Overview of the Temporal Pooler

Sagan Bolliger edited this page 24 days ago · [5 revisions](https://github.com/numenta/nupic.research/wiki/Overview-of-the-Temporal-Pooler/_history)

**Pages 22**



* [**Home**](https://github.com/numenta/nupic.research/wiki)
* [**AI & ML Reference Material**](https://github.com/numenta/nupic.research/wiki/AI-&-ML-Reference-Material)
* [**AI & ML Talks**](https://github.com/numenta/nupic.research/wiki/AI-&-ML-Talks)
* [**Behavior and Pooling Requirements**](https://github.com/numenta/nupic.research/wiki/Behavior-and-Pooling-Requirements)
* [**Capacity experiment: setup and validation**](https://github.com/numenta/nupic.research/wiki/Capacity-experiment:-setup-and-validation)
* [**Capacity Test Results**](https://github.com/numenta/nupic.research/wiki/Capacity-Test-Results)
* [**Capacity Test Results (obsolete)**](https://github.com/numenta/nupic.research/wiki/Capacity-Test-Results-(obsolete))
* [**Current Research Tasks**](https://github.com/numenta/nupic.research/wiki/Current-Research-Tasks)
* [**Extraneous Predictions**](https://github.com/numenta/nupic.research/wiki/Extraneous-Predictions)
* [**Installation on EC2**](https://github.com/numenta/nupic.research/wiki/Installation-on-EC2)
* [**Meeting and Progress Notes**](https://github.com/numenta/nupic.research/wiki/Meeting-and-Progress-Notes)
* [**Meeting Notes from September 22**](https://github.com/numenta/nupic.research/wiki/Meeting-Notes-from-September-22)
* [**Modification to sensory motor temporal memory (Layer 4) algorithm**](https://github.com/numenta/nupic.research/wiki/Modification-to-sensory-motor-temporal-memory-(Layer-4)-algorithm)
* [**Open Questions**](https://github.com/numenta/nupic.research/wiki/Open-Questions)
* [**Overview of the Temporal Pooler**](https://github.com/numenta/nupic.research/wiki/Overview-of-the-Temporal-Pooler)
* [**Research Glossary**](https://github.com/numenta/nupic.research/wiki/Research-Glossary)
* [**Sensorimotor Inference Algorithm**](https://github.com/numenta/nupic.research/wiki/Sensorimotor-Inference-Algorithm)
* [**Sensorimotor Inference Algorithm Discussion**](https://github.com/numenta/nupic.research/wiki/Sensorimotor-Inference-Algorithm-Discussion)
* [**Temporal Pooler Overview**](https://github.com/numenta/nupic.research/wiki/Temporal-Pooler-Overview)
* [**Temporal Pooler Simultaneous Tandem Learning Experiments**](https://github.com/numenta/nupic.research/wiki/Temporal-Pooler-Simultaneous-Tandem-Learning-Experiments)
* [**Union Pooler Experiment Results**](https://github.com/numenta/nupic.research/wiki/Union-Pooler-Experiment-Results)
* [**Union Pooler Psuedocode**](https://github.com/numenta/nupic.research/wiki/Union-Pooler-Psuedocode)
* [Show 7 more pages…](https://github.com/numenta/nupic.research/wiki/Overview-of-the-Temporal-Pooler)

**Clone this wiki locally**



[**Clone in Desktop**](https://windows.github.com/)

This document describes the algorithm in [union\_temporal\_pooler.py](https://github.com/numenta/nupic.research/blob/master/htmresearch/algorithms/union_temporal_pooler.py) in nupic.research

**Objective of this Writeup**

The purpose of this writeup is to describe the current temporal pooler implementation in nupic.research (as of Oct 26, 2015). We provide some background and intuition, but the primary goal is to guide the reader through the specific implementation we are currently working with. Please see the note at the end regarding previous historical implementations of temporal pooling.

**Motivation**

The purpose of the temporal pooler is to produce representations of temporal sequences that are both stable and unique. By saying that the representation is stable we mean that the representation changes little in response either to the addition of noise in the input or to different temporal instances of the same semantic class. Thus when a previously-learned sequence is presented to the TP, we want the output to vary more slowly than the input since the TP pools multiple temporal instances in the input into fewer semantic classes represented in the output. When this happens, the slow-changing output can serve as a semantic label for the input sequence it receives.

However, stability alone is not a sufficient requirement for a useful TP: one can trivially build a perfectly stable TP that always produces the same output regardless of its input. In addition to stability, the TP must produce unique representations. By uniqueness we mean that inputs that represent semantically different categories should produce different outputs. In the context of streaming data this means that presentation of different learned sequences should produce different stable outputs for the duration of those sequences.

We know from empirical studies of the neocortex that stability and uniqueness both increase while traveling up the cortical hierarchy. Thus, for example, in neocortical region V1, at the bottom of the visual hierarchy, neurons are sensitive of edges at particular orientations and change rapidly in response to any change in the visual field, and hence exhibit low stability. Additionally, the uniqueness in V1 is also low, since many semantically unrelated inputs will contain the same edges and lead to the same neural activations. Whereas in IT, at the top of the visual hierarchy, neurons represent highly semantic categories like “car” that are stable to many forms of noise, perspective change, and variation between perceptually different instances of the same semantic class. Here uniqueness in increased as small changes in the input that cross the boundaries of semantic categories produce large differences in neural activation. Thus, when designing an algorithmic implementation of pooling, it should have the property that when combined in a hierarchy, successive pooling layers should create representations that share this neural property of increasing both stability and uniqueness.

In addition to the general motivation of creating a semantic hierarchy, there is also a specific practical limitation to the functioning of the temporal memory that the TP solves, namely the issue that the TM is brittle to small sequence variations. In the TM the context of every input is exclusively represented in the set of cells active in each column. Since an unpredicted input will result in a column burst, a single unpredicted input will erase all previous context for the sequence. Thus, the TM is unable to generalize between sequences that are substantially the same but differ in small ways such as due to deletions, repetitions, or transpositions of input states.

The TP is able to solve this by creating a temporally stable representation of the input that is somewhat robust to temporary loss of context in the TM. Thus, when the TM loses its context due to the sorts of noise discussed previously, feedback connections from the TP to the TM place the cells in the TM that correspond to the sequence in a predictive state allowing the TM to pick up where it left off at the next time step.

**Algorithm Details**

**Activation Rules**

The experimental temporal pooling algorithm currently implemented in nupic.research can be thought of algorithmically as a variation of the spatial pooler with a union pooler stacked on top. A union pooler, as will be described in more detail below, is an algorithm that produces an SDR union of its inputs over time so as to increase temporal stability. Note that although this is a natural separation of the steps of the algorithm, we believe that a single population of cells implements both algorithmic steps. The neural details are an area of active research and are not covered here.

1. Modified Spatial Pooling Process

The first component of the temporal pooler implements a modified spatial pooler. It receives two inputs from the preceding TM layer: 1) the set of cells in the TM that are active (activeInput), and 2) the subset of those active cells that were predicted previously (predictedActiveInput). Column activations are determined via the connections of the proximal dendrites as per the standard SP algorithm, but with one exception: the inputs to each column are a weighted average of the active inputs and the predicted active inputs. Thus, we calculate

overlapActive = calculateOverlap(activeInput)

overlapPredictedActive = calculateOverlap(predictedActiveInput)

totalOverlap = overlapActive \* weightActive + overlapPredictedActive \* weightPredictedActive

Here totalOverlap is the input to the standard SP. The output of the SP is the set of columns with the strongest inputs determined via the standard SP winner-take-all competition. This allows us to give cells that were successfully predicted a larger impact on the output of the TP. The motivation for doing this is that after learning, states that are predicted by the TM are temporal instances that belong to the same semantic category as the preceding states, and therefore should form a part of the stable semantic representation produced by the TP.

1. Union Pooling Process

The second component of the temporal pooler implements a union pooling process. This process introduces several new concepts. A union-pooled representation can be thought of as an operation that is performed on the SDR produced by the column activations of the previous modified SP step and outputs another SDR of the same size. In the UP process, each bit of the UP has a scalar state called its persistence. The persistence values of all UP bits start at 0. Then, in each time step, every bit has its persistence decayed. The current implementation uses exponential decay, although other functions are possible.

Additionally, if a modified SP column is active, the persistence of its corresponding UP bit is increased by a nonlinear function of the overlapPredictedActive component of the input to that column. It is important to note that while unpredicted active cells do contribute to the activity of the columns in the modified SP, only those active cells that were predicted by the TM in the input to modified SP contribute to increased persistence of the corresponding bit in the UP.

The current implementation uses a sigmoidal nonlinearity for the mapping between overlapPredictedActive and the amount of persistence to add. For every active column c in the modified SP step, the persistence of the corresponding bit c in the UP is modified as:

persistence[c] +=

baseLinePersistence +

extraPersistence/(1 + exp(-(overlapPredictedActive[c] - thresh)))

where overlapPredictedActive[c] is the number of connected synapses on the proximal dendrite of modified SP column c to active cells that were predicted in the previous time step. Thus, when a modified SP column is active, the corresponding UP bit has its persistence increased by a minimum of baseLinePersistence and a maximum of baseLinePersistence + extraPersistence.

1. Sparsification

The output SDR of the TP is determined via a winner-take-all competition based on the persistence values of the bits in the UP. The output of the TP at a given time step is the set of bits with the top X% persistence in that time step. Here X is a configurable parameter that defaults to 20%.

**Learning Rules**

Learning in the TP occurs through changing the synapse permanence in the modified SP. This is done according to three rules: a modification of the standard SP learning rule, plus two additional rules.

1. Modified SP Learning

Similar to the SP learning algorithm, when the SP column is active, its synapses with active predicted pre-synaptic cells have their permanence increased and all other synapses for that column have their permanence decreased. The difference from the standard SP learning rule is that only active predicted cells have their permanence increased with the TP whereas all active cells do with the standard SP. The reason for doing this is that we only want to create stable representations of learned sequences, which in turn produce predicted activations.

1. Forward Learning

In forward learning those columns corresponding to active bits in the output SDR of the TP (the SDR that results after performing the sparsification in step #3 above) have the permanence of their synapses with active predicted pre-synaptic cells increased. The means that permanence can be increased even if the corresponding modified SP column is inactive. Additionally, there is no permanence decrement rule for inactive pre-synaptic cells.

The result of forward learning is that columns that have in past time steps contributed sufficient persistence to the UP will learn to also become active by the current state, further reinforcing the persistence of the corresponding TP output bit over the duration of the sequence.

1. Backward Learning

Like in forward learning, the “post-synaptic cells” that determine whether learning takes place are the bits of the TP output SDR rather than the modified SP columns. However, instead of taking inputs from the cell states at the current time step, the backward learning rule iterates over a fixed number of time steps into the past, and increases synapse permanence for any synapse for which the pre-synaptic cell is active predicted in any of those past time steps. Similarly to forward learning, there is no permanence decrement rule.

Whereas forward learning causes TP output bits that were active early in a sequence to continue to be active later in the sequence, backward learning has the opposite effect of causing TP output bits that are active late in a sequence to become active earlier in the sequence. Because the learning rules only operate on active predicted cells, the TP will only form stable representations of sequences that have previously been learned and therefore predicted by the TM. This prevents the TP from forming stable representations of random inputs that do not represent repeated temporal structure found in the input.

Pseudocode for the learning rules follows:

For each time step t:

For each column c:

For each potentially connected synapse s:

# Standard SP learning rule

If columnActive(t, c):

If synapseActive(t, s):

increasePermanence(s)

Else:

decreasePermanence(s)

# Forward learning rule

if outputSDRActive(t, c) and synapseActivePredicted(t, s):

increasePermanence(s)

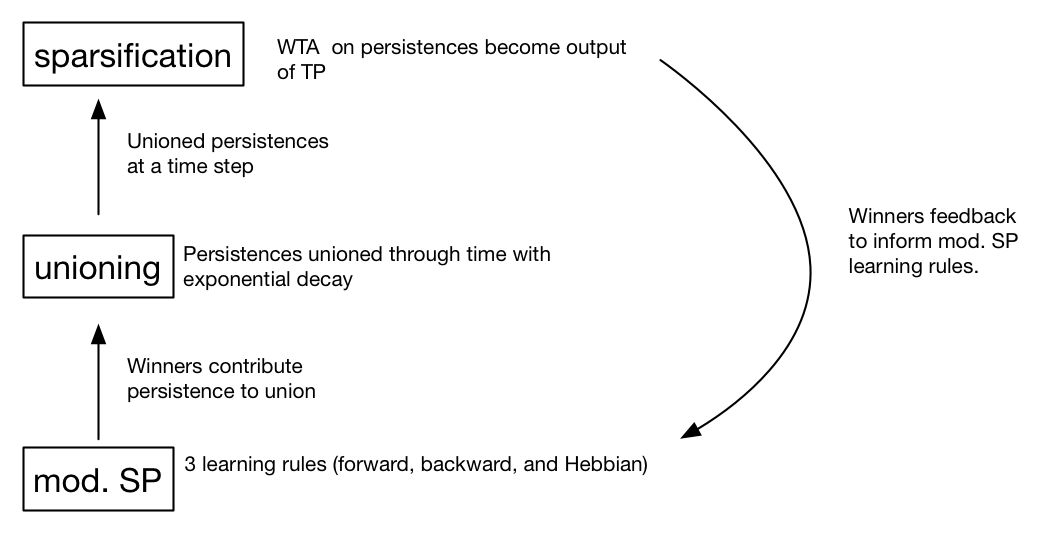
# Backward learning rule

for t’ in [t-n, t-n+1, … t-1]:

if outputSDRActive(t’, c) and synapseActivePredicted(t’, s):

increasePermanence(s)

Flow diagram of components of the temporal pooler:



**Discussion**

The question remains of how our proposed TP algorithm leads to increased stability and uniqueness while also solving the issues with brittleness that we previously discussed. Without learning, stability is trivially increased by keeping UP bits active longer than they otherwise would. Consider a scenario in which each UP bit is active for an average of five time steps after its corresponding modified SP column becomes active. Therefore, whereas in each time step the set of active modified SP columns will be nearly 100% different (less some small amount of accidental SDR overlap), the set of UP active bits will be 80% stable between time steps.

However, this 80% stability has a major pitfall, namely that every 5 time steps the set of active UP bits will have completely changed. So the stability that is gained through the UP alone is not sufficient to create stable labels of sequences with lengths longer than the persistence time of UP bits. In order to counteract this we require learning. With learning, modified SP columns that initially happen to be active at a single point in the sequence form synapses with cells that are active both in both previous and subsequent time steps. As a result those modified SP columns will become activated by states at different time steps in the sequence and will in turn contribute persistence to their corresponding UP bits at multiple times adding to their stability. So long as a modified SP column is active at least once before its correspond UP bit loses enough permanence to become inactive, there is no limit to the total duration for which a UP bit may remain active.

Uniqueness in the context of temporal sequences, on the other hand, is the result of the exclusive influence of active cells on UP bit persistence. When given two learned sequences with small differences, the sets of active cells in the TM step differ radically after the first difference in the sequences even as the column activations remain substantially the same. Since only the active cells activate the modified SP columns, they, and their corresponding UP bits, also completely differ, producing uniqueness.

Brittleness to small temporal errors in a learned sequence, as discussed previously, can be ameliorated through feedback connections from the UP bits down to the cells of the TM. In particular, once an unexpected state is received, without feedback connections, the SP columns corresponding to the state burst, and as a result all prior context is lost. However, with the addition of the TP, the UP SDR state can persist across such errors. If we include feedback connections that “project” from each UP bit to the apical dendrite of each cell and function in a similar manner to distal dendrites in putting cells into their predictive state, then once a state that belongs to the original sequence is presented, instead of bursting, as it would otherwise, the presence of a predicted cell causes only that cell in the column to become active, thus restoring the context that would otherwise have been lost.

One might object that the TP process merely “smears” activations over time rather than creating a compressed label for the temporal sequence like the SP does for spatial patterns. Consider for instance a SP column that is activated by the co-occurrance of cells A, B, C, D, and E. This column will be robust to the loss of a few cells, but after a certain point a very degraded input, for instance just A and D, will no longer cause the column to be active at all. This nonlinear response is necessary for creating a compressed representation. However, consider the case when the sequence A->B->C->D->E occurs in the input and is a learned sequence of the TM. With the inclusion of feedback, the TM is robust to deletions of states. However, if we delete 60% of the states like in the SP case leaving only the sequence A->D, the resulting TP representation will only differ by 60%. Thus it seems that there is no nonlinear response to degradation of temporal patterns like there is for spatial patterns, which in turn limits the ability of the TP to perform compression.

The resolution to this seeming limitation comes from viewing the TP in the context of hierarchy. What the TP accomplishes is converting a temporal pattern into a spatial pattern. The result is that the SP in the next hierarchical region receives what was originally a temporal sequence A->B->C->D->E as a spatial pattern consisting of A, B, C, D, and E simultaneously. Once the temporal pattern is represented spatially, the standard SP learning algorithm is able to create a compressed representation of the pattern through its column activations, which do exhibit nonlinear response properties.

**History of the term "temporal pooler"**

Temporal pooling has been an active area of research for HTMs and Numenta for several years. The meaning of temporal pooling and the overall goals of temporal pooling have been largely consistent. However the term "temporal pooler" has been used for a number of different implementations and looking through the code and past documentation can be somewhat confusing.

The original CLA Whitepaper used the term temporal pooler to describe a particular implementation (let's call this temporal pooling version 1). This implementation was intricately tied in with sequence memory. As such the sequence memory and temporal pooling were both referred to as "temporal pooling" and the two functions were confounded. In NuPIC the code files TP.py and TP10x.py implement sequence memory but use the old terminology. The newer files, such as temporal\_memory.py, avoid the term "temporal pooling" and just implement sequence memory.

In 2014 a second version of temporal pooling was implemented in nupic.research. This code is not in NuPIC (just in nupic.research). Some of the key ideas in that version were discussed in this email from Jeff: <http://lists.numenta.org/pipermail/nupic_lists.numenta.org/2014-January/002662.html>. Some experiments using this technique were done and reported here: <https://github.com/numenta/nupic.research/wiki/Capacity-Test-Results> and here: <https://github.com/numenta/nupic.research/tree/master/projects/sensorimotor/cosyne_2015>

In 2015 we are working with a third version of temporal pooling. This third version is what is described in this writeup.