Spatial Pooling Algorithm

Chapter Revision History

The table notes major changes between revisions. Minor changes such as small clarifications or formatting changes are not noted.

Version	Date	Changes	Principal Author(s)
0.4		Initial release	S. Ahmad
0.5	Feb 2017	Update to current algorithms	M. Taylor & Y. Cui
0.56	Jan 2024	Corrected description of boostFunction(c)	C. Lai

Important Note to Readers:

The following text gives details of the Spatial Pooling algorithm, including pseudocode and parameters. We highly recommend that you access some of the other Spatial Pooling resources available in order to understand the high-level concepts and role of Spatial Pooling in biology, and in HTM. You can find links to the latest Spatial Pooling resources in the Spatial Pooling chapter.

Spatial Pooling Algorithm Details

We first present some important terms, then the high-level steps, followed by details with pseudocode.

Terminology

- Column: An HTM region is organized in columns of cells. The SP operates at the column-level, where a column of a cells function as a single computational unit.
- Mini-column: See “Column”
- Inhibition: The mechanism for maintaining sparse activations of neurons. In the SP this manifests as columns inhibiting nearby columns from becoming active.
- Inhibition radius: The size of a column’s local neighborhood, within which columns may inhibit each other from becoming active.
- Active duty cycle: A moving average denoting the frequency of column activation.
- Overlap duty cycle: A moving average denoting the frequency of the column’s overlap value being at least equal to the proximal segment activation threshold.
- Receptive field: The input space that a column can potentially connect to.
- Permanence value: indicates the amount of growth between a mini-column in the Spatial Pooling algorithm and one of the cells in its receptive field
- Permanence threshold: If a synapse’s permanence is above this value, it is considered fully connected. Acceptable values are [0,1].
- Synapse: A junction between cells. In the Spatial Pooling algorithm, synapses on a column’s dendritic segment connect to bits in the input space. A synapse can be in the following states:
    - Connected—permanence is above the threshold.
    - Potential—permanence is below the threshold.
    - Unconnected—does not have the ability to connect.

Spatial Pooling algorithm 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 permanences 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, we repeat from step 3.

Spatial Pooling Pseudocode

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

The various data structures and supporting routines used in the code are defined in Table X at the end.

Phase 1 – 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.

```
1. for c in columns
2.     overlap(c) = 0
3.     for s in connectedSynapses(c)
4.         overlap(c) = overlap(c) + input(t, s.sourceInput)
5.     overlap(c) = overlap(c) * boost(c)
```

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.

```
6.  for c in columns
7.      minLocalActivity = kthScore(neighbors(c), numActiveColumnsPerInhArea)
8.      if overlap(c) > stimulusThreshold and
9.         overlap(c) ≥ minLocalActivity then
10.         activeColumns(t).append(c)
```

Phase 4 – Update synapse permanences and internal variables

This final phase performs learning, updating the permanence values of all synapses as necessary, as well as the boost values and inhibition radii. The main learning rule is implemented in lines 14-20. 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.

Lines 21-27 implement boosting. There are two separate mechanisms in place to help a column learn connections. If a column does not win often enough (as measured by activeDutyCycle) compared to its neighbors, its overall boost value is set to be greater than 1 (line 22-23). If a column is active more frequently than its neighbors, its overall boost value is set to be less than one. The boostFunction is an exponential function that depends on the difference between the active duty cycle of a column and the average active duty cycles of its neighbors. If a column's connected synapses do not overlap well with any inputs often enough (as measured by overlapDutyCycle), its permanence values are boosted (line 24-27). Note that once learning is turned off, boost(c) is frozen.

Finally, at the end of Phase 4 the inhibition radius is recomputed (line 28).

```
11. for c in activeColumns(t)
12.     for s in potentialSynapses(c)
13.         if active(s) then
14.             s.permanence += synPermActiveInc
15.             s.permanence = min(1.0, s.permanence)
16.         else
17.             s.permanence -= synPermInactiveDec
18.             s.permanence = max(0.0, s.permanence)
19. for c in columns:
20.     activeDutyCycle(c) = updateActiveDutyCycle(c)
21.     activeDutyCycleNeighbors = mean(activeDutyCycle(neighbors(c))
22.     boost(c) = boostFunction(activeDutyCycle(c), activeDutyCycleNeighbors)
23.     overlapDutyCycle(c) = updateOverlapDutyCycle(c)
24.     if overlapDutyCycle(c) < minDutyCycle(c) then
25.         increasePermanences(c, 0.1*connectedPerm)
26. inhibitionRadius = averageReceptiveFieldSize()
```

Algorithm Parameters

The Spatial Pooling algorithm has many parameters that affect the dimensionality, learning mechanisms, and overall performance. Here we discuss some parameters in detail, and how various values influence Spatial Pooling. The roles of the parameters were previously described in the pseudocode, and a full list of the algorithm parameters, data structures, and routines can be found in Tables 1 and 2 below.

Spatial Pooling Algorithm Structure

The column dimensions (columnDimensions) as specified in the Spatial Pooling algorithm parameters define the dimensions for the HTM region. If we allocate 4096 columns, the region is an array of 4096 columns. However, for a vision system, the same

number of columns could be used as a two-dimensional array, so the region's columnar structure would be 64x64. We can also specify a three-dimensional topology.

While the column dimensions control the output shape, the inputs to the columns are controlled by specifying the Spatial Pooling algorithm's input parameters. The input dimensions (inputDimensions) are specified just like the column dimensions, and the number of dimensions must match. The input space that a column can potentially connect to—i.e., the receptive field of the column—is controlled with the potential radius (potentialRadius) parameter. This value will determine the spread of a column's influence across the HTM layer. A small potential radius will keep a column's receptive field local, while a very large potential radius will give the column global coverage over the input space.

Inhibition

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.

We can however specify the density of active columns, with either localAreaDensity or numActiveColumnsPerInhArea. During inhibition these parameters will be used to calculate the maximum number of columns to remain ON within a local inhibition area. With the former parameter, you specify a density, so the actual number of active columns will change as columns' change the sizes of their receptive fields. Using the latter parameter, the density will fluctuate as the receptive fields change, but the max number of active columns remains fixed.

Learning

The learning rate can be specified with the synapse permanence increment and decrement amounts – typically the former is larger than the latter.

Column Activity

Although a column may win out in competition, its activation must be greater than a threshold (stimulusThreshold) in order to become active. The intent here is to prevent noise from activating columns, which is helpful towards the spatial pooling goal of avoiding trivial patterns.

Boosting can be helpful in driving columns to compete for activation. Boosting is monitored by both the activity and overlap duty cycles (activeDutyCycle(c) and overlapDutyCycle(c), respectively). Following inhibition, if a column's active duty cycle falls below the active duty cycles of neighboring columns, then its internal boost factor (boost(c)) will increase above one. If a column's active duty cycle arises above the active duty cycles of neighboring columns, its boost factor will decrease below one This helps drive the competition amongst columns and achieve the spatial pooling goal of using all the columns. Before inhibition, if a column's overlap duty cycle is below its minimum acceptable value (calculated dynamically as a function of minPctOverlapDutyCycle and the overlap duty cycle of neighboring columns), then all its permanence values are boosted by the increment amount. A subpar duty cycle implies either a column's previously learned inputs are no longer ever active, or the vast majority of them have been "hijacked" by other columns. By raising all synapse permanences in response to a subpar duty cycle before inhibition, we enable a column to search for new inputs.

Parameters, Data Structures, and Routines

The following tables summarize the Spatial Pooling algorithm data structures, routines, and parameters, including recommended parameter settings for typical use cases.

Columns	List of all columns.
columnCount	The total number of columns in the Spatial Pooling algorithm, and the HTM region. *This is task dependent but we recommend a minimum value of 2048.*
input(t,j)	The input to this level at time t. input(t, j) is 1 if the j'th input is on.
overlap(*c*)	The Spatial Pooling algorithm overlap of column c with a particular input pattern.
activeColumns(*t*)	List of column indices that are winners due to bottom-up input.
numActiveColumnsPerInhArea	A parameter controlling the number of columns that will be winners after the inhibition step. *We usually set this to be 2% of the expected inhibition radius. For 2048 columns and global inhibition, this is set to 40. We recommend a minimum value of 25.*
inhibitionRadius	Average connected receptive field size of the columns.
neighbors(c)	A list of all the columns that are within inhibitionRadius of column c.
stimulusThreshold	A minimum number of inputs that must be active for a column to be considered during the inhibition step. *This is roughly the background noise level expected out of the encoder and is often set to a very low value (0 to 5). The system is not very sensitive to this parameter; set to 0 if unsure.*
boost(c)	The boost value for column c as computed during learning – used to increase the overlap value for inactive columns.
boostStrength	A number greater or equal than 0.0 to control the strength of boosting. No boosting is applied if boostStrength=0.
synapse	A data structure representing a synapse, containing a permanence value and the source input index.
potentialPct	The percent of the inputs, within a column's potential radius, that are initialized to be in this column's potential synapses. *This should be set so that on average, at least 15-20 input bits are connected when the Spatial Pooling algorithm is initialized. For example, suppose the input to a column typically contains 40 ON bits and that permanences are initialized such that 50% of the synapses are initially connected. In this case you will want potentialPct to be at least 0.75 since 40*0.5*0.75 = 15.*
connectedPerm	If the permanence value for a synapse is greater than this value, it is said to be connected. *This is usually set to 0.2. The Spatial Pooling algorithm is not sensitive to this parameter.*
potentialSynapses(c)	The list of potential synapses and their permanence values for this column.

connectedSynapses(c)	A subset of potentialSynapses(c) where the permanence value is greater than connectedPerm. These are the bottom-up inputs that are currently connected to column c.
synPermActiveInc	Amount permanence values of active synapses are incremented during learning. *This parameter is somewhat data dependent. (The amount of noise in the data will determine the optimal ratio between synPermActiveInc and synPermInactiveDec.) Usually set to a small value, such as 0.03*
synPermInactiveDec	Amount permanence values of inactive synapses are decremented during learning. *Usually set to a value smaller than the increment, such as 0.015.*
activeDutyCycle(c)	A sliding average representing how often column c has been active after inhibition (e.g. over the last 1000 iterations).
overlapDutyCycle(c)	A sliding average representing how often column c has had significant overlap (i.e. greater than stimulusThreshold) with its inputs (e.g. over the last 1000 iterations).

Table 1. Variables and data structures used in the Spatial Pooling pseudocode and NuPIC implementation. For the parameters we include settings that work well under a wide range of scenarios (italicized above).

kthScore(cols, k)	Given the list of columns, return the k'th highest overlap value.
updateActiveDutyCycle(c)	Computes a moving average of how often column c has been active after inhibition.
updateOverlapDutyCycle(c)	Computes a moving average of how often column c has overlap greater than stimulusThreshold.
averageReceptiveFieldSize()	The radius of the average connected receptive field size of all the columns. The connected receptive field size of a column includes only the connected synapses (those with permanence values >= connectedPerm). This is used to determine the extent of lateral inhibition between columns.
maxDutyCycle(cols)	Returns the maximum active duty cycle of the columns in the given list of columns.
active(s)	True if synapse s is active, i.e. the input connected to synapse s is ON.
increasePermanences(c, s)	Increase the permanence value of every synapse in column c by a scale factor s.
boostFunction(c)	Returns the boost value of a column. The boost value is a positive scalar value. It is above one if the active duty cycle is below the mean active duty cycles of neighboring columns. It is less than one if a column has higher active duty cycle than its neighbors

Table 2. Supporting routines used in the Spatial Pooling pseudocode. They may have different names in the NuPIC codebase.