Numenta Node Algorithms Guide – NuPIC 1.7

NuPIC 1.7 includes early implementations of the second generation of the Numenta HTM learning algorithms. These algorithms are available as two node types: `SpatialPoolerNode` and `TemporalPoolerNode`.

This version of the algorithms is early and still considered alpha. Many parameter combinations have not yet been tested.

This guide provides a high-level description of the algorithms and explains how to use the parameters.

Other Information Sources

The core concepts, such as spatial pooling, temporal grouping and node hierarchy, are covered in the following white papers and videos:

- *Hierarchical Temporal Memory,* available on the Numenta website at: http://www.numenta.com/for-developers/education/general-overview-htm.php
- *The HTM Learning Algorithms* , available on the Numenta website at: http://www.numenta.com/for-developers/education/algorithms.php
- *HTM Algorithms* , a video of a talk by Numenta founder Dileep George given at the 2008 HTM Workshop: http://www.numenta.com/for-developers/education/ algorithms.php

NuPIC includes several examples that demonstrate how to use the new nodes. See the Vision Framework (`share/vision`), and the examples `node_prediction`, `mocap` (motion capture), and `waves`.

The parameters and their default are listed in the nodeHelp for each node, which you can use as follows:

```
>> from nupic.network import *
>> nodeHelp("SpatialPoolerNode")
>> nodeHelp("TemporalPoolerNode")
```

Node Overview

With these node types, two nodes are required to implement a learning-node level in the hierarchy: a spatial pooler and a temporal pooler.

The input to a level, regardless of its position in the hierarchy, is a temporal sequence of patterns. The goal of the level is to form pools or groups of these patterns so that patterns belonging to the same group are likely to be variations of the same thing. For example, in the visual world, a source of variation is the relative motion between objects and the viewer. Another source of variation is random noise.

If a level can group patterns that correspond to variations of the same original pattern, then the groups are be invariant to such variations. Once the level has formed enough groups of patterns (formed an invariant representation), the level can start producing outputs - summarizing each input it receives by which invariant group to which it belongs to.

Spatial Pooling and Temporal Pooling

A level uses two kinds of pooling mechanisms to form its invariant representations. These two kinds of pooling mechanisms roughly correspond to the two kinds of variations discussed above: variations due to motion and variations due to noise.

- Spatial pooling is performed by spatial pooler nodes. Spatial pooling is useful for removing noise from a pattern, but does not in isolation solve the invariance problem. Spatial pooling can be thought of as a quantization process that maps a potentially infinite number of input patterns to a finite number of quantization centers.
- Temporal pooling is performed by temporal pooler nodes. This mechanism pools patterns using their temporal proximity. If pattern A is frequently followed by pattern B, the temporal pooler can assign them to the same group. Temporal pooling is very powerful because it can pool together patterns that are very dissimilar. In a vision example, this mechanism can group together a corner and a shifted corner. Temporal pooling can learn a wider class of invariances than spatial pooling, but cannot operate on an infinite number of points; for this reason, spatial pooling precedes temporal pooling to quantize the input space and offer a finite alphabet to the temporal pooler.

A level contains two nodes to perform the two tasks:

- Spatial pooler node: Learns a mapping from a potentially infinite number of input patterns to a finite number of quantization centers. The output of the spatial pooler is in terms of its quantization centers.
- Temporal pooler node: Learns temporal groups - groups of quantization centers - according to the temporal proximity of occurrence of the quantization centers of the spatial pooler. The temporal pooler outputs the temporal groups that it has learned.

These nodes do not implement a pre-determined function, but must learn an output mapping by observing their inputs. Therefore, both nodes must undergo a learning phase before they switch to an inference phase.

The rest of this document briefly discusses each node and gives a reference to the parameters, with emphasis on potential effects of changing each parameter.

SpatialPoolerNode

The `SpatialPoolerNode` has two phases of operation.

- During the learning phase, it stores coincidence patterns, optionally after sparsifying them. A pattern that is stored represents a co-occurrence of the temporal groups from the spatial pooler's child nodes.
- During the inference phase, the node produces an output for every input pattern by comparing the input pattern against all the stored patterns. The output is a vector that denotes the degree of match of the input pattern to all the stored patterns.

The key tuning parameters for `SpatialPoolerNode` are `maxCoincidenceCount`, `maxDistance`, `sparsify`, and `sigma`.

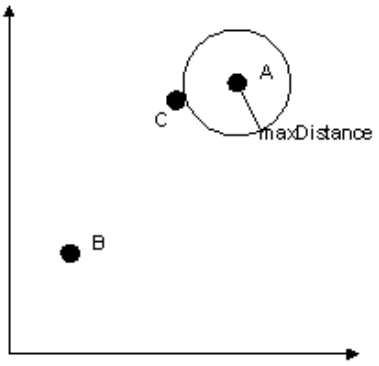
SpatialPoolerNode Parameters

maxCoincidenceCount — Determines the maximum number of coincidence patterns that can be stored in a node. Specified at the time of node creation, determines the maximum number of coincidence patterns that can be stored in a node.

You can think of this parameter as the capacity of the node. This is a key parameter affecting network performance. Storing too few coincidence patterns can result in loss of accuracy due to loss of information. Storing too many coincidence patterns can result in lower generalization and longer training times.

maxDistance — Specifies the distance by which an input pattern has to differ from a stored pattern in order to be considered as a different pattern for storage. An incoming pattern that is within the `maxDistance` of any of the stored patterns is not stored. `maxDistance` should be higher for noisy inputs.

For example, in the figure below, results are in a 2-dimensional space. Point A and B are clearly different points; however, point C is close to point A. If the value of `maxDistance` is large enough to have the circle include point C, then the two are considered two occurrences of the same point. If `maxDistance` is small and the circle does not enclose C, then the two are considered two different points.



sparsify — Specifies whether a stored pattern is sparsified or not. Sparsifying a pattern means setting most components to zero. In domains like vision and speech,

sparsifying the stored coincidence patterns increases the recognition and generalization performance.

sigma — During a node's inference stage, each input pattern is compared to the stored patterns assuming that the stored patterns are centers of radial basis functions with Gaussian tuning. The `sigma` parameter specifies the standard deviation of this Gaussian. Select a parameter value based on the noise in the environment. Keep `sigma` high for noisy situations, and low for non-noisy situations.

clonedNodes — If this parameter is true, all spatial pooler nodes in a region use the same set of coincidence patterns. Sharing of coincidence patterns among different nodes in a region reduces storage requirements and learning time when such sharing is appropriate. This is usually set to True for vision tasks.

inferenceMode — If this parameter is True, the node is in inference mode.

learningMode — If this parameter is True, the node is in learning mode. Only one of `learningMode` and `inferenceMode` can be True at a given time.

coincidenceCount — Number of coincidence patterns that are actually stored. The number of stored coincidence patterns can be less than the specified `maxCoincidenceCount` depending on the amount of training and parameter settings. If the `activeOutputCount` is not equal to the `maxCoincidenceCount` at the end of a training session, it can indicate a problem with the training, for example, that `maxDistance` is too high or not enough data were submitted to the HTM Network.

Spatial Pooler Node Inputs

bottomUpIn — The input to the spatial pooler node from its children. The input typically comes from child temporal pooler nodes or from a sensor.

Spatial Pooler Node Outputs

bottomUpOut — The output vector of the spatial pooler node. This output is meaningful only when the spatial pooler is in inference mode. The output vector indicates how well an input pattern matches stored coincidence patterns.

TemporalPoolerNode

The `TemporalPoolerNode` pooling algorithm has three main components: a grouping algorithm for creating groups based on temporal coherence, a capability for learning higher order sequences, and time-based inference (TBI).

The temporal pooler node has two phases of operation.

- During the learning phase, this node learns the transitions in its input space and forms temporal groups. The temporal groups are formed by partitioning the graph corresponding to the temporal transitions. Temporal grouping is done when the node is switched to inference mode. Depending on the configured learning complexity, higher-order temporal information can be stored in each temporal group.
  The default behavior of the node is to ignore higher order information and just perform temporal grouping based on first-order statistics.

- During the inference phase, the node produces an output that reflects its current degree of membership in each temporal group, either based on the current input or based on the sequence of inputs received until that point. The node can also provide predictions on its inputs using the predict command.

For first-order learning the key tuning parameters are `requestedGroupCount` and `transitionMemory`. For higher-order learning, `sequencerModelComplexity`, `sequencerWindowLength`, and `sequencerWindowCount` are important tuning parameters. The following inputs, outputs and parameters determine the learning and inference processes in the temporal pooler nodes.

Temporal Pooler Node Parameters

requestedGroupCount — This creation-time parameter determines the maximum number of temporal groups that will be formed. This number must be less than or equal to the size of the `bottomUpOut` output. If set to zero, then it is set to the size of the `bottomUpOut` output.

The actual number of temporal groups formed will normally be equal to the `requestedGroupCount`. In some cases the actual number of temporal groups formed can be less than the `requestedGroupCount` For example, if `sequencerWindowCount` is 1 and if the number of coincidence patterns that the Temporal pooler node is operating on is less than `requestedGroupCount` then the node can assign at most one coincidence per group. This can be an indicator of problems with the parameter settings or the training of the preceding spatial pooler nodes.

This parameter can affect the performance of the network in many ways. Setting `requestedGroupCount` too low compared to the number of coincidence patterns results in over-grouping of the coincidence patterns. This can lead to over-generalization and reduced accuracy in classifying patterns. Setting `requestedGroupCount` to be equal to the number of coincidence patterns results in groups that contain one coincidence pattern each. That setting results in explicit memorization and poor generalization.

transitionMemory — Determines how far back in time the temporal pooler node looks for temporal transitions. Having a high `transitionMemory` has the effect of smoothing out the temporal transitions so that temporal jitter and repeated states in the input sequence do not produce undesired behavior. It is expected that nodes at the bottom of the node hierarchy need smaller values, and nodes toward the top need larger values of transition memory.

equalizeGroupSize — If this parameter is True, the temporal pooler node attempts to form temporal groups that are roughly equal in size.

temporalPoolerAlgorithm — A string that selects the method of computing output probabilities. This parameter affects only the inference behavior of the node, and has no impact on what is learned.

- When set to `maxProp`, computes a more peaked score for the group based on the current input only.

- When set to `sumProp`, computes a smoother score for the group based on the current input only.
- When set to `tbi`, computes a score using Time-Based Inference (TBI) which uses the current as well as past inputs. See Understanding Time-Based Inference in *Advanced NuPIC Programming*.
- When set to `hardcoded`, the node uses pre-determined temporal pooling that is appropriate for vision tasks. With this "short-cut" setting it is not necessary to learn the temporal pooler level and the learning phase can proceed faster. Please see the experiments in share/vision/experiments for examples of using this setting.

sequencerWindowCount

The temporal pooler node looks for sequential information in multiple stages. The `sequencerWindowCount` parameter controls the number of stages the pooler uses for the discovery of sequential information. If this value is set to 1 (default), a first-order temporal model is formed. More sequential information is extracted from the data with increasing `sequencerWindowCount`. However, there is no straight-forward correspondence between the temporal order and the `sequencerWindowCount`.

Use higher values of `sequencerWindowCount` and `sequencerModelComplexity` (see below), for applications where most of the information is in the temporal dimension.

sequencerWindowLength

Specifies the number of iterations (training inputs) in each window that the temporal pooler node uses for the discovery of sequential information. The temporal statistics are accumulated for these many iterations for each window.

If a very long sequence is available as the input, different segments of the input sequence can be used for each window. If the input data is not long enough to be chunked into different windows while still giving sufficient samples for accumulating temporal transition counts, then the same data can be used for the different windows. In this case, the temporal pooler node is making multiple passes over the same data to discover higher order sequential information.

sequencerModelComplexity

Determines the temporal `order' complexity of the learned temporal groups. For higher values more sequential information is stored in the transition matrices and the nodes.

The `sequencerModelComplexity` required depends on the nature of the application. For invariant object recognition, aggressive grouping can be more important compared to storing sequential information. For such cases, `sequencerModelComplexity` can be kept to zero. For some applications, all the information is contained in the temporal dimension. The `sequencerModelComplexity` parameter should be kept high for such applications.

clonedNodes — Indicates whether the temporal groups of a node are shared by all nodes in a region. Sharing reduces storage requirements and learning time when such sharing is appropriate. This is usually set to True for vision tasks.

inferenceMode — Determines whether the node is in inference mode or not. See `learningMode` below.

learningMode — Determines whether the node is in learning mode. Only one of `learningMode` and `inferenceMode` can be True at a given time.

The following parameters are available from the node after learning:

coincidenceVectorCounts — Array of integers that shows how often each coincidence was encountered during learning.

groupCount — Number of groups actually generated.

groups — Set of coincidences belonging to each group, returned as a list of coincidence indices.

TAM — Time adjacency matrix, returned as a sparse matrix.

Temporal Pooler Commands

predict — Predicts the likelihood of coincidences or groups for a certain number of steps in the future. The command takes two arguments:

- 'coincidences' or 'groups' that indicates whether to return likelihoods for groups or coincidences (default is 'coincidences')
- an integer number of steps to predict (default is 1).

The command returns a matrix that has as many rows as the number of steps requested, and whose number of columns is either the number of coincidences or the number of groups, depending on the mode.

This command works only in inference mode, with `temporalPoolerAlgorithm` set to `TBI`.

Temporal Pooler Node Inputs

bottomUpIn — The input to the temporal pooler node from its children. The input typically comes from child spatial pooler nodes.

resetIn — This input can be used to reset the transition history during the learning mode and the time-based-inference history during the inference mode. Typically, this signal comes directly from the sensor.

Temporal Pooler Node Outputs

bottomUpOut — The output of the temporal pooler node is a vector of reals. It represents the likelihood that the input belongs to each of the temporal groups of this node.

Comparing Zeta1Node and TemporalPoolerNode / SpatialPoolerNode

The `Zeta1Node` algorithm differs from the `TemporalPoolerNode` / `SpatialPooler-Node` algorithm in a number of ways:

- Two nodes instead of one. `Zeta1Node` included both a spatial and a temporal pooler. With the new design, two nodes perform the two functions. This allows you to have either a hierarchy of pairs, or to have input from multiple spatial poolers go in to one temporal pooler node.
- Sparse spatial pooler. `SpatialPoolerNode` implements a new sparse spatial pooling algorithm. The node learns sparse distributed representations of its input vectors. These representations are more memory efficient, require fewer stored spatial patterns, and require less training time than the spatial pooler in `Zeta1Node`. As a result, sparse representations scale better to large problems than the Zeta1 algorithms.
- Temporal pooler. The temporal pooler has a new grouping algorithm for creating groups based on temporal coherence and a new capability for learning higher-order sequences.