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

Performance Spatial Pooler between Global and Local Inhibition

Tran Quang Trung
tranquangtrung.vgu@gmail.com

Nguyen Thanh Quang
thanhquang0912@gmail.com

***Abstract* - Each region in the cortex receives input through millions of axons from sensory organs and from other cortical regions. It remains a mystery how cortical neurons learn to form specific connections from this large number of unlabeled inputs in order to support further computations. Hierarchical temporal memory (HTM) provides a theoretical framework for understanding the computational principles in the neo-cortex. HTM spatial pooler was created to model how neurons learn feed forward connections. The spatial pooler method is converting the arbitrary binary input patterns into sparse distributed representations (SDRs) using competitive Hebbian learning's rules and homeostasis excitability control mechanisms. In this paper, one of the Spatial Pooler's key parameters, which is the "inhibition", will be described. The main part is to show the differences between the "local" and "global" inhibition and how and what kind of affects they contribute to the process to the Spatial Pooler learning algorithm.**

Keywords—hierarchical temporal memory, spatial pooler, global inhibition, local inhibition, LearningAPI, .NET Core

I. INTRODUCTION

The brain continuously receives vast amounts of information about the external world through peripheral sensors that transform changes in light luminance, sound pressure, and skin deformations into millions of spike trains. Each cortical neuron has to make sense of a flood of time-varying inputs by forming synaptic connections to a subset of the presynaptic neurons. The collective activation pattern of population of neurons contributes to our perception and behavior. A central problem in neuroscience is to understand how individual cortical neurons learn to respond to specific input spike patterns, and how a population of neurons collectively represents features of the inputs in a flexible, dynamic, yet robust way.

The HTM spatial pooler is inspired by working principles of the neo-cortex. It relies on competitive Hebbian learning (Hebb, 1949), homeostasis excitatory control mechanisms (Davis, 2006), topology of connections in sensory cortices (Udin and Fawcett, 1988; Kaas, 1997) and other physiological and anatomical properties of the neo-cortex. Unlike previous sparse coding studies, the HTM spatial pooler is not designed to minimize a single objective function of reconstruction error, but rather to achieve a set of computational properties that support further computation with SDRs. In addition to preserving the semantic similarity of the input space, the SP continuously adapts to statistics of the sensory input streams to ensure that the resulting representations are both efficient and noise robust. The sparsity of the SP output is tightly controlled around a target level. This stability of output sparsity reduces the recognition errors of downstream neurons.

There are many parameters settings that can be contributed to the learning progress of the HTM spatial pooler and the result from the learning process. The goal of this paper is to mention to the settings of the parameter "inhibition" and its relative parameter which would effect to the HTM spatial pooler learning process.

II. MATERIALS AND METHODS

A. HTM Spatial Pooler Overview

Firstly, this part will cover the basic algorithm of HTM Spatial Pooler. The spatial pooler converts inputs with arbitrary sparsity to SDRs with fixed sparsity and dimensionality. The spatial pooler consists of a set of mini-columns. Cells in the same mini-column share the same feed forward receptive fields (Buxhoeveden, 2002).

The overall Spatial Pooler can be summarized to the following steps:

1. Start with an input consisting of a fixed number of bits. These bits might represent sensory data or they might come from another region elsewhere in the HTM system.

2. Initialize the HTM region by assigning a fixed number of columns to the region receiving this input. Each column has an associated dendritic segment, serving as the connection to the input space. Each dendrite segment has a set of potential synapses representing a (random) subset of the input bits. Each potential synapse has a permanence value. These values are randomly initialized around the permanence threshold. Based on their permanence values, some of the potential synapses will already be connected; the permanence are greater than than the threshold value.

3. For any given input, determine how many connected synapses on each column are connected to active (ON) input bits. These are active synapses.

4. The number of active synapses is multiplied by a "boosting" factor, which is dynamically determined by how often a column is active relative to its neighbors.

5. A small percentage of columns within the inhibition radius with the highest activations (after boosting) become active, and disable the other columns within the radius. The inhibition radius is itself dynamically determined by the spread of input bits. There is now a sparse set of active columns.

6. The region now follows the Spatial Pooling (Hebbian-style) learning rule: For each of the active columns, we adjust the permanence values of all the potential synapses. The permanence values of synapses aligned with active input bits are increased. The permanence values of synapses aligned with inactive input bits are decreased. The changes made to permanence values may change some synapses from being connected to unconnected, and vice-versa.

7. For subsequent inputs, repeating step 3.

B. Spatial Pooler Coding Phase Progress

This section contains the detail coding progress for Spatial Pooler function, broken down into four phases: initialization, overlap, computation, and learning. After initialization (phase 1), every iteration of the Spatial Pooling algorithm's compute routine goes through three distinct phase (phase 2 through phase 4) that occur in sequence.

Phase1- Initialize Spatial Pooling algorithm parameters

Prior to receiving any inputs, the Spatial Pooling algorithm is initialized by computing a list of initial potential synapses for each column. This consists of a random set of inputs selected from the input space (within a column's inhibition radius). Each input is represented by a synapse and assigned a random permanence value. The random permanence values are chosen with two criteria. First, the values are chosen to be in a small range around connectedPerm, the minimum permanence value at which a synapse is considered "connected". This enables potential synapses to become connected (or disconnected) after a small number of training iterations. Second, each column has a natural center over the input region, and the permanence values have a bias towards this center, so that they have higher values near the center.

Phase 2 – Compute the overlap with the current input for each column

Given an input vector, this phase calculates the overlap of each column with that vector. The overlap for each column is simply the number of connected synapses with active inputs, multiplied by the column's boost factor.

Phase 3 – Compute the winning columns after inhibition

The third phase calculates which columns remain as winners after the inhibition step. localAreaDensity is a parameter that controls the desired density of active columns within a local inhibition area. Alternatively, the density can be controlled by parameter numActiveColumnsPerInhArea. When using this method, the localAreaDensity parameter must be less than 0. The inhibition logic will ensure that at most numActiveColumnsPerInhArea columns become active in each local inhibition area. For example, if numActiveColumnsPerInhArea is 10, a column will be a winner if it has a non-zero overlap and its overlap score ranks 10th or higher among the columns within its inhibition radius.

Phase 4 – Update synapse permanences and interval variables

This final phase performs learning, updating the permanence values of all synapses as necessary, as well as the boost values and inhibition radius. For winning columns, if a synapse is active, its permanence value is incremented, otherwise it is decremented; permanence values are constrained to be between 0 and 1. Notice that permanence values on synapses of non-winning columns are not modified.

III. ALGORITHM PARAMETERS

As mentioned from the start of the paper, the Spatial Pooler algorithm has many parameters that affect the dimensionality, learning process and overall performance. Here some key parameters will be discussed in detail, and how various values influence Spatial Pooling.

A. Spatial Pooling Algorithm Structure

The column dimensions parameter (columnDimensions) as specified within the Spatial Pooling set of rules parameters define the size for the HTM region. If there are allocated 4096 columns, the vicinity is an array of 4096 columns. However, for a vision system, the equal quantity of columns may be used as a two-dimensional array, so the place's columnar structure would be 64x64. A three-dimensional topology may be specified. While the column dimensions manipulate the output shape, the inputs to the columns are controlled by way of specifying the Spatial Pooling set of rules's input parameters. The input dimensions (inputDimensions) are specified much like the column dimensions and the number of dimensions must match. The input area that a column can doubtlessly join to—i.E., the receptive discipline of the column—is managed with the capacity radius (potentialRadius) parameter. This price will determine the unfold of a column's influence throughout the HTM layer. A small ability radius will maintain a column's receptive discipline local, at the same time as a very large ability radius will give the column global coverage over the input area.

B. Inhibition

The mechanism for maintaining sparse activations of neurons. In the Spatial Pooler this manifests as columns inhibiting nearby columns from becoming active. There are two properties of the inhibition in Spatial Pooler learning process that would be explained and showed the differences about them.

- Global Inhibition: with this property, the most active columns are selected from the entire layer.
- Local Inhibition: the winning columns are selected with respect to columns' local neighborhoods

One of the core properties of the Spatial Pooler seems to be that it creates fixed-sparsity representations. With the global inhibition the Spatial Pooler undoubtedly fulfills this property as it simply achieves s% sparsity by activating the columns with the s% highest input overlaps.

A local inhibition mechanism ensures that small fractions of the Spatial Pooler mini-column that receive most of the inputs are active within the local inhibition radius. Synaptic permanencies are adjusted according to the Hebbian rule: for

each active Spatial Pooler mini-column, active inputs are reinforced and inactive inputs are punished. The local inhibition implements a k-winners-take-all computation. At any time, only a small fraction of the mini-columns with the most active inputs become active.

With global inhibition (globalInhibition=True), the most active columns are selected from the entire layer. Otherwise the winning columns are selected with respect to the columns' local neighborhoods. The former offers a significant performance boost, and is often what we use in practice. With global inhibition turned off, the columnar inhibition takes effect in local neighborhoods. Column neighborhoods are a function of the inhibition radius (inhibitionRadius), a dynamically calculated measure internal to the Spatial Pooling algorithm that is a function of the average size of the connected receptive fields of all columns. The receptive field of columns can be controlled in part by the potential radius parameter above; it cannot be set explicitly because the receptive fields of Spatial Pooling algorithm columns (and HTM cells in general) are dynamic.

The former Global Inhibition offers a significant performance boost, and is often what we use in practice. With the global inhibition is off, the columnar inhibition takes effect in local neighborhoods. Column neighborhoods are a function of the inhibitions radius (inhibitionRadius), a dynamically calculated measure internal to the Spatial Pooling algorithm that is a function of the average size of the connected receptive fields of all columns. The receptive field of columns can be controlled in part by the potential radius parameter above; it cannot be set explicitly because the receptive fields of Spatial Pooling algorithm columns (and HTM cells in general) are dynamic.

IV. Testing Scenario and Result

In this section, the testing progress which used to compare between the effect of global and local inhibition to the learning process, will be described. All the results from this test was taken directly from the real Spatial Pooler learning process.

The input sources were used in the test is the mnistImage. The images below shown the mnistImage using for the testing.

Figure 1: Input source mnistImage

Before feeding the image input through the spatial pooler processing, the image have to be binarized. This task is done by using BinarizeImage() method.

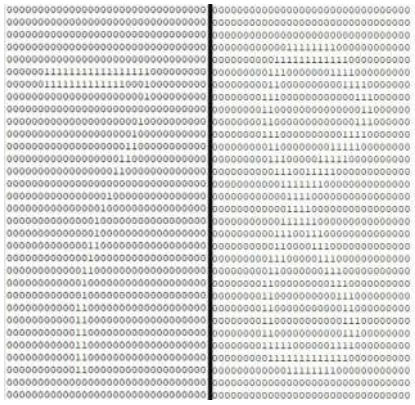

Figure 2: Input image after being binarized (number 7 and 8)

A. ***First test case (ExperimentInputSizeAndColumnDimension())*** – *input size/column dimension*

The scenario for this testing is to compare the performance speed of spatial pooler with different input size and column dimension. The other parameters are set as follow:

Parameter	***Value***
POTENTIAL_RADIUS	10
POTENTIAL_PCT	0,75
LOCAL_AREA_DENSITY	-1,0
STIMULUS_THRESHOLD	0,0
SYN_PERM_INACTIVE_DEC	0,01
SYN_PERM_ACTIVE_INC	0,1
SYN_PERM_CONNECTED	0,1
MIN_PCT_OVERLAP_DUTY_CYCLES	0,001
MIN_PCT_ACTIVE_DUTY_CYCLES	0,001
NUM_ACTIVE_COLUMNS_PER_INH_AREA	2% OF TOTAL COLUMN
DUTY_CYCLE_PERIOD	100
MAX_BOOST	10.0
IS_BUMPUP_WEAKCOLUMNS_DISABLED	true

The data is tested with 5 different input size 16x16, 32x32, 40x40, 48x48, 54x54 and 64x64. Each of the input value is paired with 8 different column dimension values: 16, 32, 40, 48, 54, 64, 72 and 80. The iteration value for this test is 300.

The Test Method for this test case is: *PerformanceOfSpatialLearningTest()* . For each pair of input size and column dimension this test method performs both

Local inhibition algorithm and Global inhibition. After that, the output files are created, including result of performance speed, hamming distance, final active column as well as bitmap image of this active column array. The effect of input image size and column dimension on the speed of learning process is shown:

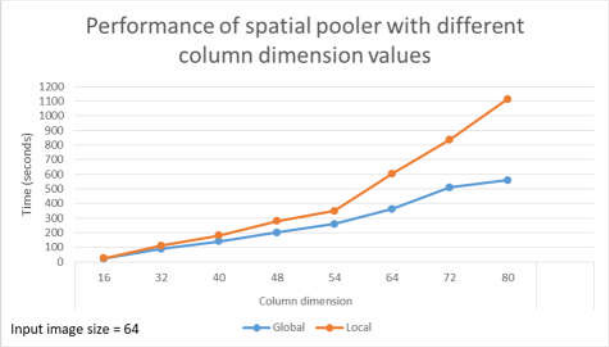

Figure 3: Performance of spatial pooler with different column dimension of 64bit image with max_boost =10 and duty_cycle_period = 1000. Input image size is 64

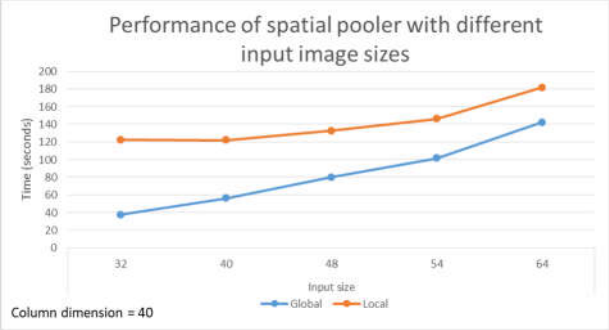

Figure 4: Performance of spatial pooler with different input image size with fixed column dimension = 40, max_boost = 10 and duty_cycle_period = 1000

The above graphs demonstrate that the higher the values of input size and column dimension, the longer the spatial pooler need to finish the work. In case of using local inhibition algorithm, column dimension has greater impact on completion duration of spatial pooler. On the other hand, the impact of input vector size is mostly similar with both global inhibition and local inhibition. Below is the summarized result table for all case in this test:

Input Image	*Column Dimension*	*Global Inhibition (seconds)*	*Local Inhibition (seconds)*
64x64	16	23	25,44
	32	88,58	111,3
	40	141,96	181,56
	48	202.06	280,74
	54	259,9	348,85
	64	362,39	605,1
	72	510,44	837,76
	80	560,77	1115,96

Figure 5: Impact of column dimension on speed of learning

Column Dimension	*Input size*	*Global Inhibition (seconds)*	*Local Inhibition (seconds)*
40	32x32	37,09	122,21
	40x40	56,03	121,75
	48x48	79,81	132,31
	54x54	101,22	145,89
	64x64	141,96	181,56

Figure 6: Impact of input image size on speed of learning

B. ***Second test case (ExperimentMaxBoostAndDutyCyclePeriod())*** *– max boost and duty cycle period*

The scenario for this testing is to figure out if "MAX_BOOST" and "DUTY_CYCLE_PERIOD" parameters have effects on the performance of spatial pooler.

"MAX_BOOST" is a maximum number of boost factor. This parameter is used to boost the overlap values of columns. This is useful because it helps inactive columns have a chance to "express" themselves and become more competitive with other columns. Each column's overlap gets multiplied by a boost factor before it gets considered for inhibition. The actual boost factor for a column is number between 1.0 and "max boost".

" DUTY_CYCLE_PERIOD" is a period constant used to calculate new duty cycles. Higher values make it take longer to respond to changes in boost process. Lower values make it more unstable and likely to oscillate. In other words, it is the parameter that decides how long before the Spatial Pooler columns start changing due to boosting algorithms. When the Spatial Pooler learns it moves the winning columns points closer to the input point. There is no correct value for this parameter. It depends on how the purpose of the Spatial Pooler and the input sources.

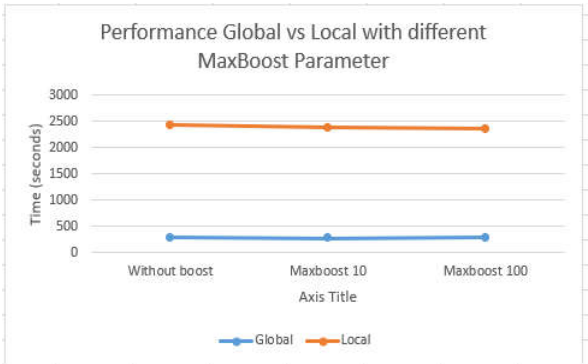

Figure 7: Performance of spatial pooler with different values of Max Boost

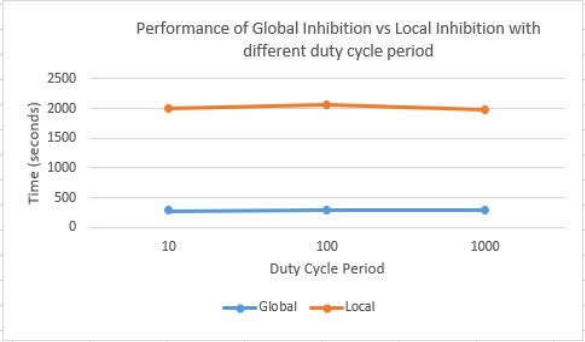

Figure 8: Performance of spatial pooler with different values of Duty Cycle Period

From figure 4 and figure 5, the learning speed of spatial pooler is not affected by values of parameter "MAX_BOOST" and "DUTY_CYCLE_PERIOD". The scenario has been done in other pairs of input size / column dimension, all results show that the speed of spatial pooler learning vary not much with different "MAX_BOOST" and "DUTY_CYCLE_PERIOD" parameter.

In addition, the result of sparse distributed representation of different *max_boost* and *duty_cycle_period* is shown below:

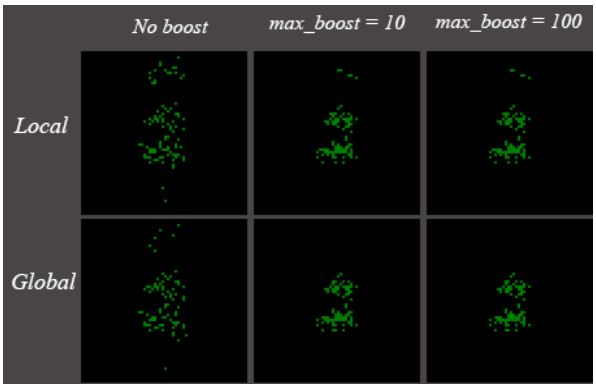

Figure 9: Sparse Distributed Representations of 32bit image input with column dimension = 64, with and without max_boost value, for local inhibition and global inhibition algorithm.

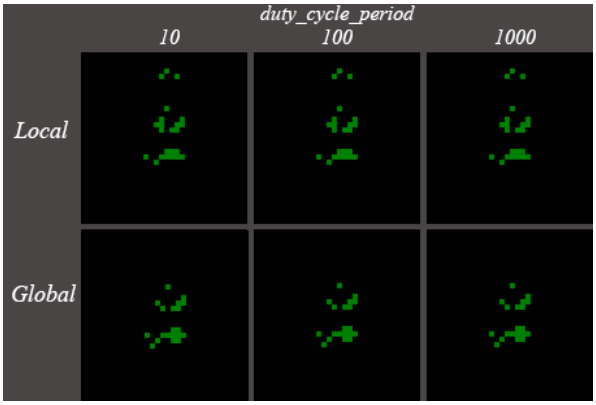

Figure 10: Sparse Distributed Representations of 32bit image input with column dimension = 32, max_boost = 100, with different value of duty_cycle_period, for local inhibition and global inhibition algorithm.

From the images above, it is clear that local inhibition algorithm sparse distributed representation gives more detail than global inhibition algorithm result. Moreover, with *max_boost* parameter enables, the active columns tend to form into smaller cluster.

C. *Third test case (ExperimentPotentialRadius()) – potential radius*

Potential_radius is the parameter which determines the extent of the input that each column can potentially be connected to. This can be thought of as input bits that are visible to each column, or a 'receptive field' of the field of vision. A large enough value will result in global coverage, meaning that each column can potentially be connected to every input bit.

Input Image	*Potential radius*	*Time (seconds)*
32x32	4	777,91
	8	1693,98
	10	2082,91
	16	2731,69
64x64	8	1871,26
	16	2774,01
	24	3743,97
	32	5015,19

Figure 11: Impact of potentialRadius on speed of learning

From the result table, depending on the input image size, with the suitable potentialRadius value, the Spatial Pooler will result the better period – faster learning process. But with the large potentialRadius, it will results the unstable and the Spatial Pooler learning will be broken and give the wrong SDRs, which shown in the Figure 13. below.

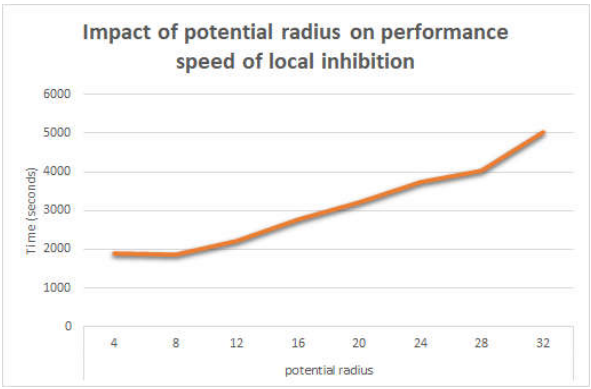

Figure 12: Impact of potential_radius parameter on performance speed of local inhibition algorithms.

Moreover, the number active columns in SDR space are not equal to the default 2% of total column. It indicates that the parameter POTENTIAL_RADIUS has significant impact on the sparsity (percent of active columns) of Spatial Pooling as seen below:

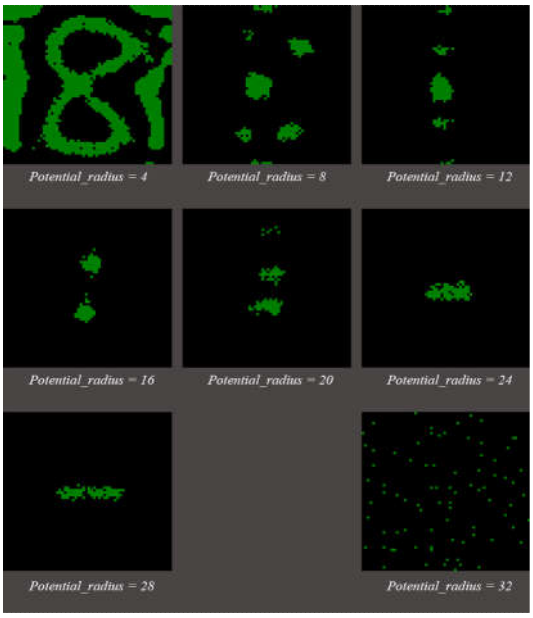

Figure 13: Sparse Distributed Representations of 64bit image with different potential_radius parameter. From potential_radius with value equal or bigger than half of input size, the output SDRs is broken with active columns scatter all over the space.

V. Discussion and Conclusion

In this paper, we compare the performance speed between two spatial pooler inhibition algorithms: local inhibition and global inhibition. From the test results, it is shown that:

- Completion duration of each algorithm is positively correlated with the magnitudes of two parameters input image size and column dimension.
- Column dimension has greater impact on performance speed of spatial pooler when using local inhibition algorithms. In local inhibition, each column must be compared to other columns in its neighborhoods, which significantly increases the time needed for scenario with high value of column dimension.
- *Max_boost* and *duty_cycle_period* have no impact on the computing speed of both global inhibition and local inhibition algorithms.
- The lower the *potential_radius* value, the faster the performance speed of local inhibition algorithms, the higher the sparsity representations.

As a conclusion, using local inhibition algorithms results in slower performance speed of spatial pooler, especially in scenario with large value column dimension. Besides, the spatial pooler does not guarantee fixed sparsity SDRs when using local inhibition algorithms. However, local inhibition still plays an important part in hierarchical temporal memory theory and should not be neglected.

References

[1] Ahmad, S., and Hawkins, J. (2015). Properties of sparse distributed representations and their application to hierarchical temporal memory

[2] Yuwei Cui, Subutai Admed, and Jeff Hawkons (2017) The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding.

[3] Hawkins, J. et al. 2016. Biological and Machine Intelligence. Release 0.4. Accessed at http://numenta.com/biological-and-machine-intelligence/.

[4] Damir Dobric (2019), Hierachical Temporary Memory implementation in C#/.NET Core. Accessed at https://github.com/ddobric/neocortexapi