

Information Technology Course

Module Software Engineering

by Damir Dobric / Andreas Pech

Inhibition of Spatial Pooler

Trung Luu Nghiem Bao

***Abstract***

***Columns Inhibition is an important step in spatial pooling algorithm. The inhibition calculates and decides columns, which would be set as “active” to represent for the input in Sparse Distributed Representation. It helps to reduce the number of resources needed to represent the input signal. The inhibition can be implemented globally on whole sparse distributed space or locally within their neighborhood to choose number of columns to be activate.***

***In this project, the different scenarios are set up to compare the result of active columns after the inhibition using local inhibition. The project is built and implemented with .NET core 2.0 framework.***

# INTRODUCTION

Spatial pooling represents the information of input space in sparse space. This is an important algorithm because the learning and predicting mechanism is based on the sparse distribution. The spatial pooling includes three different steps [1]:

* Overlaps calculation.
* Columns Inhibition.
* Update internal variables and synapse permanence.

Column inhibition is the main task in spatial pooling to decide the active columns, which would be used to represent input information.

Columns Inhibition divided into two approaches. In Global inhibition the fixed number of columns, which have highest value of overlaps would be chosen as “active”, and others would be “inactive” in the whole region of sparse space. On the other hand, in local inhibition, it is more complicate to decide the “active” columns, the local Inhibition depends much on initial parameters because the distribution of overlaps for each neighborhood is not the same. In following chapter, the implementation of local inhibition with different topologies and input sizes is introduced to analyze the variation of output after local inhibition.

# IMPLEMENTATION

To analyze the affect of different parameters on Columns Inhibition, an image of “digit 7” is used:



Figure 1: Original Image

Besides, some default parameters is set based on practical experiments:

DUTY\_CYCLE\_PERIOD = 100

MAX\_BOOST = 10.0

Where “DUTY\_CYCLE\_PERIOD” is the period to calculate duty cycle. Higher period increases the response time of changes in boosting, while lower period create unstable in learning speed of spatial pooling [2].

“MAX\_BOOST” is used to increase the overlap values of columns. “MAX\_BOOST” is used when duty cycle is equal to 0.

The data is prepared with 3 different sizes from original image 32x32, 64x64 and 128x128 in combination with 5 topologies 10x10, 20x20, 30x30, 60x60 and 100x100. The binarized images of original picture is used in the test.

The process is described in pseudo code:

*binarizedImage(originalImage,newWidth,newLength);*

*For each(binaryImage in binaryImages){*

*For each(topology in topologies){*

*var activeInput = ReadCsvFile(path\_to\_binary\_image);*

*While(i < iteration){*

*activeColumns = Compute(activeInput);*

*getHammingDistance(activeColmns[k],activeColum ns[k-1]);*

*}*

*drawBitmap(activeColumns);*

*}*

*}*

# RESULT

The result of each size of image is shown:

Figure 2: Speed of Spatial Pooling with different Topology of 32bit image

Figure 3:Speed of Spatial Pooling with different Topology of 64bit image

Figure 4:Speed of Spatial Pooling with different Topology of 128bit image

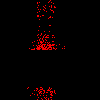
The graphs show that independently from input sizes, the first stable state remains until iteration 52, then the boost factor affects to the overlaps of columns after each iteration and creates oscillation in calculating active columns in SDR space. On the other hand, the oscillation iterations of each image sizes are different. The result is summarized in the table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Image | Topology | 1st Stable State  (from-to) | Oscillation Iteration  (from-to) | 2nd Stable State  (from-to) |
| 32-bit | 10x10 | 2-51 | 52-82 | 83-end |
| 20x20 | 3-51 | 52-81 | 82-end |
| 30x30 | 3-51 | 52-79 | 80-end |
| 60x60 | 3-51 | 52-88 | 89-end |
| 100x100 | 3-51 | 52-88 | 89-end |
| 64-bit | 10x10 | 2-51 | 52-63 | 64-end |
| 20x20 | 2-51 | 52-62 | 63-end |
| 30x30 | 2-51 | 52-63 | 64-end |
| 60x60 | 3-51 | 52-63 | 64-end |
| 100x100 | 3-51 | 52-65 | 66-end |
| 128-bit | 10x10 | 2-51 | 52-55 | 56-end |
| 20x20 | 2-51 | 52-57 | 58-end |
| 30x30 | 2-51 | 52-56 | 57-end |
| 60x60 | 3-51 | 52-57 | 58-end |
| 100x100 | 3-51 | 52-57 | 58-end |

The increasing in size of input image decrease the unstable iterations. It states that the speed of learning in spatial pooler depends on the size of an input vector.

Moreover, the result of sparse distributed representation of different input and topologies are display:

100x100



10x10

60x60

30x30

20x20



Figure 5: Sparse Distributed Representations of 32bit image



100x100

30x30

10x10

20x20

60x60



Figure 6: Sparse Distributed Representations of 64bit image

100x100



60x60

30x30

20x20

10x10

Figure 7: Sparse Distributed Representations of 128bit image

Independently from topology, the sparsity (percentage of active columns) in different sizes of input image is different. The sparsity is independent from the topology applied for representing the input image. The sparsity of active columns is approximately 2%, 3% and 28% for 32-bit, 64-bit and 128-bit image, respectively. The changes in sparsity is affected by the inhibition-radius parameter.

# CONCLUSION

After running the local inhibition with different input sizes and topologies, it is shown in the result that the sparsity and stability of inhibition is less depends on topology than on input size. With same input size and different topologies, the stability and sparsity vary not much. However, the topology affects the time of learning. In local inhibition, the process goes through all the columns and its neighborhood to find active columns and for larger topology, it needs more times to go through all the columns.

For future development, new approaches of local inhibition would be implemented to reduce the workload of current algorithm.

# REFERENCES

|  |  |
| --- | --- |
| [1] | J. Hawkins, "Hierarchical Temporal Memory (HTM) whitepaper," Numenta, Inc., 2011. |
| [2] | J. e. a. Hawkins, "Biological and Machine Intelligence (BAMI)," 2016. [Online]. Available: https://numenta.com/resources/biological-and-machine-intelligence. |