Incorporating A Continuous Bayesian Learning Rule Into A Discrete Hierarchical Temporal Model

*

Leo Zeitler
KTH Royal Institute of Technology
Machine Learning
Stockholm, Sweden
llze@kth.se

Jakob Heyder
KTH Royal Institute of Technology
Data Science
Stockholm, Sweden
heyder@kth.se

***Abstract*—With the Hierarchical Temporal Memory (HTM) Model the company Numenta developed a cortical algorithm that is based on biological evidence while being scalable. Yet it comes with several simplifications that have a high level of abstraction and include the drawback that additional parameters like thresholds need to be incorporated. One of these simplifications is the application of binary activation values and synaptical weights. In this work we aim to extend the given approach to a continuous stochastic network that uses a Hebbian-like Bayesian learning rule which adapts to changes in the input statistics while reducing the set of network parameters. Although our framework permits the mathematical investigation of the network dynamics, we were not able to reproduce the original results with a similar scalability. We are convinced that a custom hardware and software implementation of the Bayesian approach could increase the efficiency of the matrix calculations and make it feasible for larger simulations.**

***Index Terms*—BCPNN, HTM, Bayesian, neural network, Numenta**

I. Introduction

Cortical algorithms and information processing inspired by cortical function and structures in the mammal brain have become a flourishing research topic and got increasing attention in the last five years, not least because of the media attention the Human Brain Project [1] received. However, it is still very difficult to develop a solution that handles the trade-off between scalability and abstraction reasonably well. The Numenta foundation developed the Hierarchical Temporal Model (HTM) that intends to mimic the architecture and signalling of the human neocortex while being scalable and efficient [2]–[4]. Yet the solution comes with several simplifications compared to mathematical biological models, inter alia the assumption that synaptical weights and neural activity are modelled binarily; plasticity is implemented via a continuous value that is increased or decreased about a constant value and clipped off at zero or one. This delta value, called *permanence value*, can be chosen arbitrarily and is another additional parameter whose tweaks can change significantly the performance of the algorithm. This work aims to embed a Bayesian plasticity rule that is automatically adapting to changing input statistics, as it was proposed and implemented in the Bayesian Confidence Propagation Neural Network (BCPNN) [5]–[7]. The goal is to compare the approaches regarding their respective convergence behavior. We hypothesised that the network will achieve similar results, while increasing interpretability and reducing the number of model parameters that are needed to be tuned. The HTM and BCPNN as well as our adaptions are described in Section II. The methods and the experimental setup applied in this scientific work are described in Section III. Unfortunately, we were not able to adapt the Numenta framework in a way that shows a similar performance as the original approach. Yet through our modifications, in particular the usage of continuous activation values and weights, we formed nevertheless a theoretical framework that maps the HTM to a Bayesian propagation network which can be investigated mathematically. Our results and the outlook to future research questions are discussed in Section IV and V.

II. Model

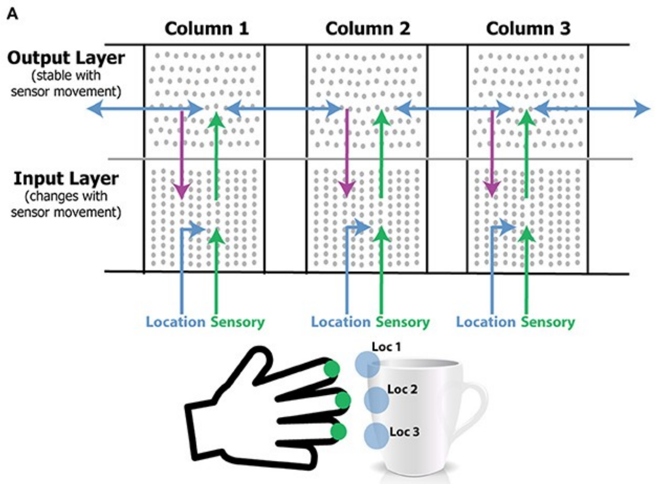
In this section we describe the HTM system as published by Hawkins et. al. [3], [8], [9] and the BCPNN [6], [7]. After presenting the core components we will explain the derivation and adaption of the HTM model with the incorporated Bayesian learning rule.

A. Hierarchical Temporal Memory

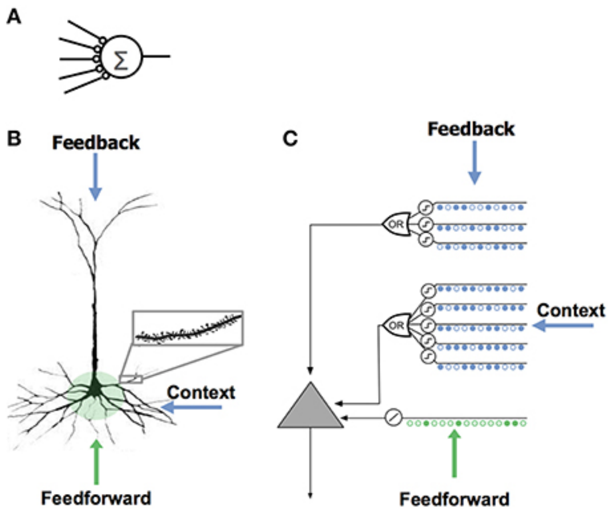
HTM is an unsupervised, online learning algorithm which is biologically constrained. As a key unit it uses the HTM neuron, a pyramidal neuron, that establishes connections via a local, Hebbian-like learning rule. While there exist multiple versions of the framework, this work focuses on the network architecture introduced by Hawkins et. al. [3].

The HTM network as depicted in Figure 1a consists of one or multiple cortical columns, each processing a subset of the input space. A cortical column is composed of an input and an output layer. The main components are the HTM neuron, Temporal Memory (TM) and Column Pooler (CP).

The input layer of each cortical column constitutes an implementation of the TM. It consists of HTM neurons arranged



(a) The HTM network architecture contains one or more laterally connected cortical columns. Each column receives feed-forward sensory input from a different subset of the input space. The input layer combines sensory input with a modulatory location input to form sparse representations that correspond to features at specific locations on the object. The output layer receives feed-forward inputs from the input layer and converges to a stable pattern representing the object. Feedback from the output layer to the input layer allows the input layer to predict what feature will be present after the next movement of the sensor.



(b) The dendrite model with the differences to conventional McCulloch-Pitts neurons and biological neurons. The HTM-Neuron models proximal input (feed-forward), apical/lateral modulatory input (feedback) and distal modulatory input (context).

Fig. 1: Figure (a) depicts the HTM architecture and Figure (b) shows illustration of the HTM-Neuron model. The figures are taken from Hawkins et. al. [3], [8].

in mini-columns, where each mini-column receives the same feed-forward input which represents object sensations. The distal input to the TM is regarded as the location signal of the sensation on the object (see Section II-A2). Within each mini-column a cell learns to represent an input pattern (sensation) within the context of a location.

The output layer also contains HTM neurons, but they are not aligned in mini-columns. It implements the CP algorithm, which receives feed-forward input from the input layer and distal input of the previous time step from other output cells of the same cortical column as well as from neighboring cortical columns (see Section II-A3). During learning, the set of cells in the output layer remains active over multiple input patterns. Thus, each object representation in the output layer has an associated set of input layer activations, which represent sensory features at locations. During inference, convergence is achieved via two means. One is by integration over time as the sensor moves relative to the object, and the other is via lateral connections between columns that are simultaneously sensing different locations on the same object.

1) HTM neuron: The HTM neuron as depicted in Figure 1b can be utilised to learn sequences and make predictions over time in a single layer network [8]. It incorporates dendritic properties of pyramidal cells, which have proximal segments, that transmit feed-forward signals, and apical and basal segments, that receive contextual input. The proximal dendrites represent feed-forward input that can activate a neuron. Input received over the basal or apical dendrites amounts to context information, referred to as *modulatory input*, which will depolarize the cell. This is modeled as a predictive state, which causes the cell to fire sooner than non-depolarized cells and thereby inhibits them. Thus the HTM neuron can be in three different states: active, predictive or inactive. As basal and apical segments work similarly we will describe only the basal calculation here. The learning rules and activation formulas of the different states are further described in the next section II-A2 of the TM.

2) Temporal Memory (TM): The TM uses the HTM neuron to learn input in different contexts. In order to do so, a cell creates segments that receive modulatory input from basal or apical connections. Instead of having continuous weights between neurons and segments, the Numenta approach uses binary synapses, defining the existence of a certain link. The connections a segment can possibly establish to other neurons in the TM are defined in a set called *potential synapses*. A value referred to as *permanence value* gives a measure of reinforced activity. It is increased about the constant value p^+ for correctly predicted cell activity, whereas it is decreased about p^- for predicted cells that were not receiving feed-forward input. If the permanence value, which is forced to be between zero and one, is larger than a threshold θ_p the synapse becomes active.

Assuming we have N mini-columns in the input layer with M cells per mini-column and an N^{ext} dimensional sparse

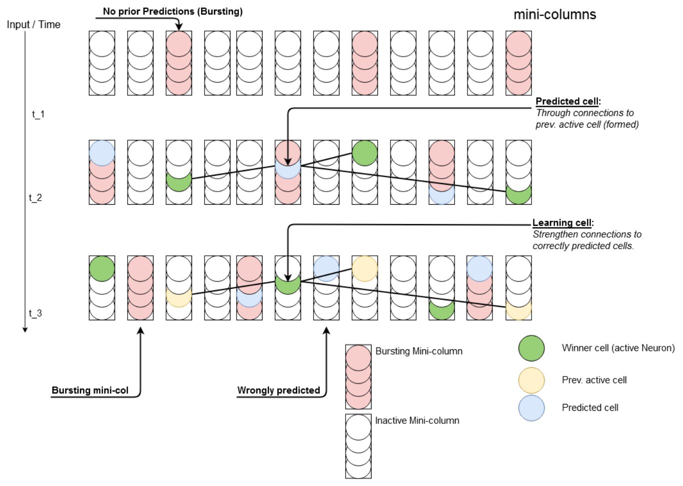


Fig. 2: Illustration of the TM. Each active mini-column is sparsified to individual cells representing the input in the specific temporal context. The three possible cases for a column are shown: (1) It has correctly predicted cells and is active, (2) It has no predictive cells and bursts and (3) It is inactive and has falsely predicted cells. The learning in every active segment reinforces connections to prior active cells and decrements inactive connections. Additionally, new synapses can be grown for each segment.

vector B^t representing the modulatory input at time step t. The set of active cells is represented by the $M \times N$ binary matrix A^t at time step t. Similarly, π^t represents the set of predictive cells. Further, let the N^{ext}-dimensional sparse vector D^d_{ij} denote the permanence values to the modulatory input of the d-th distal segment for the i-th cell in the j-th mini-column. $\tilde{D}^d_{ij}$ denotes the binary vector representing the subset of connected synapses if a connection threshold is applied to the permanence values.

There are two phases in the TM algorithm. Firstly, there is an inference phase to calculate the cell states based on the activated mini-columns and modulatory input, and secondly, a learning step which updates the permanence values and possibly adds new segments.

Inference phase: We denote the active mini-columns, which are identical to the feed-forward input with W^t. Then the activation of a cell can be calculated as follows:

$$a^t_{ij} = \begin{cases} 1 & \text{if } j \in \mathbf{W^t} \text{ and } \pi^t_{ij} = 1 \\ 1 & \text{if } j \in \mathbf{W^t} \text{ and } \sum_i \pi^t_{ij} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The cell will be active if the mini-column receives feed-forward input and it is in a predictive state or alternatively if the mini-columns is active and none of the cells within was predicted (referred to as *bursting*). The set of predicted cells π^t for time step t is calculated:

$$\pi^t_{ij} = \begin{cases} 1 & \text{if } \exists_d \, \|\tilde{D}^d_{ij} \circ B^t\|_1 > \theta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

A cell will be in a predictive state if at least one segment is active. A segment is active if there are at least θ (threshold) pre-synaptic cells in the modulatory input with a connected synapse. This is calculated with an element-wise multiplication, indicated by $\circ$.

Learning phase: HTM uses Hebbian-like learning by increasing the permanence values for correct predictions and punishing incorrect predictions. It only learns on segments for cells that were predicted and became active, i.e. winner cells of the mini-column. In columns that bursted, i.e. had no predictions and thus activated all neurons, it will select the neuron as winner that was closest to becoming active to represent the context in a newly added segment. For selected segments the permanence is increased by a larger p^+ if the segment was active and decreased by a smaller p^- if the segment was inactive:

$$\delta D^d_{ij} = p^+ D^d_{ij} \circ B^t + p^- D^d_{ij} \circ (1 - B^t) \tag{3}$$

The matrices δD^d_{ij} are added to the current matrices of permanence values at every time step. This deals with cells that became active (selected segments). Additionally a small decay is applied to active segments of cells that did not become active (Equation 4). This can happen if segments were mistakenly reinforced by chance.

$$\delta D^d_{ij} = p^- D^d_{ij} \text{where } a^t_{ij} = 0 \text{ and } \|\tilde{D}^d_{ij} \circ B^t\|_1 > \theta \tag{4}$$

3) Column Pooler (CP): The CP algorithm maps a union of patterns in the input space to a constant activation. It has an inference and learning mode, where the former implements a voting mechanism by combining feed-forward and modulatory support to infer active neurons in the output layer.

During learning it keeps the activation constant and learns connections to a union of input patterns on proximal connections by strengthening the permanence values of active synapses and punishing inactive ones. Additionally, it has modulatory connections where the vector B^t represents the previous activation in the output layer (i.e. A^{t-1}). The permanence values are adapted similarly to the TM.

During inference, cells with sufficient feed-forward activation and the most distal/lateral support from the previous time step become active. Lets denote W^t as the set of cells with enough feed-forward input (greater threshold). Then the active cells are selected by sorting over the modulatory support (activations):

$$a^t_i = \begin{cases} 1 & \text{if } i \in W^t \text{ and } \xi^{t-1} \leq p^{t-1}_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where p^{t-1}_i represents the number of active modulatory segments in the previous time step, and the s-th highest number of active segments is denoted as ξ^{t-1}.

B. Bayesian Confidence Propagation Neural Network

The Bayesian learning rule that is derived from a naive Bayesian classifier for a one-layer feedback neural network was first proposed by Lansner [5] and was extended to an online training approach in an attractor network by Sandberg et al. [6], [7]. The fundamental key concept is the interpretation of input and output values as confidence in detecting a specific feature and the posterior probabilities of outcomes, respectively. This allows the application of a Hebbian-like Bayesian learning rule that uses the probabilities of activation and co-activation to update the respective bias terms and weights. Assuming that we have n input values X_i we can calculate the posterior probability $\hat{\pi}_j$ of the j-th unit via

$$\begin{aligned} \log \hat{\pi}_j &= \log P(y_j) + \sum_i^n \log \left(\frac{P(y_j, x_i)}{P(y_j)P(x_i)} \right) P_{X_i}(x_i) \\ &= \beta_j + \sum_i \log w_{ij} P_{X_i}(x_i) \end{aligned} \tag{6}$$

where $P_{X_i}(x_i)$ represents the a priori probability. The Bayesian perspective permits the derivation of a stochastically motivated learning rule.

A straight-forward implementation is given by counting the frequency of a particular neuron being active [5]. We assume we have a given number z of training patterns $\boldsymbol{\xi}^p$ with their components ξ_i^p. We obtain

$$c_i = \sum_p^z \xi_i^p \tag{7}$$

$$c_{ij} = \sum_p^z \xi_i^p \xi_j^p \tag{8}$$

which gives us the estimated activation and co-activation probabilities $p_i = \frac{c_i}{z}$ and $p_{ij} = \frac{c_{ij}}{z}$. Since we want to apply the logarithm to these probabilities we need to pay special attention to counters that are zero. If neurons i and j have been active before ($c_i, c_j \neq 0$), but they have never been active simultaneously, we model the inhibition by setting w_{ij} to a large negative value, e.g. $\log \frac{1}{z}$. If one of the neurons has neither been active (e.g. $c_i = 0$), there is no co-dependence and the weight is set to $w_{ij} = 0$. Similarly, since the bias term models the excitability of the cell, it should be set to a large negative value, e.g. $\log \frac{1}{z^2}$, if $c_i = 0$. The weights are then calculated through

$$w_{ij} = \begin{cases} 1 & \text{if } c_i = 0 \text{ or } c_j = 0 \text{ or } i = j \\ \frac{1}{z} & \text{if } c_{ij} = 0 \\ \frac{c_{ij} z}{c_i c_j} & \text{otherwise} \end{cases} \tag{9}$$

while the bias terms can be estimated via

$$\beta_i = \begin{cases} \log \frac{1}{z^2} & \text{if } c_i = 0 \\ \log \frac{c_i}{z} & \text{otherwise} \end{cases} \tag{10}$$

In the following we will refer to this approach as *summing Bayesian* learning rule.

Sandberg proposed an incremental update approach to deal with continuously arriving data using exponentially smoothed averages Λ_i for activation and Λ_{ij} for co-activation (see [6], [7]). Since the capacity of an auto-associative attractor network is limited, the learning rule aims to give more weight to recent than to remote information, while smoothing out and adapting to longer trends. The mathematical formulation of the training procedure can be described as followed

$$\frac{d\Lambda_i}{dt} = \alpha(((1 - \lambda_0)\hat{\pi}_i(t) + \lambda_0) - \Lambda_i(t)) \tag{11}$$

$$\frac{d\Lambda_{ij}}{dt} = \alpha(((1 - \lambda_0^2)\hat{\pi}_i(t)\hat{\pi}_j(t) + \lambda_0^2) - \Lambda_{ij}(t)) \tag{12}$$

$$\beta_i = \log \Lambda_i(t) \tag{13}$$

$$w_{ij} = \frac{\Lambda_{ij}(t)}{\Lambda_i(t)\Lambda_j(t)} \tag{14}$$

where λ_0 describes some small additional background noise to avoid taking the logarithm of zero, and α denotes the learning rate. We will refer to this approach as *incremental Bayesian* learning rule.

The incremental learning rule has the desirable property that it converges towards $P(x_i)$ and $P(x_i, x_j)$ in a stationary environment. In case of absence of any input the weights become one for large t, representing no activation (or only background noise) of the cell. Yet the initialisation is arbitrary which is disadvantageous if only few training patterns are available or only few iterations are possible.

C. HTM Bayesian Adaptation

The HTM framework by Numenta and the probability-theory based BCPNN approach have some fundamental differences that make it challenging to simply re-use findings and proofs from Sandberg's work [6]. Particularly difficult is the adequate mapping from continuous values to binary weights, as well as the modelling of different dendrite segments to represent context. Unfortunately, as long as these fundamental assumptions remain untouched it is infeasible to use the merit of interpreting the input and output as a priori or posteriori probabilities. Thus, next to incorporate the Bayesian learning rule into the HTM our main focus is to extend the framework to use continuous weights and activation values.

A straightforward adaptation is to apply the permanence values directly as weights instead of applying an arbitrary threshold to obtain binary synapses. Thus, we can use the Bayesian learning rule for the weights without any additional transformation. Moreover, due to the fact that we use continuous weights we map the input to a continuous image space that is, if the bias terms and weights are derived via the Bayesian learning rule, between zero and one. This permits the interpretation of every segment in the input layer and every neuron in the output layer as a naive Bayesian classifier. Yet, as explained in Section II-A, every neural cell has several dendrite segments to model equal input in different contexts. In the original framework a single cell is assumed to be in a predictive state as soon as it has at least one active segment.

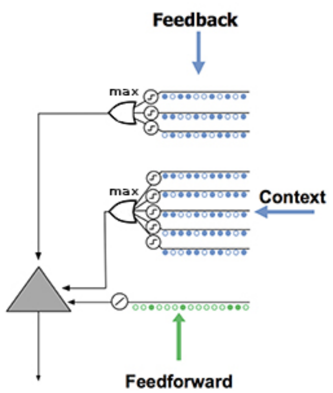


Fig. 3: The adapted dendrite model that takes the maximum over the continuous activity values of all dendrites to determine the cell's activity. For visualisation purposes we changed the original figure, presented in 1b

To avoid using an additional threshold to define an activation value for a cell in the prediction step we take the maximum activation probability of all segments. Let $(\hat{\pi}_1^i ... \hat{\pi}_k^i)$ be the activation values of all segments related to a particular neuron, calculated via equation 6. The activation a_i of neuron i can then be calculated through $a_i = \max(\hat{\pi}_1^i, ..., \hat{\pi}_k^i)$. Figure 3 shows the adaptation of the HTM neuron that was used to determine the cell's activity.

As explained in Section II-A2, next to the prediction step there is the activation step which is used to determine, through feed-forward input, which cells remain or are forced to become active or inactive. Unfortunately, to distinguish between predicted and non-predicted neurons it is necessary to define a threshold θ_a above which a cell is seen to be predicted. Assuming there are l neural cells per mini-column, it is reasonable to set this threshold to $\frac{1}{l}$. Using an uninformative prior every neuron is equally likely to be active. The activation of all cells is normalised by the sum over all activation values in a mini-column to induce competition between contexts. Cells in mini-columns that receive feed-forward input remain in their respective state, whereas all cells in all mini-columns without any proximal input are set to zero. To model the bursting of a mini-column (see Section II-A2), the cell with the lowest number of segments is chosen to be active and set to one. After calculating the posteriori activation of the cells according to eq. 6, the moving averages or the counters are updated and the weights and bias terms are re-calculated. Figure 4 gives a schematic diagram to visualise the learning concepts. Figure 4a explains the activity and learning concept used by the HTM, whereas Figures 4b and 4c exemplify the summing and incremental Bayesian approach, respectively.

The output layer works similarly. Proximal, distal and internal distal weights and bias terms are adapted through the Bayesian learning rule. The posteriori activation is mainly influenced by the proximal input that is calculated via eq. 6. To model forgetting of activation values that have not received sufficient activation over a given amount of time we implemented a decay value ϕ (named *forgetting*) applied to all all neurons in the output layer before the activation is updated. If the activation drops below a given threshold θ_o the neuron is assumed not to participate in a given firing pattern.

Special attention should be paid to any kind of support used in the input and output layer, either incorporated via context-dependent feedback information from the output to the input layer (transmitted through apical dendrites, in the following also referred to as apical information, see Section II-A1) or through activation of other output cells in the same or other cortical columns in the L2. We chose to model support as an additional weight that is multiplied to the unsupported activation; that is if $\hat{\pi}_j$ is the unsupported activity of the cell j and $\hat{\pi}_j^{sup}$ the support (e.g. feedback via apical dendrites), both calculated via eq. 6, the cell's activation $\bar{\pi}_j$ is defined through

$$\bar{\pi}_j = \hat{\pi}_j \hat{\pi}_j^{sup} \tag{15}$$

Although the total activation is likely to be decreased since we multiply probability values between zero and one it has the desirable property that activation with large support is only marginally decreased whereas low support forces the activation towards zero. Unfortunately, it is not possible to interpret the values as pure a priori / posteriori probabilities no more. Yet the weighted probabilities can be seen to represent the joint probability of support and activation assuming they are independent (which is a strong and usually violated assumption). Support in the output layer is only used during object prediction in inference mode. The prediction is dependent on all neurons with a supported activation above a support threshold θ_s.

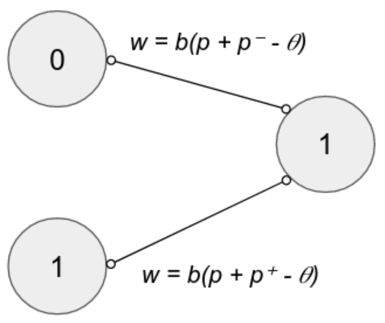
III. Methods

A. Experimental setup

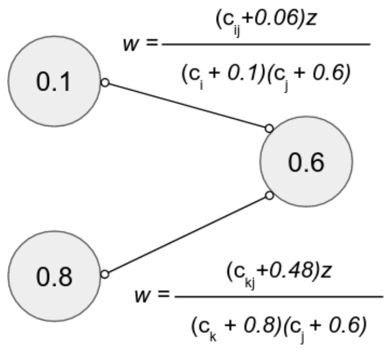
This work uses a similar experimental setup as Hawkins et. al. [3], which is depicted in Figure 5. The network structure consists out of a single cortical column with 1024 mini-columns and a varying number of cells per mini-column. The output layer has between 128 and 1024 neurons. The maximal number of dendrite segments was set to 16 (the justification is given in Section IV). Sparsity, one of the Numenta key features, was implemented through limiting the feed-forward input to only ten mini-columns at any activation step.

Cells in the input layer L4 transmit their activity via proximal connections to the output layer L2. Support in L2 is modelled via interconnections between neurons. The output layer projects back to L4 via apical connections. To investigate the effect of the incorporated support, some experiments do not use feedback information. Yet all connections are continuously learnt throughout the training phase.

The network is trained on a library of objects that is provided by Numenta. Since the original framework was proposed to recognise objects through touch at a location, each input is considered to be a sensation-location pair. A sensation itself is a vector $\mathbf{s}$ with $s_i \in \{0, 1\}$ that has a particular location on the

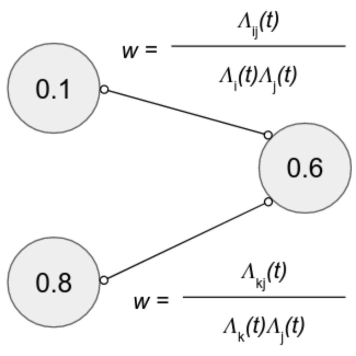


(a) The learning rule used by Numenta. $b(x)$ denotes the binary function that is 1 if $x > 0$ and 0 otherwise. Note that we assume p^- to be negative.



0.1
0.6
0.8
$w = \frac{(c_{ij}+0.06)z}{(c_i + 0.1)(c_j + 0.6)}$
$w = \frac{(c_{kj}+0.48)z}{(c_k + 0.8)(c_j + 0.6)}$

(b) The learning rule implemented by the summing Bayesian approach. $c_i, c_j, c_k, c_{ij}, c_{ij}$ are the activity counters described in eq. 7 and eq. 8 and z is the number of patterns that have been learnt so far.



0.1
0.6
0.8
$w = \frac{\Lambda_{ij}(t)}{\Lambda_i(t)\Lambda_j(t)}$
$w = \frac{\Lambda_{kj}(t)}{\Lambda_k(t)\Lambda_j(t)}$

(c) The learning rule implemented by the incremental Bayesian approach. $\Lambda_i, \Lambda_j, \Lambda_k, \Lambda_{ij}, \Lambda_{kj}$ are described in eq. 11, 12 and 14.

Fig. 4: The different learning concepts.

object assigned. During testing, the network senses each object at $K = 5$ locations, thus, we use five sensation-location pairs; each of these are taken from a given example set provided by Numenta. Although every object is represented by a unique set of these pairs, any of the given sensory input is shared across several objects.

We performed a large parameter search (to obtain reasonable values for our more sophisticated investigations, see below), in which we changed the learning rate α, forgetting ϕ, number of output cells n_o, number of cells per mini-column n_m, size of the pattern representing the learnt object in the output layer (called *sparse distributed representation (sdr) size*) s_{sdr}, and the activation threshold for the output layer θ_o. We opted to focus on the incremental Bayesian approach. To obtain good outcomes in a reasonable amount of time the parameter search was conducted using the incremental Bayesian learning rule, and the parameters that provided good results were verified for the summing Bayesian approach. For the whole parameter search we used support implemented through the interconnections in the output layer and apical feedback information in the input layer. Due to the limited amount of time available we let the threshold for the supported activation in the output layer constant to $\theta_s = 0.01$ and reset it to zero if there was no neuron with a sufficient supported activity.

The set of active output cells (we define a cell to be active if it has an activity greater than the threshold θ_o) represents the object recognised by the network. During inference we say that the network unambiguously recognises an object if all neurons of the output representation are active while only s_{sdr} neurons are active in total.

The network is trained by iterating seven times over the feature-location set of an object for the Bayesian implementations, while the original approach used only three repetitions (as it was proposed by Hawkins et. al. [3]). We increased the number of iterations due to the fact that the incremental Bayesian approach uses a step-size parameter α that requires more updates to adapt to the correct input statistics. Yet on the other hand, the benefit of this approach is that we were able to derive the input statistics automatically instead of tuning permanence increments/decrements individually per connection type. Activation patterns of the output layer for a new object are randomly initialised and kept constant over the learning phase of a particular object. In inference phase, every sensation-location pair of the object is presented to the network. The activity pattern in the output layer is classified and compared to the stored activity patterns in the network after each sensation.

For our experiments we used a virtual machine with 16GB RAM and a CPU with four virtual cores.

B. Experimental paradigm

In a first set of experiments we investigated whether the network implementations recognise the object correctly for different parameter settings. The goal was to determine whether the adaptations exhibit a similar ability as the original approach to assign correctly a given sequence of sensations to an object. For that, we used the number of steps (sensations at a location) needed to converge to the correct representation as a measure of performance. If the network did not converge after the input sequence (with sensations at $K = 5$ locations) or converged to the wrong pattern the number of steps was set to infinity.

Next to the convergence we were interested in the network behavior and dynamics, in particular in the average cell activity of the object representation and of all cells in the output layer. We assumed that these simulations permit an insight to the parameter ranges that are on one hand reasonable to correctly recognise the object, and on the other hand increase the interpretability of the outcome (the cell activity of the

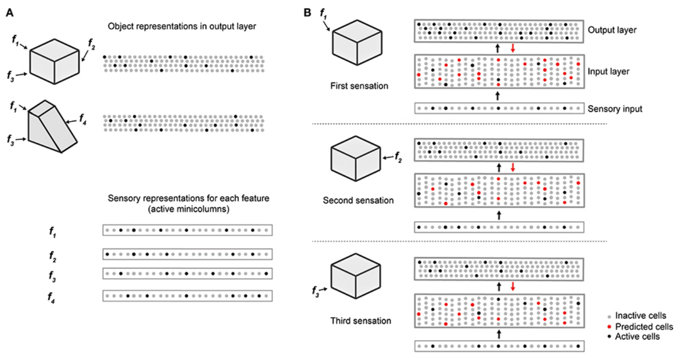


Fig. 5: An illustrative example of the experimental setup including sensory representations and exemplified object inference in the original HTM network (both, figure and caption taken from Hawkins et. al. [3]). (A) Two objects (cube and wedge). For each object, three feature-location pairs are shown (f1 and f3 are common to both the cube and wedge). The output layer representations associated with each object, and the sensory representations for each feature are shown. (B) Cell activations in both layers caused by a sequence of three touches on the cube (in time order from top to bottom). The first touch (at f1) results in a set of active cells in the input layer (black dots in input layer) corresponding to that feature-location pair. Cells in the output layer receive this representation through their feed-forward connections (black arrow). Since the input is ambiguous, the output layer forms a representation that is the union of both the cube and the wedge (black dots in output layer). Feedback from the output layer to the input layer (red arrow) causes all feature-location pairs associated with both potential objects to become predicted (red dots). The second touch (at f2) results in a new set of active cells in the input layer. Since f2 is not shared with the wedge, the representation in the output layer is reduced to only the cube. The set of predicted cells in the input layer is also reduced to the feature-location pairs of the cube. The third touch (at f3) would be ambiguous on its own, however, due to the past sequence of touches and self-reinforcement in the output layer, the representation of the object in the output layer remains unique to the cube. Note the number of cells shown is unrealistically small for illustration clarity.

neurons). Moreover, the results allow the inspection of the influence a parameter has on the output activity. If it is large, changes with respect to the parameter should be done with caution. Using the experimental results, we are able to give a guideline for different parameter settings.

As the activity represents a posterior probability of a neuron participating in a firing pattern (or weighted probability if support is used), the average cell activity of the object representation should ideally increase if a particular feature is not shared among the learnt objects. On the other hand, the confidence can drop if a sensation-location pair can be associated with several objects. The object we used in inference phase shared three of its sensation-location pairs with one other object, one pair with two and one with three. It is expected that a decrease in confidence happens if one of the two latter input values are passed to the network. Yet it is desirable that such a decrease is minimised as much as possible. Parameter values that lead to such a behavior are preferable in comparison to those which lead to a decreasing cell activity, yet slower in comparison to the other representations. Here, we examine the difference between the summing and the incremental Bayesian learning rule (see II-B), the impact of apical information as well as the effect of the parameters ϕ, θ_o and α.

IV. Results

We quickly discovered that our approach comes with the drawback of large computational costs for both, time and memory. The original Numenta approach could make use of simple lookup tables and pass only the indices of active neurons between input and output layer. In contrast, our approach requires to have a value kept in memory for every weight, bias and activity as well as the corresponding data structures that track the input statistics (i.e. the moving averages or the counters). This made it necessary to heavily restrict the maximum number of segments per cell. While the original implementation used a maximum of 255 segments we needed to cut it short to 16 to avoid running out of memory and to run the experiments in a reasonable time. Thus, eventually old contexts were overwritten which had an impact on the performance of the algorithm. An intensive parameter search was not possible, and we tried to limit the parameter space to a minimum.

With the reduced set of parameter settings we were able to infer correctly the object in question when up to five objects were stored in the network. However, when saving three or more objects the parameter search became exhaustive, as most of the tried settings led to either bad results or to a network that was not able to learn the objects at all; that means that in inference phase none of the cells in the ouput layer had sufficient activity to be counted as active. We presume that if more objects are learnt by the network, fewer parameter combinations are able to permit a correct inference. Due to the limited time available we opted to conduct more thorough experiments with only five objects instead of running a single long-taking experiment for parameter fiddling to be able to learn more objects. Since it is the aim to detect an object among multiple, all our following results refer to our experiments with the most (five) objects stored in the network. Exemplifying parameter settings that led to the correct outcome are given in Table I. All these parameter settings work well with the incremental Bayesian learning rule, and were verified with the summing approach (the numbers of steps needed to infer correctly the object are written in the parentheses, where the ∞ marks no correct recognition). We decided to take the parameter setting given above the dashed line since the object was not recognised at the first sensation and hence has some evolution of the activity in the output layer over time. This makes the influence of different parameters on the activity of the neurons in the output layer easier to understand. Thus, we took these values for all of our following experiments (referred to as default parameters). Using the default parameters as reference permits a comparison of the outcome between different values of a single parameter while keeping the others constant.

n_{steps}	n_m	ϕ	α	θ_o	n_o	s_{sdr}
3 (2)	8	0.1	0.01	0.3	128	5
1 (∞)	8	0.0	0.01	0.3	1024	5
1 (1)	8	0.0	0.01	0.3	512	20
3 (2)	8	0.0	0.01	0.3	128	5

TABLE I: Parameter settings to detect the right object under a given number of objects. n_{steps} denotes the number of sensory input needed to recognise the object (in parenthesis for the summing Bayesian, ∞ signifies no correct inference). For the summing Bayesian learning rule the learning rate α was ignored.

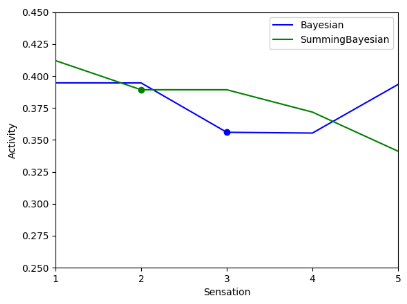
Based on hypothesis to obtain a similar performance, we started investigating the effect of different parameters on the network. Since more context-dependent representations are possible with more cells per mini-column we assumed that increasing the number of neurons in the input layer opens the possibility to learn more objects, or that it becomes easier to infer the correct objects, as the network has more possibilities to find an internal representation that makes it easier to discriminate between the objects. Interestingly, we discovered that, contrary to our expectations, it did not help to set the number of cells per mini-column as high as possible to be able to model the same input in more different contexts. Instead, if 12 cells were involved, it was not possible to derive the correct object representation. This shows the sensitivity of the Bayesian implementation to this parameter and that the algorithm outputs the right result only if the number of cells is within a certain range. We presume that this occurs due to the fact that using continuous values and apical information from the output layer open the possibility to incorrectly model sensations in a new context when it should have used an existing context. Thus, limiting the number of segments helps to avoid this special form of overfitting. This requires further and more thorough examination, which is out of scope of this work.

We conducted a second set of experiments, whose aim was two-fold: we wanted to give a guideline for choosing parameter values that lead to a good performance as well as interpretability of the results. The plots are given in Figure 6, where the dots denote the sensation at which the object was correctly inferred. Figures 6a to 6d and 6e show the change of the average activity of the object representation in L2 over the five sensation-location pairs that were used as input values. Figure 6f compares the average activity of the output representation of the object with the average output activity of all output neurons. Please note that the last two plots 6d, 6e and 6f have another scale on the y-axis. We aim to show with these results that it is possible to see the average neural activation as a measure of confidence which is differently influenced by certain parameters. Values that decrease the impact of input that is shared among several objects and hence minimise the drop in certainty are preferable. In all our experiments, the first sensation-location pair of the object that was to be inferred was shared among two other objects, the second and third with one, the fourth with three and the last input with one other object.

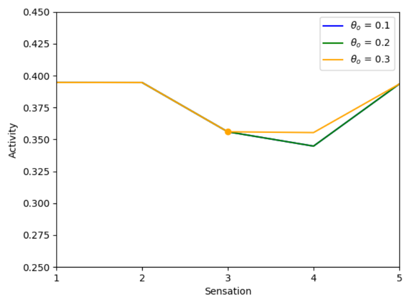
It was important to examine which implementation, i.e. the incremental Bayesian or the summing Bayesian approach, satisfies most the desired features. The graphs are given in Figure 6a. When comparing the two implementations with each other, the summing Bayesian learning rule was able to infer the right object more quickly (with less sensation-location pairs) than the incremental learning rule. On the one hand, the plot shows that both approaches learn the input statistics differently for a small data set and few training iterations. On the other hand, it is noticeable that the summing Bayesian approach does not fulfill the criteria stated above; the average activity decreases steadily although the features are not increasingly shared across the learnt objects. The incremental Bayesian approach, in contrast, exhibits a network behaviour that is similar to what we have predicted. However, the certainty decreases already after receiving a sensation-location pair that is shared with only one other object, while it remains at this confidence/average activity when the sensory input was shared more often between several objects (sensation number three and four). We figured that this happens due to the fact that some other parameters prevent the output activity to drop too quickly or too low, i.e. the threshold for the output activity θ_o and forgetting ϕ.

Using this justification, we investigated the impact of the output activation θ_o on the network. The plot is displayed in Figure 6*b*. Although all trajectories are remarkably similar, networks that were learnt with $\theta_o = 0.1$ and $\theta_o = 0.2$ were not able to recognise the object. Presumably this is due to the fact that neurons with a weaker activity remain active and are not forced to zero, even though they are not part of the pattern. As expected, we can observe a larger drop at the fourth sensation if the threshold θ_o is decreased. However, another test with $\theta_o = 0.5$ revealed that the network with a high threshold for the output activity was not able to learn any object at all. That was expected, as the average activity lays below that threshold. In all our simulations, $\theta_o = 0.3$ led often to good results, even with a changing number of input and output neurons as well as varying s_{sdr}.

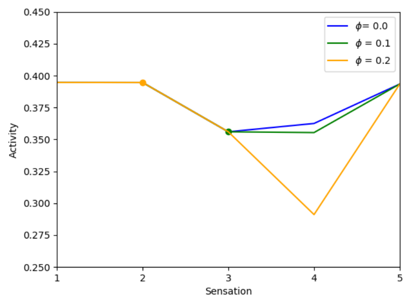
The forgetting rate ϕ had a comparatively large effect on the dynamics. If none is applied, then the average activity is much more stable than applying a larger ϕ, yet the network is not able to infer the object. Interestingly, a network with $\phi = 0.2$ was able to infer the correct object more quickly than the default parameter setup after the second sensation. It can be assumed that this is due to the fact that the neurons that are not part of the output pattern do not receive any new feed-forward input and drop quickly below the threshold. It should be noted that this also means that if unexpected or less informative input (e.g. large overlap with other sensations) is perceived, the actual true representation will lose its activation more quickly as well (see the drop at sensation 4). Thus, special care should



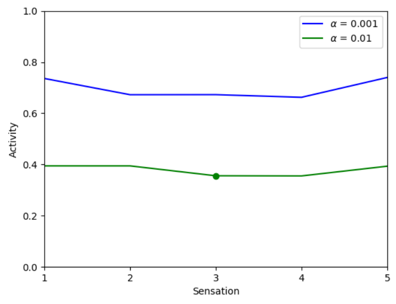
(a) Comparison of the average activity of the output representation for the incremental Bayesian and the summing Bayesian approach.



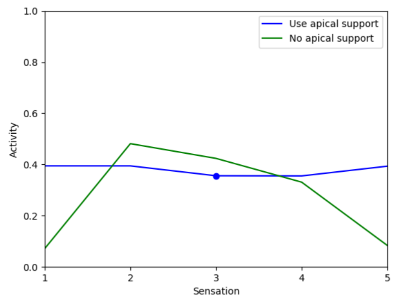
(b) Comparison of the average activity of the output representation with different values for θ_o.



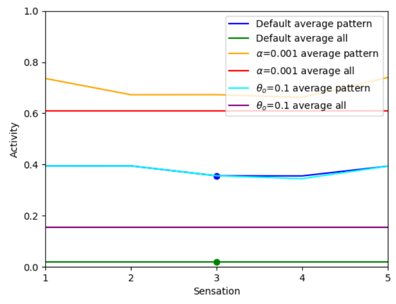
(c) Comparison of the average activity of the output representation with different values for ϕ.



(d) Comparison of the average activity of the output representation with different values for α.



(e) Comparison of the average activity of the output representation when apical information is used versus when it is not used.



(f) Comparison of the average activity of the output representation and the average activation over the whole output layer.

Fig. 6: Experiment about the average activity of the neurons in the output layer. The dots give the time step when the object was detected. All experiments used the incremental Bayesian learning rule but Figure 6a (see labels).

be taken for the choice of ϕ: small values greater 0 should be taken, whereas it is necessary to keep them low in order to minimise potential decreases in certainty as much as possible.

As another parameter of interest, we figured that the learning rate α permits some insight about how different learning intensities affect the performance Interestingly, decreasing the order of magnitude of the learning rate α increased largely the average activity of the representation without being able to infer the correct object (see figure 6d). In comparison, applying a larger value $\alpha = 0.1$ led to a failing training procedure, such that the algorithm could not learn any object at all. Hence, we can conclude that the learning rate α plays in a significant role for the discrimination of different objects.

The previous results suggest that although the average activity of the output representation might evolve similarly for different parameter values, there are some that complicate the discrimination between objects. That indicates that the average activity of all neurons in the output layer is much higher and that several cells have an activity larger than the threshold. To verify this claim we plotted the average activity of all neurons against the average activity of the output representation for the default values, changing $\theta_o = 0.1$ and $\alpha = 0.001$. The results are given in Figure 6f. As expected, the average activity over all cells is much higher for the changed θ_o and α as for the default values. It is remarkable that the average activity over all neurons for $\alpha = 0.001$ is very close to the average activity of the output representation, and therefore implies that all learnt objects have a very similar internal representation. These results evince that the chosen parameters should find a good trade-off between being able to discriminate between objects and permitting a quick derivation of the input statistics in a comparatively short training phase.

To understand how important apical information is for the correct inference of objects we compared the output activity of the representation for both, included and excluded feedback connections. Figure 6e depicts the results. Although we assumed that the incorporation of feedback support simplifies the correct object recognition, we did not anticipate such a large difference. If no apical input was used in the input layer, the network was not able to recognise the correct object, and the activity of the neurons of the object representation decreases rapidly after a steep increase. Naturally, this is no desired behavior, and we strongly dissuade from removing the incorporated feedback structures.

Lastly, we compared the time consumption of the algorithms. As we have already stated, the original Numenta implementation is much quicker while using less memory. The unchanged HTM framework takes on average 0.21 seconds to learn five objects and to infer the correct one. In comparison, the summing Bayesian learning rule takes 638.71 seconds, and the incremental version needed 475.61. Thus, there is a large discrepancy in terms of scalability between those approaches.

V. Conclusion & Discussion

In contrast to our expectations, we could not adapt the HTM algorithm while maintaining a similar performance. Due to the fact that the matrices scale up quickly the algorithm becomes very slow and eventually runs out of memory if the number of segments is not bounded to a value that is sufficiently low. Moreover, even though we did not discard any information about the input and the statistics by using continuous values, our adaptions made it not much more accurate than the original Numenta approach. This is due to the fact that we represent our activity densely while only using some few values through sparse input patterns. All versions, the original implementation and our two Bayesian variations, were able to detect the correct object if five objects were learnt by the network and the parameters were set accordingly.

The Bayesian approach does indeed decrease the number of parameters, however, it is not significant. Although we were able to pare down the parameters for the increment/decrement and the thresholds for the permanence values, we added a learning rate α and a parameter for the noise. Yet the summing Bayesian implementation does not require a learning rate and initialisation of averages and hence could make a significant reduction. Further, although we could reproduce Numenta's results with a limited number of learnt objects we were not able to do it with a similar performance (in terms of time and memory consumption) nor were we able to reduce much the set of parameters. This means we could not verify our hypothesis.

It is worth stating some presumptions why the Bayesian approach falls short in terms of performance in comparison to the the original implementation. A severe difficulty was the discrimination between objects with overlapping sensation-location pairs. As exemplified in Figure 5, some objects were almost identical but a few sensations. The strength of the Numenta approach is to learn and infer an object within just a few time steps, which is needed if only single sensations differ between the objects. Our Bayesian adaptation needs more training samples and iterations to learn the correct input statistics and hence needs adapt quickly enough to changing input. Further, we could not incorporate the full merit of the Bayesian approach due to the fact that we discard information by applying thresholds that were necessary when embedding the learning rule into the existing framework.

Although the experiments did not return the results we had hoped, we truly believe that the conducted research was nevertheless not in vain. Since the original Numenta approach was highly motivated by biology while lacking a mathematical justification, our mapping to a mathematical framework provides a theoretical scaffold that permits the investigation of the dynamical behavior. We showed that with a thorough parameter search and sufficient resources it is possible to obtain similar results as the HTM implementation. Moreover, we gave an indication of how different parameters affect the behavior of the network. Thus, it is reasonable that future research uses this work to extend mathematical investigations of the network behavior from a perspective of an attractor network. We hope that we can contribute with this work to close the gap between traditional machine learning approaches and biological models. We strongly believe that both areas can benefit from the exchange of concepts, as it combines the analysis of time-dependent data and recognition that is backed by the field of statistics with architectures that are based on findings in neuroscience.

Other projects could compare these algorithms with a more efficient implementation or using other hardware for parallelising matrix calculations (e.g. GPU). We chose to implement a comparatively slow and high dimensional matrix implementation in Python. Yet memory and time consumption could be heavily reduced with efficient lookup implementations in C++. Instead of using matrix data structures it is possible to implement an array of tuples (i, a_i) with an index i indicating the cell or the segment and the corresponding value a_i. The length of this array can be largely reduced if only cells with an activity greater than a given threshold are passed to the next computational instance, e.g. the output layer. We expect that it is possible to show that the performances can be made similar with a more efficient implementation (or that learning is possible within a reasonable time frame). This would then allow to conduct thorough experiments and to compare the approaches according to their noise tolerance and accuracy if a large number of objects were learnt.

References

[1] "Human Brain Project of the European Union," https://www.humanbrainproject.eu/en/, accessed: 2019-12-28.

[2] J. Hawkins, M. Lewis, M. Klukas, S. Purdy, and S. Ahmad, "A framework for intelligence and cortical function based on grid cells in the neocortex," *Frontiers in Neural Circuits*, vol. 12, p. 121, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2018.00121

[3] J. Hawkins, S. Ahmad, and Y. Cui, "A theory of how columns in the neocortex enable learning the structure of the world," *Frontiers in Neural Circuits*, vol. 11, p. 81, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2017.00081

[4] M. Lewis, S. Purdy, S. Ahmad, and J. Hawkins, "Locations in the neocortex: A theory of sensorimotor object recognition using cortical grid cells," *Frontiers in Neural Circuits*, vol. 13, p. 22, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2019.00022

[5] A. Lansner, *A one-layer feedback artificial neural network with a Bayesian learning rule*, ser. Trita-NA-P 8910, 1989.

[6] A. Sandberg, "Bayesian attractor neural network models of memory," 2003.

[7] A. Sandberg, A. Lansner, K. M. Petersson, and O. Ekeberg, "A bayesian attractor network with incremental learning." *Network*, vol. 13 2, pp. 179–94, 2002.

[8] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, p. 23, 2016. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncir.2016.00023

[9] Y. Cui, S. Ahmad, and J. Hawkins, "The htm spatial pooler—a neocortical algorithm for online sparse distributed coding," *Frontiers in Computational Neuroscience*, vol. 11, p. 111, 2017. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2017.00111