2022-03-03

On the Importance of the Newborn Stage When Learning Patterns with the Spatial Pooler

Dobric, D

10.1007/s42979-022-01066-4
SN Computer Science
Springer

On the importance of the newborn stage when learning patterns with the Spatial Pooler

Author Information:

Damir Dobric
Frankfurt University of Applied Sciences, daenet GmbH
ddobric@daenet.de
http://damirdobric.me

Andreas Pech
Frankfurt University of Applied Sciences
Full Professor
pech@fb2.fra-uas.de

Bogdan Ghita
University of Plymouth
Associate Professor
bogdan.ghita@plymouth.ac.uk

Thomas Wennekers
University of Plymouth
Associate Professor
thomas.wennekers@plymouth.ac.uk

Keywords: HIERARCHICAL TEMPORAL MEMORY, CORTICAL LEARNING ALGORITHM, SPATIAL POOLER, HOMEOSTATIC PLASTICITY

Declarations

Funding: Not applicable
Conflicts of interests/competing interests: Not applicable
Availability of data and material: Not applicable

Code availability:

All experiments described in this paper are implemented in C#/.NET *dotnet standard 2.1* compatible with the latest release .NET 6.0. The Hierarchical Temporal Memory framework with the Spatial Pooler used in experiments is based on the open-source project *NeocortexApi*. The source code and documentation can be found at GitHub [1]. The experiment related to the stability of the Spatial Pooler is implemented in a form of the UnitTest inside of the Microsoft Unit Testing framework integrated with Visual Studio.

The test used for the stability experiment is called *SpatialPooler_Stability_Experiment_3.* It is implemented in the source file *SpStabilityExperiments.cs*. This code generates three output CSV files:

-ActiveColumns.csv,
-ActiveColumns-plotlyinput.csv and
-Oscillations.csv.

ActiveColumns files hold the same information in a slightly different format than ActiveColumns-plotlyinput.csv. Both files contain active columns (SDR) for every trained digit in every iteration. ActiveColumns-plotlyinput.csv can be used as the input for the Python script to generate diagrams that represent active columns shown in figure 6.

The script used to generate the diagram is called draw_figure.py and can be found at the following location: */Python/ColumnActivityDiagram/draw_figure.py*

Further information about running the script can be found in the Python script. The file Oscillations.csv file is used to generate the diagram shown in Figure 1. This diagram was generated by Microsoft Excel.

Abstract:

Hierarchical Temporal Memory (HTM-CLA) - Spatial Pooler (SP) is a Cortical Learning Algorithm for learning inspired by the neocortex. It is designed to learn the spatial pattern by generating the Sparse Distributed Representation code (SDR) of the input. It encodes the set of active input neurons as SDR defined by the set of active neurons organized in groups called mini-columns. This paper provides additional findings extending the previous work, that demonstrates how and why the Spatial Pooler forgets learned SDRs in the training progress. The previous work introduced the newborn stage of the algorithm, which takes a control of the boosting of mini-columns by deactivating the Homeostatic Plasticity mechanism inside of the SP in layer 4. The newborn stage was inspired by findings in neurosciences that show that this plasticity mechanism is only active during the development of newborn mammals and later deactivated or shifted from cortical layer L4, where the SP is supposed to be active. The extended SP showed the stable learned state of the model. In this work, the plasticity was deactivated by disabling the homeostatic excitation of synaptic connections between input neurons and slightly inactive mini-columns. The final solution that includes disabling of boosting of inactive mini-columns and disabling excitation of synaptic connections after exiting the introduced newborn stage, shows that learned SDRs remain stable during the lifetime of the Spatial Pooler.

1 INTRODUCTION

The Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA) is an algorithm inspired by the biological functioning of the neocortex. It combines a combination of spatial pattern recognition and temporal sequence learning [2]. The HTM CLA is the algorithm that can be used to solve several kinds of problems like anomaly detection [3], object recognition [4], document categorization [5], Sequence Learning [6] and many others. In a nutshell, the HTM-CLA organizes neurons in layers of column-like populations built from many neurons, such that the units are connected into structures called areas. Areas, columns and mini-columns are hierarchically organized [7] and are usually connected in more complex structures, which implement higher cognitive functions like invariant representations, pattern- and sequence recognition etc. HTM CLA in general consists of two major algorithms: Spatial Pooler and Temporal Memory.

The Spatial Pooler, which is of interest in this work operates on mini-columns connected to input neurons [8]. This can be some sensory input or a set of cells activated by cortical learning of some connected population of neurons. The SP is responsible to learn spatial patterns by encoding the pattern into the sparse distributed representation (SDR). The created SDR is represented as a population of active mini-columns, which is further used as the input for the Temporal Memory (TM) algorithm.

The TM is an algorithm that is responsible for learning sequences. Experiments in previous work [9] show that the originally designed version of the Spatial Pooler is unstable. This means, that the learned patterns will be forgotten during the learning process and then learned again. Results also showed that the Spatial Pooler oscillates between stable and unstable state. Experiments show the instability in the learning process does not happen for all patterns at the same time. It always detected in a single or few patterns. Ver time unstable patterns become stable, but some new patterns can become unstable.
For example, The Spatial Pooler can keep the stable SDR_1 for pattern *p1* while SDR_2 for pattern *p2* becomes unstable and so on. Having a stable SDR is essential for all other cortical functions that rely on the generated SDR. An unstable Spatial Pooler will also cause the Temporal Memory algorithm to forget learned sequences.

The previous work [9] investigated the instability of the SP and introduced a *newborn* stage of SP. The extended algorithm prevents the original SP algorithm to enter unstable ("epileptic") behaviour. The *newborn* stage was designed to first enable the homeostatic plasticity mechanism [10] that boosts inactive columns, and then to disable it. The modification has shown better stability of the SP, but it still produced oscillations in the learning process. This paper extends the previous work and shows that the SP can become almost completely stable if the excitation of inactive synaptic connections between input neurons and mini-columns is also controlled and disabled after exiting the *newborn* stage.

2 METHODS

To analyse the learning process of the Spatial Pooler, an instance of the SP with the set of common parameters was used (see table 1).

Table 1 Spatial Pooler parameters used in experiments. Set of parameters shown in the table that are used in expeiments with the Spatial Pooler

Parameters	Value
INPUT BITS	200
COLUMNS	2048
GLOBAL_INHIBITION	true
NUM_ACTIVE_COLUMNS_PER_ INH_AREA	2% (40)
STIMULUS_THRESHOLD	0.5
SYN_PERM_INACTIVE_DEC	0.01
SYN_PERM_ACTIVE_INC	0.01
SYN_PERM_CONNECTED	0.1
MIN_PCT_OVERLAP_DUTY_CYCLES	0.001
MIN_PCT_ACTIVE_DUTY_CYCLES	0.001
POTENTIAL_RADIUS	1024
DUTY_CYCLE_PERIOD	100
MAX_BOOST	10

Experiments were done with a different number of columns. Results shown in this paper are produced by using 2048 mini-columns. In this specific case, the scalar encoder was used to encode the scalar input values, which are presented to the Spatial Pooler during the learning process. The SP was trained to remember values between 0 and 100. Before presenting an input to the Spatial Pooler, every input value was encoded with 200 bits, each value is encoded with 15 non-zero bits.
Figure 1 shows three examples of encoded scalar values used as input for SP. For more detailed information about the meaning of all parameters please see [1].

Figure 1 represents scalar values 0, 1 and 2. The encoded input value is on the right and the corresponding SDR is on the left. The grey colour in the figure represents zero-bits (background of the image) and the black colour represents the non-zero bits. Grey dots on the left represent a set of active columns after encoding the given input.

Inspired by the homeostatic plasticity mechanism [10], [11], [12], the Spatial Pooler algorithm implements a boosting of inactive columns and excitation of inactive (weak) synapses. This influences the excitation and inhibition balance of neural cells and is likely important for maintaining a stable cortical state. The functional stability of neural circuits is achieved by homeostatic plasticity. It keeps in balance the network excitation and inhibition and coordinates changes in circuit connectivity [13]. This mechanism is implemented explicitly in the Spatial Pooler [9] and it makes that all columns are uniformly used across all seen patterns.

Because this mechanism is continuously active, it can perform the boosting of mini-columns and excitation of synapses that already build learned SDRs. Once that happens the Spatial Pooler will "forget" some learned patterns. If the forgotten pattern is presented again to the SP, it will start learning it again.
To analyse the learning behaviour of the Spatial Pooler, a set of input patterns was presented to the SP instance over many iteration steps.

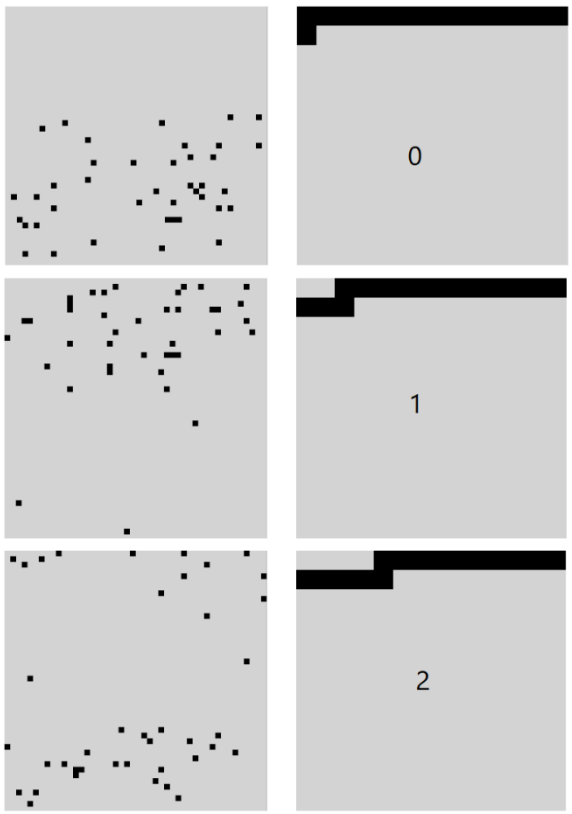

Figure 1: Examples of three input values encoded by the scalar encoder (right) and their corresponding Sparse Distributed Representation (left) encoded by the Spatial Pooler.

Every input pattern is encoded by the Spatial Pooler into SDR represented as a set of indices of active columns A_k of the given pattern in iteration *k*. In every learning step of the same pattern, the similarity between SDR in step k and step k+1 is calculated as shown in equation 1.

$$s = \frac{|A_k \cap A_{k+1}|}{\max(|A_k|, |A_{k+1}|)} \qquad (1)$$

The similarity *s* is defined as a ratio between the number of elements (cardinality) of the same active columns in SDRs generated in steps *k* and *k+1* and a maximum number of active columns in two comparison steps. The Spatial Pooler is by definition stable if the generated SDR of the same pattern does not change for the entire life cycle of the Spatial Pooler. In this case, the similarity between all SDRs of the same pattern is 100%.

Figure 2 shows the SDR of the same input pattern presented to SP in more than 25000 iterations.

The Spatial Pooler learns patterns very fast. It requires usually no more than two to three iterations to learn the presented pattern. The y-axis shows the similarity *s* of SDRs in the current iteration step and the previous step. The x-axis shows the iteration step. The similarity of 100% means the learned SDR does not change over time. After an unspecified number of iterations, the SP forgets the learned SDR and starts learning again. Every time

the SDR changes, it means the learned SDR for that pattern is changed. Because the new SDR for the pattern is created, the previously learned one is forgotten. In that case, the similarity drops from 100% to zero or some other value.

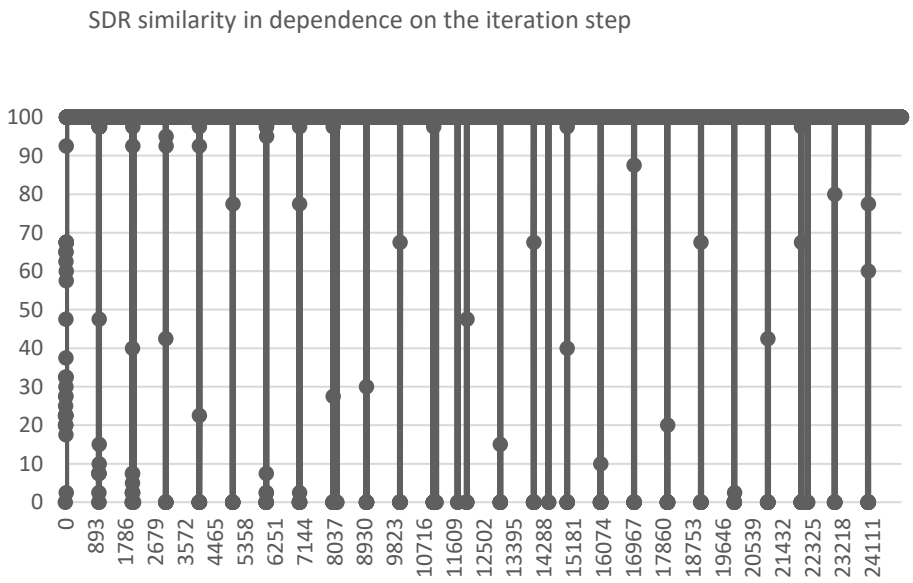

Figure 2: Unstable Spatial Pooler. SP learns the pattern and keeps the SDR unchanged for some iterations. When boosting gets active SP forgets the SDR (similarity drops) and starts learning again.

In contrast, keeping the similarity at 100% means that the learned SDR for the same input is the same for the entire iteration interval. If the similarity is less than 100%, generated SDRs of the same input are different. This indicates an unstable Spatial Pooler. As shown in Figure 2 the learned state oscillates between stable and unstable states during the entire learning time, which is not a useful behaviour for real-life applications.
This experiment clearly shows the instability of the Spatial Pooler, but it does not show any details about the encoding of the SDR. Figure 3 shows the same behaviour from a different point of view. It shows how the SDR of the same pattern is encoded in the first 300 iterations (cycles) on the example of a single input value. The Spatial Pooler generates a stable SDR right at the beginning of the learning process and keeps it stable (unchanged) for approx. 200 iterations. After that SDR will change until the Spatial Pooler enters the stable state again (not shown in the figure).
In the next experiment, the boosting was disabled by setting DUTY_CYCLE_PERIOD and MAX_BOOST to zero value. These two values disable the boosting algorithm in the Spatial Pooler.

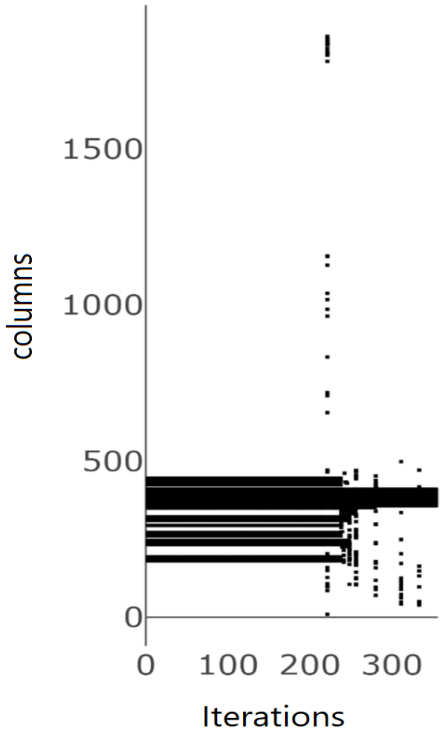

Figure 3: SDR shows active columns (SDR) of the learned input in the first 300 iterations (cycles). The learned SDR is unchanged (stable) in approx. thefirst 200 iterations. After that, it gets unstable.

Results show that the SP with these parameters produces stable SDRs as shown in Figure 4. The figure shows an example of a stable encoding of the single pattern with the disabled boosting algorithm. The SP learns the pattern and encodes it to SDR in a few iterations (typically 2-3) and keeps it unchanged (stable) during the entire life cycle of the SP instance.

By following this result, the stable SP can be achieved by disabling the boosting algorithm.
Unfortunately, without the boosting mechanism, the SP generates SDR-s with an unpredictable number of active mini-columns. Figure 5 shows two input values '0' and '6'. The x-axis represents indexes of active mini-columns, which participates in the encoding of the input value. The y-axis represents the learning iteration. The SP is stable if the SDR code does not change over time. As already mentioned, disabling boosting will cause the SP to enter the stable state as shown in Figure 5. The value '0' is encoded with approx. 40 active mini-columns and the value '6' is encoded with 4 active mini-columns. This is a significant unwanted difference. Experiments showed that some patterns can even be encoded without any active mini-column if boosting is disabled completely or early disabled.

If the number of active mini-columns in an SDR for different inputs is significantly different, the further processing of memorized SDR-s will be negatively influenced. Most operations in the Hierarchical Temporal Memory rely on the calculation of the overlap between neural cells, synapses or mini-columns [14]. In that case, SDR-s with a much higher number of active columns will statistically produce higher overlaps, which is not in balance with other SDR-s with less active cells.

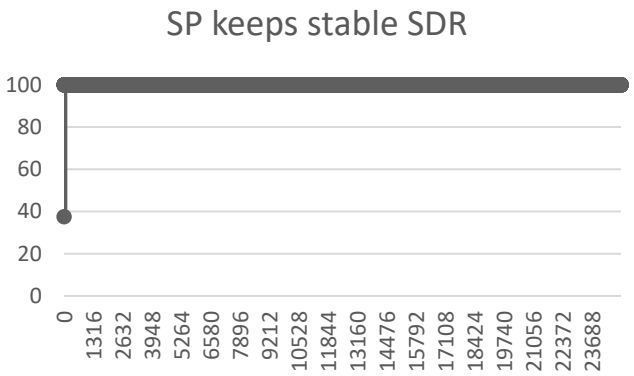

Figure 4: Spatial Pooler generates stable SDR after the boosting is disabled.

The parameter NUM_ACTIVE_COLUMNS_PER_INH_AREA defines the percentage of columns in the inhibition area, which will be activated by the encoding of every single input pattern. Inspired by the neocortex, this value is typically set at 2% [2] [15]. By using the global inhibition in these experiments by the entire column set of 2048 columns the SP will generate SDRs with approx. 40 active columns. The boosting mechanism inspired by homeostatic plasticity in the neocortex solves this problem by consequent boosting of passive mini-columns and inhibiting too active mini-columns. As long the learning is occurring, the SP will continuously boost mini-columns. Every time the boosting takes a place, some learned patterns (SDRs) might be forgotten, and learning will continue when the same pattern appears the next time.

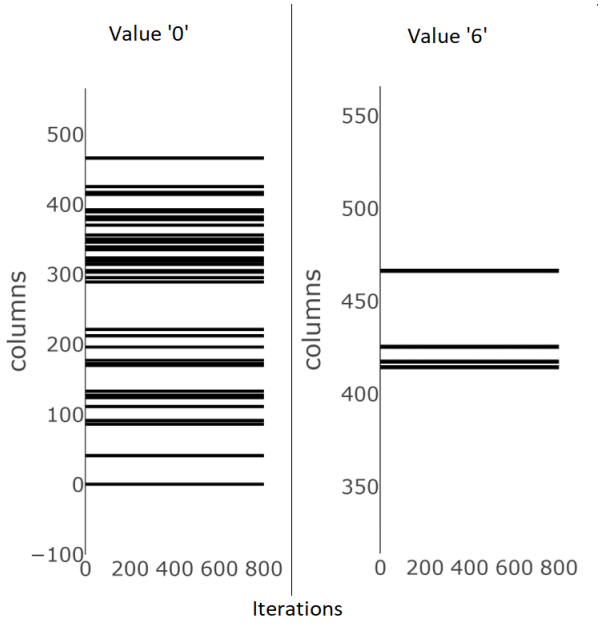

Figure 5: Two SDRs with different numbers of active mini-columns produced by Spatial Pooler with disable boosting.

It can be concluded that the stability of the SP can be influenced by the boosting mechanism. The SP can enter the stable state, but it will produce SDRs with a significantly different number of active mini-columns. In contrast, if boosting is enabled, the SP will uniformly activate mini-columns, but the learning will be unstable.

Previous findings in neural sciences [16] show that homeostatic plasticity boosting is only active during the development of a *newborn* animal and then deactivated or shifted from cortical layer L4, where Spatial Pooler is supposed to be active. The Spatial Pooler operates on sensory inputs, which are commonly connected to the cortical layer L4 [17]. By following this finding, this work extends the Spatial Pooler algorithm and introduces the *newborn* stage of the Hierarchical Temporal Memory and Spatial Pooler.

Deactivation of the boosting in homeostatic plasticity in the cortical layer L4 can also be applied to Spatial Pooler. It is still not clear exactly how this mechanism exactly works. However, by following findings in this area the same or similar mechanism inside of the SP can be adopted. Currently, in the HTM, this mechanism consists of boosting and inhibition algorithms, which operate on the mini-column level and not on the cell level inside of the mini-column. The reason for this is that SP operates explicitly on the population of neural cells in mini-columns and does not make usage of individual cells [8]. Individual cells rather play an important role in the Temporal Memory algorithm [2].

The main idea in this work, with the aim to stabilize the SP and keep using the plasticity, is to add an additional algorithm to SP, which does not influence the existing SP algorithm. The extended Spatial Pooler is based on the algorithm implemented in the new component called Homeostatic Plasticity Controller. The controller is "attached" to the existing implementation of the Spatial Pooler. After the computation in each iteration, the input pattern and corresponding SDR are passed from the SP to the controller. The controller keeps the boosting active until the SP enters the stable state, measured over the given number of iterations. During this time the SP is operating in the so-called *newborn* stage and will produce results similar to results shown in Figure 2 and Figure 3. Once the SP enters the stable state, the new algorithm will disable the boosting and notify the application about the state change. The controller tracks the participation of mini-columns overall seen patterns. After the controller notices that all mini-columns are approx. uniformly used and all seen SDRs are encoded with the approx. the same number of active mini-columns, the SP has entered the stable state. From that moment the SP will leave the *newborn* stage and continue operating as usual but without the boosting. The previous work [9] focused exclusively on the boosting of mini-columns. That is, mini-columns that do not participate enough in the learning process will be boosted. In the boosting process of the mini-column, the calculated overlap will be multiplied by some factor. Unboosted minicolumns will use the factor 1.0. This delivers acceptable results. In this work, the Homeostatic Plasticity Controller takes also the control of synapse excitation in the Spatial Pooler into account. Every mini-column creates a set of synaptic connections to the input neurons. These synapses are strengthened in every learning cycle if the synapse is connected to the active input neuron. If the mini-columns synapses do not connect often enough to the active input neurons, then the value of the permanence of synapses (known as *weight* in classic neural networks) is incremented. We call the process of this increment "weak synapse excitation".
In this work, the weak synapse excitation was also switched off to initiate the exit of the *newborn* stage of the Spatial Pooler. This makes sure that the stable Spatial Pooler will no longer get unstable in future learning iterations if some mini-column does not uniformly activate synapses across the entire input pattern set.

3 RESULTS

To validate that the Spatial Pooler algorithm can be improved to reliably generate a stable state with the help of the Homeostatic Plasticity controller, the following experiment was designed. The experiment (see Listing 1) executes 25000 iterations and presents 100 scalar values to the SP. It was repeated more than a thousand times for various configurations. The scalar encoder used in line 11 is configured with the set of parameters (line 5) described in Table 2. Every input value (0-100) will be encoded as the vector of 200 bits. Also, every single value from the specified range will be encoded with 15 non-zero bits as shown in
Figure 1 - right.

Listing 1. Using of improved SP - Pseudo code

```
 0 | function Experiment( inputSet )
 1 | begin ( I )
 2 |   p          // Set of SP parameters.
 3 |  hp,enp    // Set of HPC and encoder parameters
 6 |  isStable = false
 5 |  en←create(enp)
 6 |  hpc ←create(hp, onStateChange);
 7 |  sp ←create(i, hpc);
 8 |  FOR i = 0; i<25000
 9 |     FOREACH i IN inputSet
10 |        // Generate SDR for the input.
11 |        o ← sp.compute(encode(i));
12 |        IF isStable = true
13 |           // newborn stage exited
14 |           // Use stable SDRs. Custom code here.
15 |        ENDIF
16 |     ENDFOREACH
17 |   ENDFOR
18 | end
19 |
20 | function onStateChange(state)
21 | begin
22 |    isStable = true // Indicate the stable state
23 | end
```

The instance of the Spatial Pooler (line 7) with the common set of parameters (line 3) has been created. The same configuration was used in the experiment described in the previous section, which produced results shown in Figure 2 and Figure 3. The Homeostatic Plasticity Controller (line 6) is typically attached to the Spatial Pooler instance (line 7, second argument) and used inside of the compute method.
The Homeostatic Plasticity Controller (HPC) requires the *callback* function (line 6, second argument), which is invoked when the controller detects the stable state of the Spatial Pooler. The experiment is designed to execute any number of training iterations (line 8 defines 25000 iterations). In every iteration, the Spatial Pooler is trained with the whole set of input values **I** (line 9).

The spatial input is trained in line 11. The output of the training step in line 11 is an SDR code (set of active mini-columns) associated with the encoded input value i. Before being presented to the Spatial Pooler, the input value i is encoded by the Scalar Encoder configured with the named set of parameters shown in Table 2. The encoder is represented as a function e that converts the given scalar value to the binary array:

$$e: \mathbb{R} \longrightarrow \{0,1\}$$

The computation inside of HTM operates exclusively on binary arrays as the neocortex does it. The existing SP compute algorithm is in this work extended to invoke the algorithm implemented in the Homeostatic Plasticity Controller (HPC) shown in Algorithm 1. The HPC computation takes place after the Spatial Pooler has computed the current iteration.

The HPC Algorithm 1 starts with two inputs. The first one is the binary array of an encoded input pattern and the second one is the SDR as calculated by the SP for the given input. Table 2: Scalar Encoder parameters

Parameters	Value
W – Bits for coding of the single value	15
N – Input bits	200
MinVal	0
MaxVal	100

In the beginning, the algorithm does not perform any change in the SP. This period is called the *newborn* stage. The Homeostatic Plasticity Controller will disable the boosting in the Spatial Pooler after the minimum required number of iterations ***m*** (minCycles) is reached (line 15). When the iteration number is larger than ***m***, the boosting is disabled by setting parameters DUTY_CYCLE_PERIOD and MAX_BOOST to zero. These parameters update the boost factors for every single column in every iteration. The boost factors are used in the Spatial Pooler to increase the number of connected synapses (overlap) of inactive columns. Increased overlap of inactive columns improves the chance of the column becoming active.

Algorithm 1: Computation in HPC

01 | input: $\boldsymbol{i}$ // Set of neural cells. I.e., sensory input.
02 | output: $\boldsymbol{o}$ // Set of active columns - SDR.
03 | configuration:
04 | b // SP max boost
05 | d // SP min pct. overlap duty cycles
06 | **begin**
| // Calculate the hash value of the input of N bits.
07 | H ← $hash(\boldsymbol{i})$;
| // Calculate the number of active columns in SDR.
08 | E ← (H, $\sum_{k=0}^{M} i_k)\ o_k \in \boldsymbol{o}$)
| // The average change of num. of the act. columns.
09| $\delta \leftarrow \frac{1}{p} * \sum_{k=0}^{p-1} | \mathcal{E}_{Hk} - \mathcal{E}_{H(k+1)}| \ \mathcal{E}_H \in$ E
| // Calculate the correlation of the current and the previous output.
10| $\boldsymbol{c} = corr(\boldsymbol{o}', \boldsymbol{o}) \mid \boldsymbol{o}' \in \mathcal{H}$
| // Store input-hash and SDR pair
11| $\mathcal{H}$←(H, o)
| // Increment the counter of stable iterations for ***i***.
12| Γ← $\gamma_H + 1[\delta = 0, \boldsymbol{c} > \theta | 0.9 < \theta < 1, \gamma_H \in \Gamma]$
| // Fire stable state event
13| *StableState* [$\gamma_H = \tau, \forall\, \gamma_H \in \Gamma,\ \tau \in$ **N**]
| // Reset the counter of stable iterations for ***i***.
14| Γ← $0_H[\boldsymbol{c} \leq \theta | 0.9 < \theta < 1.0]$
| // Disable boost and synaptic excitation after specified num. of iterations.
15| inactiveColumnBoost=**off**;synapseExitation=**off** [$iteration >= m$]
16| **end**

After disabling the plasticity the algorithm starts tracking all seen patterns and their associated SDRs. To avoid the saving of the entire input dataset internally, function $hash$ calculates the hash value (line 6) over the sequence of bits of the input in the current iteration. The calculated hash-value is a sequence of bytes defined as a set H (line 7). In line 8 the tuple of the input's hash value H and the number of active columns of the corresponding SDR is associated with the set E. The set E remembers *p* tuples of every input. As discussed in the previous section, the goal is to keep the number of active columns uniform across all generated SDRs. The value δ is the average change of the number of active columns in each SDR in the cycle interval *p* (line 9).
The cycle interval *p* is the number of previous iterations used to calculate the δ. In most experiments, this value was set to five. This is an acceptable value because the SP does not change the number of active columns, once the stable state is entered.

The value δ is calculated as an average sum of deltas $\mathcal{E}_{Hk} - \mathcal{E}_{H(k+1)}$ in the last *p* iterations for the given input hash value H.

$$\delta = \frac{1}{p} * \sum_{k=0}^{p-1} | \mathcal{E}_{Hk} - \mathcal{E}_{H(k+1)}| \ \mathcal{E}_H \in \text{E}$$

Having this value zero is the first condition of the stability of the new Spatial Pooler. This value is zero if the number of active columns of the SDR of the same input does not change over time defined by the number of iterations *p*.

The second condition for stability of the Spatial Pooler is the achieving of the constant SDR for every input seen by the Spatial Pooler during the entire training process. For this reason, the set $\mathcal{H}$ is used to keep tuples (H,o) of input hash values and their SDRs. SDRs of inputs in upcoming iterations override the previously-stored tuple of the current input. There is always a single tuple (H,o) for every input inside of $\mathcal{H}$. Tuples in $\mathcal{H}$ are used to calculate the correlation $\boldsymbol{c}$ between the previous and the current SDR of the given input (lines 10, 11). If the correlation between the last SDR (output of learning cycle for the given input pattern) $\boldsymbol{o}'$ and the new (current) SDR $\boldsymbol{o}$ of the given input $\boldsymbol{i}$ is larger then the specified tolerance threshold θ (typically near 100%) and the first condition $\delta = 0$ is fulfilled, then the counter of stable iterations of the given input $\boldsymbol{i}$ is incremented (line 12). If the correlation between $\boldsymbol{o}$ and $\boldsymbol{o}'$ is under the tolerance threshold θ, the number of stable iterations for the given input pattern is reset to zero. The threshold θ can theoretically be set to 1.0. However, the SP internally always select the specific number of active mini-columns. If two mini-columns have the same overlap, they will compete for activation. The selection of competing mini-columns is a random process, which can lead to the selection of a different mini-column in learning cycles. The HPC algorithm encounters this behaviour and builds in the explicit tolerance defined by θ less than 1.0. The better solution here would be to change the SP algorithm to allow side by side activation of competing mini-columns, which would be a probably more biological way of activation than the current one.

The second condition that corresponds to the stable state of the Spatial Pooler is fulfilled if the γ_H (number of stable iterations) reach the defined threshold τ (line 13) for every seen input during the training process. In all experiments, τ = 50 was used. In most experiments, the chosen value was between 15 and 150. Every time the correlation value is less than threshold τ the counter of stable iterations γ_H for the given input is reset.
After entering the stable state all generated SDR-s should remain unchanged for the entire lifetime of the Spatial Pooler instance. The SP is defined as stable if both described conditions are satisfied: Uniform number of active cells in all SDRs and Required number of stable iterations for all SDRs is reached.

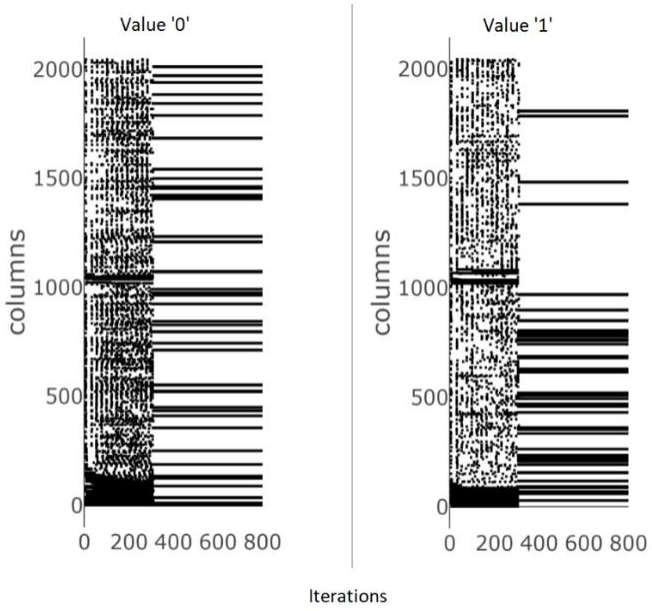

Figure 6: Spatial Pooler in the stable state representing two SDRs of two input pattern examples with the activated Homeostatic Plasticity Controller.

The implementation of the algorithm of HPC [18] continues to track the stability after the SP has reached a stable state. Results show that the extended Spatial Pooler with HPC algorithm gets always stable with the uniformly distributed number of active columns for all SDRs.

Figure 6 shows the SDRs of two coincidently used spatial input samples. Values '0' and '1' are both encoded with the stable SDR after approx. 300 iteration. As shown in the figure, generated SDRs are unstable in the first 300 hundred iterations. Active columns which encode SDRs are in the first 300 steps continuously changed. This iteration interval is called HTM *newborn* stage and it is defined by the parameter $\boldsymbol{m}$ (line 15). In this stage the stimulation of mini-columns and weak synapses is active and SDRs of all inputs are changing frequently during the learning process (first approx. 300 hundred cycles in Figure 6). After approx. 300 cycles the HPC disables the stimulation and SDRs converge very quickly to the stable state, which remains during the life cycle of the Spatial Pooler. In this experiment, tests were done with up to 30000 iterations. The SP remains stable with one exception. Some experiments show that SP can also get unstable shortly after entering the stable state.

This instability is according to the design of the HPC algorithm enforced when the currently processing input changes its SDR. The HPC will in this case reset the counter of stable iterations for the given input (line14), which will declare the SP as unstable. When this exception occurs, the learning can continue until the SP enters the stable state again for the entire life cycle of the SP instance. This unwanted behaviour occurs mostly when the chosen number of minimum required iterations $\boldsymbol{m}$ is too low. Choosing larger values for $\boldsymbol{m}$ seems to solve this exceptional behaviour but it takes a longer time to leave the *newborn* stage and enter the stable state. However, even when choosing higher values for $\boldsymbol{m}$, the SDR of a pattern might slightly change over time.

The HPC uses the tolerance threshold θ to decide when the SP gets stable or unstable. SP chooses the set of active mini-columns by sorting them by overlap. During the learning process, some synapses between columns and input neurons might increase their permanence. This can increase the overlap of some mini-columns and include them in the set of active mini-columns. Because the SP holds constant the number of mini-columns per input pattern, some previously active mini-columns will be removed from the set of active mini-columns. For example, assume that the four-active-column SDR(i,t) of the input i at the stable iteration t is C1-10, C5-10, C15-9, C20-9. The first number is the index of the column and the second one is the overlap of the column. At the current iteration, column C14 has an overlap 8. It is not included in the SDR, because 4 mini-columns with the highest overlap build the SDR. During the learning process in the next iteration, column C14 might increase its overlap from 8 to 9. In this case, the SDR(i,t+1) will become C1-10, C5-10, C14-9 and C15-9. The previously active mini-column C20 with the same overlap 9 is now replaced with C14-9. With the tolerance threshold $\theta = 1.0$ this change would cause the HPC to degrade the stable SP to the unstable one in the iteration t+1. In contrast, the tolerance threshold $\theta = 0.75$, would keep the SP stable if a single mini-column in the SDR of four mini-columns is replaced. In most tests, we figured out that $\theta = 0.975$ by 40 active mini-columns of 2048 generates stable SP. Higher θ values cause the SP to temporary becomes unstable after a stable state.

Application developers should choose a reasonable value for their specific use case. Even if this value is not ideally selected, the HPC will notify the application when the SP gets unstable. With this, any required action can be performed inside of the application.

Figure 7 shows this behaviour. The HPC was configured in this experiment to use a relatively low iteration value $\boldsymbol{m}$=30 for the minimum required number of iterations for the *newborn* stage. This is typically a very short interval for a *newborn* stage. In the first experiment Figure 7 (left) θ =1.0 was used. It means no SDR change is allowed to keep the stable state. In the second experiment Figure 7 (right) θ =0.975 was used. It means the single mini-column can be replaced by 40 active mini-columns. In the first experiment, the SP entered the stable state in iteration 129 (left, green line), but, it got temporarily unstable in iteration 391 (left, red line), because the single mini-column was replaced. In the second experiment, the SP entered the stable state at iteration 84 (right, green line). Because in the second experiment with the θ =0.975, few column-replacements during the learning process are allowed. Figure 7 (right) shows stable SP, even if some columns are replaced after entering the stable state (right, blue line). In both cases, the SP behaves the same way, but HPC uses a different threshold to decide the iteration step of stability. Please note that in described experiments stability was entered by learning different inputs.

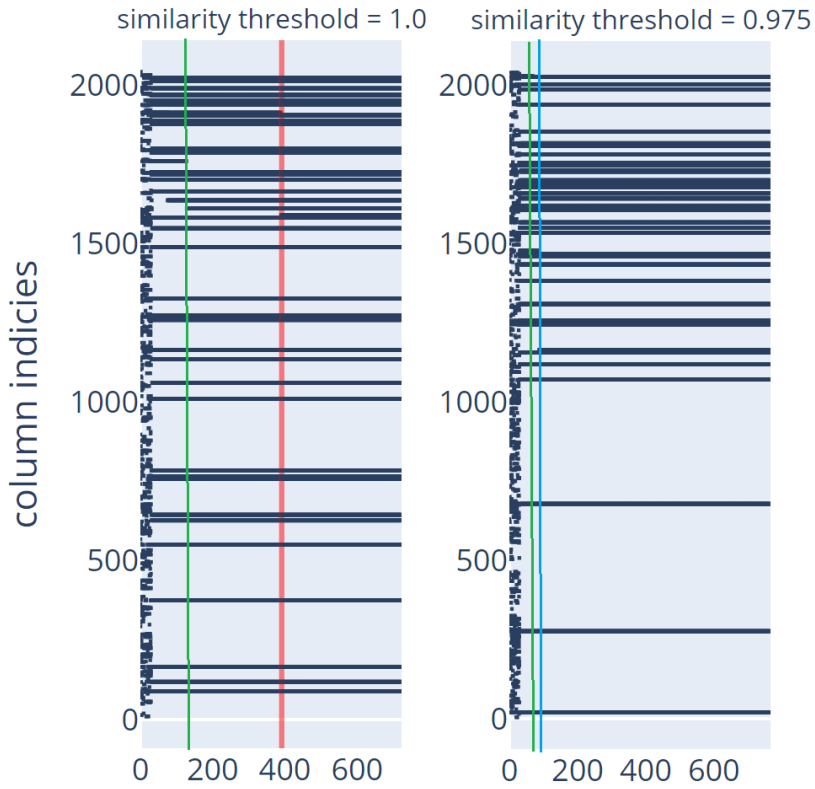

Figure 7: Spatial Pooler soon after entering the stable state become temporarily unstable for some input patterns in few learning iterations. After a few iterations, the SP becomes stable again and remains in a stable state. Used similarity threshold $\theta = 0.97$ and $\theta = 1.0$

In this work, it was also analysed how stability is reached across the entire input data set. To test this, the HPC trace was created at the iteration that occurred long after entering the stable state. The trace contains the number of stable cycles for all patterns. As shown in Figure 8, all patterns have mostly a different number of stable cycles. In this experiment, the SP entered the stable state at iteration 441 (shown previously in Figure 7) and the experiment was stopped at iteration step 4012, which is in the stable state. The minimum number of stable states 3637 was detected for the input 64 and the maximum number 3962 of stable states was detected for multiple numbers 0,1,2,3,4,5,6 and some more. As described, the SP stability in experiments was defined by $\tau = 50$. Notice, by subtracting $4012 - \tau$ we get the maximum number of stable cycles. That means that inputs with the maximum number of stable cycles entered a stable state at the very beginning of the learning process. This shows that some patterns are stabilized very early and some others need a more cycles to learn the pattern even if all patterns have been uniformly presented.

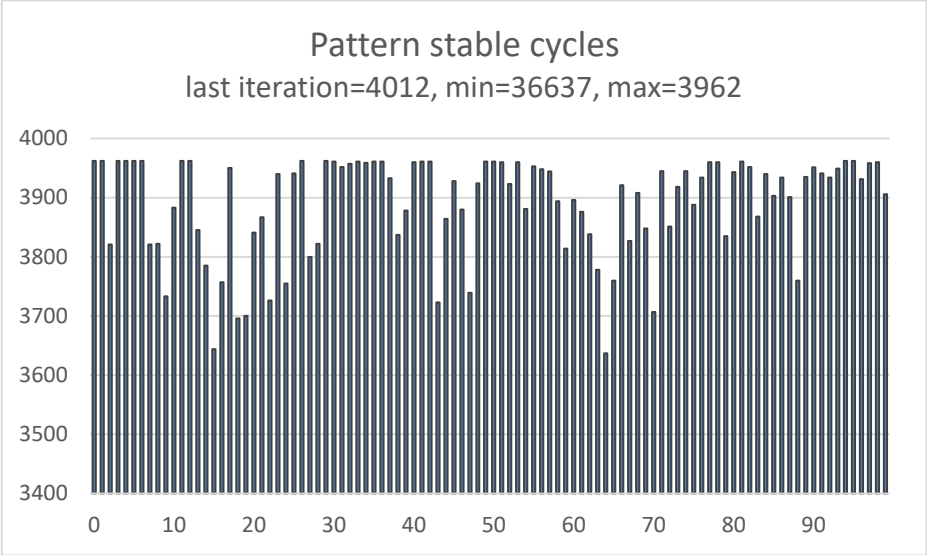

Figure 8. The internal trace of the Homeostatic Plasticity Controller showing a number of stable cycles of each generated SDR for the entire input data set.

Figure 9 shows all SDRs stored in the Spatial Pooler after entering the stable state. As mentioned, the set of encoded scalar input values 1-100 was used to train the SP. The x-axis shows input value 1-100. Y-axis shows the columns that are active for every of the given inputs. For example, black dots along the green line shows the SDR code of the scalar input value 60. When presenting the input 60 to SP, the mini-columns along the green line are activated.

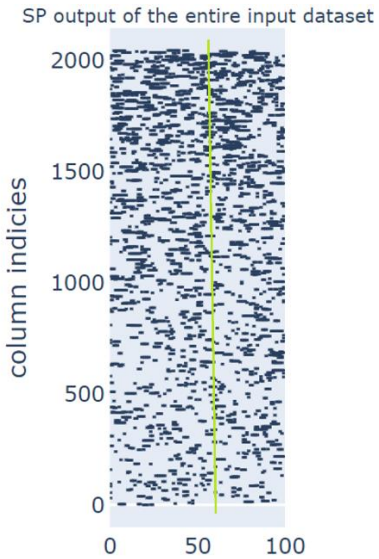

Figure 9 All SDRs at once. Representation of all generated SDRs seen in the training process of 100 input patterns. The horizontal axis shows the index of the input. The vertical axis shows the SDR. Every black "dot" represents the active mini-column.

4 CONCLUSION

The Hierarchical Temporal Memory algorithm is inspired by the neocortex and implements many known features that have roots in neuro-sciences. Nowadays many results show that the algorithm is very flexible and can solve different kinds of problems like sequence learning, anomaly detection, object recognition, classification etc. [2] [3] [4] [5] [6]. However, the reverse engineering of the neocortex is still a complex and unsolved task. All design decisions in the algorithm are based on findings in neuroscience, but some features still might be assumptions and work in progress. This paper focuses on the instability issue of the HTM Spatial Pooler algorithm when memorizing spatial patterns in an unsupervised way. As discussed, the original Spatial Pooler already integrates some sort of homeostatic plasticity mechanism discovered in previous work in neurosciences. However, the original algorithm caused instability in the learning process, which makes it very difficult to build reliable applications. This work briefly analysed this issue and provides the solution by extending the existing SP algorithm with the new component called Homeostatic Plasticity Controller (HPC). The new Spatial Pooler extended with the HPC is also motivated by findings in neurosciences, that document the activity of this mechanism during the development of the species. Inspired by this finding the new Homeostatic Plasticity Controller introduces the *newborn* stage of the Spatial Pooler. This work demonstrates how important is the high activity of plasticity in the early development phase of the cortical tissue in this work modelled as HTM. This plasticity mechanism has the disadvantage of disrupting the stable learning process, but at the same time ensures that the activity of the mini-columns is evenly distributed in the experimental tissue. In this *newborn* stage, the SP stimulates the inactive mini-columns and synapses connected to input neurons. The new HPC extension of the SP algorithm waits for a specified number of iterations (*newborn* stage) and then switches off the synaptic and mini-column stimulation mechanism waiting on the SP to enter the stable state. With this approach, the SP converges very quickly to a stable state. Applications can subscribe to the event that notifies about the state change of the SP. The immediate disabling of boosting is not a natural mechanism. However, if required, disabling of boosting can be easily adapted to slow down exiting the *newborn* stage in a more natural (biological) way. To conclude, the original version of the SP enters unexpectedly the unstable state that behaves similar to an "epileptic" state. The new SP with the HPC and the *newborn* stage controlled by HPC improves the overall quality of the learning of the SP and enables the implementation of more reliable solutions. Another work in progress in this context is related to the design of the parallel version of the HTM. The new HPC algorithm needs to be validated for parallel implementation [19].

5 REFERENCES

[1] Dobric, "GitHub," 2018. [Online]. Available: https://github.com/ddobric/neocortexapi.

[2] Hawkins, Subtei, "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex," *Frontiers in neural circuts,* 2016.

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing,* vol. 262, 2017.

[4] Maciej Wielgosza, Marcin Pietron, "Using Spatial Pooler of Hierarchical Temporal Memory to classify noisy videos," *aAGH University of Science and Technology, Krak´ow, Poland,* 20`16.

[5] Deven Shah, Pinak Ghate, Manali Paranjape, Amit Kumar, "Application of Hierarchical Temporal Memory - Theory for Document Categorization," in *IEEE Xplore,* San Francisco, 2018.

[6] "Continuous online sequence learning with an unsupervised neural network model," vol. 28 (11), p. 2474–2504, 2016.

[7] Mountcastle, "The columnar organization of the neocortex," *Journal of neurology,* vol. 120, pp. 701-22, 1997.

[8] Yuwei, Subutai and Hawkins, "The HTM Spatial Pooler, A Neocortical Algorithm for Online Sparse Distributed Coding," *Frontiers in computational neurosciences,* vol. 11, p. 111, 2017.

[9] Damir Dobric, Andreas Pech, Bogdan Ghita, Thomas Wennekers, “Improved HTM Spatial Pooler with Homeostatic Plasticity Control,” in *Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods* , Vienna, 2021.

[10] Turrigiano and Nelson, “Homeostatic plasticity in the developing nervous system,” *Nature Reviews Neuroscience,* p. 97–107, 2004.

[11] Davis and Graeme, “Homeostatic Control of Neural Activity - From Phenomenology to Molecular Design,” *Annu. Rev. Neurosci.,* 2006.

[12] Davis, Graeme, “Homeostatic Signaling and the Stabilization of Neural Function,” *Neuron,* p. 09.044, 2013.

[13] Tien and Kerschensteiner, “Homeostatic plasticity in neural development,” *ND - Neural Development,* p. 13/9, 2018.

[14] Subutai, Hawkins, “How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites,” *ResearchGate,* 2016.

[15] Luca A. Finelli ,Seth Haney,Maxim Bazhenov,Mark Stopfer,Terrence J. Sejnowski, “Synaptic Learning Rules and Sparse Coding in a Model Sensory System,” *PLOS Computational Biology,* 2008.

[16] Maffei, Nelson, Turrigiano, “Selective reconfiguration of layer 4 visual cortical circuitry by visual,” *Nature neuroscience,* pp. 1353-9, 2004.

[17] Hawkins, Subutai and Cui, “A Theory of How Columns in the Neocortex Enable Learning the Structure of the World,” *Frontiers in Neural Circuits,* vol. 11, pp. 81-81, 2017.

[18] Dobric, “Implementation of Homeostatic Plasticity Controller,” 2020. [Online]. Available: https://github.com/ddobric/neocortexapi/blob/master/NeoCortexApi/NeoCortexApi/HomeostaticPlasticityController.cs.

[19] Dobric, Pech, Ghita and Wennekers, “SCALING THE HTM SPATIAL POOLER,” *International Journal of Artificial Intelligence,* vol. 11, no. 4, 2019.