0, 1, np.float32) * 2 * Math.PI;
        var c = Math.Cos(angle);
        var s = Math.Sin(angle);

        Tuple<double, double>[] sdist = {
            Tuple.Create(c, -s),
            Tuple.Create(s, c)
        };

        rotationMatrix.Add(sdist);
    }

}
```

The constructor takes two arguments as input. The size argument defines the length of the SDR whereas the sparsity refers to the percentage of neurons being active at any given time.

The periods array is a list of distances.The length of this list defines the number of distinct grid cell modules. In the sample code the periods is of length 5 which represents a total of 5 grid cell modules. The period of a module is the distance between the centers of a grid cell receptive fields.

Each grid cells modules are of the same size but are slightly of different orientation and offsets. The “offsets” property is a two dimensional (size, 2) list of random values generated between 0 and maximum value in the periods list. Each row in the offset represents the x and y offset for the location.

The “rotationMatrix” contains the orientation angle for the each grid modules. For 5 grid modules, the “rotationMatrix” will be a size of 5.

B. *Grid Cell Encoder Algorithm*

```
Step 1: Calculate displacement
  - Apply offsets to the location i.e. (x, y)
  - Then, apply rotation

Step 2: Find radius of grid cell in each grid
module

Step 3: Convert the displacement into and out
of hexagonal coordinates

Step 4: Calculate difference between the
result of Step 3 with Step 1.

Step 5: Find the distance (hypoteneous)
between the location (result from Step 4) and
the RF center.

Step 6: Get SDR by activating the closest grid
cells in each module based on sparsity.
```

a) Step 1: Calculate displacement

```
var location = np.ndarray((1,2));
location[0][0] = x;
location[0][1] = y;

var displacement = location - offsets_;

for (var i = 0; i < partitions.Count; i++) {
    var start = partitions[i][0];
    var stop = partitions[i][1];

    var r = rotationMatrix[i];

    var rr = np.ndarray((2,2));
    rr[0][0] = r[0].Item1;
    rr[0][1] = r[0].Item2;
    rr[1][0] = r[1].Item1;
    rr[1][1] = r[1].Item2;

    displacement[$"{start}:{stop}"] =
rr.dot(displacement[$"{start}:{stop}"].T).T;
}
```

The x and y are the location for which the SDR is to be calculated. A 1x2 NDArray (location) is created from the x and y coordinate. As resulted the "offsets" which is a (size x 2) NDArray can be easily subtracted from the location.

The partition groups the displacement as per the number of grid cell modules. For example, for a SDR of size 100 and 5 grid cell modules, there will be 100 displacements and 5 partitions in the range of 0-20, 20-40, 40-60, 60-80 and 80-100. So, displacement from the range 0-20 belongs to the first grid cell module, displacement from the range 20-40 belongs to the second grid cell module and so on. So the appropriate rotation is applied to the each cells in the particulate grid cell module.

b) Step 2: Find radius of grid cell in each grid module

```
var radius = np.empty(size);
for (var i = 0; i < periods.Length; i++) {
    var start = partitions[i][0];
    var stop = partitions[i][1];

    radius[$"{start}:{stop}"] = periods[i] / 2;
}
```

For each grid cell module the radius is calculated which is the half of the period value of that module.

c) Step 3: Convert the displacement into and out of hexagonal coordinates

```
NDArray nearestArray = np.ndarray((size, 2));
for (var i = 0; i < size; i++)
{
    var grid = new
HexagonalGrid(HexagonalGridType.PointyEven,
(float)radius[i].GetValue<Double>());

    var x = displacement[i][0].GetValue<Double>();
    var y = displacement[i][1].GetValue<Double>();

    var nearest = grid.ToPoint2(grid.ToCubic(x, y));
    nearestArray[i][0] = nearest.X;
    nearestArray[i][1] = nearest.Y;
}
```

Each of the displacement (x,y) is converted to cubic and back to (x,y) which rounds to the nearest hexagonal center.

d) Step 4: Calculate difference between the result of Step 3 with Step 1.

```
var nearestMinusDisplacement = nearestArray -
displacement;
```

e) Step 5: Find the distance between the result from Step 4 and the hexagonal center

```
var distances = new double[size];
for (var i = 0; i < size; i++)
{
    var x = nearestMinusDisplacement[i][0];
    var y = nearestMinusDisplacement[i][1];
    var hypot = Math.Sqrt(Math.Pow(x, 2) +
Math.Pow(y, 2));

    distances[i] = hypot;
}
```

The hypotenuse is calculated ($h = \sqrt{p^2 + b^2}$) where h gives the distance from the point to the center of the hexagon and p and b is the value from the Step 4.

f) Step 6: Get SDR by activating the closest grid cells in each module based on sparsity

```
var activatedCells = new List<int>();
foreach (var partition in partitions)
{
    var start = partition[0];
    var stop = partition[1];
    var z = (int) (Math.Round(sparsity * (stop -
start)));


    var indexes = distances[start..stop]
    .Select((x, i) => new KeyValuePair<double,
int>(x, i))
    .OrderBy(x => x.Key)
    .Take(z)
    .Select((x, i) => x.Value + start)
    .ToList();

    activatedCells.AddRange(indexes);
}

activatedCells.Sort();
```

The "start" and "stop" variable gives the range to get distances (from step 5) for each module which is then sorted in ascending order of value and the first "z" number of distances which is calculated based on sparsity is taken. The index of these selected distances is the active index in the SDR.

III.RESULTS

For this experiment, we take three input coordinates and generate the resulting SDR. We also find the similarity matrix to find the similarity between these computed SDR. The size of the SDR is 100 for this experiment.

A. SDR Output

```
var coordinates = new double[,] {
    {100, 100},
    {100, 100.50},
    {5000, 400}
};
```

a) 100,100

The SDR for the coordinate (100,100) is as follow:

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

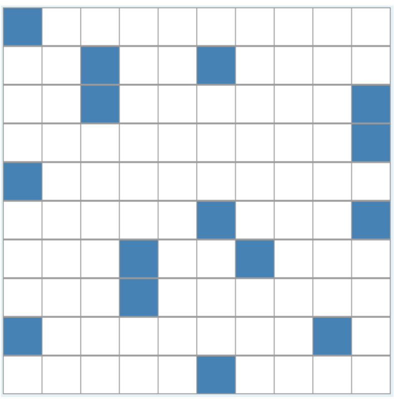

Figure X: SDR visualization for coordinate 100,100

b) 100,100.50

The SDR for the coordinate (100,100.50) is as follow:

1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0

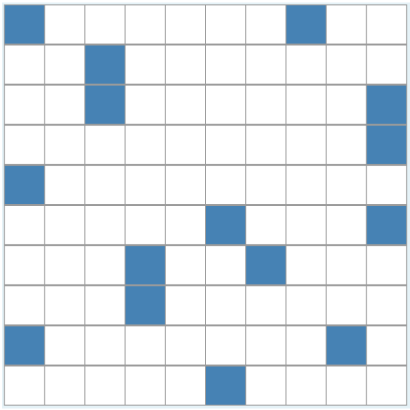

Figure XX: SDR visualization for coordinate 100,100.5

a) 5000,400

The SDR for the coordinate (100,100.50) is as follow:

1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

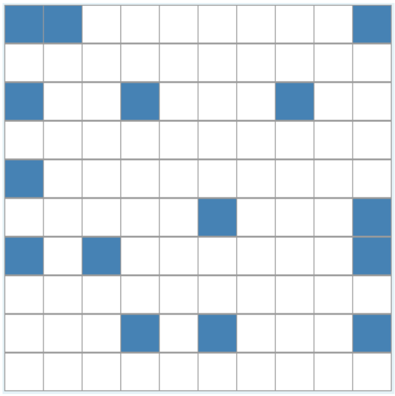

Figure XX: SDR visualization for coordinate 5000, 400

B. Similarity Matrix

Coordinates (x,y)	100 x 100	100 x 100.5	5000 x 400
100 x 100	100	93.33	26.67
100 x 100.5	93.33	100	26.67
5000 x 400	26.67	26.67	100

From the similarity matrix of the resulting SDR for three coordinates in this experiment we can observer that the similarity for the coordinate 100x100 and 100x100.5 is quite similar at 93.33%.

The similarity of the SDR 5000x400 with respect to the other two coordinates is at 26.67%.

IV. CONCLUSION

This paper gives an overview about Grid cells and also present an algorithm for the Grid Cell Encoder to produce unique SDR to represent a location in space with the use of multiple grid cell modules which are aligned in different orientation and offset.

Also, with the help of similarity matrix table it can be seen that for the coordinates that are very close to each other the grid cell encoder produces similar but different SDR.

REFERENCES

1. *How grid cells map space* (2021) *Numenta*. Available at: https://www.numenta.com/blog/2018/05/25/how-grid-cells-map-space/ (Accessed: November 21, 2022).
2. Authors Dori Derdikman *et al.* (no date) *How do we find our way? grid cells in the brain*, *Frontiers for Young Minds*. Available at: https://kids.frontiersin.org/articles/10.3389/frym.2021.678725 (Accessed: November 21, 2022).
3. Dmac, Thanh-Binh.to and Breznak (2019) *Grid cell encoder*, *HTM Forum*. Available at: https://discourse.numenta.org/t/grid-cell-encoder/5796 (Accessed: November 21, 2022).
4. Htm-Community (no date) HTM-community/htm.core: Actively developed Hierarchical Temporal Memory (HTM) community fork (continuation) of nupic. implementation for C++ and python, GitHub. Available at: https://github.com/htm-community/htm.core (Accessed: November 21, 2022).
5. (2017) *YouTube*. YouTube. Available at: https://www.youtube.com/watch?v=dgN5j16scj4 (Accessed: November 22, 2022).
6. (2018) *YouTube*. YouTube. Available at: https://www.youtube.com/watch?v=CQPswbIuCkk (Accessed: November 22, 2022).
7. Guanella, Alexis & Kiper, Daniel & Verschure, Paul. (2007). A model of grid cells based on a twisted torus topology. International journal of neural systems. 17. 231-40. 10.1142/S0129065707001093.
8. (2018) *YouTube*. YouTube. Available at: https://www.youtube.com/watch?v=mP7neeymcUY (Accessed: November 22, 2022).
9. DiogoSantosPata (no date) *Diogosantospata/gridcells: Implementation of grid cells in python and Javascript*, *GitHub*. Available at: https://github.com/DiogoSantosPata/gridcells (Accessed: November 22, 2022).