Modeling Implicit Acquisition of Sequential Information using a Neocortex-inspired Neural Network – Hierarchical Temporal Memory

Submitted By:
TAHER HABIB

Supervisor:
PROF. DR. GORDON PIPA

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Cognitive Science to:

NEUROINFORMATICS & ROBOTICS RESEARCH GROUP
INSTITUTE OF COGNITIVE SCIENCE

11-FEBRUARY-2021

51-52-53

Dedicated to Mr. Shailendra Jha who inspired me to be curious, to ask questions and to seek answers.

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Signature

Osnabrück, 11-Feb-2021

City, Date

Acknowledgements

I would like to express my gratitude to Professor Dr. Gordon Pipa for agreeing to supervise this project and introducing me to the research area of Sequence Learning.

I also extend my heart-felt gratitude to my second supervisor Sophie Lehfeldt for introducing me to this fascinating research domain on Sequence Learning in biologically-inspired Neural Networks and later agreeing to support me in carrying out this particular study. I cannot thank her enough for her valuable insights related to the research problem, her extreme levels of patience while discussing the general topic and her suggestions on other super-specific problems that I encountered; and lastly her extremely helpful inputs on writing and structuring the draft.

I also thank Johannes Nowack for some fruitful discussions on the overall research problem in this work and for proofreading parts of this thesis and giving constructive advice on editing the draft.

Lastly, I am grateful to my parents and my love for just being there for me, whenever and in whichever way I needed them.

Taher Habib

Absorb what is useful, discard what is useless and add what is specifically your own.

– Bruce Lee

Contents

Abstract

The human brain seems to acquire sequential information – mainly, recognition and prediction of temporally-correlated patterns – almost seamlessly. Moreover, the acquisition is even implicit in some cases like language [32, 33]. This thesis attempts to study this feature of the biological brain by addressing the question of implicit (unsupervised) learning of nontrivial and higher-order sequential information over time, which is also an important direction of inquiry that lacks a satisfactory solution in the problem domain of Sequence Learning.

To this end, the thesis examines a neocortical-inspired model – Numenta's Hierarchical Temporal Memory (HTM) – which deploys Hebbian plasticity based learning techniques to implicitly learn higher-order temporal dependencies in sentences (strings) generated from an artificial grammar, as Artificial Grammar Learning (AGL) is known to model the implicit acquisition mechanisms of language processing in the human brain [5, 30, 32]. The used solution model exploits two computational principles found in the brain – first, sparse distributed encoding of information (temporal, in case of sequence learning); second, use of dendrites as nonlinear, locally trainable, independent subunits that detect these sparse synchronous activation patterns in the neuronal population [1, 13]. The HTM network is shown to learn long-range dependencies implicitly in an online fashion. However, it is found to have a variable-order memory that attempts to "memorize" sample sequences, instead of acquiring the structure and syntax of the Artificial Grammar.

CHAPTER I: INTRODUCTION

I.1 Objective of Study

An environment in which events in time are independent of each other is rarely found. From the cycle of the seasons over an year's span to human speech over the span of a few seconds, most of the phenomena that a brain comes across are temporally distributed and structured. It is, therefore, necessary for an intelligent agent to be able to make sense of temporally distributed events: learn to recognize their occurrences, extract useful regularities and make use of them to predict future events. In other words, sequential organization is crucial to many adaptive cognitive functions and behavior [28, 29]. Hence, it is of central importance in cognitive science to elucidate on the question of how knowledge of temporally distributed observations is *acquired* and in what form it is *represented* in the memory.

One of the challenges that needs to be addressed in the domain of Sequence Learning is the question of implicit (unsupervised) learning of nontrivial and higher-order[1], sequential information over time. The overarching challenge here – that must be stressed – is the need to learn to recognize and predict temporally-correlated patterns in an ***implicit*** fashion, i.e. "without any requirements of awareness of either the process or the product of acquisition" [4]. Besides, the agent also needs to be receptive to changes in these patterns over time so that old patterns can be relearned in newer, novel ways or simply forgotten to learn new ones instead.

Unsurprisingly, like most other research questions in the field, the general problem of Sequence Learning has been tackled from different perspectives in the past – using both classical, statistical/machine learning based techniques and more recent, brain-inspired approaches.

[1] *Higher-order* sequences are those where contextual dependencies span over several timesteps. The term *order* refers to the Markov order of the sequence-generating process and specifically denotes the minimum number of previous timesteps that the learning algorithm needs to consider in order to make accurate predictions of the future. So, a *first-order* sequence is one where knowledge of only the current timestep is required to make predictions about the next timestep.

In the former category, better performance on the task at hand was usually the main objective. As a result, these approaches did not explicitly attempt to model the impressive ability of the neocortical mechanisms in the brain that perform implicit acquisition of sequential information; and instead they used network architectures and/or learning rules that were mostly unrelated to the brain's biology [9]. Nonetheless, in most of these attempts, nontrivial sequences with complicated temporal dependencies were learned [9, 75-77].
In case of brain-inspired studies, focus was put more on the mechanisms of implicit acquisition of temporal information, where elaborate, biologically plausible models and/or learning rules were used, but to learn sequences of rather simpler forms [17, 37, 38, 50] (see section I.3, Literature Review for details).

Consequently, methods, where brain-inspired networks are used with biologically plausible learning mechanisms to implicitly learn rather nontrivial / higher-order sequences, have not received much attention in the past. Addressing this issue is the objective of the thesis.
Specifically, the focus of this thesis is on studying the problem of sequence learning in the domain of Artificial Grammar Learning using a neocortex-inspired model (Numenta's Hierarchical Temporal Memory (HTM)) that uses a neuronal model based on active dendrites and learning mechanisms based on Hebbian (neuro-) plasticity [1]. Furthermore, the study examines how and why the HTM network learns a *sequence of transitions* – higher-order, rule-based but not entirely deterministic – from one item to the next; without the need for any external supervision or without a parameterized/formalized target function encoded inside the model itself.

I.2 Choice of Problem Domain: Artificial Grammar Learning

I.2.1 Relation of Language Acquisition to Implicit Sequence Learning

An area in which there is growing evidence of structured sequence processing and implicit learning of sequential information is language acquisition and learning [5, 30, 80]. This implies that the problem of implicit sequence learning can be (almost) equivalently studied as the problem of implicit grammar learning/acquisition, which is addressed in the domain of Artificial Grammar

Learning [32, 80]. Artificial Grammar Learning (AGL) examines mechanisms that underlie language processing in the human brain and its ability to learn made-up grammar [32, 33, 80].

I.2.2 Using Artificial Grammar to Test Implicit Sequence Learning

In 1967, A. S. Reber suggested that humans can implicitly learn artificially-constructed grammars (Artificial Grammars, as they are called in the literature) by extracting the relevant temporal structure/dependencies from the input [32]. Specifically, human participants were asked to memorize a set of strings created using an artificial grammar and were subsequently asked to categorize new strings which they had not previously been exposed to. It was only during this subsequent test phase that the participants were informed about the existence of an underlying set of rules generating the strings. The study found that the participants could identify, with above chance level accuracy, which strings were grammatically correct. However, they could not identify (verbalize) the rules that generated those grammatical strings. In other words, there was no intentional learning strategy used by the human subjects and hence, the learning was implicit.

Apart from this, there are other recent studies suggesting Artificial Grammar Learning as a relevant model for aspects of language acquisition [33].[2]
Furthermore, artificial grammars in AGL generate sequences that are higher-order and have nontrivial temporal dependencies (see chapter II, II.2 for details).

Hence, learning artificial grammars is the chosen problem domain in this thesis.

[2] It must be noted, however, that there are also some concerns [83-86] in the linguistic community on whether paradigms deployed in the Artificial Grammar Learning studies – based on finite state grammars like Reber Grammar (see chapter II, II.1) – are sufficient to account for implicit acquisition of language in humans. In [85, 86], it was argued that finite state grammars were not sufficiently expressive to capture the syntactic recursion, the modularity and hierarchical organization of constituents and phrases in human language and that a context-free grammar is the least requirement to capture these complex features. However, the current study will overlook these concerns and will, nonetheless, use finite state grammars as they generate nontrivial, higher-order sequences.

I.3 Literature Review on Sequence Learning

This section reviews the past attempts – describing their successes and failures – at solving the problem of Sequence Learning in general (and not restricted to AGL), which will further help in elaborating the need for the foundational principles of the proposed solution model.

I.3.1 Classical, Machine Learning Based Approaches

Most traditional recurrent neural networks (RNNs) – except for Long-Short Term Memory (LSTM) [9] – like time-delay neural networks (TDNNs) [76], statistical models like hidden markov models (HMMs) [75], autoregressive methods like autoregressive integrated moving average (ARIMA) [77] and other gradient-based approaches have been shown to be successful at learning a variety of difficult sequential learning tasks. However, they have not been tested for performance in non-stationary and highly dynamic environments/benchmarks [1]. See Hochreiter & Schmidhuber, 1997 [9] for a detailed review. In all these aforementioned approaches, the benchmark data was generally divided into training and testing sets, with the underlying assumption that the testing dataset's statistics were the same as that of the training dataset [34]. Based on these considerations and the fact that they used external supervision for learning, it can be concluded that they did not attempt to mimic the same (implicit) learning mechanisms as employed in the biological brain and, hence, these past attempts were biologically implausible.

LSTMs have been shown to successfully learn sequences with long time-lag dependencies also, using gating mechanisms which equip them with the ability to selectively pass information across time [9]. They have also yielded impressive results on a wide range of temporal, sequence-based, real-world problems [35, 36]. However, similar to the other RNN based approaches above, LSTMs do not attempt to model the biological brain and as such they provide little insight into the impressive ability of the neocortical neuronal mechanisms in the brain that perform implicit learning.

I.3.2 Recent, Brain-Inspired Approaches

There are also several brain-inspired approaches in the literature. For instance, in Rao & Sejnowski, 2001 [37], a biologically plausible model based on spiking neurons was studied and shown to learn a predictive model of its inputstream (a moving stimuli generating excitatory input) using temporal-difference based asymmetric Hebbian learning mechanism. However, the task used for testing the model in this study was relatively straightforward: the neurons simply learned to fire a few milliseconds before the input stimuli arrived; and developed direction selectivity to the movement of the stimuli, as a consequence. Besides, there was no stochasticity (for instance) present in the inputstream, like in this study here where an artificial grammar has been used (see chapter II, II.1).

In another study (Brea et al., 2013 [38]), a spiking recurrent network was shown to learn and recall precisely timed sequences of inputs, *but* the network used – albeit a biologically plausible one – a supervised learning rule that minimized an upper bound on the KL divergence between the model and target distribution of network activity.

In Lazar et al., 2009 [45], a bio-inspired, self-organizing recurrent network (SORN) was shown to have the ability to map identical inputs onto separable internal representation in its reservoir network via evolving the recurrent connections in an unsupervised, self-organized manner using neuroplasticity mechanisms. Inspired by this, a study was conducted by Duarte et al., 2014 [17], where SORN was deployed to solve, specifically, the AGL task. The network was trained in a supervised manner – like the previous study in [38] – in order to predict the next input in the sequence, generated from an artificial (Reber) grammar. It was demonstrated that SORN's predictions were closely comparable to those of the 3-gram model[3], indicating that SORN is somewhat capable of learning the *right* amount of contextual information i.e. trigrams of the Reber Grammar. See chapter II, II.2 for details on this characteristic feature of the Reber Grammar.

[3] An *N*-gram is a probabilistic model used for sequence modeling in natural language processing, using the statistical properties of *N*-grams. It is used for predicting the next symbol/letter/word in a sequence in the form of a $(N-1)^{th}$ order Markov process.

However, the study emphasized more on showing that (Hebbian-learning based) plastic reservoirs were more efficient and stable than static, non-plastic reservoirs in adequately representing the nontrivial temporal relations in the input sequence. Accordingly, it did not shed light on the nature of the context-based representations of the temporal relations in the inputstream, developed by the reservoir. As a result of this and the fact that external supervision was used to train the SORN network, a substantial part of the question of how the human brain implicitly acquires the structure/syntax of an artificial grammar was left unanswered.

The originators of SORN, in [45], also called for modifications in their simplistic neuronal model with more elaborate ones based on leaky integrate & fire (LIF) neurons, etc. in order to endow the neurons with intrinsic memory storage properties so that some or all of the storage of the temporal history of the input stream, relevant for making successful predictions, could be offloaded from the network's recurrent dynamics. Utilizing this suggestion, there was another study by Klos et al., 2018 [50] in which the LIF-SORN [51] network – a 2D grid of recurrently connected LIF neurons – was used to learn a repeated spatio-temporal stimulation pattern (again, a moving stimuli generating excitatory input).
It was found that, during the test phase, when cued by stimuli at the starting point (during training) of the moving spot, the network anticipated the continuation of the sequence (as revealed by its spiking activity) in a direction specific way, similar to the above study by Rao & Sejnowski [37].
Also, like the study in [37], the LIF network learned to recall the sequence without any external supervision and structuring its connectivity in a totally self-organized manner. This is remarkable since the input sequence used in this study was rather sophisticated – structure-wise – and inspired entirely from an actual experimental study done on rats' primary visual cortex. However, the task here, after all, involved learning only short-term dependencies, as in [37].

I.3.3 Conclusions from Literature Review

Plausibility of Sequence Learning via Hebbian Plasticity

None of the brain-inspired spiking neural networks above – except Duarte et al., 2014 [17] – directly addressed the problem of learning sequences that are nontrivial and higher-order i.e.,

sequences which need an *N*-gram model with $N > 2$. Nonetheless, these studies have shown that relatively simple and limited types of temporal patterns can indeed be learned via biologically plausible, Hebbian learning rules in spiking neural networks.

Encoding of Sequential Information Solely in Connection Weights

Considering the implementation details of the bio-inspired models discussed above, it can be said that the internal representations of the temporal dependencies in the inputstream, in all of them, are entirely shaped by spike-timing-dependent plasticity (STDP) and accompanying homeostatic mechanisms, which work together on evolving the optimal synaptic weights between neurons.

This highlights that most of the temporal memory information that gets learned, gets encoded in the *finely graded* synaptic weight values, as in the traditional connectionist approach of artificial neural networks [42, 43]. Of course, the exclusive dependency on the connection weights could be reduced by employing more sophisticated, neuro-biology inspired models of neurons like the LIF neurons, as used by Klos et al., 2018 [50]. However, as stated in the previous subsection, the study in [50] concentrated around predicting relatively simple, moving excitatory inputstream only.

More significantly, since the learned representations of sequential information in the input were not selectively examined in these studies, the nature and structure of the serial ordering of temporal information remain unknown. Additionally, the question of whether biologically plausible learning mechanisms can lead to disambiguous representation of input in distinct temporal context also remains unanswered; although there was evidence of the presence of context-based sequential input representation – albeit only weakly separable – in the neural activity of SORN [45].

Passive Utilization of Inhibitory Mechanisms

Despite an abundance of evidence for the significance of inhibition in *shaping neuronal circuits for optimality* [52-54], most of the brain-inspired studies above adapt *only* the excitatory connections via plasticity rules, leaving the inhibitory connections unchanged to their initialized values. It can be argued that this potentially leads to a passive, inefficient way of deploying network resources to learn optimal, separable internal representations of temporal dependencies in the input.[4]

[4] This could also be a probable reason for the need of biologically implausible, external supervision in SORN, in [17] above, for learning higher-order sequences.

In conclusion, it can be assumed that if the problem at hand is to be solved in an *entirely* unsupervised manner, the inhibitory mechanisms in the network are also required to be *actively* utilized when forming internal representations of the temporal structure in the inputstream.[5]

I.4 Choice of Solution Model: Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) [1] exploits *two* general aspects associated with computation in the brain: first, encoding information in sparse, distributed activations of neuronal ensembles and, second, use of superlinear computation in dendrites and clustering of synapses on dendrites, based on *synchronous* presynaptic activation (also called *in-branch localization* [12]). Motivation behind the use of these two computational principles in the current study are explained in the following two subsections.

I.4.1 Use of Sparse Distributed Representations

Ubiquity of Sparse Coding in the Brain

Sparseness in a neuronal population activity can be defined as the number of activated neurons with respect to the total number of neurons in the population at any given time. It has been found that information is massively distributed in the brain and only a very small fraction of neurons are active in the brain at any given time [11]. Also, it has been suggested that sparse coding may be a general tactic deployed by neural systems to augment their memory capacity at a very low energy cost [59], with evidence for sparse distributed representation of sensory information found in many brain regions like those responsible for vision [63], audition [64] and touch [65].

Encoding of Temporal Information in the Brain in Sequence Retrieval Tasks

Specific to the requirements of the current study, there have been several studies in the past [55-58] which collectively suggest that in a temporal sequence retrieval task, regions of the brain –

[5] It is worth noting that the last two conclusions from the literature review are intricately related to each other. For *totally* unsupervised learning, a network that learns to distinctly incorporate any input along with the necessary temporal context right into its activation dynamics, would also (supposedly) require an active role of its inhibitory mechanism.

particularly, hippocampus[6] – distinctly encode "object+(sequence-)position" binding in their neural activity patterns. In other words, activation patterns of the neuronal ensembles in the hippocampus do not just encode information related to an individual object in the inputstream, but also the object's temporal context. This is performed by the human brain without any external or intentional encoding strategy.

This empirical finding on brain regions, suggesting the use of network-activity based encoding for temporal context in the inputstream, is a crucial one for the current study. Since artificial grammars generate sequences that are higher-order (see chapter II, II.2), the ability of encoding the combined "object+sequential context" information in the activation patterns can be posited as a fundamental requirement on a network that learns the sequential information implicitly, without supervision.

Relating this to the aforementioned idea of sparse coding in the brain, the HTM model uses Sparse Distributed Representations (SDRs) which encode temporal dependencies between the inputs at different time points in the inputstream by representing the same input using different activation patterns, which are directly dependent on the input's temporal context (see chapter II, II.7 for details).

I.4.2 Use of Dendrites in Neuron Model

Use of SDRs to build and learn representations of input information calls for another necessary component in the network – the active dendrites framework.

Spike Coincidence Detection in Real Neurons

König et al., 1996 [20] identifies two distinct operational modes of neurons in the cortex that depend on the duration of the interval over which they effectively integrate their incoming synaptic input. First, if the said interval of integration is of the order of the mean inter-spike interval or longer, then the neurons will effectively act as temporal integrators, and they could be unreliable in transmitting information temporally. This more traditional view on the operational mode of cortical

[6] a brain region that plays a significant role in the consolidation of information from short-term to long-term memory in the brain.

neurons is predominant in the modern connectionist approach in the form of the linearly summing, point-neuron model as integrate-and-fire devices [42, 43].

In contrast, if the integration interval is shorter than the mean inter-spike interval, then the neurons will effectively act as spike-coincidence detectors relaying synchronized inputs [70]. Based on several neurophysiological studies and other theoretical arguments related to solving the *binding problem* (i.e., the problem of integrating information – of an object/event and its associated features – distributed over a population of neurons into coherent representations), it has been reasoned that coincidence detection might be the dominant mode of operation of cortical neurons (König et al., 1996 [20]).

The key thing to note here is that if the information on temporal dependencies in the inputstream is stored in distributed, sparse activity patterns which are synchronized, this cortical neuron model – based on spike-coincidence detection – may be more significant for the problem of sequence learning in general, than previously thought.

Coincidence based Pattern Discrimination and the Need for Dendrites

Specific to the current study, the aforementioned neuronal model entails the use of dendrites in the model. If the information on temporal relations is to be stored in distributed activity patterns and since there could be potentially numerous distinct contextual histories, it stands to reason that a single neuron is supposedly likely to take part in multiple ensembles, each representing a different temporal context. Besides, if the same neurons in the network are to recognize these sparse distributed patterns over time (for predictions), using presynaptic spike-coincidence, then a single neuron presumably needs to be able to recognize a multitude of independent patterns with several distinct contexts.

In direct relation to these considerations, there have been several studies in the past [12,15,16,18,19] proposing that nonlinear properties of the dendrites can enable cortical neurons to recognize several distinct patterns in neuronal populations. Additionally, several other studies [14, 46, 48, 49] have indicated that distal dendritic branches of the excitatory neurons of the neocortex possess the ability to act as dynamic computational subunits – leading to the dendritic branches acting as independent pattern recognizers.

Specifically, in Poirazi & Mel, 2001 [19], it was analytically shown that presence of nonlinear, compartmentalized and locally trainable subunits (***active dendrites***) in every neuron dramatically increased the storage capacity (i.e., number of distinct patterns recognized by a neuron) of each neuron and the network, as a consequence.

In conclusion, the HTM model also uses neurons that are coincidence detectors and that recognize patterns of synchronous activity in the population – representing temporal context in the form of SDRs (see chapter II, II.7) – using dendrites, with clustering of synapses based on synchronicity of presynaptic activations (see chapter II, II.6).

CHAPTER II: METHODOLOGY & MODEL

Formalization of Sequence Learning in AGL

II.1 Finite-State Languages

Finite-state languages serve as a good starting point for formalizing the idea of a grammar. According to Chomsky & Miller, 1958 [31], a ***finite-state language*** is a finite or infinite set of strings (*sentences*) of symbols (*words*) generated by a finite set of rules (the *grammar*). Each rule specifies the state of the system in which it can be applied, the symbol which is generated, and the state of the system after the rule is applied. In simpler terms, a finite-state language can be described with a finite set of letters (for instance, the seven letters {*A, T, P, S, X, V, Z*} as in Fig. 1) as the "vocabulary" and a set of rules for generation of sentences as the grammar. The grammar is characterized by a directed Markov process, as illustrated in Fig. 1.

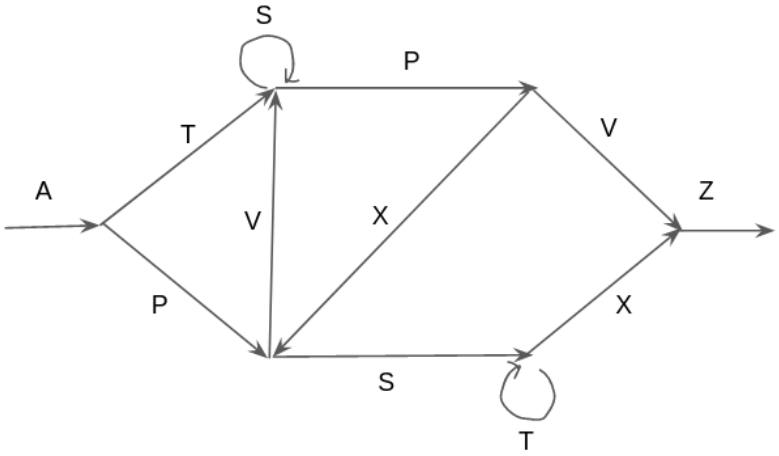


Fig. 1. Schematic state transition diagram of a finite-state language. Referred to as Simple Reber Grammar (SRG), hereafter. SRG is a set of rules for generating sequences (strings) of symbols by traversing from the leftmost node to the rightmost node in the graph, with each transition producing a symbol associated with the corresponding edge. An infinite number of sequences can be generated from this transition diagram. At nodes with multiple edges to traverse, a choice is made randomly (Inspired by Fig. 1 in [39]).

A transition from one node (or state) to another produces a letter. A sentence begins at the leftmost node with letter *A*; then, each subsequent transition produces a letter, and the sentence ends when

the rightmost node with letter *Z* is reached. The final sentence produced depends upon the path taken. At nodes where there are multiple edges to traverse, all the possible transitions are equally likely. The language, then, is simply the set of all possible sentences that the grammar can produce. Two example sentences (strings) of the artificial grammar above are: <*ATSSSPXSXZ*> and <*APVPXVSPVZ*>.

II.1.1 Simple Reber Grammar (SRG)

Henceforth, the above generative, finite Markov process in Fig. 1 will be referred to as ***Simple Reber Grammar.***

In the SRG above, the average number of incoming (also, outgoing) edges on any node is 2. Of course, the starting and ending nodes are different in this respect – with 2 outgoing and 2 incoming edges only, respectively – because every string in the grammar learning task is assumed to have a beginning and an end.[7]

II.2 Simple Reber Grammar is a Second-Order Markov Process

In Petersson et al. 2005 [39], after an elementary, formal analysis of the SRG, it was demonstrated that the Markovian process that generates symbols from SRG is second-order. Intuitively, given any two consecutive symbols from an SRG string, the probability of occurrence of the next symbols can be determined.

To take an example, if at a given timestep in a given Reber string/sentence, the current and the preceding letters are known to be *T* and *A* respectively, then the next timestep's (correct) predictions are letters *S* and *P* only. If instead only the current symbol *T* were known with no memory of the letters at previous timesteps, then it would be impossible to determine the correct predictions for the next timestep[8].

[7] One can also join the final node (before the *Z*-edge) to the starting node (after the *A*-edge) and generate a continuous inputstream of symbols from the vocabulary for the network to learn. However, learning on this task will not be examined in the current study.

[8] It is worth noting that this second-order property of the SRG is *not strictly true* in all cases. For example, if the current and preceding letters in a given string are *V* and *P* respectively, there is no way to determine the next letters until one knows the letter preceding *P*. In the case when it is *A*, the letter after *V* will be either *S* or *P*; and if it is *S* or *T* or *V*, then the letter after *V* will definitely be *Z*.

The second-order nature of the SRG was also verified by Duarte et al. 2014 [17] where they found that the best performing target distribution for assigning probability to future outcomes in a Reber String, given the context, was the 3-gram Markov model.

II.2.1 Implications for the Current Study

1. Since the SRG strings are higher-order, it can be assumed that in order for a Hebbian-learning based network to learn the SRG *implicitly* only from the observed transitions in the inputstream (at each timestep) and without any external supervision, it is necessary that the network has a storage and representational capacity enough to store and represent every symbol in its full and correct context i.e., in a way such that each symbol is stored *distinctly in different contexts*, in order for the different contextual histories to elicit distinct prediction responses[9].

   For instance, in an SRG string, no matter how the prediction responses are represented and elicited, the symbol *X* coming in as an input with the context of <...*SPX*> should be represented distinctly enough to elicit a distinct prediction response (*V* and *S*, in this case) than *X* coming in with the context of <...*PSX*>, which should elicit a different prediction response (*Z* only).
2. Due to the second order nature of the SRG, it was concluded in [39] that *a machine/algorithm that recognizes and acquires a set of trigrams generated from the SRG, can be said to have learned the Simple Reber Grammar.* Therefore, in the spirit of lexical learning framework in theoretical linguistics [80, 81], the HTM will also be tested on its ability to acquire the "lexicons" (trigrams) of the Simple Reber Grammar, apart from its ability to learn higher-order sequences in an implicit manner.

   It must be noted that there are 33 trigrams in SRG in total, as listed below:

The analysis carried out in [39] was based on estimating the convergence of the conditional entropy of the next symbol in a reber string of length *n*, as *n* tended to infinity.

[9] As discussed before in section I.4.1, there are several studies [55-58] indicating that the biological brain uses such mechanisms in various sequence retrieval tasks.

{"ATS", "ATP", "APS", "APV", "TSS", "TSP", "TPV", "TPX", "PVS", "PVP", "PST", "PSX", "SSS", "SSP", "SPV", "SPX", "PVZ", "PXV", "PXS", "XVS", "XVP", "XST", "XSX", "TTT", "TTX", "TXZ", "VSS", "VSP", "VPV", "VPX", "STT", "STX", "SXZ"}.

HTM Model Description

HTM is a neocortex-inspired neural network model that mimics the structural and algorithmic properties of the neocortex [13]. HTM essentially models learning and recalling sequences of patterns. It learns a *predictive* model of the data based on observed transitions from exposure to an inputstream in an online fashion, without any need for the input data to be broken down into separate predefined chunks for training and testing.

II.3 HTM Network

The HTM network – in an attempt to imitate the architecture of the neocortex [2] – consists of HTM neurons (described below in section II.4) organized into a set of *N* minicolumns with *M* cells per column (Fig. 2). This is the HTM network schema that learns sequence transitions from one symbol to the next in the Simple Reber Grammar, along with the suitable temporal context, using sparse distributed representations (see section II.7).

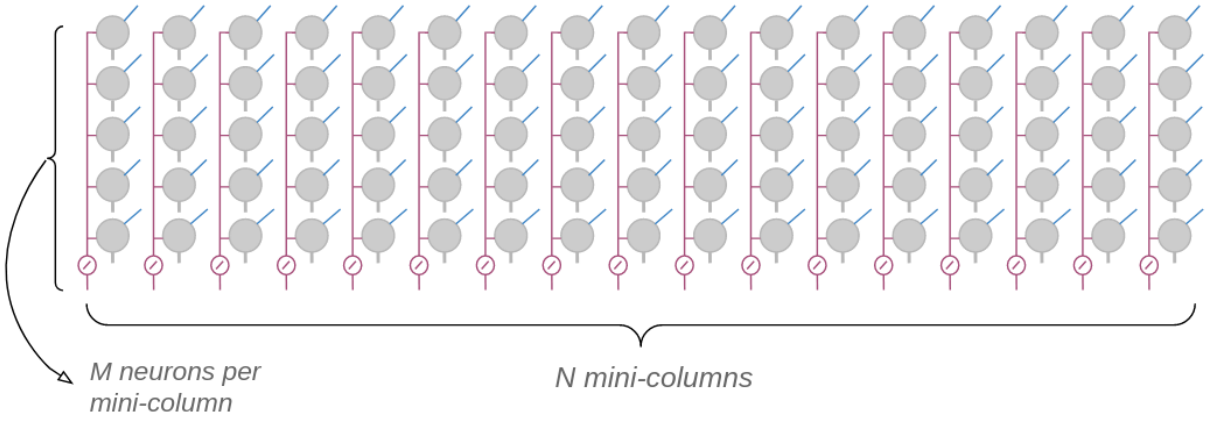


Fig. 2. Schematic layout of an HTM network. HTM cells are shown in grey circles. Each cell is marked by three small, colored segments denoting the three different components of the HTM cell – axon (grey), proximal dendrite (magenta), distal/lateral dendrite (blue). For details, see Fig. 3.

Notably, this single grid of HTM neurons is called an HTM level/region in the original model [3]. An HTM network is, then, made up of several such regions that are hierarchically stacked on top of one another with both feedforward and feedback connections.

However, in the current study, only *one* of these regions of the HTM model will be used and henceforth, the diagram above will be called an HTM network.

II.4 HTM Neuron

An HTM neuron/cell is inspired by several neuroscience studies [14, 15, 20, 46, 49] which together suggest that neurons do not perform a simple, weighted linear sum of their inputs and fire based on that sum – which is predominantly what neurons in most neural network models do [43]. Instead, an HTM neuron has dendrites – that act as independent pattern recognizers – resembling the characteristics of cortical neurons and dendrites, such as the sustained and significant depolarization of the soma after a dendritic NMDA spike. It is well known that such prolonged subthreshold depolarization of the neurons results from the co-activation of several distal synapses localized within spatial proximity of each other on any given dendritic branch [46, 49].

II.4.1 Synapses in HTM

Every HTM cell has a set of dendrites (described below in section II.5), each of which contains a set of synapses. A synapse is simply a directed connection from the "axon" of a presynaptic neuron in the network to a particular dendritic segment of another neuron in the network. However, the concept of synapse is a little bit more elaborate in HTM, as explained in the following.

Permanence Values and Synaptic Weights

Every synapse is associated with a float value between 0 and 1 (both ends inclusive) that is called as its ***permanence value.*** If the permanence value exceeds a threshold (called ***permanence/connection threshold***), such as 0.5, then the ***weight of the synapse*** is 1 and the synapse is considered as ***connected.*** In contrast, if the permanence value is at or below the permanence threshold, then the weight of the synapse is 0 and the synapse is considered as ***unconnected.*** Thus, *the synaptic weights in HTM are not continuous real values.* The connection

threshold represents the establishment of a synapse, albeit one that could easily disappear. A permanence value close to zero represents an axon and dendrite with the "potential" to form a synapse but have not commenced growing one. A permanence value of 1 represents an axon and dendrite with a fully-formed synapse.

Potential Synapses

Learning in HTM occurs by incrementing/decrementing the permanence value of a synapse using Hebbian plasticity, as explained later in section II.9. It should be noted that although an unconnected synapse has a weight 0, yet it possesses the potential to become a *connected* synapse, if repeatedly reinforced by the Hebbian plasticity mechanisms during learning such that its permanence value shoots off the permanence threshold. Thus, an unconnected synapse will, henceforth, be referred to as a ***potential synapse.*** Also, note that the number of potential synapses is usually larger than the number of connected synapses.

II.4.2 Possible States of HTM Neuron

An HTM neuron has *three* different possible states: an active state, a predictive state, or an inactive state.

Active State

A neuron is ***active*** when its proximal dendrite is active, and when it is uninhibited by any other cell in the minicolumn. This state is equivalent to a neuron firing an action potential.

Predictive State

A neuron is ***predictive*** when any one of its distal dendrites is active (see subsection II.5.2 below). This state is equivalent to a neuron being *subthreshold depolarized* due to generation of a dendritic NMDA spike [46].

Inactive State

A cell remains ***inactive*** if it is neither active nor predictive. This state is equivalent to a neuron being hyperpolarized.

Output

The *output* of a cell is always binary: it is active (fires) or not. Mathematical equations and details are given in section II.8.

II.5 Dendritic Structure of HTM Neuron

In the original HTM model [13], there are three mutually exclusive zones/sets of dendritic branches for synaptic integration, as described below.

II.5.1 Proximal Dendrites – For Feeding External Inputs

This set comprises a single dendrite that is used for feed forwarding the input coming externally from the (sensory) inputstream. The proximal dendrite has a relatively large effect on the generation of action potential by the soma and hence, it crudely models the basic receptive field of the neuron [72].

A single proximal dendrite is shared by all the neurons in a single minicolumn (see Fig. 3). This means that *all the cells in a minicolumn get the same feedforward input.*

Note: In this study, there is *no explicit modeling of the proximal dendrite* – with variable synaptic permanences – to feed the external input to the network. This is simply because it is unnecessary for the current problem, where each symbol of the Reber string can be fed directly to a *pre-chosen* set of proximal dendrites (and, in turn, to a pre-chosen set of minicolumns). Thus, in this study's model, a proximal dendrite becomes *active* when the input symbol corresponding to that proximal dendrite is present in the input stream. An active proximal dendrite then feeds its (binary) activation to all the cells in the corresponding minicolumn. Details on this are elaborated later in section II.6 and subsection II.7.1.

II.5.2 Distal/Lateral Dendrites – For Learning Sequential Transitions

This is a set of dendritic segments that contain synapses representing lateral connections between neurons in the network (see Fig. 3). These are the dendrites on a neuron that recognize distinct

patterns of activation in the network that just precede the neuron's firing. Hence, *distal dendrites are the ones responsible for learning transitions between patterns of activity.*

Several studies have shown that coincident detection of presynaptic activity on 8-20 synapses in close spatial proximity on a dendritic branch leads to a nonlinear NMDA spike in the dendrite [74, 75, 49]. This phenomenon of coincident presynaptic activity detection – through only the *connected synapses*, and disregarding *potential synapses* – on a dendritic segment of a cell in HTM theory leads to the particular dendritic segment becoming active [1]. More specifically, a distal segment becomes active when the number of synapses connected to the currently active cells in the network exceeds a certain threshold (called the ***coincidence/NMDA threshold***). In this way, a small set of localized *connected* synapses on a dendrite can essentially act as pattern detectors. The detection of an SDR pattern on a dendrite causes the cell to enter the predictive state. For an illustrative example, see section II.6.
This behavior is akin to the one described in Antic et al., 2010 [46], where a neuron enters a slightly depolarized state – but not enough to generate an action potential (subthreshold depolarization) – whenever there is a dendritic NMDA spike, resembling a cortical up state that is sustained for a longer time span (~ 100ms) than the depolarization caused in the event of an action potential that is only sustained for a relatively shorter timespan (~ 10ms) [46].

More importantly, a slightly depolarized cell fires earlier than it would otherwise, if it only receives enough feedforward input subsequently (via proximal dendrites). This early firing in the cell leads to ***inhibition*** of other cells in the minicolumn, which forms the basis of sparse coding of information in the activity of the network [13] (see section II.6).

It is important to note that the actual activation pattern that is detected by a distal segment can (and generally does) consist of a potentially large number of neurons since it is distributed over a large population of cells, as a whole. Even so, it can be shown that detection of activation of only a few of these cells (8-20 neurons, as studies above have shown) is sufficient to reliably recognize the activation pattern distinctly (see chapter IV, IV.3.2).

II.5.3 Apical Dendrites – For Top-Down Feedback

This is another set of dendritic segments that is functionally similar to the distal segments, in that it also "generates an NMDA spike", upon recognizing a pattern, that slightly depolarizes the soma. However, this set of segments is used to establish a top-down expectation based on the feedback from other HTM regions/layers.

Note: In this study, there are *no apical dendritic segments* on any of the cells in the network.

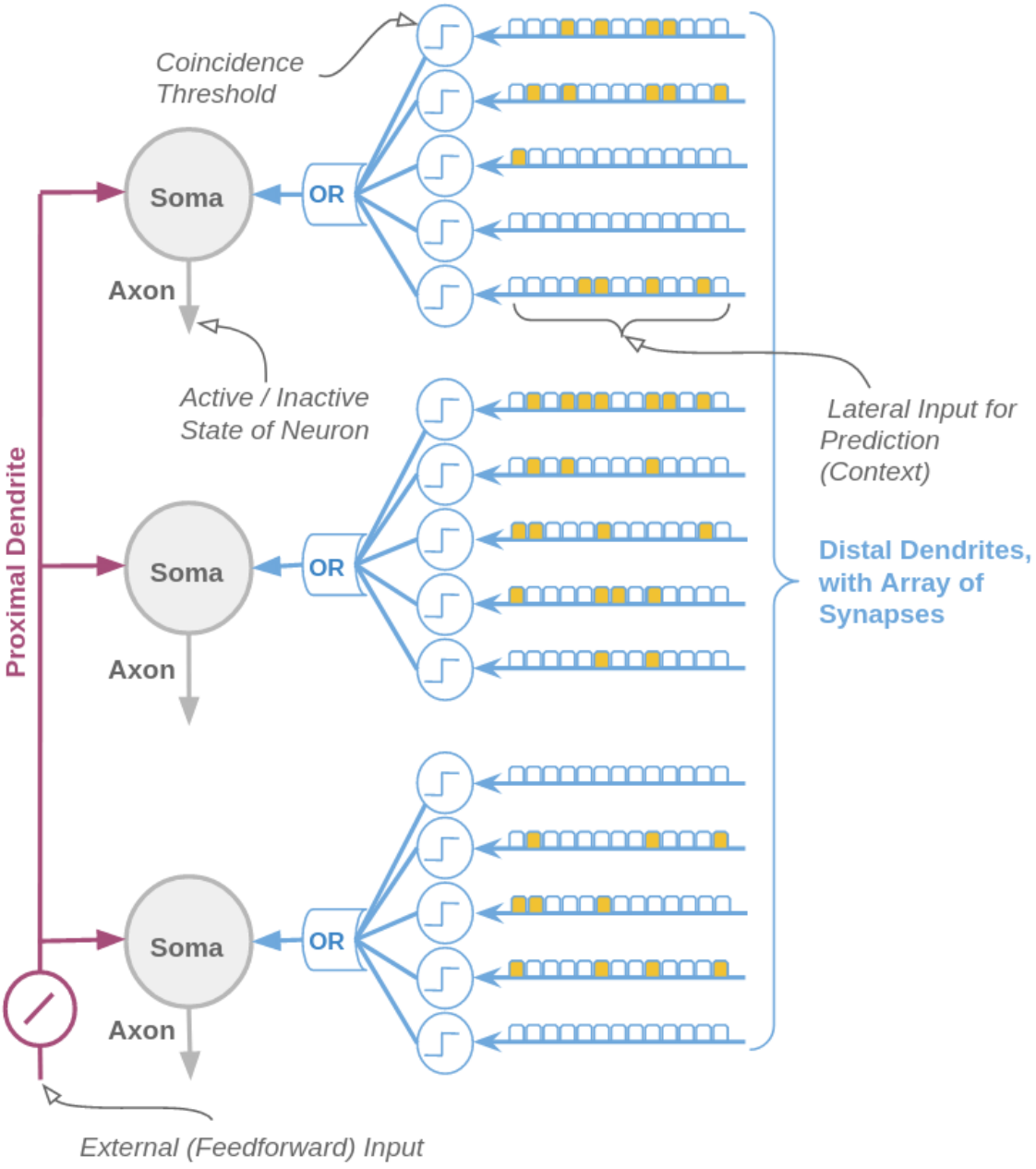


Fig. 3. Three HTM cells in a minicolumn of the HTM network; with a single proximal dendrite shared by all the 3 cells and with 5 distal dendrites on each cell. Each cell has a binary output – active or inactive. Proximal dendrite is used to feedforward the external input to an entire minicolumn. Every distal dendritic segment on a cell has (an array of) synapses connecting with a subset of other cells in the network. Hence, a single cell might have multiple connections to another cell via different dendritic segments. In the diagram above, a filled synapse denotes connection to an *active* presynaptic cell. An unfilled synapse denotes connection to an inactive presynaptic cell. A cell becomes predictive if any one (OR operation) of its lateral dendritic segments is active. A lateral dendrite becomes active when the coincidence threshold is reached with enough coincident presynaptic activity on the corresponding dendrite. (Inspired by Fig. 1C in [13])

II.6 Minicolumn in HTM: Excitation & Inhibition of Neurons

This section discusses the mechanisms of activation and inhibition of cells in the HTM network and, in turn, illustrates the underlying process of generation of sparse representations of inputs.

As stated before in II.5.1, proximal dendrites are used in HTM to feedforward the external input in the network; which is the same input for all the neurons in a particular minicolumn.
Consider a particular minicolumn of HTM with three neurons, as depicted in Fig. 4(a) below. Let this minicolumn's proximal dendrite receive feedforward input at timestep t. Also, consider the three independent sets of distal dendrites on the neurons with connected synapses on each dendrite (as shown in Fig. 4(a), left); and where some of the synapses have received presynaptic activity from the previous timestep's (t-1) activation in the network.

Let the coincidence threshold θ of the network be set to 9. Clearly, no distal dendrite on any of the three neurons have *enough* number of synapses receiving presynaptic activation from the previous timestep i.e., active synapses on any of the distal segments are not higher than or equal to 9 (in Fig. 4(a), left). As a result, none of the distal segments in the entire minicolumn are active and consequently, no neuron in the minicolumn is predictive.
Accordingly, all the cells in the minicolumn will become active in the current timestep upon receiving the activation from their shared proximal dendrite (in Fig. 4(a), right). This phenomenon is called ***bursting of a minicolumn*** in HTM.

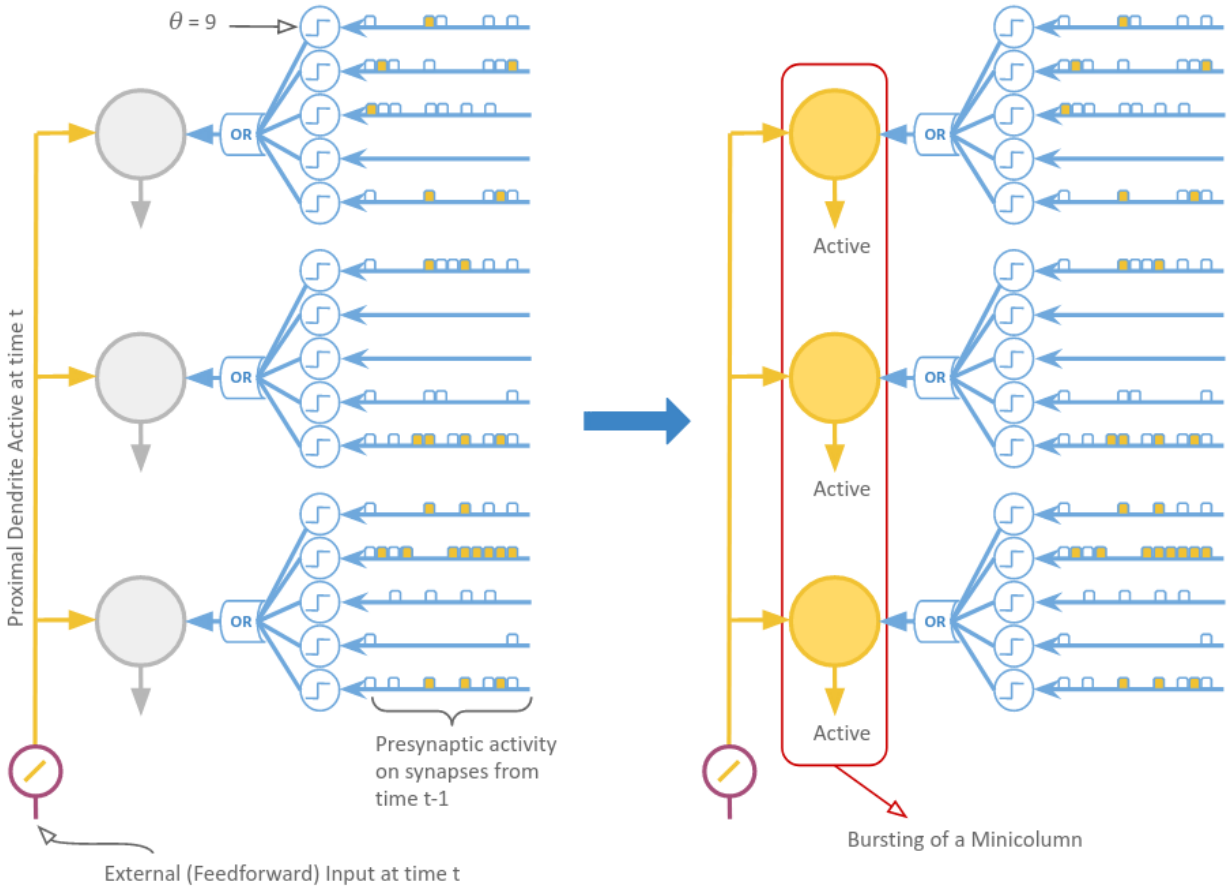


Fig. 4(a). (*Left*) Three HTM cells in a minicolumn of the HTM network with their shared proximal dendrite receiving feedforward input at timestep *t*. The most number of connected synapses on a distal dendrite that receive presynaptic activation from timestep *t*-1 is six (on the third cell's second segment). Since the coincidence threshold θ is set to 9, none of the distal dendrites on any of the three cells in the minicolumn becomes active. Accordingly, no cell in the minicolumn is in a predictive state. (*Right*) As a consequence, there is no inhibition occurring in the minicolumn and hence, all the cells in the minicolumn fire upon receiving feedforward input via their proximal dendrite (also called *bursting of minicolumn*).

Now, for the sake of argument, let the coincidence θ be lowered from 9 to 6. This time, the second distal dendrite on the third neuron in the minicolumn has *just enough* number of synapses receiving presynaptic activation from the previous timestep i.e., active synapses on this distal segment is equal to θ. As a result, this particular distal segment is active and consequently, the third neuron in the minicolumn is predictive (in Fig. 4(b), left).

Accordingly, the predictive third cell inhibits the first two cells (even though they receive the same feedforward activation from the proximal dendrite) in the minicolumn; thereby leading to only one

cell in the minicolumn becoming active at the current timestep *t* (in Fig. 4(b), right). This is the basis of the generation of sparse activation patterns in HTM.

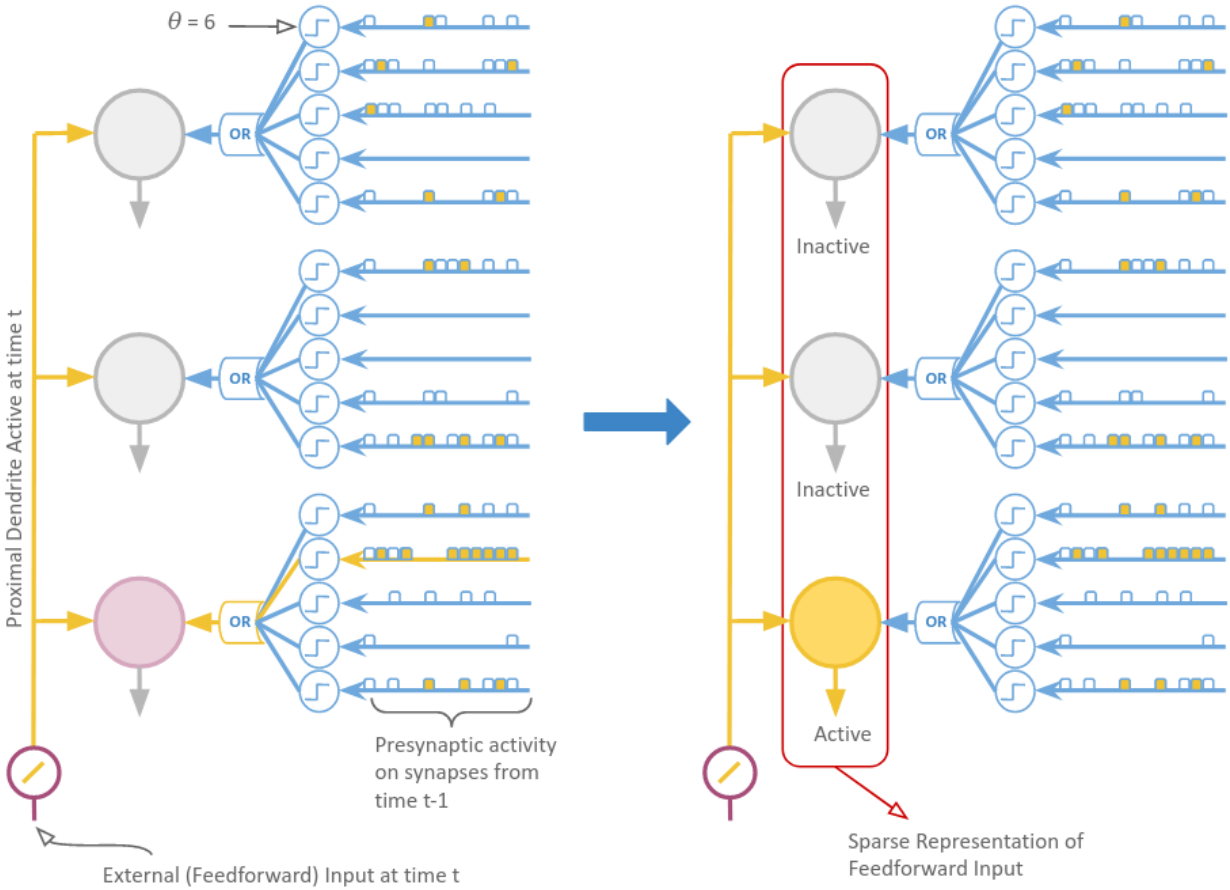


Fig. 4(b). (*Left*) Three HTM cells in a minicolumn of the HTM network with their shared proximal dendrite receiving feedforward input at timestep *t*. The most number of connected synapses on a distal dendrite that receive presynaptic activation from timestep *t*-1 is six (on the third cell's second segment). Since the coincidence threshold θ is set to 6, the second distal dendrite on the third cell in the minicolumn becomes active (in yellow). Accordingly, the third cell becomes predictive (in pink). (*Right*) As a consequence, it inhibits the other two cells in the minicolumn and hence, only the third cell in the minicolumn fires upon receiving feedforward input.

It is important to note that the distal dendrite that led to the sparse activation in the minicolumn became active upon detecting a certain activation pattern in the population from the previous timestep. In other words, this particular distal segment can be said to be "storing" a certain sequential transition of coincident activation patterns in the network; while at the same time the

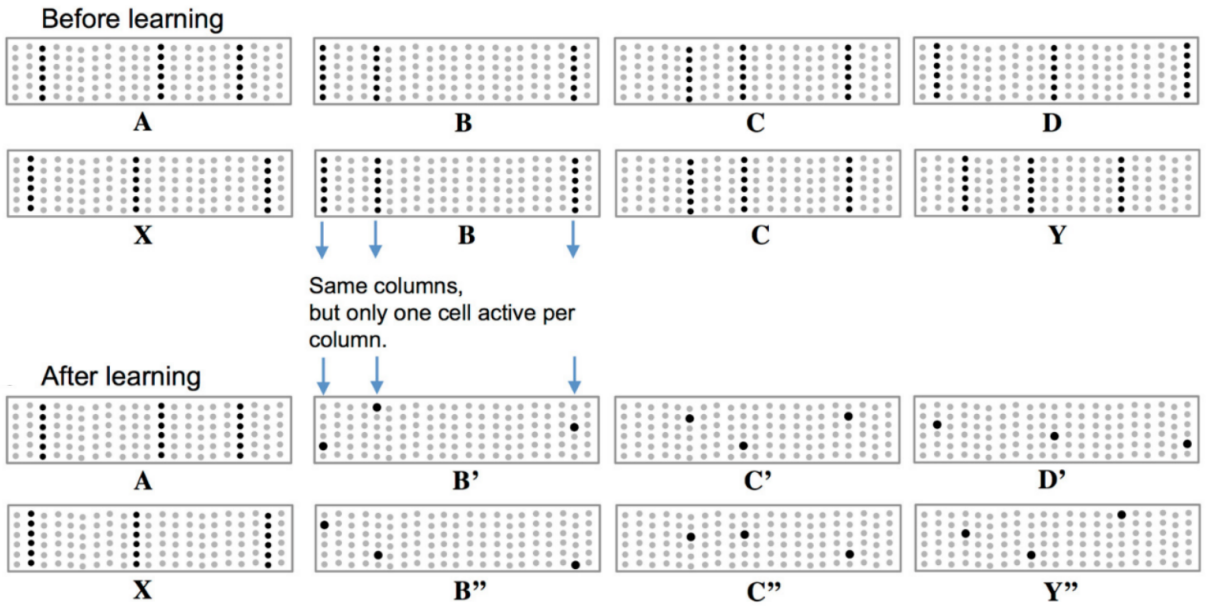
activation of this distal dendrite led to a new sparse activation in the network, which signifies the particular temporal context that the activated distal segment was able to detect.

This section dealt with sparse activation in a single minicolumn of HTM. The next section discusses how sparse activation patterns in a collection of minicolumns lead to the representation of the same inputs differently, in different temporal contexts.

II.7 Sparse Distributed Representations in HTM

In order to represent and learn sequences with contextual dependencies that span several timesteps – so called *higher-order* sequences – HTM relies on a composition of *two* levels of sparse representations, simultaneously. One represents a sparse code for the feedforward input to the network at any given timestep; while the other represents an even *sparser* code for the same feedforward input, but in a particular temporal context. This is illustrated with the help of an example in the following.

Consider the following sequence learning example[10] where the network learns two abstract sequences <*ABCD*> and <*XBCY*> consisting of symbols {*A, B, C, D, X, Y*} (Fig. 5).



[10] The example is adopted directly from Hawkins & Ahmad, 2016 [13].

Fig. 5. *Top:* Each sequence element in the input sequences <*ABCD*> and <*XBCY*> invokes a sparse set of minicolumns (here, 3) out of the total 21 minicolumns. All the 6 cells in a minicolumn become active (black) prior to learning the sequences. *Bottom:* After learning the two sequences, the inputs invoke the same mini-columns but only one cell is active in each column, labeled B', B'', C', C'', D' and Y''. Time flows from left to right. All inactive cells are grey. (Figure adapted from Hawkins & Ahmad, 2016 [13], Fig 2B, 2C)

II.7.1 Sparsity within Minicolumns of Network

As shown in Fig. 5 (top), every distinct letter is represented by a *distinct* sparse set of randomly chosen minicolumns. In the depiction, every letter at each timestep is represented by 3 distinct minicolumns of the network. The network contains a total of 21 minicolumns. In general, an input symbol at timestep t would invoke k minicolumns out of the total set of N, where k is a small fraction of N.

This is the first level of sparse representations in HTM, which is at the level of minicolumns. In the current study, a set of k proximal dendrites (or k minicolumns) gets randomly pre-selected for each of the seven symbols/letters in the SRG. Hence, when an input letter from the Reber Grammar appears in the inputstream at timestep t, it activates all of its corresponding k proximal dendrites. Consequently, there is no explicit modeling of synaptic growth on the proximal dendrites of the model setup in this study[11]. Expressed differently, *there is no learning of sparse representations of the input at the level of minicolumns.*

It is important to note that once an occurrence of a particular input symbol activates a certain set of k minicolumns, it will always choose the same set of k minicolumns on every subsequent occurrence of that symbol in the inputstream. This means that *the set of k chosen minicolumns for*

[11] In the original version of HTM [13], an algorithm called Spatial Pooler (SP) invokes – via an intricate intercolumnar inhibition mechanism that evolves as a result of Hebbian learning – a sparse set of minicolumns to represent the feedforward input at the current timestep. Recall from subsection II.5.1 that all the cells of a particular minicolumn share the same proximal dendrite. The SP, as a result of invoking activations in minicolumns in response to the current input, models synaptic growth in the proximal dendrites of the HTM cells.
Since the current study's model does not require it for the AGL task, the Spatial Pooler algorithm is not discussed in any detail here. For details, refer to [73].

any particular symbol is independent of the temporal context and only depends on the particular symbol at the current timestep.

II.7.2 Sparsity within Cells of Minicolumn

After having chosen a sparse set of *k* minicolumns for the current timestep's feedforward input, the second level of sparse representation comes at the level of individual cells within these *k* minicolumns in order to represent the temporal context.

At any particular timestep, a subset of *k* cells in the *k* active minicolumns (one cell per minicolumn) will represent information regarding the temporal context of the current input. This subset of chosen cells within the *k* minicolumns is learned using synaptic growth during the learning process where connections are created (and subsequently reinforced) between active cells from the previous timestep to those in the current timestep via Hebbian plasticity mechanisms. Details are discussed in section II.9.

As depicted in Fig. 5 (bottom), after learning the sequences <*ABCD*> and <*XBCY*>, input symbols *B*, *C*, *D*, *Y* are represented by sparser activations (only one cell per minicolumn). Additionally, note how the representation of *B* and *C* are different in different temporal contexts of *A* and *X*. Furthermore, since symbols *A* and *X* do not have any preceding temporal context, their occurrence in the input always invokes all the 6 cells in all of their corresponding 3 minicolumns.

II.7.3 Prediction in HTM Using SDRs

Once relevant synaptic connections are learned i.e., when potential synapses between cells in a sequential transition have become connected synapses, the HTM network generates predictions based on the input and the learned connections at the current timestep. These predictions comprise the collection of cells in the network that have a predictive state at the current timestep. Recall from II.5.2, that distal dendritic segments on each cell are responsible for the cell turning from inactive to predictive state.

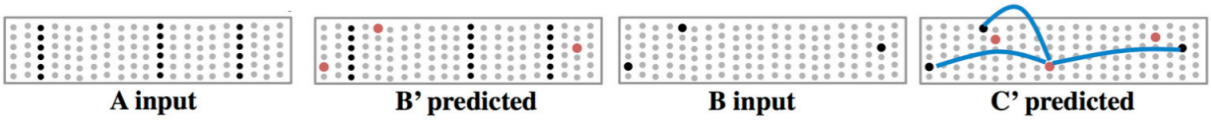


Fig. 6. Prediction in HTM based on temporal context. Active cells and predictive cells are depicted in black and red, respectively. Inactive cells are grey. Time flows from left to right. Lateral dendritic connections on one of the predictive cells of *C'* are shown in the rightmost figure in blue. (Figure adapted from Hawkins & Ahmad, 2016 [13], Fig 3A)

As depicted above in Fig. 6, when the sequence <*ABCD*> has been learned by the network, the input *A* (which has no prior context and leads to firing of all the cells in the 3 minicolumns) leads to prediction of *B'* (3 red cells in the 3 minicolumns of *B*). This SDR of *B'* stores the temporal context of *A* and when *B* finally comes in the inputstream at the next timestep, only the 3 predictive cells of *B* (representing *B'*) become active, inhibiting other cells in the minicolumn as explained in II.6. The occurrence of *B'* leads to the prediction of *C'* and so on.

As a follow-up remark, it is important to note that had letter *C* come in the inputstream after *A* (by chance or due to noise), then the 3 minicolumns associated with *C* would have *burst up*, since none of the cells in the 3 minicolumns of *C* were predicted. This phenomenon is akin to the network showing surprise – higher cell activity – on receiving unanticipated input.

Multiple Predictions when Temporal Context is Ambiguous

The HTM model can also handle cases where the temporal context for a symbol is unspecified. As shown in Fig. 7, if the overlapping subsequence <*BC*> is given as input to the network – after learning both <*ABCD*> and <*XBCY*> – then since *B* was not predicted (i.e. *B* was unexpected), all the 3 minicolumns associated with *B* burst up. However, this leads to the prediction of both *C'* and *C''*.

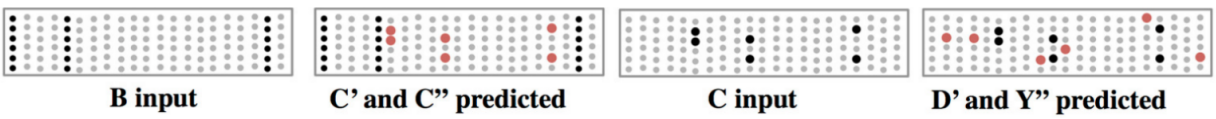


Fig. 7. Multiple Predictions in HTM in the absence of specific context or ambiguity. Active cells and predictive cells are depicted in black and red, respectively. Inactive cells are grey. Time flows from left to right. (Figure adapted from Hawkins & Ahmad, 2016 [13], Fig 3B)

Subsequently, when *C* comes in the inputstream, all of the predictive cells of *C* become active. This, in turn, leads to the prediction of both *D* and *Y*. In this way, the HTM network has the ability to hold multiple predictions too.

Multiple Predictions when Input is Not Fully Deterministic

More significantly, the ability to make multiple predictions is necessary in cases where there is inherently some stochasticity in the inputstream. This is obviously true for the present problem domain of AGL. As mentioned earlier in section II.1, wherever there are more than one possible transitions from a given node in the SRG, all the possibilities are equally likely to occur in the following transition and hence, there is an inherent stochasticity in any input Reber String.

As shown later in chapter III, III.3, the HTM network automatically learns to predict all of the next possible symbols in the input string. This is also the used standard for measuring performance of the network on the AGL task, as described later in section II.11.

II.8 Formalization of Activation & Prediction in HTM

This section lays out a mathematical formalization of activation and prediction of cells in the HTM network. Basically, it provides mathematical notation to the activation, prediction and inhibition mechanisms that were more pictorially illustrated before in section II.6.

As stated earlier in II.3, the HTM network consists of *N* minicolumns with *M* cells per minicolumn. Hence, let $A^t = \{a_{ij}^t\}$ and $P^t = \{p_{ij}^t\}$ be *M*x*N* binary matrices representing the active state and the predictive state of the *MN* neurons in the network at timestep *t,* respectively.

Each cell is associated with a set of distal/lateral segments, D_{ij}, such that D_{ij}^d represents the d^{th} segment of the i^{th} cell in the j^{th} column. Each distal dendritic segment contains a number of potential/connected synapses, representing distal/lateral connections to the other *MN* cells. Therefore, $D_{ij}^d$is also an *M*x*N* matrix with permanence values – always bound between 0 and 1 (both

inclusive) – of each of the potential/connected synapses on the dendrite *d*. Let $\hat{D}^{d}_{ij}$ be the *M*x*N* binary matrix containing only the *connected* synapses[12].

Furthermore, let W^t be a binary 1x*N* matrix containing 1s at the indices of *k* active columns (also called ***active columns***) associated with the input symbol at time *t*, and 0s otherwise. In other words, W^t represents which proximal dendrites are active at timestep *t* and is therefore, a representation of the sparsity at minicolumns' level, as explained earlier in subsection II.7.1. Each input symbol is represented uniquely by a binary 1x*N* vector with *k* 1s for each symbol in the artificial grammar such that there is no overlap of any 1's position in the N-vector for two distinct symbols[13]. *Accordingly, W^t also represents the feedforward input to the HTM network at each timestep t.*

II.8.1 Activation of Cells

As stated before in subsection II.5.1, all the cells in a mini-column share the same feedforward input. Activation of the i^{th} cell in the j^{th} column is computed using:

$$a_{ij}^{t} = \begin{cases} 1; & if\ j \epsilon W^t\ and\ p_{ij}^{t-1} = 1 \\ 1; & if\ j \epsilon W^t\ and\ \sum_i p_{ij}^{t-1} = 0 \\ 0; & otherwise \end{cases}$$

… (Eq. 1, adapted from [13])

In natural language:

- If there is a cell in predictive state from the previous timestep in an active column in W^t, the first part of Eq. 1 activates that particular predictive cell in the current timestep *t*. This phenomenon is also described as the cell (*i*, *j*) being ***correctly predicted.*** This is exactly equivalent to the process (stated before in subsection II.5.2) where a slightly depolarized cell fires earlier than it would otherwise, leading to *inhibition* of other cells in the

[12] Every distal dendrite has a certain capacity of containing a maximum number of *connected* synapses. The number of *connected* synapses on a distal segment cannot exceed this number (see model parameters in section II.12). However, the number of *potential* synapses on a distal segment are not restricted in count.

[13] As stated before in II.7.1, W^t is uniquely pre-selected for each symbol of the SRG in this study, before the HTM network starts to learn.

minicolumn. This is the basic principle underlying generation of SDRs in the network, as illustrated in section II.6.

- Else, if no cell in an active column is in a predictive state, the second part of Eq. 1 activates all cells in that column (also known as *bursting* of the minicolumn, as illustrated in II.6).
- The cell remains *inactive*, otherwise.

II.8.2 Prediction of Cells

Prediction of the i^{th} cell in the j^{th} column is computed using:

$$p_{ij}^{t} = \begin{cases} 1; & if\ \exists d, \left\| \hat{D}_{ij}^{d} \odot A^{t} \right\| > \theta \\ 0; & otherwise \end{cases}$$

… (Eq. 2, adapted from [13])

where, $\odot$ represents element-wise multiplication of the two matrices and $\|.\|$ represents scalar addition of all entries in the resulting matrix. θ represents the coincidence/NMDA threshold.

In natural language:

- A cell enters a predictive state if at least one of its distal dendritic segments is active.
- As stated earlier in II.5.2, a distal dendrite is active if the number of *connected* synapses with active presynaptic cells is more than the coincidence/NMDA threshold at a given point in time.

II.9 Formalization of Learning in HTM[14]

In this section, the Hebbian learning mechanisms of growing and updating synaptic connections between neurons in the network are formalized. Learning in HTM is performed via generic neuroplasticity mechanisms based on the Hebbian rule – *cells that fire together, wire together*.

[14] While the activation and prediction rules presented in the previous section are identical to the corresponding ones presented in Hawkins & Ahmad, 2016 [13], the learning rules are not. The learning rules are adapted from Numenta's digital book called Biological and Machine Intelligence [3], chapter 6 on Temporal Memory.
This version is different from the one in [13] (and also [1]) in that it is optimized for speeding up the simulation for large networks. In this version, the network is not initialized with a complete set of synapses across every dendritic segment on every cell. Instead, cells are randomly selected in an unpredicted (bursting) minicolumn and initialized with potential synapses on a dendrite by sampling from previously active cells. Notably, this version of the learning rules is also the one that most other HTM studies and applications use.

II.9.1 Initialization of Network

The HTM network starts with *M*x*N* HTM cells where each cell has a certain maximum number of distal dendrites on them. However, none of the distal dendritic segments is initialized with synapses. This approach of starting the learning process with no synapses initialized on any distal segments of any cell in the network is adopted to speed up the learning process for large network sizes. Synapses on distal dendrites are initialized on the go, as learning commences.

The following subsections detail the learning rules of sequence-transitions' memory that utilize *only* the distal dendritic segments on HTM cells. Thus, a distal dendritic segment is simply referred to as dendrite or dendritic segment in the following subsections.

II.9.2 When a Minicolumn Bursts (No Cell in the Column is Predicted)

Since the network starts with no dendritic segments initialized on any cell (resulting in no predictions occurring), bursting of entire minicolumns is the dominant form of network activity in the beginning of the learning process.

If an active column *j* in W^t bursts, new synaptic connections are learned on a dendrite of a cell in the column which will represent the temporal context in the future, in case the current sequence transition repeats. This cell on which new synapses are learned is called the ***winner cell in the minicolumn.*** Collection of all the winner cells in a particular timestep *t* are maintained in *M*x*N* binary matrix $C^t = \{c_{ij}^t\}$.

Best Matching Dendrite is Found

For every bursting column in W^t, a dendritic segment *d* is chosen on a cell *i* such that it has the maximum number of active synapses, at any permanence level between 0 and 1. Simply put, the chosen segment *d* is closest to being active i.e., it has the most input even though it is below threshold.

Furthermore, in order to prevent any spurious matching of a relatively very small number of synapses (like only 1 or 2 synapses receiving presynaptic activity) on a dendritic segment of a cell in a minicolumn, there is a lower threshold called the ***learning threshold*** which signifies the minimum number of active synapses that any dendritic segment must have in order to be chosen as a learning segment.

The corresponding equation for selection of the dendrite is:

$$\forall j \in W^t \left(\sum_i p_{ij}^{t-1} = 0\right) and \left(\left\|\bar{D}_{ij}^d \odot A^{t-1}\right\| = max_i\left(\left\|\bar{D}_{ij}^d \odot A^{t-1}\right\|\right)\right)$$

… (Eq. 3, adapted from [13])

where, $\overline{D}_{ij}^{d}$ is a binary matrix containing only the positive (>0) entries in D_{ij}^{d} and the max value on the RHS is at least equal to the learning threshold.

Thereafter, the permanence values of synapses with presynaptic activity on the selected segment *d* are reinforced using Eq. 4, as follows:

$$\Delta D_{ij}^d = p^+(D_{ij}^d \odot A^{t-1}) - p^- D_{ij}^d$$

… (Eq. 4, adapted from [13])

Specifically, all the synapses on the dendrite *d* have their permanence values decreased by a small value p^- and then synapses with active presynaptic cells are rewarded by increasing their permanence values by a larger value p^+. $p^+ = 2p^-$ in this study.

The cell *i* in the bursting minicolumn *j* then becomes the winner cell of the column i.e., $c_{ij}^t = 1$.

In cases where the number of connected synapses on the best matching dendrites is lower than the dendrite's maximum capacity, new synapses are "grown" (initialized) to connect to a subset of the previous timestep's winner cells. The count of the newly formed synapses is upto the remaining capacity of the dendrite to have connected synapses.

Best Matching Dendrite is Not Found

However, since there are no synapses initialized on any of the dendrites in the network, no best matches to the active cells from the previous timestep are found in the beginning.

In such cases where no best matching dendrites are found, the network randomly selects a cell *i* with most number of uninitialized dendritic segments – breaking ties randomly – and selects an uninitialized segment on this cell to "grow" new synapses to a subset of the previous timestep's winner cells only. This chosen cell *i* in the bursting minicolumn then becomes the winner cell of that column. The newly grown synapses are initialized at a certain initial permanence value of 0.25 $\pm$0.002 (normally distributed)[15].

It is important to note that new synapses are initialized only to a subset of the previous timestep's winner cells – that already represent the relevant context from the timesteps before – and not to all (or a random subset thereof) of the previous timestep's active cells. Again, this shortcut is taken only to speed up the learning process.

II.9.3 When a Cell is Correctly Predicted in a Minicolumn

If a cell was correctly predicted, then the dendritic segment that was active and caused the cell to become predictive is selected by:

$$\forall j \in W^t \ (p_{ij}^{t-1} > 0) \ and \left\| \hat{D}_{ij}^{d} \odot A^{t-1} \right\| > \theta$$

… (Eq. 5, adapted from [13])

The selected dendritic segment *d* is then reinforced using Eq. 4 above; and, the corresponding cell index (*i, j*) is added to the winner cells C^t i.e., $c_{ij}^{t} = 1$.

II.9.4 When a Cell is Predicted in a Minicolumn But Remains Inactive Subsequently

It may also happen that a cell gets (incorrectly) predicted at a particular timestep but remains inactive at the subsequent timestep – if any of its dendritic segments gets mistakenly reinforced by chance (which is likely to happen in the beginning of the learning process) or if the cell was part of the network's multiple predictions at a given timestep.

In this case, a very small decay p^{--} is applied to active dendritic segments of cells that did not become subsequently active, using:

[15] In the original HTM implementation of [1], [3] and [13], new synapses are initialized with permanence values at a single point value of 0.21, without any gaussian spread around it.

$$\Delta D_{ij}^{d} = p^{--} D_{ij}^{d} \ , \ if \ a_{ij}^{t} = 0 \ \ and \ \left\| \hat{D}_{ij}^{d} \odot A^{t-1} \right\| > \theta$$

… (Eq. 6, adapted from [13])

p^{--} = -0.2p^{-} in this study.

In the end, for each dendrite *d* of every cell (*i*, *j*), the matrix ΔD_{ij}^{d} is added to the current matrix of permanence values at every timestep.

II.9.5 When the External, Incoming Input Repeats[16]

When a given letter/symbol in a Reber string repeats (for example, the repetition of *"S"* in <*ATSSSP...*>), it is reasonable that all the cells in the corresponding minicolumns of the network would burst upon encountering a repetition i.e. the second or third repetition of the letter/symbol, since this would be considered as a novel transition, at first, by the network. This bursting phenomenon would lead to an unintended and non-optimal side effect with the existing learning rules laid out above.

This can be explained as follows with the example of the repetition of *"S"* in Reber string <*ATSSSP...*>:

Upon the bursting of *"S"*-minicolumns after the second repetition (transitions *A*→*T*→*S*→*S*) for the very first time in the inputstream, a set of *k "S"*-cells will be chosen as winner cells to represent the context <*ATSS...*>. However, upon the bursting of *"S"*-minicolumns after the third repetition in the current string (transitions *A*→*T*→*S*→*S*→*S*) – or higher number of repetition in any future Reber string – it is almost certain that a different set of *k "S"*-cells will be chosen as winner cells to represent the context <*ATSSS...*> since the process of selection of winner cells upon bursting of a minicolumn is random. This will lead to different *"S"*-representations for the different repetitions of *"S"* – the second, third, fourth and so on.

[16] This mechanism is not implemented in the original HTM model of [1], [3] and [13]. Instead, the HTM model uses what is called *backtracking*, which essentially tries to find out which starting point in the past leads to the current input with as most context as possible. In other words, this process locks onto the current set of inputs by assuming that the sequence started up to N steps ago on start cells. This particular feature of the HTM algorithm that gives it the best chance of making correct predictions going forward – especially in a continuous learning setting – has been unfortunately omitted from all of Numenta's published work.

It is clear to see that all these different repetitions of *"S"* are indeed the same *"S"* in the Reber grammar and, thus, it stands to reason that they must all be represented identically in the SDR-scheme of representation.

In an attempt to not deviate too far from the biological plausibility of the learning rules, it is proposed that the network, upon encountering a repetition in the input string, should use the same set of cells to grow new dendrites or reinforce existing synapses as the cells which were winner cells in the same repeating active minicolumns ($W^t = W^{t-1}$) from the previous timestep. In notational form: If $W^t = W^{t-1}$, then $C^t = C^{t-1}$.

The selection of a matching dendrite is done using:

$$\forall (i,j) \in C^t \, (c_{ij}^t = 1) \; and \; (\left\| \bar{D}_{ij}^d \odot C^{t-1} \right\| = max_i(\left\| \bar{D}_{ij}^d \odot C^{t-1} \right\|)) \quad \ldots \text{(Eq. 7)}$$

The selected dendritic segment *d* is then reinforced using Eq. 4 above.

In case where no matching dendrite is found, the network initializes a new dendritic segment on cell *i* to "grow" new synapses to connect to the previous timestep's winner cells. As before, the new "grown" synapses are initialized at a certain initial permanence value of 0.25 ± 0.002 (normally distributed).

Following this simple strategy ensures that the same SDR representation is used for every letter's representation. Taking the example of string <*ATSSSP...*> above, the network will form the same SDRs for all of the three repetitions of *"S"* in the string.

Notably, there are only two letters *"S"* and *"T"* which can repeat subsequently in a string from the Reber Grammar of Fig. 1.

II.9.6 Pruning Unused Dendrites[17]

Besides having a set of connected and/or potential synapses, every dendrite on every cell in the network has a certain duty cycle too.

Essentially, a ***dendrite's duty cycle*** tracks if the dendrite is being used by the network for learning the sequential information. Accordingly, there is an *upper limit* to the number of timesteps for which a dendrite in the network is allowed to exist without having been either active or initialized with new synapses. This is referred to as the ***maximum dendrite dormancy.***

In other words, every new dendrite in the network exists for a fixed number of timesteps – equal to 'maximum dendrite dormancy' value – before it gets pruned for being inactive or for not being used for synapse growth. This process is proposed in order to keep a check on the network growing too many dendrites that store redundant / spurious / irrelevant temporal contexts. This could be crucial with regards to the learning of Simple Reber Grammar where there are only a small finite number of distinct temporal contexts (trigrams) to learn in total.

Determining Maximum Dendrite Dormancy for Grammar Learning Task

Determination of the size of the period of *maximum dendrite dormancy* – specifically, for the task of learning of SRG – could be made based on an estimate of the maximum number of possible timesteps between two occurrences of any given trigram in the grammar. This is due to the fact that *maximum dendrite dormancy* represents the maximum number of possible timesteps between the occurrence of the same SDR (i.e. temporal context) in the inputstream.

The SRG has a total of 33 possible trigrams, which are not all equally frequent in a large collection of randomly generated strings from the SRG due to the fact that there are inherent "asymmetries" in the directed graph of Fig. 1 that generates the SRG strings. For instance, the top left node has 3 incoming and 2 outgoing edges.

[17] This mechanism, also, is not implemented in the original HTM model of [1], [3] and [13]. Instead, the HTM theory (slightly) punishes synaptic permanence on every dendrite that bears enough presynaptic activity to match the previous timestep's active cells but yet it fails to become active in the current timestep. Eventually if the synaptic permanence reaches zero for a given synapse on a given dendrite, then that synapse is deleted from the memory of the HTM system.

Fig. 8 below depicts the mean counts (with the corresponding standard deviations) for all the 33 trigrams in the SRG in a set of 10,000 randomly generated Reber strings, with an average of approximately 770 unique strings in each collection. Clearly, trigram *"PVZ"* is the most frequently occurring trigram in the collection, with occurrences almost equal to half of the number of strings in the set. Then, there are 12 trigrams (*"ATS"*, *"ATP"*, *"APV"*, *"APS"*, *"SSS"*, *"SSP"*, *"SPV"*, *"SPX"*, *"PXV"*, *"PXS"*, *"TXZ"* and *"SXZ"*) which have a lower frequency of occurrence than that of *"PVZ"* and occur almost as many times as a quarter of the number of strings in the set. The remaining 20 trigrams occur with a frequency that is almost equal to 1/8th the number of strings in the collection[18].

Using this frequency distribution of trigrams, it is logical to estimate the maximum number of timesteps between two occurrences of any given trigram in the grammar based on the average number of timesteps between two occurrences of the *least frequent* trigrams in the Reber grammar.

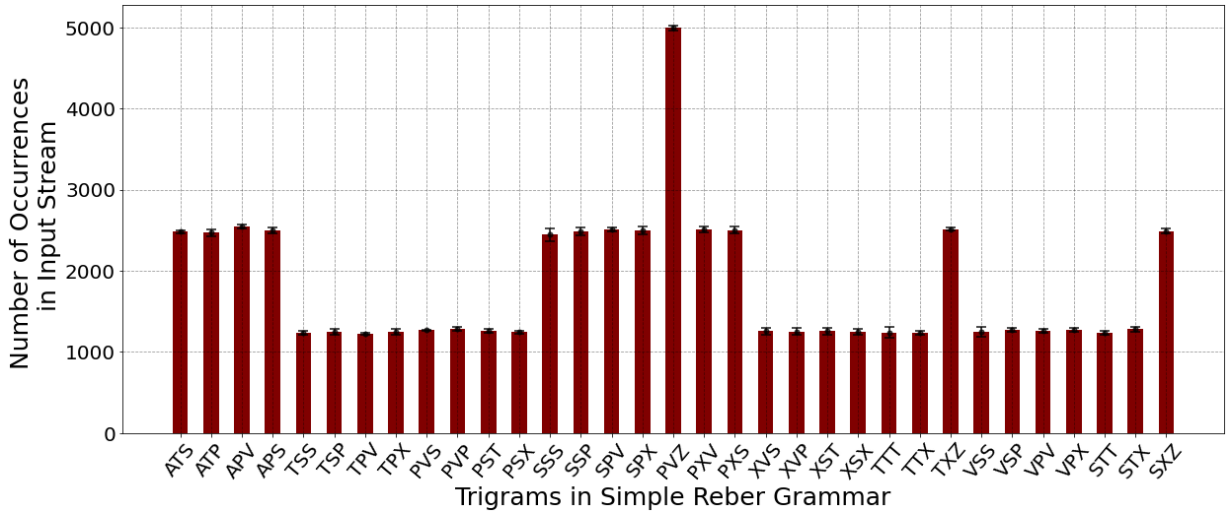


Fig. 8. Frequency distribution of all the 33 possible trigrams of the Simple Reber Grammar in any large collection (10000, in this figure) of randomly generated strings from the SRG.

Hence, if N_s is the total number of strings in the inputstream, then each of the least frequent trigrams repeats approximately $N_s/8$ times in the inputstream. As a result, a trigram with $N_s/8$ total occurrences in the inputstream will repeat every 8th string, if it is assumed that all trigrams are

[18] This overall pattern in frequency distribution of the trigrams in SRG was reliably replicated with other sizes (e.g., 5,000 and 20,000 strings) of the set too.

roughly uniformly distributed throughout the entire length of the inputstream. Based on this, the maximum dendrite dormancy can be reasonably estimated to be at least 8 times the average length of a randomly generated Reber string.

However, since the last assumption (regarding uniform distribution of trigrams) may be easily broken in a not-large-enough random set of Reber strings (as used in this study), a safe and conservative estimate of the maximum dendrite dormancy would be 3 times of the above value. Thus, the current study will use a 'maximum dendrite dormancy' value of 24 times the average length of a random Reber string.

In conclusion, every time a new dendrite gets initialized with synapses, or when an existing dendrite "grows" new synapses to previous timestep's winner cells, or when an existing dendrite becomes active (leading to the predictive state of its cell), the dendrite's duty cycle gets renewed to its 'maximum dendrite dormancy' value. If a certain dendrite's duty cycle falls to zero, it gets automatically pruned by the network.

Experimental Setup

II.10 Methodology Used in Simulated Experiments

In this study, the HTM network is exposed to a set of Reber strings generated using Fig. 1. The input stream is a stream of letters from the Reber grammar where every letter is encoded into a binary N-vector with k entries as 1s and the rest as 0s (see Table 1 below for parameter values). This N-vector is fed into the network (with N minicolumns in total), via the proximal dendrites, at each timestep.

The network then learns a predictive model of the inputstream based on the observed transitions in an online fashion i.e., there are no separate training and test phases, instead the performance is evaluated at each timestep of the inputstream right from the beginning.

Since the learning task at hand is one of acquiring the rule-based, syntactical structure of the Reber Grammar – instead of continuous learning of sequential information – every start of a new Reber string marks a resetting of the learning process in the network. In other words, every occurrence of

letter *"A"* in the inputstream is the point where the network halts learning the previous transition (*Z→A*, *"Z"* from the previous string's end to *"A"* in the current string's beginning) and restarts learning transitions from *A→ …* and onwards.

Consequently, the network does not make any predictions at the final *"Z"* of every Reber string.

II.11 Performance Evaluation

As stated earlier in chapter II, II.7.3, since the AGL task is characteristically stochastic, the HTM network must learn to predict all the possible next letters at any particular timestep.

II.11.1 Prediction Accuracy (PAR) & Prediction Performance Ratios (PPR)

One straightforward way to estimate the performance of the network is to check if the next letters predicted by the network are actually expected or not. However, since the predictions in HTM have an SDR representation, measuring the accuracy of the prediction is inherently challenging.

This is explained in the following:

1. The HTM network will start making predictions for the possible next inputs after having initialized and subsequently reinforced some distal dendrites in the early stages of the inputstream. In order to measure how accurate these *one-step* predictions are, an elementary approach is to take the ratio of the number of minicolumns with a correctly predicted cell and the number of minicolumns with any predicted cell – correct or otherwise. This evaluation metric is, hereinafter, called ***prediction accuracy ratio (PAR).*** Thus,

$$PAR = \frac{\#\ Correctly\ predicted\ columns}{\#\ Total\ Predicted\ columns} \quad \ldots \text{(Eq. 8)}$$

Following this, the prediction accuracy will be 1 (or 100%) if # *Correctly predicted columns* = # *Predicted columns.* However, this approach to evaluate the HTM network's predictions comes with an unintended flaw. Suppose there are two actually expected predictions – say *"V"* and *"S"* after an initial sequence <*AP*> – and the network only correctly predicts one of the letters, then the # *Correctly predicted columns* will be equal to the # *Predicted columns*, and the prediction accuracy would be 1, which is clearly wrong. An evaluation

metric very similar to this is used for evaluating the performance of HTM in Cui et al., 2016 [1] and Hawkins & Ahmed, 2016 [13].

2. An alternative approach – which seems more suitable than Eq. 8 but is also not completely flawless – is to take the ratio of the number of minicolumns with a correctly predicted cell and the actual expected number of minicolumns which must be predicted. This evaluation metric is, hereinafter, called ***prediction performance ratio (PPR).*** Thus,

$$PPR = \frac{\#\ Correctly\ predicted\ columns}{\#\ Expected\ prediction\ columns} \quad \ldots \text{(Eq. 9)}$$

Using this approach, the evaluation of the network's prediction would be rightly calculated as 0.5 (or 50%) in cases where there are two expected predictions and the network only correctly predicts one of them.

However, this approach has a flaw too: if the network predicts additional minicolumns besides the expected prediction columns, then this measure would still estimate a perfect prediction performance of 1.0 at the particular timestep, leaving the incorrectly predicted minicolumns unaccounted. Notably, this error is accounted for by PAR.

In light of the aforementioned considerations, this study will evaluate the network's performance using both of the metrics (PAR and PPR). Therefore, it must be noted that the best one-step predictions are those in which both PAR and PPR are 1.0.

Besides, whenever PAR < PPR (and none equal to 0), then the network has incorrectly predicted some undesirable minicolumns besides correctly predicting some or all of the expected minicolumns. Similarly, whenever PPR < PAR (and none equal to 0), then the network has failed to predict some of the expected minicolumns. Clearly, the two scores are always simultaneously zero at any particular timestep in the inputstream.

II.11.2 Prediction Performance Per String (P3S)

Another useful metric of evaluating the performance of the network is to compute the prediction performance on each input Reber string. Hence, ***prediction performance per string (P3S)*** of the

HTM network on a given Reber string is defined as the average of the prediction performance (PPR) on every letter in the particular Reber string.

Intuitively, this metric roughly measures how much the network has learned / acquired the "syntax" of the Reber Grammar or the grammaticality of a particular input string.

II.12 Model Parameters

The following table lists the various network parameters and their corresponding values used in the simulated experiments. The parameters are mostly different – and better suited to the learning task of this study – than the canonical values used in other HTM models and applications in the literature.

Parameter	**Value**
Number of active minicolumns per letter, k	32
Number of minicolumns (total), N	32*7 = 224[19]
Number of cells per minicolumn, M	16
Max. number of distal dendrites per cell	128
Max. number of *connected* synapses per distal dendrite	49[20]
Initial synaptic permanence	0.25 ± 0.002 (gaussian distributed)
Synaptic permanence decrement, p^-	0.10
Synaptic permanence increment, p^+	0.20
Synaptic permanence decay, p^{--}	-0.02
Connection threshold for synaptic permanence	0.50
NMDA/coincidence-detection threshold, θ	18

[19] There are 7 distinct letters in the Reber Grammar – $\{A, T, P, S, X, V, Z\}$.

[20] The choice of maximum number of connected synapses is based on the requirement that no two different contextual SDRs can be learned on the same distal dendrite segment (*synaptic clustering* [12]). Hence, while this number is kept greater than k, it is one less than $k+\theta$, since θ is the minimum number of connected active synapses that a distal dendrite requires in order to become active and recognize a certain SDR in the population.

Learning threshold	12
Max. dendrite dormancy	24*Average length of Reber string in the inputstream

Table 1. Model Parameters for HTM used in this study.

CHAPTER III: SIMULATION RESULTS

III.1 Learning a Single Reber String

In order to illustrate the various architectural aspects and learning strategy of HTM in a simplistic form, this section discusses the learning of a single Reber string <*APVPVZ*> by the HTM network. The network is exposed to 50 repetitions of the string <*APVPVZ*>, with a total of 300 timesteps / characters in the inputstream.

The task is relatively straightforward since there is no stochasticity involved and there is only one letter to be predicted at each timestep.

III.1.1 Performance on Learning a Single Reber String

Fig. 9 (Top) shows the P3S score (in %) of the network on the entire inputstream. Evidently, the HTM network learns the sequence completely after 11 repetitions.

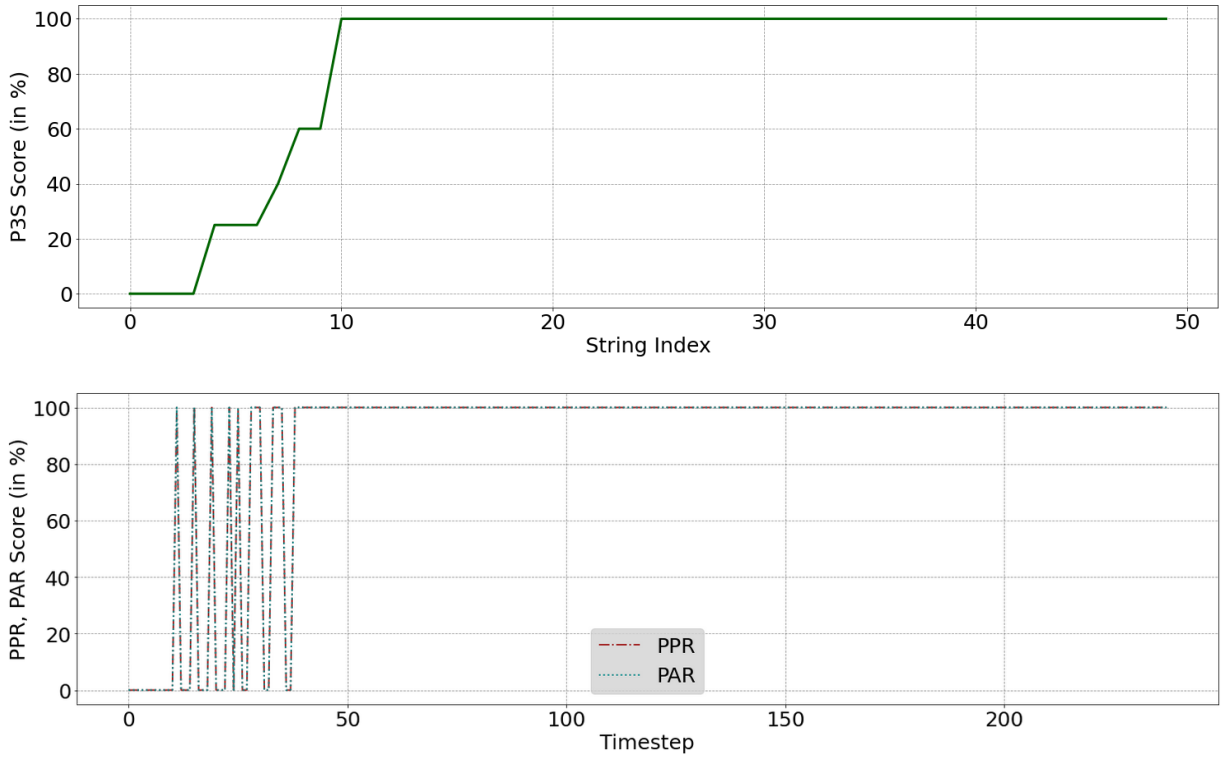


Fig. 9. *Top:* Prediction Performance Per String (in %) of the HTM network on the task of learning a single Reber string <*APVPVZ*>, repeated 50 times in the inputstream. The string index of the first and last string is 0 and 49, respectively. *Bottom:* Prediction Performance and Prediction Accuracy scores (in %) of the HTM network on the same task. The two scores are identical at each timestep.

Fig. 9 (Bottom) shows the PPR and PAR scores (in %) of the network on the entire inputstream. Clearly, the PAR and PPR scores of the network's performance are identical at every timestep since there is only a single possible next letter to be predicted at each timestep.

III.1.2 Network Gradually Disambiguates Different Temporal Contexts

Fig. 10(a) below depicts the network's activity for the 7th repetition – when the network has not yet completely learned the sequential transitions – of the string <*APVPVZ*>. Recall that every letter has k=32 minicolumns for representation; and, thus, there are 32 minicolumns that are either bursting or have sparse activity at each timestep.[21] The bursting of *"A"* minicolumns is due to the resetting of the learning process, as stated earlier in the methodology section II.10.

[21] In 1st repetition, all the minicolumns corresponding to every letter at each timestep burst up, as expected.

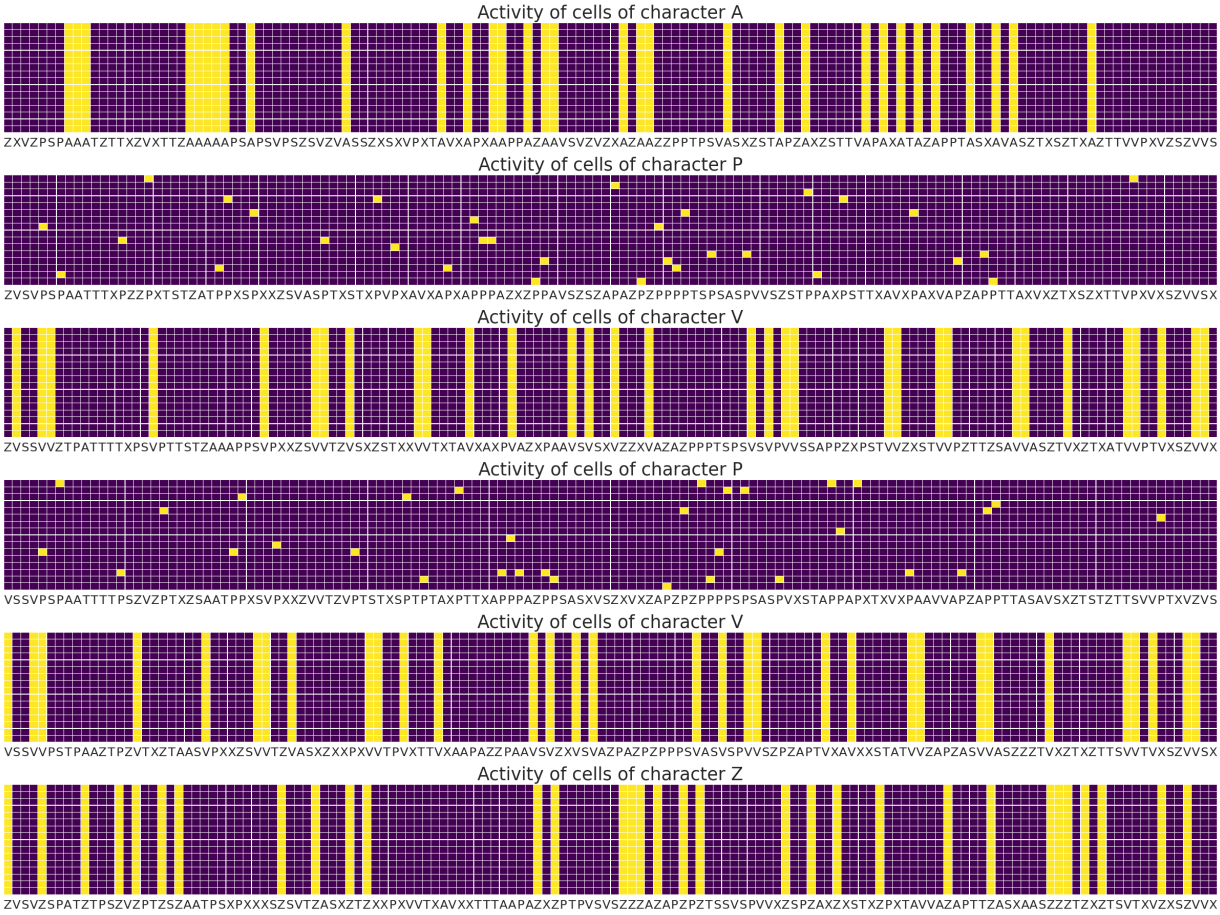


Fig. 10(a). Activity of the network at each timestep of the 7th repetition of the Reber string <*APVPVZ*>. Top to bottom: *"A"* to *"Z"*. Cells in yellow are active, the rest in violet are inactive. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

Evidently, the network has learned the transitions $A \rightarrow P$ and $V \rightarrow P$ with two different SDR representations for *"P"*. Moreover, the network has not yet learned the transition of $A \rightarrow P \rightarrow V$ and $\ldots \rightarrow V \rightarrow P \rightarrow V$ (V-minicolumns bursting). This is due to the fact that the same letter *"P"* comes prior to both of the occurrences of *"V"* and hence, it takes longer for the network to disambiguate the temporal context coming from the previous timestep. Furthermore, as a consequence of this ambiguity in the temporal context of *"V"*, the network wrongly predicts *"P"* at the penultimate *"V"*, instead of *"Z"*, as shown in Fig. 10(b) below.

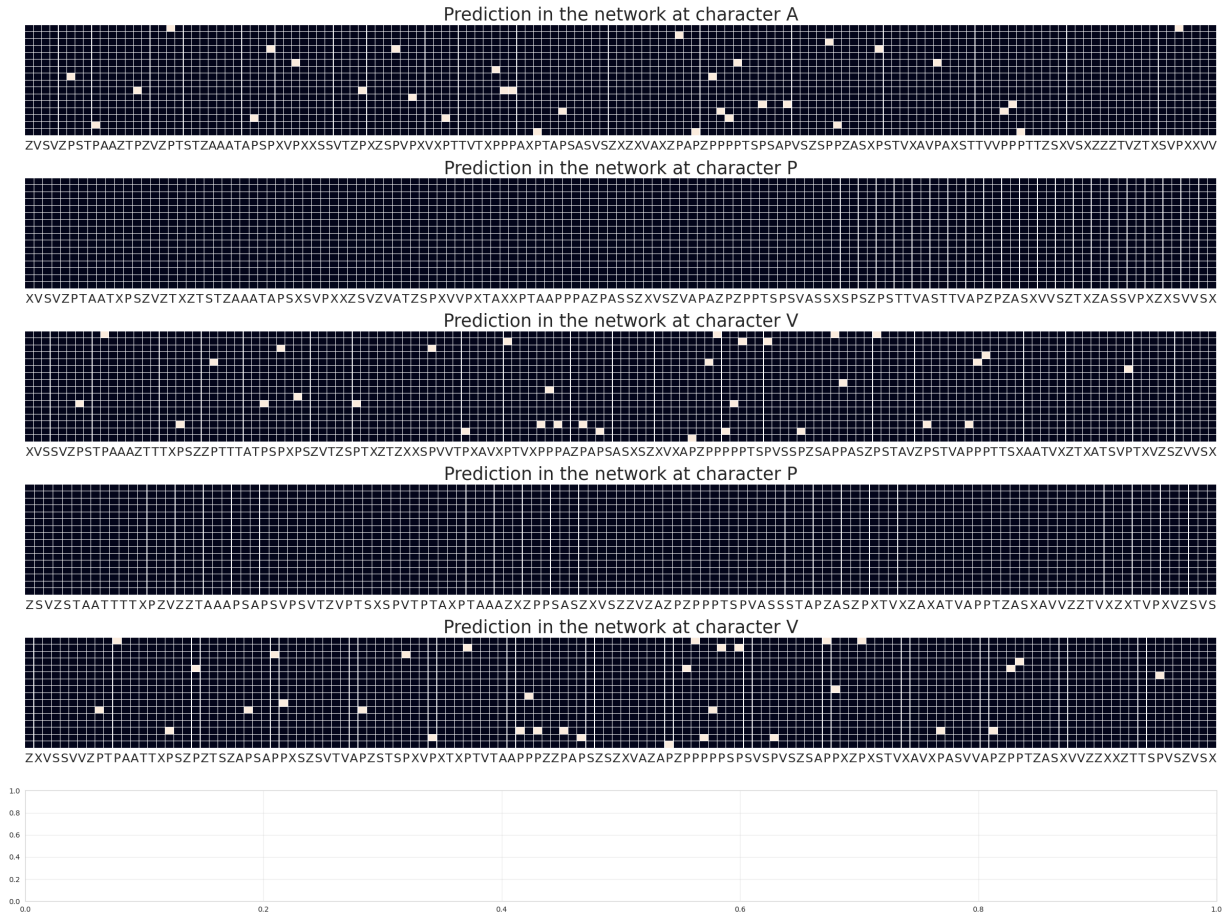


Fig. 10(b). Predictions in the network at each timestep of the 7th repetition of the Reber string <*APVPVZ*>. Top to bottom: *"A"* to *"Z"*. There are no predictions at the ultimate step *"Z"*. Cells in white are predictive, the rest in black are inactive. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

In Fig. 10(c) below, the network's activity for the last (50th) repetition of the string <*APVPVZ*> is shown. Primarily, the activity has now become sparse, with only 1 cell per minicolumn active at each timestep, except at letter *"A"* due to the resetting. Furthermore, it is especially notable how the network has selected different SDRs for *"P"* and *"V"* in their respective different contexts.

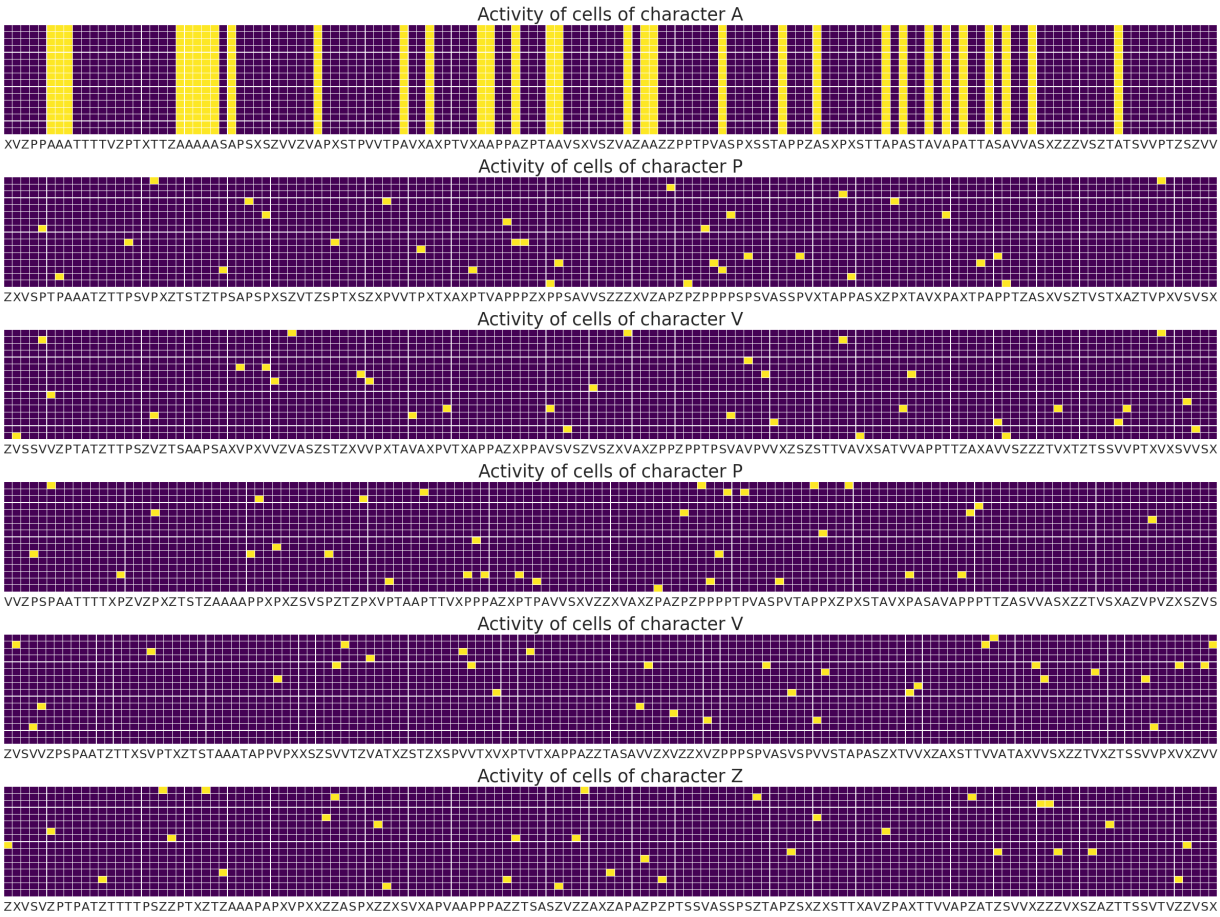


Fig. 10(c). Activity of the network at each timestep of the last (50^{th}) repetition of the Reber string <*APVPVZ*>. Top to bottom: *"A"* to *"Z"*. All the letters in the string, except *"A"*, are represented with very sparse activations, representing the respective temporal contexts. Cells in yellow are active, the rest in violet are inactive. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

III.1.3 Dendritic Growth Profile

Fig. 11 below depicts the number of dendrites on each of the 3584(=224*16) cells of the network after the 50 repetitions of <*APVPVZ*>. Cells of *"A"* do not grow any dendrites, since the *Z→A* transition is not learned by the network. There are two cells initialized with one dendrite each in each one of the 32 minicolumns of *"P"* and *"V"* and only a single cell with one dendrite initialized in every *"Z"*-minicolumn. The rest of the letters have no dendrites initialized on any of their cells.

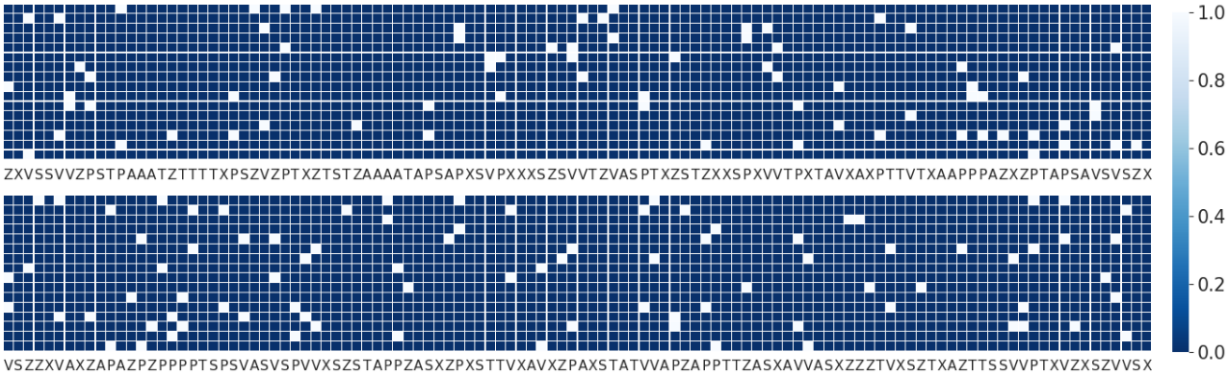


Fig. 11. Dendritic growth on each of the 3584 cells in the network at the end of 50 repetitions of the Reber string *<APVPVZ>*. The entire 224 minicolumns panel has been divided into two equal-sized panels (top panel: 0-111; bottom panel: 112-223). Dendrites have been initialized on cells in only *"P"*, *"V"* and *"Z"* minicolumns. The rest of the minicolumns (in navy blue) corresponding to other letters have no dendrites initialized on any of their cells. All the 32 minicolumns of both the letters *"P"* and *"V"* have 2 cells with one dendrites each for the two learned transitions. Only a single cell is initialized with a dendrite in every *"Z"* minicolumn.

Clustering of "Functionally Equivalent" Synapses on Individual Dendrites

For *"P"*-minicolumns, the two dendrites have each learned the transitions $A \rightarrow P$ and $V \rightarrow P$ respectively. For *"V"*-minicolumns, the two dendrites have each learned the transitions $A \rightarrow P \rightarrow V$ and $V \rightarrow P \rightarrow V$ respectively. And finally, the one dendrite in each of the *"Z"*-minicolumns has learned the final $V \rightarrow Z$ transition. A sample of each of these 5 different types of dendrites is shown in Fig. 12 (a), (b) and (c) below.

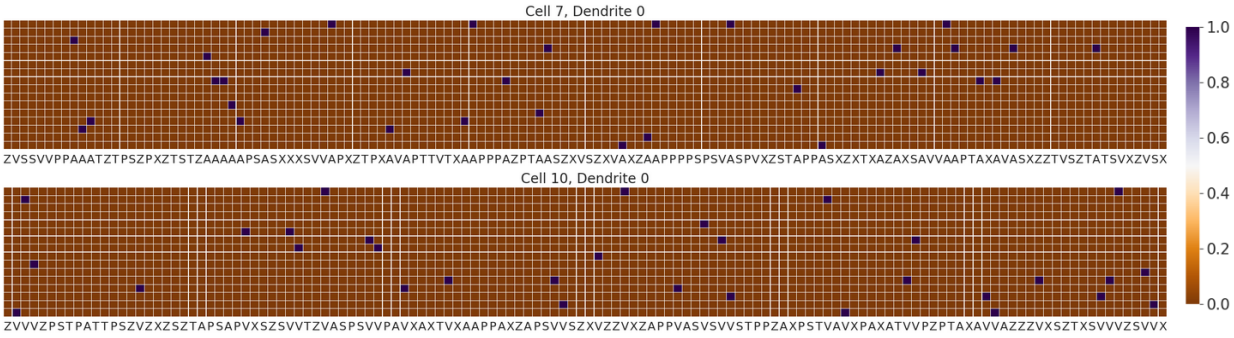


Fig. 12(a). The two dendrites on cells 7 and 10 of the 9th minicolumn (*"P"*-minicolumn) in the network at the end of 50 repetitions of the Reber string *<APVPVZ>*. *Top*: Dendrite with 32 fully-formed (permanence value 1.0) synaptic connections to *"A"* cells. *Bottom:* Dendrite with 32 fully-formed synaptic connections to *"V"*

cells (representing context *A→P→V*). For clarity sake, only around 40% of the minicolumns (randomly selected) without any initialized synapses are shown here.

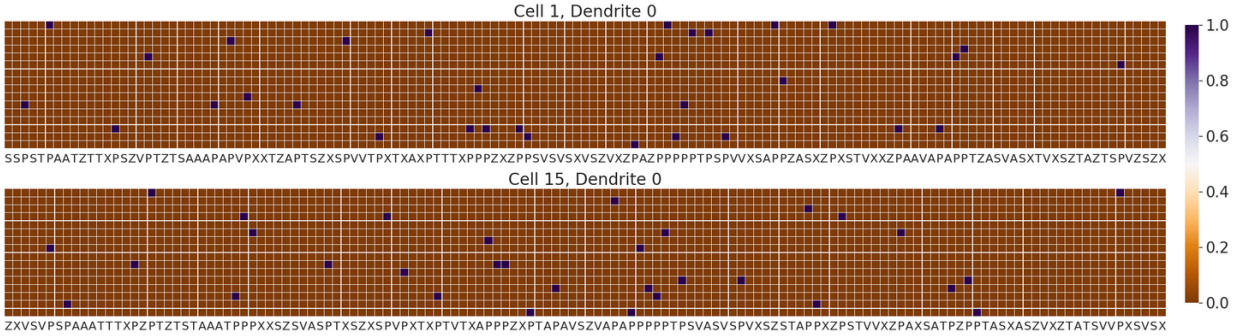


Fig. 12(b). The two dendrites on cells 1 and 15 of the 3rd minicolumn (*"V"*-minicolumn) in the network at the end of 50 repetitions of the Reber string *<APVPVZ>*. *Top*: Dendrite with 32 fully-formed synaptic connections to *"P"* cells (representing context *V→P*). *Bottom:* Dendrite with 32 fully-formed synaptic connections to *"P"* cells (representing context *A→P*).

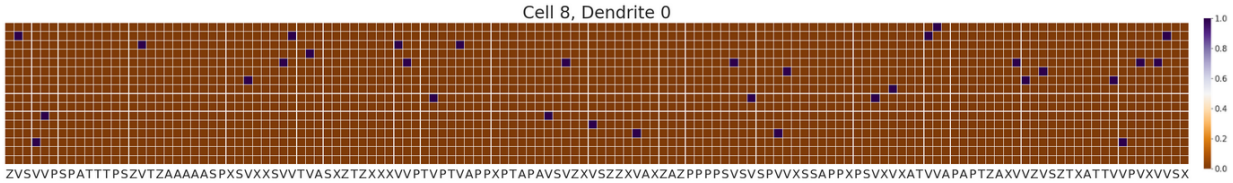


Fig. 12(c). The one dendrite on cell 8 of the 1st minicolumn (*"Z"*-minicolumn) in the network at the end of 50 repetitions of the Reber string *<APVPVZ>*; with 32 fully-formed synaptic connections to *"V"* cells (representing context *V→P→V*).

Incidentally, Fig. 11 and Fig. 12 together demonstrate a significant feature of the HTM learning algorithm. Every dendrite in the network specializes in storing a particular temporal context. In other words, there is clustering of functionally equivalent synapses i.e., synapses storing a certain temporal context, on single dendritic branches of the individual cells in the network. In this way, every cell in the HTM model acts as a recognizer of multiple activation patterns; which is also one of the prominent features of biological cortical neurons [14, 19, 48].

III.2 Learning Long-Range Dependencies

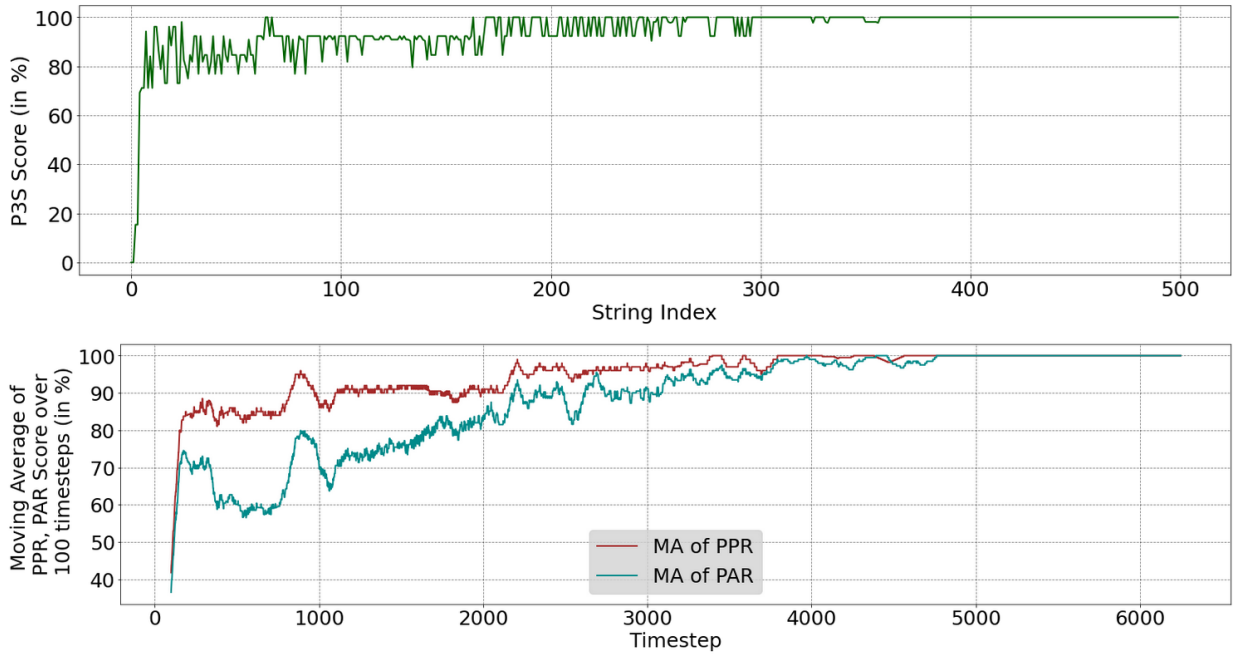
This section presents results on the remarkable ability of the HTM model to implicitly learn long-range dependencies in a sequential stream of input symbols.

The inputstream in this experiment consists of an equal number of occurrences of the following four strings: {<*A**S**PVSPXVPXSX**S**Z*>, <*A**T**PVSPXVPXSX**T**Z*>, <*A**V**PVSPXVPXSX**V**Z*>, <*APVSPXVPXSXZ*>}, with 500 strings in total (the four strings being randomly shuffled) and 6750 characters / timesteps long. The first three strings have a long-range dependency between the second and the penultimate letter (marked by bold case, the two letters are the same), whereas the last one has no long-range dependency. However, all the sample strings share a common substring (marked by an underline).[22]

III.2.1 Performance on Learning Long-Range Dependencies

In Fig. 13 (top), the P3S score (in %) of the network is shown over the length of the entire inputstream; while Fig. 13 (bottom) depicts the 100-timesteps moving average of the PPR and PAR scores (in %). The network achieves a perfect P3S score after about 360th strings. Similarly, perfect PPR and PAR scores are attained in less than 5000 timesteps.

The moving average of PPR and PAR scores over 100 timesteps is performed simply for the sake of clarity in depiction of the improvement in performance of the network over time as it is otherwise difficult to discern in the regular, timestep-wise plot of PPR and PAR scores (shown in Fig. 14)



[22] The four sample strings are not Reber strings.

Fig. 13. *Top:* P3S score (in %) of HTM as the network is exposed to an inputstream of 500 strings in total consisting of an equal number of occurrences of the set of four strings used in this section (see text above). The HTM network achieves 100% P3S score after exposure to less than 400 strings from the set. *Bottom:* 100-timesteps moving average of PPR and PAR scores (in %) of the network. The network attains a 100% averaged PPR and PAR scores in less than 5000 timesteps.

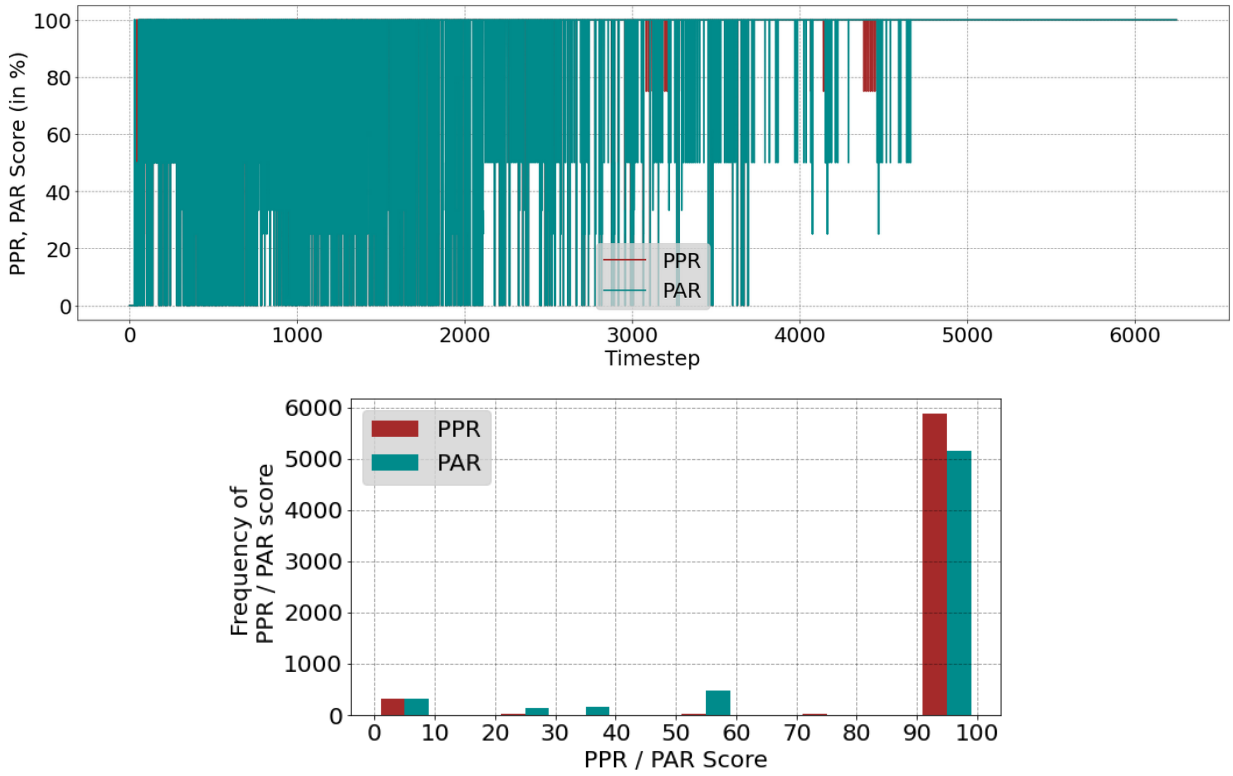


Fig. 14. *Top:* PPR and PAR scores (in %) over the length of the entire inputstream of 500 strings in total consisting of an equal number of occurrences of the set of four strings used in this section (see text above). *Bottom:* Frequency distribution of PPR and PAR scores in the figure from *top*.

It is evident from Fig. 13 (bottom) that the network's prediction accuracy (PAR score) struggles initially – the number of predictions made by the network for the next timestep is more than the required number – as it strives to differentiate the different long range dependencies from each other.

III.2.2 HTM Learns Distinct SDRs for Encoding Long-Range Dependencies

Fig. 15 below depicts SDRs of the first *"P"* letter from the common substring shared by the four different sample strings. Expectedly, HTM uses distinct SDRs to encode the different temporal contexts in which the letter *"P"* occurs in the inputstream. The same is true for the other letters of the common substring. Hence, in essence, HTM utilizes SDRs to distinctly encode all the different letters in the common substring in the context of the different long-range dependencies.

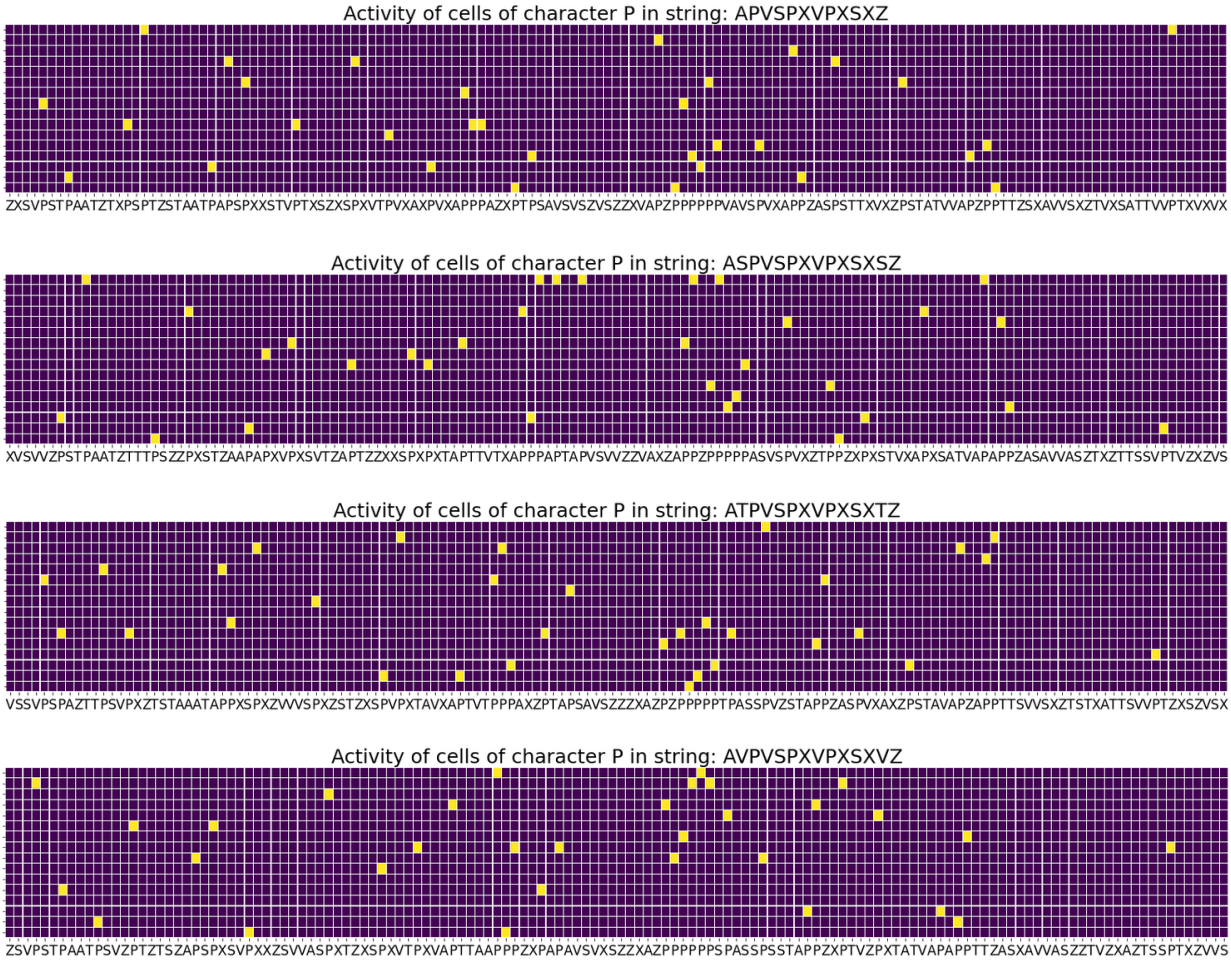


Fig. 15. *Top to bottom:* Four distinct SDRs of the first letter *"P"* in the commonly shared substring in the four sample strings used in this section (see text above). The HTM network uses distinct SDRs for storing distinct temporal contexts. The same is found to be true for other letters of the common substring. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

Although this capacity of HTM to represent different temporal contexts using different SDRs proves successful in allowing the network to automatically learn arbitrarily long long-range dependencies in the inputstream, it has an unforeseen drawback too. The HTM network also generates distinct SDRs for the long-range dependency letters {*S, T, V*} at the second and the penultimate timesteps (see Fig. 16 below).

While this distinction can certainly be understood in terms of the two distinct positions of the letters in the strings – for instance, distinct SDRs for letter *"S"* at the second and the penultimate steps because the two *"S"*s are different in terms of their positions in the sample string – it can be argued that the network is not recognizing them as the "same" letters. Put differently, it can be posited that the HTM model does not "identify" a long-range dependency as a single, collective component spanning over multiple timesteps, since it only learns transitions in sequences from one element to the next.

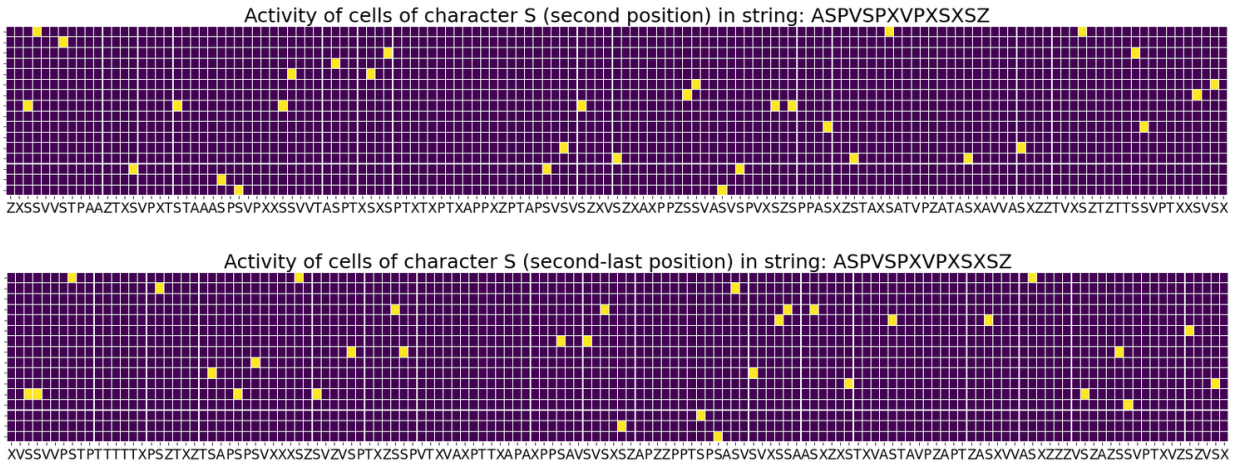


Fig. 16. The HTM network learns two distinct SDRs for the letter *"S"* at the second and the penultimate steps in the same string *<ASPVSPXVPXSXSZ>*. The same is found to be true for the other two long-range dependency letters *"T"* and *"V"*. It can be argued that the network fails to identify a long-range dependency as a single constituent of the string, spanning multiple timesteps in the inputstream. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

This observation becomes rather unsurprising upon reexamination of the HTM model's learning strategy. The HTM network simply looks at the previous timestep's activation patterns in the network and grows synapses connecting the previously active cells to the active cells in the current timestep. In other words, HTM's memory only spans one timestep and the network *solely* relies on SDRs to store the temporal context from a variable number of steps in the history of the sequence.

Hence, HTM can be said to have a *variable-order* memory which enables it to learn/memorize sequences as a whole, instead of recognizing temporal dependencies spanning over a small number of timesteps in subsequences, as one constituent unit in a sequence.

In the next section, more evidence will accumulate in support of this argument.

III.3 Learning / Acquisition of Simple Reber Grammar

This section discusses the performance of HTM on the task of implicitly learning and acquiring the rules of the Reber Grammar.
For the simulation, the HTM network is exposed to a randomly generated set of 2000 Reber strings from the Reber Grammar in Fig. 1. Results of the grammar learning task for a total of 10 trials, each over a different collection of 2000 randomly sampled Reber strings (with no upper bound on an individual string length) are described in the following.

The average total length of the inputstream is 15971 ± 100 characters/timesteps, with a distribution of length of an individual string as shown in Fig. 17. Besides, there are a total of 259 ± 9 unique Reber strings, on average, that the network is exposed to.

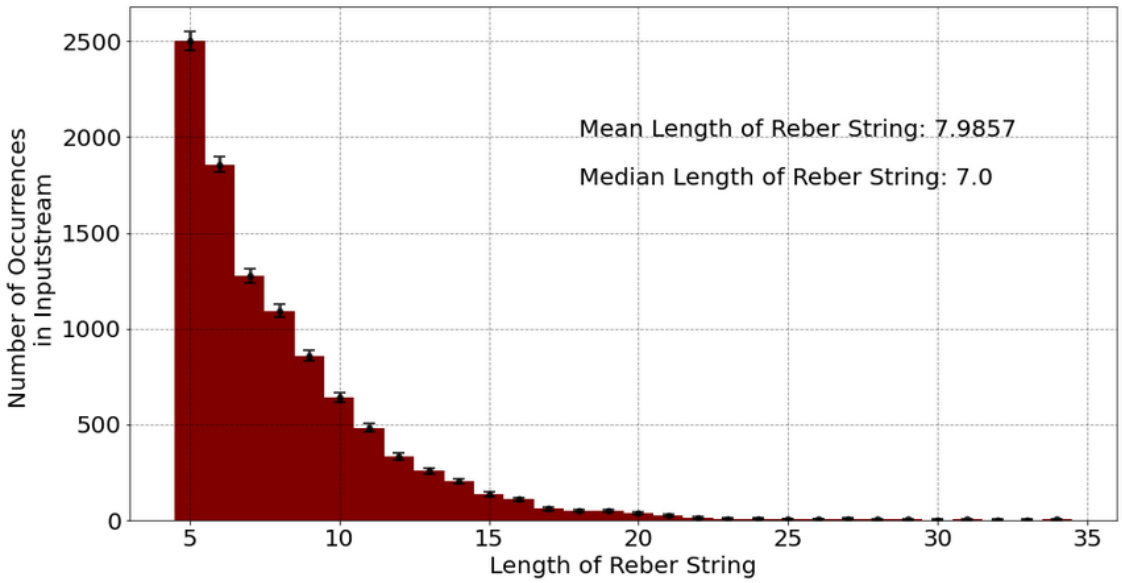


Fig. 17. Distribution of the length of an individual Reber string (on average) in the 10 different sets of 2000 randomly generated Reber strings on which the HTM network is trained for the Reber Grammar learning / acquisition.

III.3.1 Performance on Learning SRG

In Fig. 18 (top), the P3S score of the network is shown over the length of the entire inputstream (averaged over the 10 trials); and Fig. 18 (bottom) depicts the 100-timesteps moving average of the PPR and PAR scores (also averaged over the 10 trials). The network achieves a P3S score of approximately 95% and an average PPR and PAR score of 94% and 93% respectively, in the final 10% of the inputstream.

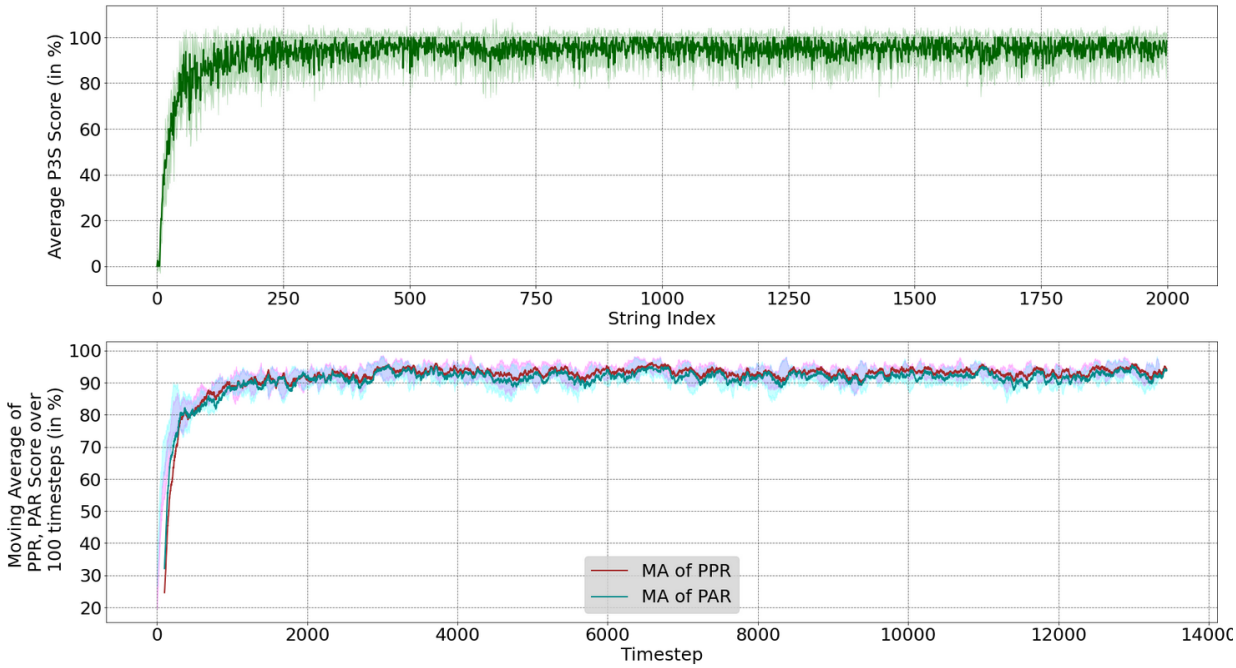


Fig. 18. *Top:* P3S score (in %) of HTM, averaged over 10 trials, as the network is exposed to Reber strings in the inputstream[23]. The task is to learn Simple Reber Grammar. The inputstream consists of a randomly generated set of 2000 Reber strings, with around 259 unique Reber strings in the set. The HTM network achieves over 80% P3S score after exposure to only about 150 Reber strings. *Bottom:* 100-timesteps moving average of PPR and PAR scores (in %) of the network, averaged over the 10 trials. The averaged PPR and PAR scores of the network reach over 90% after 2000 timesteps or 300 Reber strings.

III.3.2 Failure to Learn / Acquire the SRG

It is plain to see from Fig. 18 above that the network fails to achieve perfect scores on the grammar learning task until the very end. Upon further analysis, it is found that the network repeatedly drops to 0% PAR and PPR scores – which means no predictions are made – even at later stages of the inputstream. See Fig. 19 (top) below where the PPR and PAR scores of the network are shown over the length of the entire inputstream for one of the 10 sets of 2000 Reber strings.

The drop to 0% PAR and PPR scores is primarily due to the fact that newer SDRs are continually learned in the network, even at later stages of the inputstream; and this makes the subsequent one-step predictions impossible.

[23] The overshooting of the spread of error beyond 100% at some string indices is due to the fact that each trial has a different string at any given index, leading to a higher variance in the P3S score.

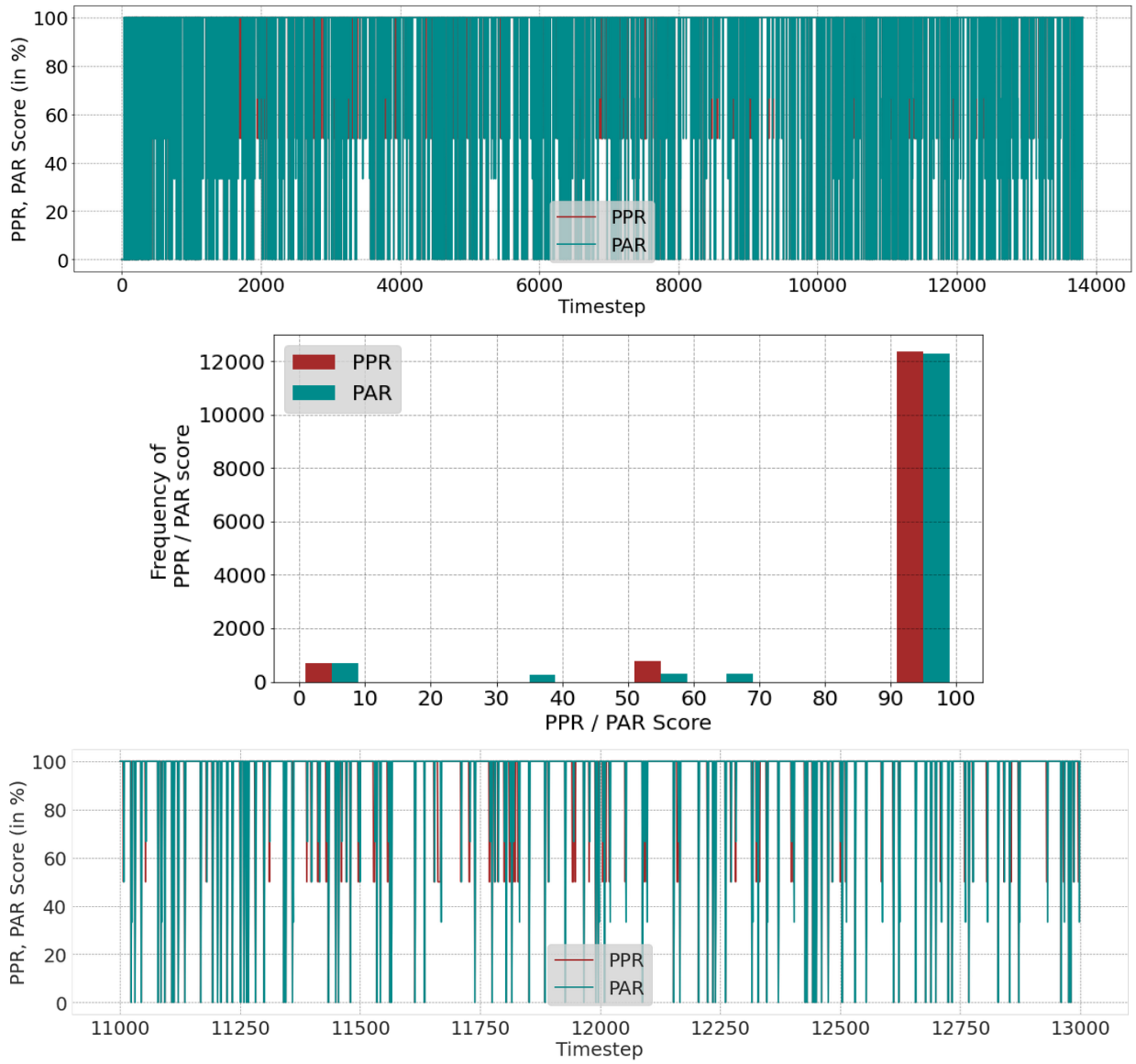


Fig. 19. *Top:* PPR and PAR scores (in %) over the length of the entire inputstream for one of the 10 trials on a randomly generated set of 2000 Reber strings. *Middle:* Frequency distribution of PPR and PAR scores in the selected trial. *Bottom:* Zoomed-in version of the *top* figure between 11000-13000 timesteps, simply for the sake of clearly depicting the PPR and PAR scores for a certain section of the *top* figure.

As an example, consider the 1851st Reber string <*ATPXVPXVSPVZ*> from the inputstream. Fig. 20(a.1) and Fig. 20(b.1) show the activations and predictions in the network for the first 6 letters "*ATPXVP*" of the string, respectively. Fig. 20(a.2) and Fig. 20(b.2) show the activations and

predictions in the network for the last 6 letters "*XVSPVZ*" of the string, respectively. The color scheme in the prediction panels are chosen such that cells of the same color in any particular prediction panel signify all the predicted cells for the same (single) letter.

As can be readily observed, the network has very different SDR representations of *"P"*, *"X"* and *"V"* for the two occurrences of the *"PXV"* trigram; thus, corroborating the assertion that the HTM network does not learn the trigrams generated from the SRG.

The SDR representation of *"V"* in the second *"PXV"* trigram is newly formed and does not generate any predictions at the 8th timestep in the string (see Fig. 20(b.2), second panel from top) . This is validated by the first occurrence of *"V"* where the network has made the correct predictions of both *"P"* and *"S"* (shown in Fig. 20(b.1), second last panel from top). As a result of the new SDR for *"V"*, the subsequent occurrence of *"S"* in the string leads to bursting activity in the network (shown in Fig. 20(a.2), third panel from top).

And, as a consequence of the bursting of *"S"*-minicolumns in the 9th timestep in the string, the network yields all the possible predictions – in all the possible distinct contexts – after *"S"* i.e., prediction of all the possible SDRs of *"P"*, *"S"*, *"X"* and *"T"* (shown in Fig. 20(b.2), third panel from top). At this stage, due to more number of predicted minicolumns than expected, the PAR score of the network at the current timestep drops below 100% (to 50%, in this case); while the PPR score stays at 100%.

This event of multitude of predictions in the network (with multiple possible context) runs down the subsequent predictions and activations in the network – leading to the network behaving as if it has "lost memory" of most of the temporal context of the Reber string. This behavior of the network is seen in the final three timesteps of the Reber string <*ATPXVPXVSPVZ*> (see Fig. 20(a.2)).

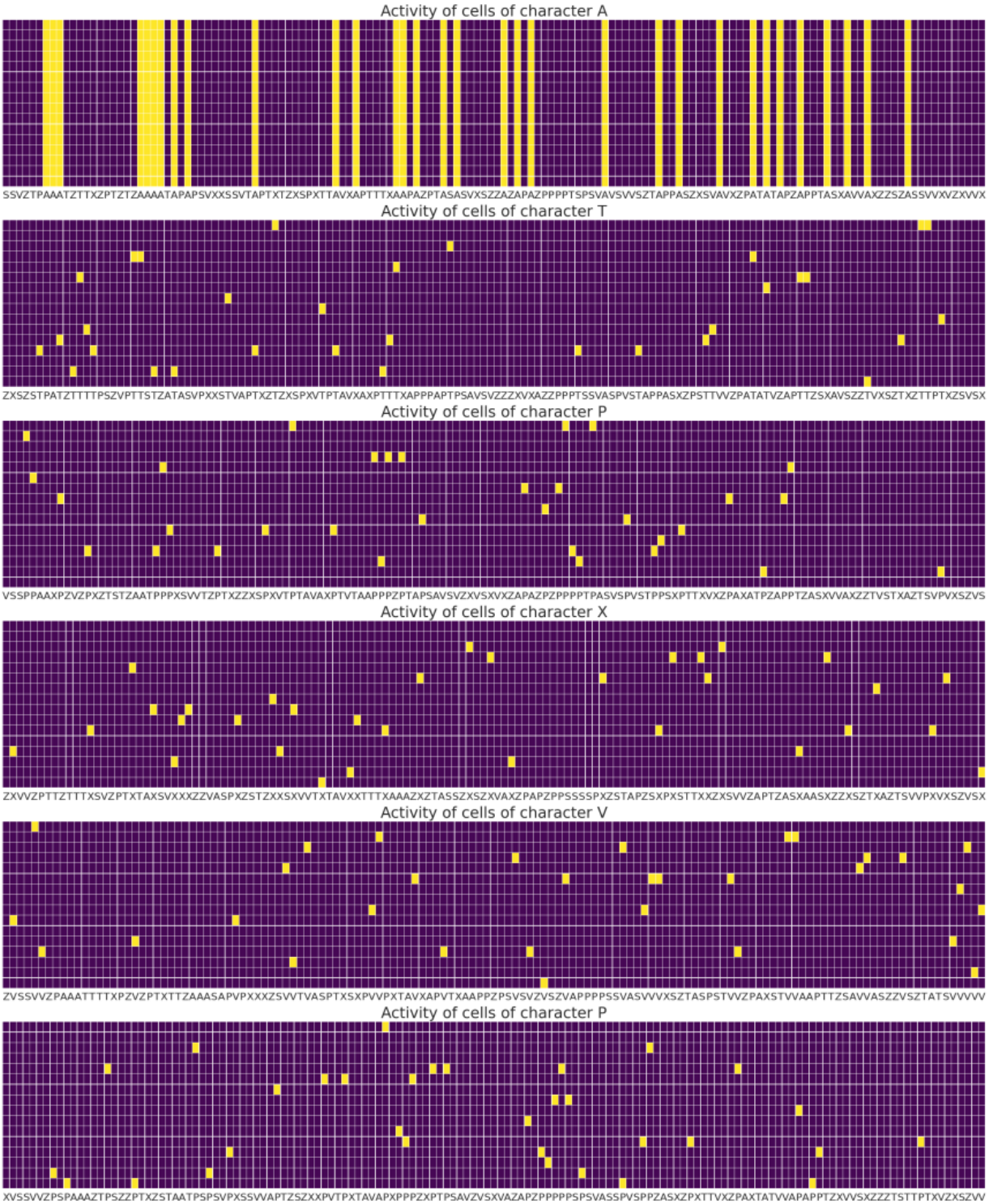


Fig. 20(a.1). Activations in the network for the first 6 letters "*ATPXVP*" of the 1851st string <*ATPXVPXVSPVZ*> in the inputstream of one of the 10 trials on a randomly generated set of 2000 Reber strings. For clarity sake, only around 40% of the inactive columns (randomly selected) are shown here.

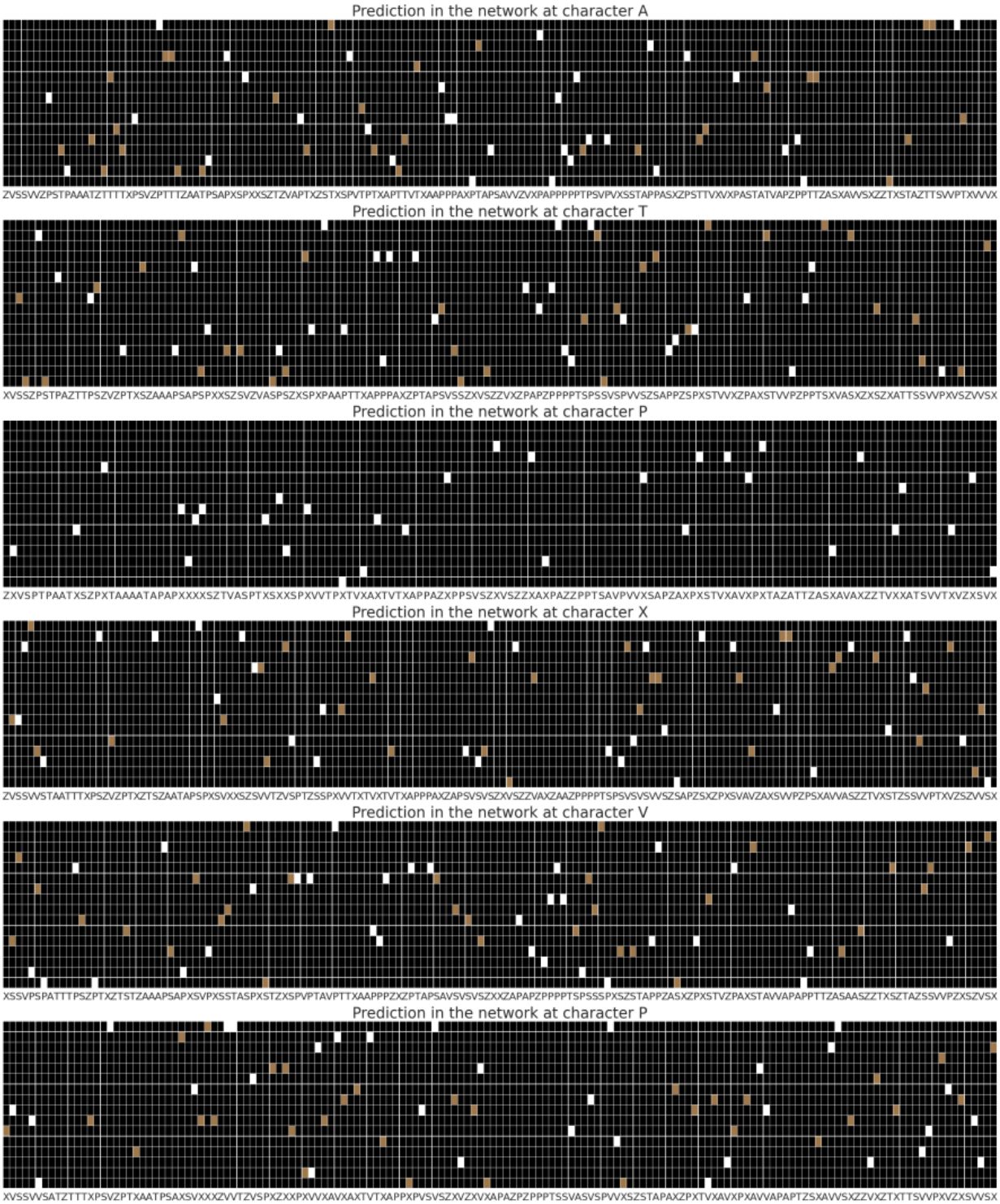


Fig. 20(b.1). Predictions in the network for the first 6 letters "*ATPXVP*" of the 1851st string <*ATPXVPXVSPVZ*>. Cells of the same color in any particular panel signify predicted cells for a single letter. For clarity sake, only around 40% of the unpredicted columns (randomly selected) are shown here.

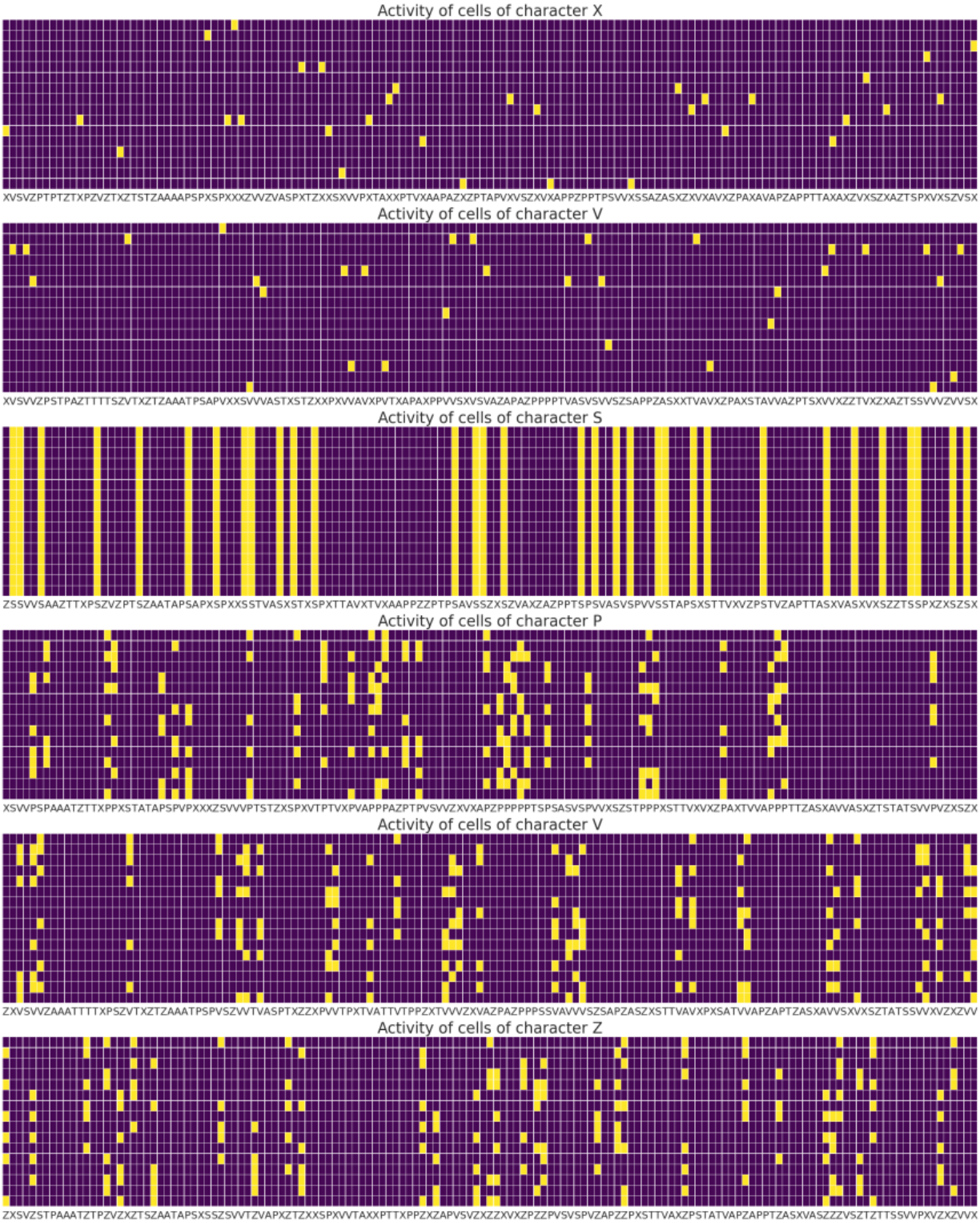


Fig. 20(a.2). Activations in the network for the last 6 letters "*XVSPVZ*" of the 1851st string *<ATPXVPXVSPVZ>*.

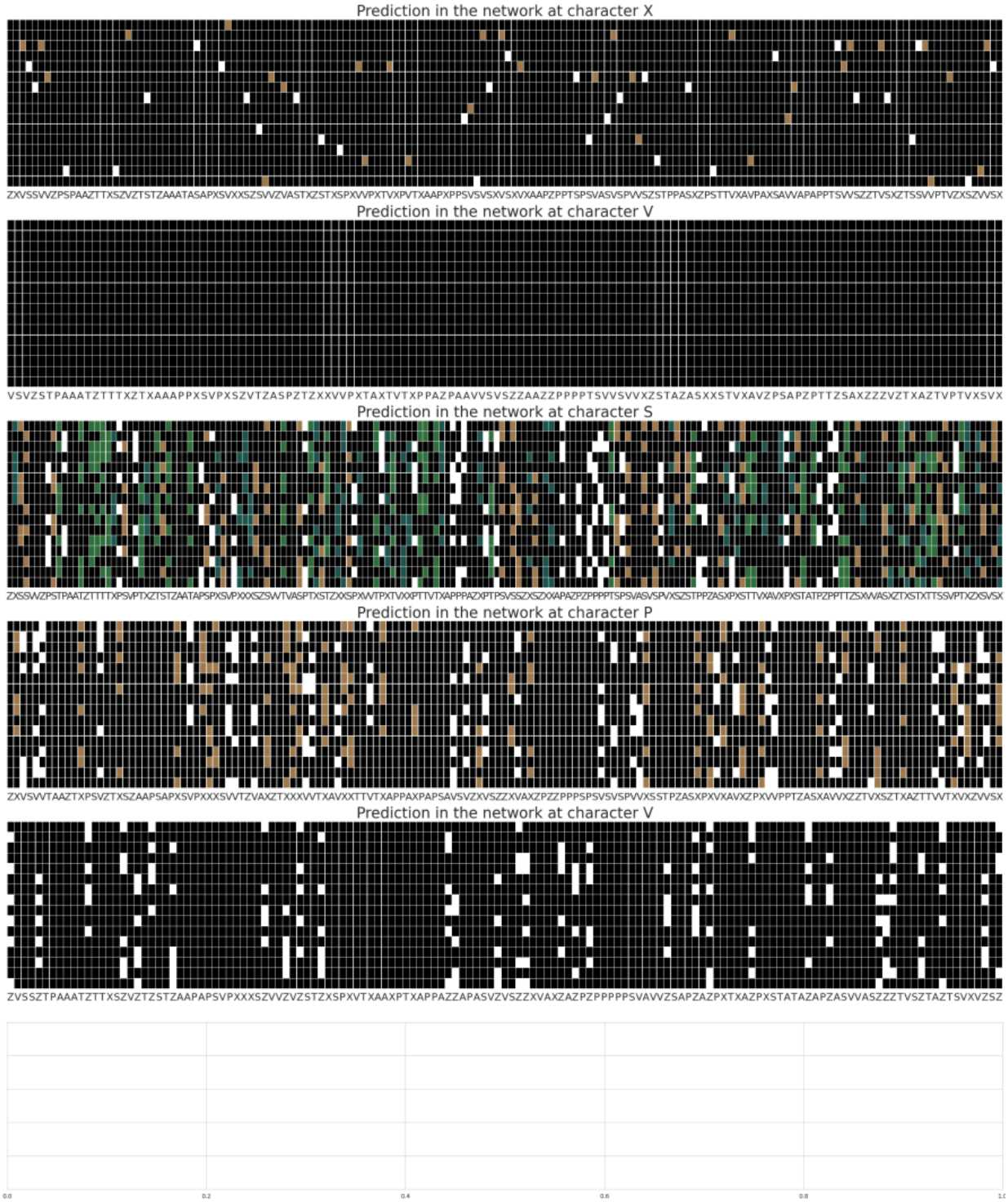


Fig. 20(b.2). Predictions in the network for the last 6 letters "*XVSPVZ*" of the 1851st string <*ATPXVPXVSPVZ*>. Cells of the same color in any particular panel signify predicted cells for a single letter.

Furthermore, the network often fails to correctly predict half of the expected minicolumns, leading to a 50% PPR score for around 1000 timesteps in the inputstream (see Fig. 19 (middle)). Upon further inspection, it is revealed that this is also a ramification of the network's ability to continually learn newer SDRs for every letter, as and when it appears with a newer temporal history in the context of the entire Reber string and regardless of its occurrence in the context of a certain trigram.

From the results so far, it seems likely that HTM does not learn the Simple Reber Grammar by acquiring its trigrams (see chapter II, II.2.1); but instead learns and stores every possible transition in the SRG in the temporal context of the entire Reber string. The next two subsections give some concrete evidence supporting these claims.

III.3.3 HTM Does Not Learn / Acquire Trigrams of SRG

As mentioned earlier in chapter II, II.2.1, there are 33 trigrams possible in the Simple Reber Grammar. In other words, if HTM were to learn the SRG by acquiring only these 33 trigrams, then the total number of transitions learned by the HTM network would be 33 only. In other words, the HTM network would only need to learn 33 different SDRs and correspondingly grow 33**k* dendrites to store them[24]. Additionally, since the current study utilizes a learning mechanism where same SDRs are formed for consecutively repeating letters in a Reber string (see chapter II, II.9.5), the number of distinct SDRs required are only 31(=33-2). This is because the network does not learn a separate SDR for the trigrams *"TTT"* and *"SSS"*.
Hence, if the HTM model were to *only acquire the trigrams of SRG*, there would be a total of 31*32 = 992 dendrites "grown" in the network (since *k*=32 in this study, see section II.12).

In Fig. 21 (top), the total number of dendrites in the network (averaged over the 10 trials), after exposure to each Reber string in the inputstream, is shown. Clearly, the network grows almost double the requisite number of dendrites.

[24] Since every letter of the Reber Grammar is encoded using *k* minicolumns, every dendrite in the network – storing a certain transition in the Reber Grammar from one letter to the next – has *k* copies, one each in every *k* minicolumn of the particular letter. For an illustration, see Fig. 11 in subsection III.1.3.

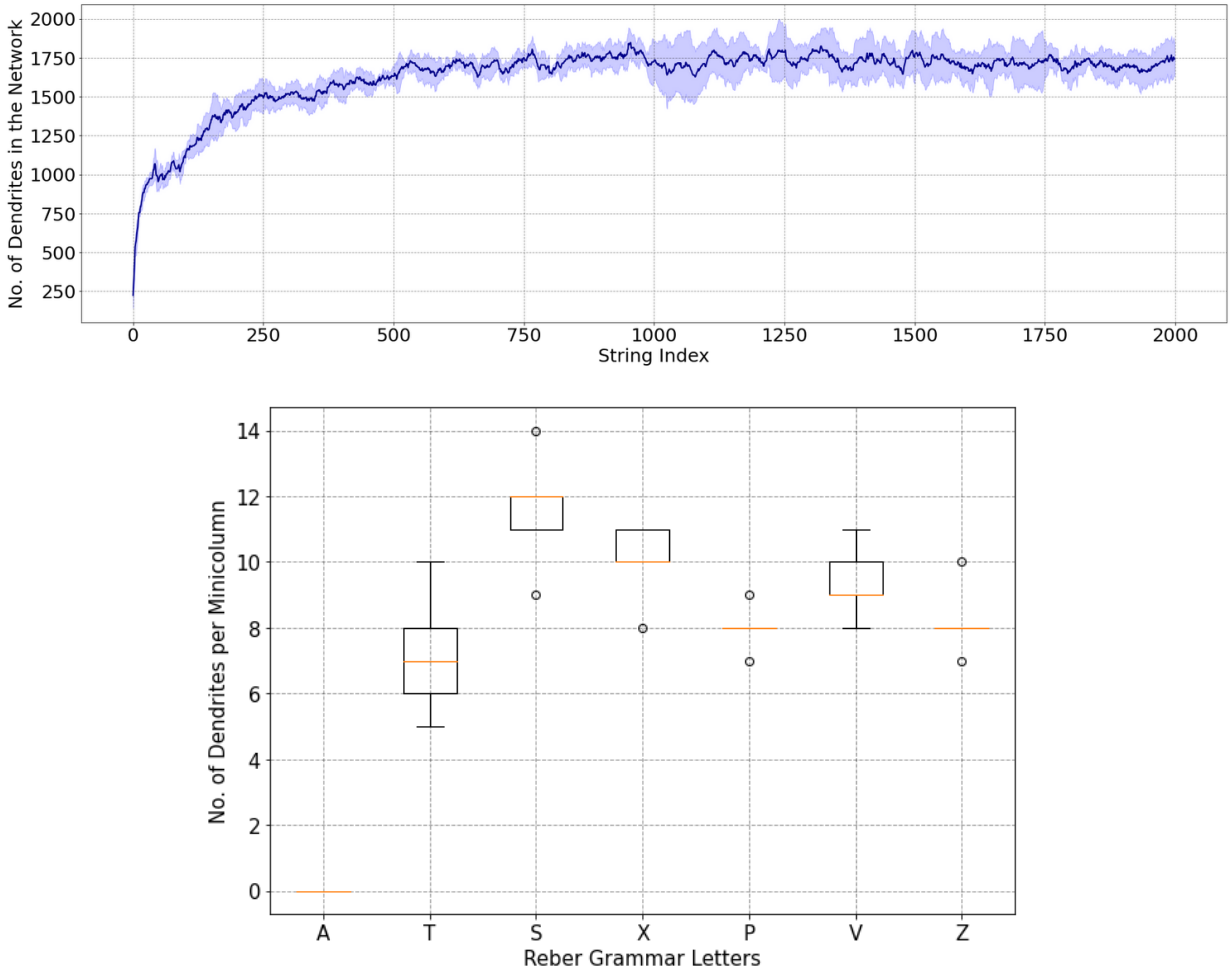


Fig. 21. *Top:* Number of dendrites in the HTM network (averaged over 10 trials) as it is exposed to an inputstream consisting of 2000 randomly generated Reber strings, with around 259 unique Reber strings in the set. The network grows approximately 1750 dendrites by the end, even though the second-order process of SRG requires only 992 dendrites in the network in total. *Bottom:* Number of Dendrites per minicolumn of each of the seven letters in the Reber Grammar. The orange horizontal bar depicts the median count. Since the $Z \rightarrow A$ transition is never learned, minicolumns of *"A"* do not have any dendrites initialized on any of their columns. On an average, 9 dendrites are grown per minicolum of every letter of the grammar, excluding *"A"*; `when only an average of around 5 (≈(992/6)/32) dendrites per minicolumn of every letter is required to learn the SRG.`

Fig. 21 (bottom) shows the number of dendrites "grown" per minicolumn for each letter of the Reber Grammar. To take an example, the network has learned a total of 12 different contexts for the letter *"S"*, on average. However, there are only seven different trigrams (*"ATS"*, *"TSS"*, *"VSS"*,

"PVS", *"XVS"*, *"APS"*, *"PXS"*) that represent the letter *"S"* in all possible contexts. Similarly, the network learns more than the necessary number of SDRs for other letters in the grammar too.

In conclusion, it can be said that HTM is unable to acquire the exact set of contextual representations to correctly learn the Simple Reber Grammar. In simpler terms, the network fails to acquire the "lexicons" of the Simple Reber Grammar and to identify the (second-order) "syntax" of the Reber grammar (see subsection II.2.1).

III.3.4 HTM Attempts to "Memorize" Sample Reber Strings

In this subsection, the HTM network is again exposed to another inputstream consisting of 2000 randomly generated SRG strings (distribution of the string length is shown in Fig. 22 below). However, this time no dendrites are pruned in the network. In other words, the maximum dendrite dormancy in the network is set to infinity.

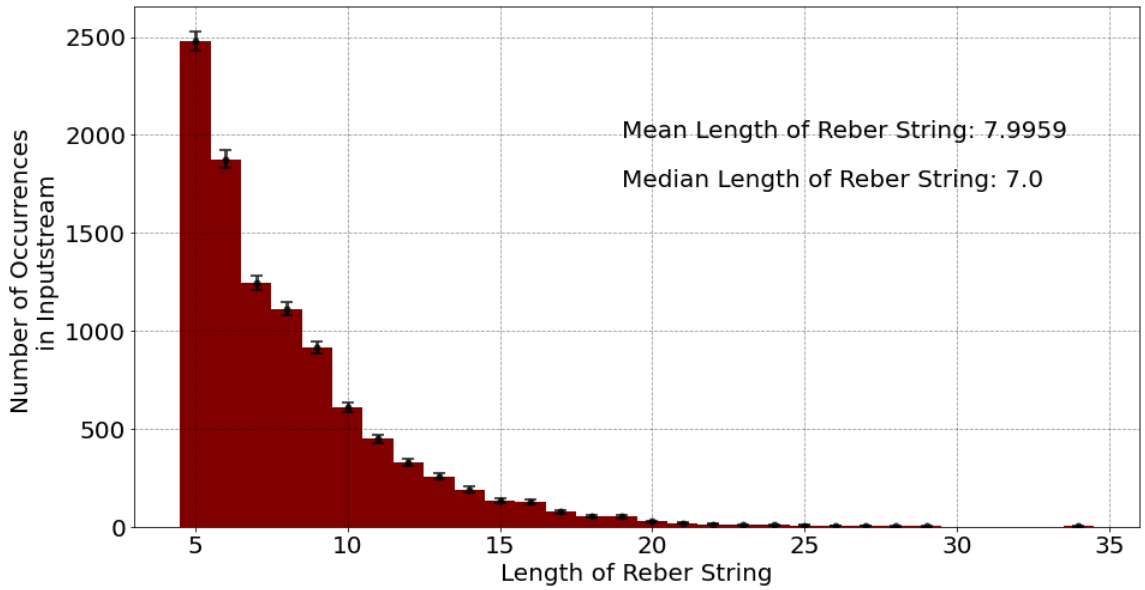


Fig. 22. Distribution of the length of an individual Reber string in the 10 different sets of 2000 randomly generated Reber strings which the HTM network is exposed to for learning of the Simple Reber Grammar in this subsection.

If the HTM network actually learns SDRs in the *context of the entire Reber string*, then it can be hypothesized that it would keep growing the number of stored SDRs / transitions in the network (unrestrictedly, until it reaches the limit set by 'max. distal dendrites per cell' (see section II.12)) as it gets exposed to the inputstream since there are potentially an infinite number of Reber strings possible. A different way of stating the previous hypothesis is to say that HTM attempts to "memorize" the example Reber strings from the inputstream; thereby, resulting in a continual increase in the number of stored SDR patterns as exposure to new input strings continues.

As depicted in Fig. 23, evidence from the simulation clearly upholds the previous claim. The HTM network continues to grow new dendrites (with an upward trend of growth, for longer inputstreams) – and learn new SDRs – as it is exposed to sequential transitions in the inputstream.

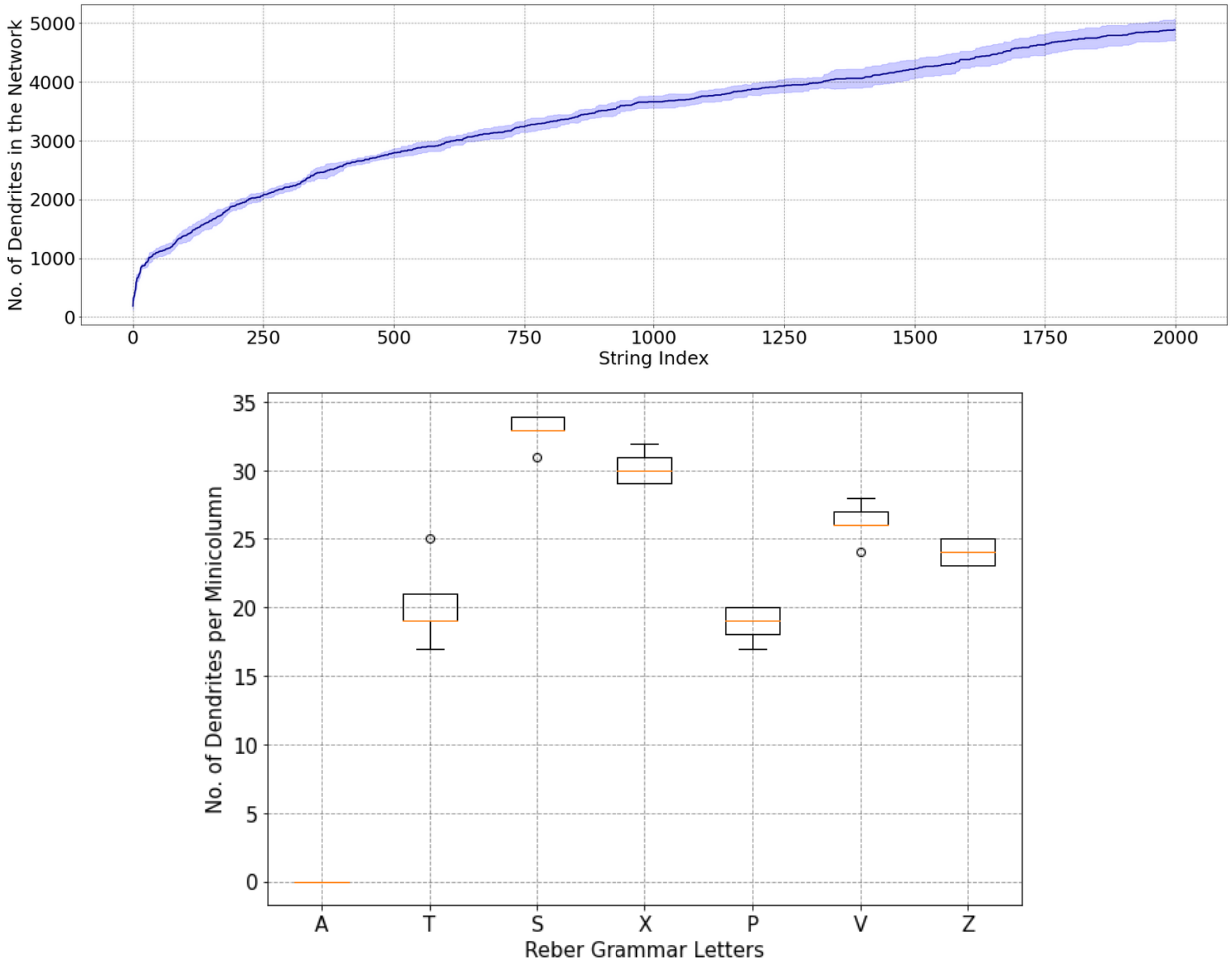


Fig. 23. *Top:* Number of dendrites in the HTM network (averaged over 10 trials) as it is exposed to an inputstream consisting of 2000 randomly generated Reber strings. With 'maximum dendrite dormancy' set to infinity, the number of dendrites in the network keep on growing until the end, which supports the hypothesis that HTM attempts to "memorize" the input Reber strings, instead of learning the "syntax" (second-order) of Reber Grammar. *Bottom:* Number of Dendrites per minicolumn of each of the seven letters in the Reber Grammar. The orange horizontal bar depicts the median count. Minicolumns of *"A"* do not have any dendrites initialized on any of their columns since the $Z \rightarrow A$ transition is never learned. On an average, 25 dendrites are grown per minicolum of every letter of the grammar, excluding *"A"*, which is five times the requisite average value of 5.

In addition, the growth in the number of dendrites is directly proportional to the prediction performance of the network. As the number of dendrites grow unrestrictedly, the prediction performance of the network also improves gradually, as shown in Fig. 24. The network attains a

P3S score of approximately 99% and an average PPR and PAR score of 98% each, towards the end of the inputstream.

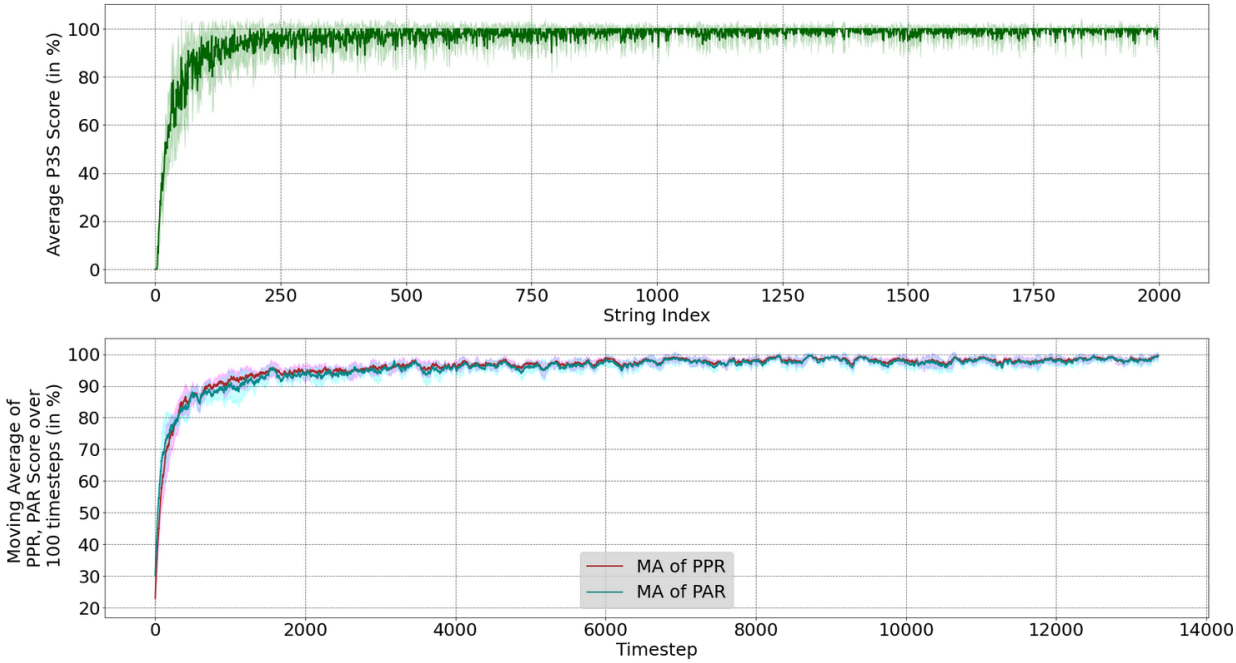


Fig. 24. *Top:* P3S score (in %) of HTM, averaged over 10 trials, as the network is exposed to a randomly generated set of 2000 Reber strings in the inputstream[25]. The network achieves an average of over 99% P3S score in the final 10% of the inputstream. *Bottom:* 100-timesteps moving average of PPR and PAR scores (in %) of the network, averaged over the 10 trials. The averaged PPR and PAR scores of the network reach approximately 98% in the last 10% of the inputstream.

From the experimental evidence presented in this and the previous section, it can be justifiably concluded that *HTM memorizes sample Reber strings, instead of extracting the second-order "syntax" of the artificial Reber Grammar.* As a result of this, the HTM model learns the Reber Grammar in an approximate fashion only.

[25] As stated earlier in the footnote of Fig. 18, the overshooting of the spread of error beyond 100% at some string indices is due to the fact that each trial has a different string at any given index.

CHAPTER IV: DISCUSSION & OUTLOOK

IV.1 Conclusions from the Study & Future Outlook

The thesis aimed at modeling the mechanisms of implicit acquisition of sequential information in the biological brain. Accordingly, a brain-inspired neural network, deploying biologically plausible Hebbian learning rules, was utilized to implicitly learn nontrivial / higher-order sequences.
The used HTM model was shown to successfully perform sequence learning in an implicit and online fashion; simply by exposure to transitions in the inputstream. HTM achieved this by using two very general computational and structural aspects of the brain: first, HTM exploited sparse and distributed encoding of temporal information in the sequence in the activation patterns of neurons arranged vertically in minicolumns; and second, HTM utilized the nonlinear properties of dendrites detecting synchronous activity, thereby enabling a single neuron to recognize several distinct activation patterns in neuronal ensembles.

While the HTM model could learn higher-order sequences – such as those in the Simple Reber Grammar – using sparse distributed representations, it is evident from the results in this study (see chapter III, section III.5) that HTM lacks a mechanism to learn certain long-range dependencies as single, constituent units in the sequence. In more technical terms, the previous statement signifies that HTM lacks mechanisms to process and learn non-adjacent, recursively embedded dependencies such as the ones found in *context-free grammars* [82].
This drawback of HTM is also the reason behind its failure to acquire the second-order nature of the Reber Grammar. (see subsections III.3.2 and III.3.3).

As found in this study, this limitation of HTM is clearly due to the flaw in the learning strategy used (to acquire sequential transitions in any inputstream), where the HTM network utilizes activation patterns from the previous timestep only, in order to infer the temporal context of the sequence. The HTM network is, therefore, *exclusively* dependent upon the different possible combinations of

sparse representations[26] in the network in order to learn the temporal context of an incoming input. As a result, HTM can also be called a variable-order sequence memory network.

Additionally, as a direct consequence of the stated drawback above, it can be theorized that HTM will also fail in acquiring / learning the more complex Embedded Reber Grammar (ERG), a benchmark for testing an algorithm's ability to learn long time-lag dependencies [9]. An ERG is an extension of the SRG that generates two types of strings: <*A****T***<*SRG String*>***TZ***> and <*A****P***<*SRG String*>***P****Z*>, with long time-lag dependencies marked in bold. Based on the experimental evidence from this study (section III.2), it is rather obvious that HTM would fail to discern (distinguishably) the long-range dependencies correctly; and instead, it would try to memorize the ERG strings as they are, just like it did for the SRG strings.

Unsurprisingly enough, humans have been known to demonstrate a capacity (innately) to perceive and learn complex, recursively embedded, hierarchical structures (in language) and other types of long-range dependencies [78, 79], and hence, it stands to reason that implicit acquisition of long time-lag dependencies should also be tractable via a computational framework like HTM which is architecturally and algorithmically inspired by the biological brain.
In particular, a potential research direction in this regard is to integrate hierarchies of HTM regions to process time-varying inputs on different timescales. Recall from chapter II, section II.3, that the HTM network used in this study is actually a "region" of the HTM model. A prospective approach forward could, thus, come from integrating several such regions on top of one another in a hierarchical fashion with outputs of lower-order regions – learning transitions at a smaller timescale – becoming inputs of higher-order regions – learning long time-lag transitions and/or recursively nested temporal dependencies – along with a certain mechanism of information transfer across these different hierarchies via feedforward and feedback connections. Although this idea seems promising, it has not been explored in any sufficient detail as it is beyond the scope of the current study.

[26] See subsection IV.3.1 for details on the exact representational capacity of an HTM network.

IV.2 HTM as a Promising Modeling Framework for Other Cortical Phenomena

Despite the aforementioned drawbacks of HTM in modeling the brain's neuronal mechanisms for implicit organization of sequential information, HTM still seems to be a promising computational framework for explaining a wide range of cortical phenomena and for exploring several other advanced abilities of the neocortical microcircuit within the sensory cortex. Some of these prospects are discussed in the following.

IV.2.1 HTM Aligns with the *Synaptic Clustering* Hypothesis of Memory Formation in the Brain

The idea of using active dendrites framework in HTM, with in-branch localization of synapses detecting a certain SDR pattern of activation, was postulated to be a likely candidate for computational and memory storage in the brain in an earlier review by Kastellakis et al., 2015 [12]. In the review, it was concluded with substantial evidence that *synaptic clustering* – clustering of synapses on a dendritic branch based on synchronicity of presynaptic activations – on individual dendritic branches should play a key role in any theory of memory formation in the brain.

IV.2.2 SDRs in HTM are Robust & Generalizable; and Provide Explanation for Ubiquitous Sparsity in the Brain

From studies conducted on macaques and cats, it has been observed that upon repeated stimulation by a time-varying, sequential stream of sensory input, cells in the primary visual cortex of the animals exhibited highly sparse responses suggesting that the sparse neuronal activity is highly informative and redundancy in a cortical column is low (Yen et al., 2007 [68], Vinje & Gallant, 2002 [69]).

This empirical study is in direct agreement with the foundational principle behind the use of SDRs in HTM to encode temporal contextual information in highly sparse activation patterns in neocortical minicolumns, as learning in the network goes on.

Besides, in the machine learning literature also, it was recently found in Clay et al., 2020 [66] that semantically meaningful and stable representations of a continuously streamed 3D environment were learned in the form of sparse activation patterns in the network – in an entirely unsupervised manner – by a deep reinforcement learning agent trained on sparse rewards. Besides arising spontaneously in the process, the sparse representations were found to be more robust to noise and better for generalizability.
HTM was also demonstrated to be quite robust to noise and highly tolerant to faults in the network in Cui et al., 2016 [1].

IV.2.3 HTM utilizes Binary Synaptic Weights which are Better-Suited for Long-term Memory Storage in the Brain

There is also evidence to suggest that biological synapses are rather more stochastic in their efficacy than being stable, precise and finely-graded in the long-term (as they are commonly found in the mainstream connectionist approach [42, 43]).

In Faisal et al., 2008 [40], it was found that there is a substantial amount of variability in synaptic efficacy between neurons in the brain due to cellular noise and/or the recent activation history of the synapses. Hence, a direct mapping of precise numerical weights from a connectionist network onto synapses in the brain would become inconsistent [41].

Likewise, evidence was found for a low-resolution of synaptic weight values on longer timescales (Petersen et al., 1998 [44]). The study focussed on hippocampus and inquired whether the synaptic efficacy of individual synapses is controlled by activity in a graded or all-or-none manner. Experimental evidence strongly suggested that changes in synaptic strengths required for inducing long-term potentiation (LTP) were relatively abrupt (all-or-none)[27]. In other words, individual

[27] It is well worth noting that there are two ways in which this all-or-none mechanism of changes in synaptic efficacy can help the brain in solving the common problems related to long-term information storage in a *noisy* world, as the authors of the above study [44] also note. To put concisely, the first point is that memory elements that store information in a binary (all-or-none) format are much more robust to noise since sufficiently frequent repeated restoration after perturbations caused by the presence of cellular noise of generic magnitude can preserve "digital" information over an arbitrary length of time in a noisy environment. Second, storage of information in a "digital" format can also resolve the problem of not storing information

synaptic connections are scarcely more than binary-valued connections when it comes to information storage and the development of neuronal connectivity on long timescales.

The above studies together suggest that assuming precisely-graded connection strengths between whole neurons may not be the right way to look at how the brain stores learned information[28]. Like before, the use of binary synaptic weights (controlled via finely-graded permanence values on which Hebbian learning is applied) in HTM is evidently supported by these previous arguments.

IV.2.4 HTM's Superlinear Dendritic Integration Could Potentially Solve the Feature-Binding Problem

In an interesting modeling study by Legenstein & Maass, 2011 [71], it was demonstrated that a spiking neuron model – with nonlinear summation of synaptic input at dendritic branches and standard STDP-based learning – could solve the *feature binding problem*[29] with superlinear dendritic integration. Specifically, the study showed that *specific* dendritic branches can *specialize* in recognizing a given combination of neuronal ensemble activities – where the activity of the

---

under wrong conditions by having an appropriate threshold for deciding whether to store the information or not. The evidence from the study suggests that there is indeed a real threshold for potentiation at individual synapses which can, presumably, be reached only with suitably correlated (temporally) pre- and post-synaptic activity and would be very difficult, otherwise, to be modified by irrelevant background neural noise.

[28] Apparently, this is a probable reason for the need of external supervision in SORN, in [17], for learning higher-order sequences (see chapter I, I.3.2 and I.3.3). Furthermore, because the idea of binary synaptic weights for storing long-term memory was not realized in the previous studies of [17] and [50], they ended up using other biologically implausible weight normalization techniques like *synaptic normalization* to counter uncontrolled seizure-like synchronous bursts of activity due to unbounded growth of synaptic weights via STDP alone (see [45] for details).

[29] The feature binding problem is a nonlinear problem of integrating information of an object/event stored in a distributed fashion over a population of neurons or independent brain modules into unitary percepts. For instance, assume that part of a brain encodes information on colors – say, yellow and red – in highly distributed neuronal ensembles; and another part of the brain encodes information on shapes – say, circle and triangle – in a similar fashion. The feature binding problem is then the problem of how the brain combines different features of the sensory input like color, shapes, sounds, etc. into a single unique perception or experience.
It must be noted that features need not be delineated categories only but could also be any continuously-graded percept with varying degrees of perception. In the current study, relevant features of symbols in the inputstream are: the letter/symbol itself and its sequential position in the Reber string.

neurons representing different features was either rate-based (uncorrelated firing rates) or synchronous (like in the current study).

This is exactly the working principle behind HTM's use of active dendrites, as explained in chapter I, I.4.2, where spike-coincidence based pattern discrimination in an ensemble of neurons, with superlinear dendritic integration, is utilized to bind features ("object + sequential position") of the inputs in the inputstream.

While the different features encoded by distinct neuronal ensembles in [71] were categorical (shapes and colors, see footnote 29 before), it is undoubtedly also possible for the features to be temporal – as they were in this study's AGL task.

IV.3 Other Computational Aspects of HTM

IV.3.1 Storage & Representational Capacity of the Network

When talking about the capacity of an HTM network, there are two important aspects to consider. On the one hand is the ***storage capacity*** which denotes how many sequential transitions an HTM network of a given size can store. This is essentially the *number of different SDRs that an HTM network can learn and store.*

On the other hand, there is a ***representational capacity*** which denotes the *number of SDRs that an HTM network can create and use to represent distinct sequential elements with distinct temporal contexts.*

Storage Capacity

The storage capacity can be calculated in a straightforward fashion by counting the number of distinct SDR patterns that a neuron can recognize and multiplying that with the total number of neurons in the network, discounting the ones which are copies of the same neuron, but in a different minicolumn. Thus,

$$Storage\ Capacity = \frac{Max.\ number\ of\ distal\ dendrites\ per\ cell * M * N}{k} \quad \text{… (Eq. 10)}$$

The storage capacity scales linearly with the number of dendrites per neuron and the number of cells per minicolumn. It must be noted that the use of 'max. number of distal dendrites per cell' as proxy for the number of distinct SDRs on a cell is based on the assumption that every dendrite stores only 1 distinct SDR – as it was in this study.

Representational Capacity

The representational capacity of an HTM network is also easy to calculate. If there are k minicolumns to represent a given input symbol, with M cells in each minicolumn, then – assuming that only 1 cell per minicolumn is used to represent the symbol in a certain sequential context – the particular input symbol can be represented differently in M^k ways. Thus,

$$Representational\ Capacity\ =\ M^k \qquad \ldots \text{(Eq. 11)}$$

Remarkably, the representational capacity of an HTM network is practically unlimited for any large HTM network. It is also noteworthy that this prodigiously large representational capacity of the network (endowed by the architecture of the network) is the reason behind HTM's ability to store any variable-order temporal context and learn very long-range temporal dependencies in the inputstream.

For the parameter values used in this study, the storage and representational capacities of the HTM network are 14,336 and (more than) 3.4×10^{28} respectively.

IV.3.2 Reliability of Pattern Detection using SDRs

Sparse Distributed Representations provide the HTM network with the ability to reliably recognize a pattern using only a subsampled version. In other words, the HTM network can recognize a large [30] and distributed activation pattern by matching only a small subset of the active cells in the larger pattern. Because SDRs are sparse and distributed, it makes robust recognition possible with only 8-20 synapses [13]. A quantitative argument in support of the previous statement is presented below.

[30] Involving a lot of neurons from the population.

Consider a population of, say, 2000 HTM neurons in a network, where SDRs comprise around 2% (40) of the cells at any given time i.e., there are only about 40 cells active at any given point in time. Also, if a single dendrite of a neuron in the network forms connected synapses to, say, 20 of the 2000 active cells, and the threshold for generating an NMDA spike is at 10, then the dendrite will detect an SDR pattern when any 10 of the 20 synapses receive presynaptic activation simultaneously. An HTM neuron should be able to *correctly* detect when a particular SDR occurs in the population.

It is reasonable to believe that the dendrite could falsely detect many other random SDR patterns that share the same 10 active cells and hence lead to a wrong prediction. However, due to sparsity, the probability of a false match for a different random SDR pattern using the same 10 cells is very small – approximately 4.94363 x 10^{-13}, to be exact.
This probability of a false match is computed using the following formula:

$$\frac{\sum_{b=\theta}^{s} \binom{s}{b} \times \binom{n-s}{a-b}}{\binom{n}{a}}$$

… (Eq. 12, adapted from [13])[31]

where, n represents the total number of cells in the population; a represents the number of active cells at a given time, for sparse patterns $a << n$; s represents the number of connected synapses on a dendritic segment and θ is the NMDA threshold.

For the parameter values used in this study, the probability of a false match is approximately 3.4 x 10^{-27}.

IV.4 Other Limitations of HTM

Some other limitations of the HTM model are listed below:

[31] For a derivation of the equation, refer to [74].

- The HTM model utilizes the dendritic structure only for having independent, locally-thresholding subunits for pattern recognition and completely ignores other global nonlinearities and modulations that are evident in dendrites in the brain, such as the ones stated in London & Hausser, 2005 [18].

- Besides other biologically-inspired mechanisms of inducing long-term potentiation and depression in synapses, the neuronal model with dendrites used in Legenstein & Maass, 2011 [71] is closer to mimicking real biological neurons since it contains both supralinear and linear integration of presynaptic activity in the neuron, as found experimentally by Poirazi et al., 2003 [48]. On the other hand, the HTM model currently does not have any mechanism for linear integration of synaptic input and a neuron in HTM only exhibits supralinear integration via spike-coincidence detection [13]. While it was argued in chapter I, I.4.2 that spike-coincidence detection may be a more significant mode of operation for neurons relaying temporal information, it is certainly a disadvantage to not have neurons which can linearly integrate their inputs, especially when the neural code is mediated by average firing rate [48].

References

[1] Cui, Y., Surpur, C., Ahmad, S., and Hawkins, J. (2016). Continuous online sequence learning with an unsupervised neural network model. *Neural Computation 28, 2474-2504.* doi:10.1162/NECO_a_00893.

[2] Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain.* Apr;120 (Pt 4):701-22. doi: 10.1093/brain/120.4.701. PMID: 9153131.

[3] Hawkins, J. et al. (2016-2020). Biological and Machine Intelligence. Release 0.4. Accessed at https://numenta.com/resources/biological-and-machine-intelligence/.

[4] Reber, A. S., Walkenfeld, F. F., & Hernstadt, R. (1991). Implicit and explicit learning: individual differences and IQ. *Journal of experimental psychology. Learning, memory, and cognition, 17(5), 888–96.*

[5] Petersson, K. M., & Hagoort, P. (2010). The neurobiology of syntax: beyond string sets. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences, 367(1598), 1971–83.*

[6] Altmann, G. T. M., Dienes, Z., & Goode, A. (1995). Modality independence of implicitly learned grammatical knowledge. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 21*(4), 899–912.

[7] Mathews, R. C., Buss, R. R., Stanley, W. B., Blanchard-Fields, F., Cho, J. R., & Druhan, B. (1989). Role of implicit and explicit processes in learning from examples: A synergistic effect. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 15*(6), 1083–1100.

[8] Gers F. A., Schmidhuber J., Cummins F. Learning to Forget: Continual Prediction with LSTM. (2000). *Neural Computation 12, 10, 2451-2471.* doi:https://doi.org/10.1162/089976600300015015

[9] Hochreiter S., Schmidhuber J. (1997). Long short-term memory. *Neural Computation, 9(8):1735–1780.*

[10] Dehaene, S., Meyniel, F., Wacongne, C., Wang, L., and Pallier, C. (2015). The neural representation of sequences: from transition probabilities to algebraic patterns and linguistic trees. *Neuron 88, 2–19.* doi: 10.1016/j.neuron.2015.09.019

[11] Long, L. N. (2016). Toward human-level massively-parallel neural networks with Hodgkin-Huxley neurons. In B. Steunebrink, P. Wang, & B. Goertzel (Eds.), *Artificial General Intelligence - 9th International Conference, AGI 2016, Proceedings* (pp. 314-323).

[12] Kastellakis, G., Cai, D.J., Mednick, S.C., Silva, A.J. & Poirazi, P. (2015). Synaptic clustering within dendrites: an emerging theory of memory formation. *Prog. Neurobiol. 126, 19–35.*

[13] Hawkins, J., and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Front. Neural Circuits 10:23.* doi: 10.3389/fncir.2016.00023

[14] Polsky, A., Mel, B. W., and Schiller, J. (2004). Computational subunits in thin dendrites of pyramidal cells. *Nat. Neurosci. 7, 621–627.* doi: 10.1038/nn1253

[15] Branco T, Clark BA, Häusser M. (2010). Dendritic discrimination of temporal input sequences in cortical neurons. *Science 329:1671–75.*

[16] Li S., Liu N., Zhang X., et al. (2019). Dendritic computations captured by an effective point neuron model. *Proc. Natl Acad. Sci. USA 116, 15244–15252.*

[17] Duarte R., Series P., Morrison A. (2014). Self-organized artificial grammar learning in spiking neural networks. In: *Proceedings of the 36th Annual Conference of the Cognitive Science Society. p. 427–432.*

[18] London M., Hausser M. (2005). Dendritic computation. *Annu. Rev. Neurosci.* 28, 503–532.

[19] Poirazi P., Mel B. W. (2001). Impact of active dendrites and structural plasticity on the memory capacity of neural tissue. *Neuron 29, 779–796.*

[20] König P., Engel A. K., Singer W. (1996). Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends Neurosci. 19, 130–137.*

[21] Takahashi N, Kitamura K, Matsuo N, et al. (2012). Locally synchronized synaptic inputs. *Science*.335(6066):353-356.

[22] Fu M, Yu X, Lu J, Zuo Y. (2012). Repetitive motor learning induces coordinated formation of clustered dendritic spines in vivo. *Nature*. 483(7387):92-5.

[23] Aswolinskiy W, Pipa G. (2015). RM-SORN: a reward-modulated self-organizing recurrent neural Network. *Front. Comput. Neurosci.* 9:36. doi: 10.3389/fncom.2015.00036

[24] Lang, K., Waibel, A., and Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks, 3:23–43*

[25] Williams R., Zipser D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications, pages 433–486.*

[26] Robinson, A.J. and Fallside, F. (1991). A recurrent error propagation network speech recognition system. *Computer Speech and Language, 5(3), 259–274.*

[27] O'Reilly, R. C., Frank, M. J. (2006). Making working memory work: A computational model of learning in the frontal cortex and basal ganglia. *Neural Computation, 18,* 283–328.

[28] Melloni L., Schwiedrzik C. M., Müller N., Rodriguez E., Singer W. (2011). Expectations change the signatures and timing of electrophysiological correlates of perceptual awareness. *Journal of Neuroscience 31:1386-1396.* doi: https://doi.org/10.1523/JNEUROSCI.4570-10.2011.

[29] Summerfield C., de Lange F. P. (2014). Expectation in perceptual decision making: neural and computational mechanisms. *Nature Reviews Neuroscience 15:745-756.* doi: https://doi.org/10.1038/nrn3838.

[30] Saffran J.R., Aslin R.N., Newport E. L. (1996). Statistical learning by 8-month-old infants. *Science 274:1926–1928.* doi: https://doi.org/10.1126/science.274.5294.1926.

[31] Chomsky, N., Miller, G. A. (1958). Finite state languages. *Information and Control, 1(2), 91–112.* https://doi.org/10.1016/S0019-9958(58)90082-2

[32] Reber, A. S. (1967). Implicit learning of artificial grammar. *Journal of Verbal Learning and Verbal Behaviour, 6, 855-863.*

[33] Gomez R. L., Gerken L. (2000). Infant artificial language learning and language acquisition. *Trends in Cognitive Sciences, 4, 178-186.*

[34] Ben Taieb S., Bontempi G., Atiya A. F., Sorjamaa A. (2012). A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems Applications, 39(8), 7067–7083.*

[35] Lipton Z. C., Berkowitz J., Elkan C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv.1506.00019[cs.LG]*

[36] Greff K., Srivastava R., Koutnik J., Steunebrink B. R., Schmidhuber, J. (2015). LSTM: A search space Odyssey. *arXiv.1503.04069.*

[37] Rao R. P., Sejnowski T. J. (2001). Predictive learning of temporal sequences in recurrent neocortical circuits. In *Proceedings of the Novartis Found. Symp., 239, (pp. 208–229; discussion 229–240).*

[38] Brea J., Senn W., Pfister J.P. (2013). Matching recall and storage in sequence learning with spiking neural networks. *Journal of Neuroscience, 33(23), 9565–9575.*

[39] Petersson K., Grenholm P., Forkstam C. (2005). Artificial grammar learning and neural networks. *In Proceedings of the cognitive science society.*

[40] Faisal A. A., Selen L. P. J., and Wolpert D. M. (2008). Noise in the nervous system. *Nat. Rev. Neurosci. 9, 292–303. doi: 10.1038/nrn2258*.

[41] Maass W., and Zador A.M. (1999). Dynamic stochastic synapses as computational units. *Neural Computation. 11, 903–917.*

[42] Rumelhart D. E., McClelland J. L., the PDP Research Group. (1986). Parallel Distributed Processing: Explorations in the microstructure of cognition. *Volume I. Cambridge, MA: MIT Press.*

[43] LeCun Y., Bengio Y., Hinton G. (2015). Deep learning. *Nature, 521 , 436–444.*

[44] Petersen C.C.H., Malenka R.C., Nicoll R.A., Hopfield J.J. (1998). All-or-none potentiation at CA3–CA1 synapses. *Proceedings of National Academy of Science USA 95, 4732–4737.*

[45] Lazar A, Pipa G, Triesch J. (2009). SORN: a self-organizing recurrent neural network. *Frontiers in computational neuroscience 3: 23.* doi: 10.3389/neuro.10.023.2009.

[46] Antic S. D., Zhou W. L., Moore A. R., Short S. M., Ikonomu K. D. (2010). The decade of the dendritic NMDA spike. *Journal of neuroscience research*, *88*(14), 2991–3001.

[47] Medinilla V., Johnson O., Gasparini S. (2013). Features of proximal and distal excitatory synaptic inputs to layer V neurons of the rat medial entorhinal cortex. *The Journal of physiology*, *591*(1), 169–183.

[48] Poirazi P., Brannon T., and Mel B. W. (2003). Pyramidal neuron as two-layer neural network. *Neuron 37, 989–999. doi: 10.1016/S0896-6273(03)00149-1*.

[49] Major G., Larkum M. E., Schiller J. (2013). Active properties of neocortical pyramidal neuron dendrites. *Annual Review Neuroscience 36, 1–24.* doi: 10.1146/annurev-neuro-062111-150343.

[50] Klos C., Miner D., Triesch J. (2018). Bridging structure and function: A model of sequence learning and prediction in primary visual cortex. *PLoS Computational Biology 14(6): e1006187.*

[51] Miner D., Triesch J. (2016). Plasticity-Driven Self-Organization under Topological Constraints Accounts for Non-random Features of Cortical Synaptic Wiring. *PLoS Computational Biology 12(2):e1004759.*

[52] Isaacson J.S., Scanziani M. (2011). How inhibition shapes cortical activity. *Neuron, 72, pp. 231-243*.

[53] Chen S., Kim A., Peters A. (2015). Subtype-specific plasticity of inhibitory circuits in motor cortex during motor learning. *Nature Neuroscience 18, 1109–1115.*

[54] Anastasiades P.G., Marlin J.J., Carter A.G. (2018). Cell-type specificity of callosally evoked excitation and feedforward inhibition in the prefrontal cortex. *Cell Rep., 22 (2018), pp. 679-692*.

[55] Hsieh L. T., Gruber M. J., Jenkins L. J., Ranganath C. (2014). Hippocampal activity patterns carry information about objects in temporal context. *Neuron*, *81*(5), 1165–1178.

[56] Hsieh L. T., Ranganath C. (2015). Cortical and subcortical contributions to sequence retrieval: Schematic coding of temporal context in the neocortical recollection network. *NeuroImage*, *121*, 78–90.

[57] Kumaran D., Maguire E.A. (2006). The dynamics of hippocampal activation during encoding of overlapping sequences. *Neuron; 49(4):617-29.*

[58] Schendan H.E., Searl M.M., Melrose R.J., Stern C.E. (2003). An FMRI study of the role of the medial temporal lobe in implicit and explicit sequence learning. *Neuron; 37(6):1013-25.*

[59] Kanerva P. (1988). Sparse Distributed Memory, *MIT Press, Cambridge, MA.*

[60] Azevedo F. A., Carvalho L. R., Grinberg L. T., Farfel J. M., Ferretti R. E., Leite R. E., Jacob Filho W., Lent R., Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *J. Comp. Neurol.* 513, 532–541.

[61] Bartol Jr T. M., Bromer C., Kinney J., Chirillo M. A., Bourne J. N., Harris K. M., Sejnowski T. J.. (2015). Nanoconnectomic upper bound on the variability of synaptic plasticity. *Elife, 4.*

[62] Nguyen T. (2010). Total Number of Synapses in the Adult Human Neocortex. *Undergraduate Journal of Mathematical Modeling: One + Two: Vol. 3: Iss. 1, Article 14.*

[63] Vinje W.E., Gallant J.L. (2000). Sparse coding and decorrelation in primary visual cortex during natural vision. *Science*. 287 (5456): 1273–1276.

[64] Hromádka T., Deweese M.R., Zador A.M. (2008). Sparse representation of sounds in the unanesthetized auditory cortex. *PLOS Biol*ogy 6 (1): e16.

[65] Crochet S., Poulet J.F.A., Kremer Y., Petersen C.C.H. (2011). Synaptic mechanisms underlying sparse coding of active touch. *Neuron*. 69 (6): 1160–1175.

[66] Clay V., König P., Kühnberger K.-U., Pipa G. (2020). Learning sparse and meaningful representations through embodiment, *Neural Networks*, ISSN 0893-6080.

[67] Sherrington, C.S. (1906) Integrative Action of the Nervous System, *Yale University Press*.

[68] Yen S.-C., Baker J., Gray C. M. (2007). Heterogeneity in the responses of adjacent neurons to natural stimuli in cat striate cortex. *Journal of Neurophysiology 97*, 1326–1341.

[69] Vinje W. E., Gallant J. L. (2002). Natural stimulation of the nonclassical receptive field increases information transmission efficiency in V1. *Journal of Neuroscience 22,* 2904–2915.

[70] Abeles M. (1982). Role of the cortical neuron: integrator or coincidence detector? *Israel Journal of Medical Sciences, 18(1):83-92. PMID: 6279540.*

[71] Legenstein R., Maass W. (2011). Branch-Specific Plasticity Enables Self-Organization of Nonlinear Computation in Single Neurons. *Journal of Neuroscience, 31(30):10787–10802*.

[72] Spruston N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience 9, 206–221.*

[73] Cui Y., Ahmad S. and Hawkins J. (2017). The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding. *Frontiers in Computational Neuroscience 11:111.*

[74] Ahmad S., and Hawkins J. (2016). How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites. *arXiv:1601.00720 [q–bio.NC].*

[75] Fine S., Singer Y., & Tishby N. (1998). The hierarchical hidden markov model: Analysis and applications. *Machine Learning, 32(1), 41–62*.

[76] Waibel A., Hanazawa T., Hinton G., Shikano K., & Lang K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoust., 37(3), 328–339.*

[77] Durbin J., Koopman S. J. (2012). Time series analysis by state space methods (2nd ed.). *New York: Oxford University Press.*

[78] Rohrmeier M., Fu Q., Dienes Z. (2012) Implicit Learning of Recursive Context-Free Grammars. *PLoS ONE 7(10): e45885.*

[79] Saffran J.R., Senghas A., Trueswell J.C. (2001). The acquisition of language by children. *Proceedings of National Academy of Science, U.S.A. 98, 12874–12875*.

[80] Petersson K. M. (2005). On the relevance of the neurobiological analogue of finite-state architecture. *Neurocomputing, 65-66, 825-832.*

[81] Schmit N. (2000). Lexical chunks. *ELT Journal*, Volume 54, Issue 4, Pages 400–401.

[82] Chomsky N., Schutzenberger M.P. (1963) The algebraic theory of context-free languages. *Computer programming and formal systems. North–Holland, Amsterdam. pp. 118–161.*

[83] Chomsky N. (1956). Three models for the description of language. *IRE Transactions on Information Theory IT-2: 113–124.*

[84] Chomsky N. (1957). Syntactic Structures. *The Hague: Mouton.*

[85] Lobina D.J. (2011). Recursion and the competence/performance distinction in AGL tasks. *Language and Cognitive Processes.*

[86] Lobina D.J., Garcıa-Albea J.E. (2009). Recursion and cognitive science: data structures and mechanisms. *Proceedings of the 31st Annual Conference of the Cognitive Science Society.* Austin, TX: Cognitive Science Society. pp. 1347–1352.

Code Availability

Implementation of the Hierarchical Temporal Memory model along with other conducted experiments of this study can be found at: https://github.com/TaherHabib/working-memory-model. The programming language used is Python 3.8.5. In addition to Python's standard modules, the following packages were also used:

- numpy (1.18.5)
- pandas (1.1.5)
- matplotlib (3.3.1)
- scipy (1.5.2)